

RETI DI CALCOLATORI

Sommario

Introduzione	8
Scopo del corso.....	8
Cos'è una rete	8
Caratteristiche comuni di una rete.....	8
Caratteristiche specifiche delle reti di calcolatori	8
Applicazioni	9
Requisiti.....	9
Connettività	9
Affidabilità	10
Amministratori di reti (management).....	10
Architetture di rete e protocolli	10
Interfacce.....	11
Protocolli	11
Incapsulamento.....	12
Architettura ISO/OSI	12
Architettura TCP/IP	14
Architettura internet.....	15
Network API	16
Modello Client-Server tramite UDP.....	17
Modello Client-Server tramite TCP.....	18
Performance.....	19
Definizione di bandwidth	20
Definizione di throughput	21
Latenza	21
Delay vs throughput	21
Jitter	22
Reti di computer	22
Link e frequenze.....	23
Codifica e simboli.....	23
Teorema di Nyquist-Shannon.....	24
Teorema di Shannon-Hartley.....	25
Tipi di codifiche.....	26
Codifica NRZ.....	26

Codifica NRZI.....	27
Codifica Manchester.....	27
Codifica MLT-3.....	27
Codifica 4B/5B.....	28
Frame.....	28
Protocollo orientato ai byte.....	29
Protocollo orientato ai bit	30
Protocollo PPP.....	30
Errori.....	30
Calcolo delle probabilità.....	31
Detezione degli errori	31
Cyclic Redundancy Check	33
Trasmissione affidabile	36
Protocollo stop-and-wait.....	37
Protocollo sliding window	38
Ethernet – IEEE 802.3	40
10base5.....	41
Fattori di velocità minima per i cavi di rete	43
10baseT.....	43
Reti miste (mixed).....	43
Altre varianti	43
Frame Ethernet.....	44
Formato del pacchetto.....	44
Indirizzi Ethernet	44
Algoritmo di ricezione Ethernet.....	45
Algoritmo di trasmissione Ethernet.....	45
Post collisione – backoff esponenziale.....	47
Efficienza della Ethernet.....	47
Collegamenti wireless.....	49
Spread Spectrum	50
Diverse tecnologie wireless	51
WIFI – IEEE 802.11	53
IEEE 802.11 – Collision avoidance	54
IEEE 802.11 – Formato dei frame.....	57
IEEE 802.11 – Distribuzione.....	58
Bluetooth – 802.15.....	59

BLE (Bluetooth Low Energy)	61
Internet	61
Switching e Forwarding	61
Switch.....	61
Approccio switching and forwarding.....	64
Bridge e Switch LAN	67
Distributed Spanning Tree	69
Internetworking.....	72
Formato dei pacchetti.....	73
Frammentazione e ricomposizione	74
Indirizzi globali	76
IP Datagram Forwarding.....	77
Subnetting	78
Indirizzamento senza classi	79
IP fowarding rivisitato	82
Indirizzi IP per usi speciali (RFC 5735).....	84
Protocolli IP ausiliari (ancillary)	84
Protocollo di traduzione degli indirizzi (ARP)	84
Formato del pacchetto ARP	86
ARP proxy	86
Configurazioni degli host e DHCP.....	87
Protocollo di segnalazione e gestione della rete (ICMP).....	88
Tipi di messaggi di richiesta ICMP	88
Reti IP private e NAT	90
Funzionamento NAT di base (“cono pieno” = “full cone”)	90
Funzionamento NAT di base (“cono pieno” con PAT)	91
Applicazioni dei NAT	91
“Firewall” naturale e port forwarding.....	91
Hole Punching (STUN).....	92
Hole Punching (TURN)	93
Preoccupazioni relative al NAT.....	93
Routing.....	94
Routing vs forwarding	94
Algoritmi di routing.....	95
Vettore di distanza (distance vector)	96
Algoritmo di Bellman-Ford	97

Problema del conteggio all'infinito	98
Protocollo di informazione di routing (RIP)	99
Stato dei collegamenti (link state)	99
Instradamento sul percorso più breve (Dijkstra).....	100
Protocollo Open Shortest Path First (OSPF)	101
Internetworking avanzato.....	103
Instradamento inter-dominio.....	103
Propagazione del percorso (Route Propagation)	104
EGP e BGP.....	105
Integrazione del routing inter-dominio e intra-dominio.....	106
Prevenzione dei loop BGP e policy routing	107
Problemi di BGP	108
IP di nuova generazione (IPv6)	108
Formato del pacchetto.....	108
Priorità	109
Flow label	109
Transizione da IPv4 a IPv6	109
Internet Multicast	110
Multicast in IP	111
Routing multicast basato sulla sorgente (source-based)	112
Instradamento multicast ad albero condiviso per gruppi (group-shared trees).....	113
Protocollo di instradamento multicast a vettore di distanza (DVMRP)	113
Multicast indipendente dal protocollo (PIM)	114
Backbone multicast (MBONE).....	116
Protocolli end-to-end	116
Livello di trasporto	116
Indirizzamento end-to-end in TCP/IP.....	117
Socket.....	117
Modello client/server	117
Tipi di comunicazione	118
UDP.....	119
Calcolo del checksum.....	119
API per i servizi UDP	119
Demultiplexer semplice	120
TCP.....	120
Controllo del flusso vs controllo della congestione.....	121

Problemi end-to-end	121
Segmento TCP	121
Diagramma di stato di TCP	123
Rivisitazione della finestra scorrevole.....	126
Protezione contro il wraparound	127
Mantenere la pipe piena.....	128
Attivazione della trasmissione	128
Sindrome della finestra stupida (Silly Window Syndrome)	128
Algoritmo di Nagle.....	129
Ritrasmissione adattiva e calcolo del timeout.....	130
Prestazioni del TCP	131
Controllo della congestione e allocazione delle risorse	131
Problemi di allocazione delle risorse.....	131
Criterio di valutazione con allocazione efficace delle risorse	133
Criteri di valutazione con allocazione equa delle risorse.....	134
Disciplina di accodamento FIFO con caduta della coda (tail drop)	135
Disciplina di accodamento con accodamento equo	135
Controllo della congestione di TCP	137
Finestra di congestione (congestion window)	138
Meccanismi di evitamento della congestione	143
Rilevamento precoce casuale (RED)	143
Evitare le congestioni basate sulle sorgenti.....	145
Sicurezza della rete.....	146
Obiettivi e terminologie.....	147
Terminologie sulla sicurezza	147
Tipi di attacchi.....	148
Servizi di sicurezza.....	150
Meccanismi di sicurezza.....	151
Modelli per la sicurezza di rete.....	151
Modello del canale insicuro (Dolev-Yao).....	152
Posizionamento della sicurezza	153
Sicurezza nei vari livelli.....	153
Crittografia	155
Crittografia simmetrica	155
Principio di Kerckhoff.....	156
Crittoanalisi.....	156

Cifrari di flusso	158
RC4.....	159
One-Time Pad	160
Cifrari a blocchi.....	160
AES.....	161
Modalità di funzionamento dei cfrari a blocchi	162
Integrità e autenticazione dei messaggi	164
Autenticazione dei messaggi tramite crittografia simmetrica	165
Codice di autenticazione del messaggio (MAC).....	165
Funzioni di hash	165
Algoritmo di Sicure Hash (SHA).....	166
Uso degli Hash Digest nell'autenticazione dei messaggi.....	166
Wired Equivalent Privacy (WEP).....	167
Crittografia a chiave pubblica	169
RSA.....	170
Altri crittosistemi a chiave pubblica.....	172
Sicurezza degli schemi a chiave pubblica	172
Firme digitali.....	172
Certificati a chiave pubblica.....	174
Sicurezza delle e-mail	176
Modello PGP	177
S/MIME (Secure/Multipurpose Internet Mail Extensions).....	179
Posta Elettronica Certificata (PEC).....	179
Gestione delle chiavi e autenticazione delle entità	179
Autenticazione del messaggio e dell'entità	180
Procedure di autenticazione in X.509	180
Diffie-Hellman Key Agreement	182
Attacco Man-in-the-middle	182
Accordo di chiave sicura da stazione a stazione	183
Sicurezza Web.....	183
Sicurezza a livello di trasporto (protocollo SSL/TLS)	183
Architettura TLS.....	184
Protocollo Record.....	185
Protocollo Handshake.....	186
Fase 1.....	186
Fase 2.....	186

Fase 3.....	187
Fase 4.....	188

Introduzione

Scopo del corso

Quando si vuole gestire una rete di calcolatori, nuova o preesistente, non basta avere solo i componenti per farla funzionare, ma bisogna avere anche delle conoscenze e delle tecniche precise. Bisogna quindi sapere chi utilizzerà la rete (utenti, non solo umani) e come, chi la ha progettata e come, nonché bisogna capire quali sono le funzionalità di essa per poter progettare anche delle applicazioni che la utilizzino. In base a queste analisi (complesse) la rete viene realizzata e strutturata, con varie architetture, hardware e software. Questo è l'ambito del corso.

Cos'è una rete

Una rete è composta da più componenti fisiche e non (computer, server, LAN, ecc.), chiamate nodi, messe in comunicazione e in funzione tra loro. Ogni linea di comunicazione e ogni nodo della rete hanno una loro funzione all'interno di essa, per cui scollarne uno significa perdere qualcosa di essa e comprometterla.

Anche le vecchie linee telefoniche del XIX secolo, in cui c'erano delle centraliniste che spostavano i cavi da una cella a un'altra per mettere in comunicazione due utenti sullo stesso canale, possono essere considerate una rete. Un altro esempio di rete sono quelle neurali, in cui più neuroni sono collegati tra loro.

Una rete non è quasi mai un'entità statica, ma è mutevole nel corso del tempo (es. nelle linee telefoniche di una volta in cui i cavi si spostavano da una cella a un'altra). Alcune reti sono resilienti, ovvero se un componente si guasta ve n'è un altro che lo può sostituire (es. rete neurale). Statisticamente, una rete non è mai funzionante al 100%, per cui essa dovrebbe essere sempre in grado di sopportare ai guasti e da questo concetto nascono le reti auto-configuranti.

Caratteristiche comuni di una rete

Una rete è composta da un insieme di nodi (indipendenti) e un insieme di informazioni intercambiabili che formano un grafo e utilizzano canali di comunicazione per interagire tra di loro. Questo grande insieme connesso consente di svolgere dei task che un singolo nodo non è in grado di fare. I nodi possono essere qualsiasi cosa: computer, persone, celle, neuroni, ecc., come anche i canali di comunicazione possono essere elettrici, ottici, chimici, virtuali, ecc.

Caratteristiche specifiche delle reti di calcolatori

Una rete di calcolatori moderna è pensata per essere generale (*general purpose*), quindi adatta a diversi tipi di applicazioni. Di conseguenza non è ottimizzata per una specifica applicazione. Più applicazioni possono coesistere, dunque una rete deve essere flessibile e aperta, ovvero che deve potersi adattare a nuove future feature che possono essere aggiunte dinamicamente al sistema.

Queste esigenze sono nate perché in passato ogni azienda produceva le sue, quindi non erano *vendor independent* e di conseguenza non potevano essere connesse a altre reti se non quelle con componenti solo di quella azienda (componenti private proprietarie). Se l'azienda poi falliva o subiva modifiche, poteva mettere a repentaglio le reti create perché potevano non avere più supporto tecnico e dunque gli utenti si ritrovavano a dover cambiare tutta la rete con altre formate da componenti standard pubbliche (flessibili a altre reti).

Le reti di calcolatori devono avere i loro requisiti, le loro funzionalità, principi generali (segnali, architetture di rete, protocolli, sicurezza) e applicazioni (ISO/OSI, TCP, ecc.).

Applicazioni

Le reti internet funzionano attraverso applicazioni come World Wide Web, email, social network online, streaming di audio e video, condivisione dei file, messaggistica istantanea, ecc. Ognuna di esse appena menzionate è specializzata per svolgere un determinato compito.

Esempio WWW: applicazione principale per Internet, che utilizza ipertesti (documenti, ecc.) generati dinamicamente, URL, HTTP, TCP, messaggi, ecc. Ogni calcolatore utilizza schede di rete per comunicare e scambiare messaggi e informazioni con altri calcolatori, quindi vi sono dei protocolli (TCP/IP) per far sì che riescano ad interagire tra di loro. In realtà, ogni protocollo TCP/IP ha al suo interno degli altri protocolli che definiscono altre informazioni, infatti per una singola comunicazione (una richiesta URL) si necessita di 17 messaggi.

Esempio conferenza video: applicazione multimediale che include streaming audio e video, ipertesti, coordinazione, ecc. In questo caso non si ha un sito o URL, però utilizza le stesse regole e protocollo del TCP/IP.

Requisiti

Per i requisiti si hanno diversi punti di vista in base a chi fa uso della rete, perché vi sono diversi tipi di utenti fruitori di essa e non è detto che siano gli utenti finali. Infatti, tra gli altri utenti (e che interessano a questo corso) vi sono:

- i programmatori di applicazioni, sono gli sviluppatori delle applicazioni di rete e che implementano i Client-Server web, i Client-Server di posta elettronica, le app, ecc. Essi si occupano di tutte quegli applicativi complessi delle reti odierne. Essi devono garantire anche sicurezza, prestazioni, semplicità, banda alta, il minor ritardo, poca latenza, mobilità (un dispositivo deve essere poter spostato con facilità e garantire il servizio anche in luoghi diversi), ecc.
- i designer di rete, ovvero i progettisti della rete, che tramite un budget ristretto devono sapere sfruttare le risorse al meglio possibile, senza guastare l'efficienza del prodotto finale. Devono saper effettuare delle scelte progettuali in modo da garantire la miglior esperienza per l'utente, chiunque esso sia. La rete che dovranno realizzare dovrà essere equa per tutti gli utilizzatori di essa, dovrà rispettare delle regole, dovrà essere affidabile in modo che abbia il minor numero di guasti possibili e che necessiti di poca manutenzione, ecc.
- i provider di rete, che si occupano della manutenzione della rete nel corso del tempo, facendo in modo che continui a funzionare come sempre. Non è detto che siano le stesse persone che la progettano (anche per questo la rete deve essere progettata bene, anche in termini di costi).

Connettività

Ogni nodo è collegato a un altro con un cavo o link, di solito di tipo punto-punto. Questo collegamento può essere implementato con qualsiasi tipo di tecnologia (fibra ottica, cavo di rete, piccione viaggiatore, ecc.). Delle volte si ha un mezzo condiviso (bus) in cui tanti nodi sono collegati tra di loro e in questo caso si dovranno implementare anche dei protocolli di fairness comunicativa in modo che non vi siano sovrapposizioni.

Esistono inoltre delle reti commutate (*switched*), che sono una generalizzazione di quella appena descritta e che implementa tante linee punto-punto che collegano più nodi a dei nodi “speciali” chiamati commutatori o switch, che permettono di far comunicare diverse linee specializzate con altre linee dedicate. Queste reti, una volta collegate tra di loro, formano a livello base dei collegamenti punto-punto, poi vi sono i circuiti delle reti di commutatori e tra queste varie reti di switch vi possono essere collegati, tra altri nodi specializzati, dei router, ovvero degli instradatori (*gateway*). In questo modo, combinando tra loro le reti, si formano delle reti sempre più grandi, infatti una rete di calcolatori può essere definita come una rete ricorsiva con due o più nodi collegati tra loro che possono essere punto-punto o su mezzo condiviso, oppure con due o più reti

collegati da un nodo comunicante in comune. Questa espansione crea normalmente una vera e propria gerarchia di reti (con la possibilità di aggiungere o togliere dinamicamente parti di esse) e che comporta a un problema di indirizzamento delle informazioni attraverso i nodi.

Per i designer di rete e per i provider è importante capire come gestire le risorse e farle comunicare. Esistono diversi meccanismi, come multiplexing/de-multiplexing, oppure il Synchronous Time-division Multiplexing (fasce temporali o di dati trasmessi in slot predeterminati), o il Frequency Division Multiplexing (FDM), ecc. Questi, tuttavia, hanno limitata efficienza e scalabilità, per cui si opta per altri meccanismi, ad esempio il Statistical Multiplexing, in cui i dati vengono trasmessi in base alla domanda di ciascun flusso (*flow*, ovvero pacchetti vs messaggi) che viene gestito tramite FIFO, Round-Robin o code con priorità (Quality-of-Service, QoS), oppure si utilizzano servizi o pattern, che consentono di gestire più pacchetti e canali logici. Questi pattern di servizi sono sostanzialmente di due tipi: comunicazione di flusso (es.: stream video/audio) o di comunicazione ad anagramma/pacchetti (es.: request/reply).

Affidabilità

Nel contesto delle reti ci sono diversi aspetti qualitativi, che non riguardano la funzionalità, ma l'affidabilità. Infatti, vi sono molteplici problematicità tra cui la perdita dei pacchetti, la loro corruzione o duplicazione durante l'invio, oppure gli attacchi alla sicurezza, o i guasti alla rete, o i ritardi della comunicazione, ecc. Per questo motivo si introducono meccanismi (SW) che gestiscono questi errori e cercano di correggerli. Si parla di *bit errors* quando vengono invertiti 0 e 1 e di *burst errors* quando vi sono diversi errori consecutivi.

Amministratori di reti (management)

Gli amministratori di reti hanno il compito di gestire la rete (configurazioni, monitoraggio, riparazioni, ecc.). Spesso le configurazioni di reti molto grandi non sono fatte da un singolo amministratore, né è sicuro che siano esperti (es.: nel caso di home network). Per questo motivo devono essere auto configurabili e resilienti a fallimenti (*failures*).

Architetture di rete e protocolli

Le reti sono di solito sviluppate a strati (*layer*), in quanto sono in generale molto complesse e difficilmente implementabili con un'unica soluzione software/hardware. Gli strati sono software che si sviluppano attorno all'hardware e ai livelli sottostanti, le cui funzionalità diventano più astratte a mano a mano che ci si allontana dall'HW. Ognuno degli strati, quindi, aggiunge nuove funzionalità (API) utilizzabili dai livelli superiori, sfruttando le API e i servizi dei livelli inferiori.

Di solito le pile (*stack*) presentano molteplici livelli, ma supponiamo di avere un'architettura semplificata a quattro strati come in figura. Lo stato inferiore rappresenta l'hardware, ovvero l'aspetto fisico, che contiene

Application programs
Process-to-process channels
Host-to-host connectivity
Hardware

tutte le funzionalità per gestire fisicamente il livello (es.: come viene codificato il segnale, come sono i livelli di tensione, frequenze da utilizzare, regole con cui accedere al mezzo, ecc.) e si occupa inoltre della trasmissione dei dati attraverso il canale fisico. Lo strato immediatamente superiore, è quello che consente di creare la struttura dei dati, di scomporla in pacchetti più piccoli e sfrutta il livello sottostante per l'invio di essi, mentre l'host che riceve farà l'operazione opposta, ovvero quella del riasssemblaggio dei pacchetti e di interpretazione di ciò che ha ricevuto. Lo strato superiore, invece, definisce quali processi devono essere trasmessi, dunque aggiunge delle informazioni al pacchetto da trasmettere. Queste informazioni serviranno poi allo strato finale, quello applicativo, per comunicare all'utente quanto ricevuto/inviato.

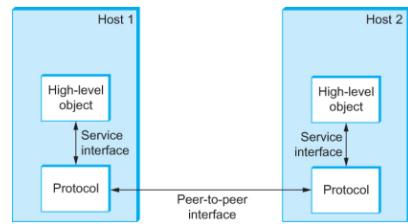
In realtà vi possono essere più protocolli nello stesso livello e differiscono in base al tipo di comunicazione (flusso o pacchetti) e quindi l'architettura dovrà avere API differenti. Ad esempio per request/reply basta avere una comunicazione a pacchetti, in quanto in uno verrà trasmessa una request e l'host provvederà a inviare la reply sottoforma di altro pacchetto al mittente. Mentre, se si vuole trasferire un file (es.: http o mail) si necessita di un servizio affidabile, dunque si richiede l'apertura di una connessione stream/flusso e il servizio viene svolto da un canale di streaming di messaggi, posto al livello del processo (dell'architettura descritta prima).

Application programs	
Request/reply channel	Message stream channel
Host-to-host connectivity	
Hardware	

Interfacce

Un'interfaccia (di protocollo) è l'insieme di funzionalità offerte da ognuno dei moduli/livelli dell'architettura. Nella realtà le interfacce utilizzate sono di due tipi: quella di servizio (API di un livello che viene offerta ai livelli superiori) e quella *peer* (paritaria) per i sistemi distribuiti per poter comunicare tra di loro.

In ogni architettura vi è un'interfaccia uguale, che consente di poter interpretare il messaggio ricevuto dall'altro host.

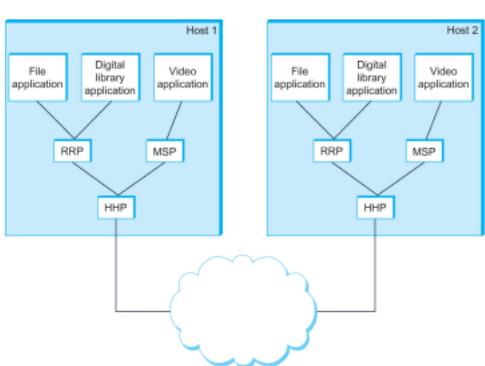


Protocolli

Fondamentalmente un protocollo è un insieme di regole che definiscono le interfacce sia dei livelli superiori, che quelli alla pari (*peer*). Dunque, un protocollo dice sia come questi servizi debbano essere offerti, sia come devono essere utilizzati, quindi si riferisce sia alla struttura verticale per la comunicazione e funzionalità attraverso i livelli dell'architettura, sia per la comunicazione orizzontale dello stesso strato (anche di host differenti). Quindi i protocolli sono i blocchi da costruzione di un'architettura di rete e sfruttano sia le interfacce di servizio, sia quelle peer. In realtà questo termine viene un po' "abusato", in quanto delle volte ci si riferisce anche a come è implementato il servizio di un certo livello, ma non è proprio così.

Quando si ha a che fare con macchine diverse, con architetture completamente diverse e implementate da persone diverse (es.: una è basata su little endian e una sui big endian, o su 32 bit o 64, ecc.), i moduli dei protocolli devono riuscire comunque a "parlarsi e capirsi", quindi la definizione dei protocolli devono essere chiare e standard (e *open* se possibile). Queste specifiche vengono fornite in varie maniere, ad esempio sottoforma di documenti che descrivono a parole quello che devono fare, oppure sottoforma di implementazione in pseudocodice o di diagramma, e devono essere interoperabili, ovvero quando due o più protocolli implementano la specifica in modo accurato.

Un ente che definisce i protocolli è la Internet Engineering Task Force (IETF).



Affianco un esempio di grafo di un protocollo. Con la sigla RRP si intende request-reply protocol, con MSP message stream protocol e con HHP host-to-host protocol. Il grafo può essere visto un po' come l'insieme dell'immagine delle interfacce e quella dell'architettura.

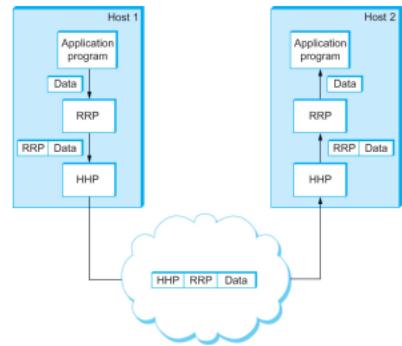
Esso descrive in pratica quali dipendenze vi sono tra i protocolli, infatti, gli archi dicono quali livelli utilizzano i servizi degli altri, ad esempio il RRP usa quelli del HHP e viceversa quando si sale.

Le due architetture hanno la medesima struttura funzionale, in quanto virtualmente la comunicazione deve avvenire in modo simile affinché le informazioni vengano codificate e interpretate correttamente. L'utente non deve accorgersi della differenza di architettura, né di come è implementata la comunicazione (mezzo fisico).

Incapsulamento

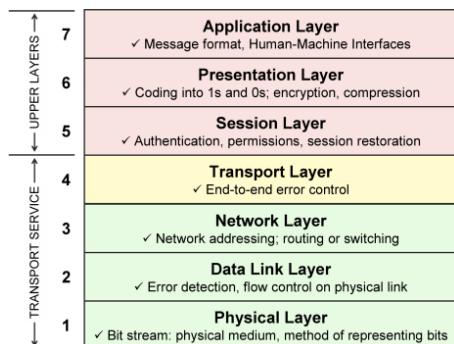
Come appena visto nell'esempio, i moduli di macchine diverse devono comunicare con strati architetturali simili e che implementano gli stessi protocolli. Perché ciò avvenga devono avere a disposizione le stesse informazioni, dunque ogni livello procede a incapsulare il pacchetto da inviare con informazioni relative al proprio protocollo (ovvero dati non intrinsechi a ciò che si sta per inviare/ricevere). Ogni servizio possiede e necessita di informazioni diverse dagli altri (es.: counter di richiesta, numero del pacchetto ricevuto, ecc.), altro motivo per cui gli strati devono funzionare allo stesso modo. Inoltre, queste informazioni che vengono aggiunte non servono all'utente, ma solo al protocollo che dovrà utilizzarle, e vengono chiamate intestazioni (*header*), mentre il resto dei dati del pacchetto costituirà il *payload*.

Quando un nuovo pacchetto deve essere spedito viene composto un nuovo header relativo alle informazioni dei protocolli di quello strato e viene aggiunto al payload contenente le info precedenti (messaggio) e poi il tutto viene passato allo strato successivo, fino al livello fisico, che poi provvede all'invio del pacchetto. Quando quest'ultimo arriva a destinazione, l'architettura host provvede a scompattare il pacchetto leggendo l'intestazione, processando le informazioni tramite appositi servizi e poi se tutto va a buon fine passa il payload al livello superiore, il quale svolge le medesime azioni.



Architettura ISO/OSI

Questo tipo di modello di architettura, realizzato da OSI (Open System Interconnection) e promosso da ISO (International Organization for Standardization) da cui deriva l'acronimo, non è da considerarsi uno stack preciso/concreto, ma un modello standardizzato nato negli anni '70 per uniformare le architetture di rete e fare in modo che tutti gli strati si presentassero nello stesso ordine e che avessero gli stessi protocolli. Quindi la ISO ha realizzato questo insieme di regole (quindi non l'effettivo modello), in modo che i costruttori avessero un pattern comune da seguire.



Il modello è sostanzialmente diviso in tre sezioni (in realtà esse sono riassumibili in due perché quella al centro fa da tramite/pontile) con in totale 7 livelli. La sezione più in basso (*lower layers*) è quella implementata dai sistemi operativi e di solito la si trova già implementata, mentre dal quinto livello compreso in su si hanno i livelli fuori dal kernel (*user space*) e riguarda la parte applicativa (library, linguaggi di programmazione, ecc.).

I tre livelli inferiori (di colore verde) sono implementati su tutti i nodi della rete, mentre il livello di trasporto (giallo) e i livelli superiori (rossi) vengono tipicamente eseguiti solo sugli host finali e non sugli switch e i router intermedi.

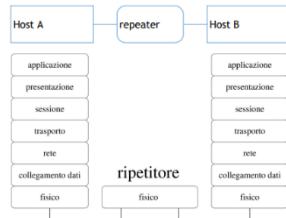
I livelli della rete sono:

- livello fisico, che riguarda l'HW e ciò che consente ai messaggi di viaggiare fisicamente, nonché la codifica dei bit e gli eventuali problemi di encoding/decoding. Si occupa, inoltre, di uniformare i segnali in arrivo, in modo che dispositivi che trasmettono con tecnologie differenti possano comunicare senza problemi. Se si utilizzano tecnologie wireless, ad esempio, si dovranno definire qui le frequenze di trasmissione. A livello di interfaccia non si ha quasi nulla, infatti ci si occupa solo di ricevere un buffer di dati e di inviarli tramite un mezzo fisico;
- livello datalink, detto anche livello di collegamento, si occupa gestire l'accesso al mezzo (es.: come codificare i bit, quando e quanto trasmettere, turni, ecc.) e di controllare eventuali problemi di

trasmissione. Questo livello deve accorgersi di eventuali errori e per accertarsene deve mettere dei dati di controllo sulla sua intestazione (talvolta cerca di sistemarli). Altra funzionalità che offre è quella di indirizzamento dell'host (importante per i mezzi condivisi). Questo livello lavora con pacchetti definiti come stream di bit (*frame*);

- livello network (di rete), presenta maggiore astrazione e consente la comunicazione tra reti differenti in quanto identifica gli host indipendentemente dal tipo di interfaccia che hanno e gestisce l'instradamento tra i nodi tramite commutazione a pacchetto. I pacchetti sono anche le "unità" di questo livello, ovvero i payload del frame del datalink;
- livello di trasporto, implementa la codifica da layer a layer e presenta funzionalità che permettono a dei processi (e non più host) di comunicare, in particolare sa a quali processi assegnare determinati dati e per fare questo utilizza il de-multiplexing. Contiene dei controlli sul livello di flusso per non saturare i nodi (e per controllare la ricezione dei dati), in quanto nei livelli inferiori non si ha nessun controllo sulla capacità di ricezione dei nodi. In realtà il modello ISO/OSI prevederebbe quest ultimo controllo al secondo livello (infatti i modem lo rispettano), ma non sempre è così. I pacchetti di questo livello si chiamano datagrammi o segmenti;
- livello di sessione, in cui con sessione si intende qualcosa che ha un inizio e una fine e in questo caso serve per coordinare delle comunicazioni (es.: audio e video). Questo livello sincronizza diversi flussi;
- livello di presentazione, è il layer in cui i dati possono essere modificati, di solito per motivi di codifica, ad esempio little endian/ big endian, stringhe, ecc. e quindi converte delle informazioni in formati standard interoperabile, comprese compressioni, cifrature e tutto ciò che riguarda la manipolazione dei dati in sé;
- livello applicativo, vi sono le regole di comunicazione tra le applicazioni e applicativi (es.: http);

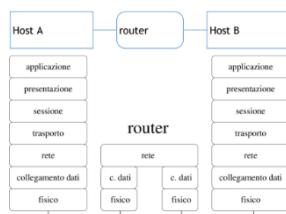
Non è detto che tutti i dispositivi implementino tutti gli strati, ciò che è importante è che siano presenti nello stesso ordine. Ad esempio, i ripetitori (*repeater*) sono dispositivi che implementano solo il livello fisico e quindi possono solo passare i bit codificati da una parte a un'altra della rete, infatti essi ricevono un segnale codificato (di qualunque tipo esso sia), lo decodificano per ricostruire lo stream di bit e interpretarlo (senza rilevazione di eventuali errori) e poi lo ricodificano per inviarlo all'altra parte. NON sono da confondere con gli amplificatori di segnale, che svolgono una funzione molto simile.



Invece, nella famiglia di dispositivi che implementano solo i primi due livelli, vi sono ad esempio i *bridge*. Essi non lavorano solo con i singoli bit come i repeater, ma con i frame derivanti da bluetooth, ethernet, ecc.

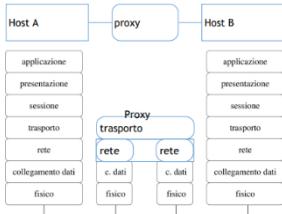
Anch'essi una volta decodificato e interpretato il messaggio lo rinviano ricodificato dall'altra parte, solo che in questo caso vi è il controllo sulla presenza di eventuali errori e la loro correzione prima di inoltrare il messaggio ricevuto. Talvolta è presente anche un controllo di flusso, ovvero si comunica al mittente che sta inviando troppi messaggi rispetto a quelli che possono essere consumati. Un caso particolare di bridge è lo *switch*, ovvero un bridge che possiede più di due porte e dunque effettua dei controlli su dove inviare i frame. Questi dispositivi appena descritti sanno come lavorare con differenti tecnologie di comunicazione e di segnale e con formati header diversi, ma non sanno cosa ci sia scritto dentro al messaggio inviato.

Tra i dispositivi che utilizzano fino al terzo livello dell'architettura vi sono i *router*, che connettono reti diverse in modo da sembrare la stessa. Essi implementano la tecnica dell'instradamento per sapere a quale rete inviare i pacchetti (tipi di dati con cui lavorano). Per fare ciò essi prendono il frame, ne tolgono l'header, guardano il contenuto per sapere dove indirizzare il messaggio e ricostruiscono un altro frame



con un'altra interfaccia di livello 3. In realtà i router hanno anche altri compiti, ma questo è quello principale.

Al livello 4 (trasporto) vi sono ad esempio i *proxy*, che lavorano con datagrammi (comunicazioni senza connessione) e segmenti (comunicazioni con connessioni). Questi dispositivi



ricevono una richiesta di connessione da un host e la ridirezionano a un altro, il quale fornirà il suo servizio. Sono utili per superare i firewall, controllare il contenuto del traffico e altro. Essi lavorano con messaggi, dei quali vengono sostituiti i frame e le intestazioni livello rete, mentre vengono mantenuti il protocollo applicativo e l'intestazione livello trasporto.

Architettura TCP/IP

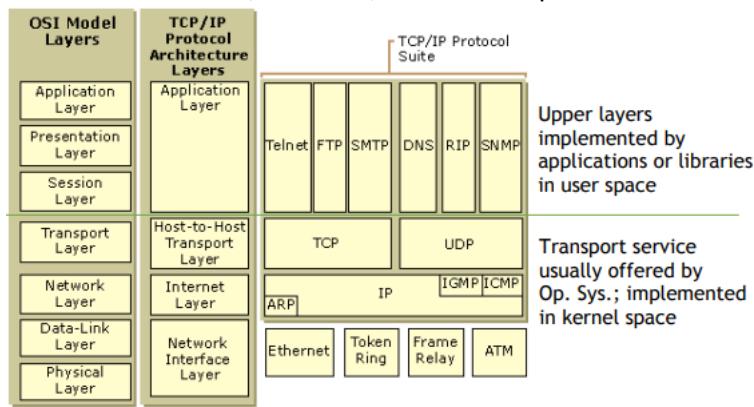
Di seguito un esempio di applicazione concreta di architettura di rete, in particolare di uno stack TCP/IP di Internet e di come sono strutturate le sue funzionalità (caso reale di Internet). Questo particolare tipo di architettura è più semplice del modello ISO/OSI, in quanto alcuni degli strati del modello precedente sono condensati in uno unico, ad esempio il livello data-link e il fisico dell'ISO/OSI che costituiscono il livello network del TCP/IP, il quale accorda entrambi i compiti svolti dai due strati dell'architettura precedente. Esso quindi si occupa degli aspetti di collegamento (sia fisico che astratto della comunicazione), ad esempio il protocollo Ethernet che specifica sia come accedere al mezzo, sia di come gestire gli errori, i frame, la codifica dei bit, ecc. e non si ha un solo standard Ethernet di riferimento, ma molteplici e dipendenti dalla tecnologia implementata (bluetooth, wi-fi, ecc.), per cui è molto importante saperli gestire correttamente.

Al terzo livello si ha l'IP (Internet Protocol), ovvero uno dei livelli più importanti e che danno anche il nome all'architettura. Esso si occupa di gestire l'instradamento dei pacchetti e in TCP/IP non si assume che gli strati inferiori siano affidabili, per cui questo livello non si aspetta che il data-link gli fornisca un canale affidabile e con i pacchetti corretti: come essi arrivano, tali restano. Ciò è dovuto al fatto che facendo assunzioni così lasche/deboli, tantissime tecnologie possono essere utilizzate come base della comunicazione, infatti se si assumesse che il canale inferiore fosse affidabile, giusto, preciso, ecc., si potrebbero creare reti molto semplici e lo stack avrebbe pochissime implementazioni, invece facendo ipotesi che sotto non sia affidabile si possono utilizzare molte più applicazioni. La gestione degli errori viene fatta in un secondo momento. In questo livello si hanno protocolli ancillary come ARP, ICMP, IGMP, ecc., che non servono a trasportare i dati, ma a far funzionare la rete (es.: routing).

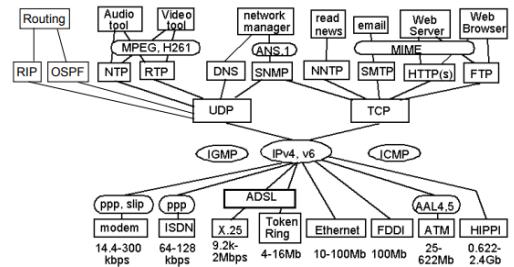
Al livello 4 dell'OSI si ha il livello di trasporto, mentre nel TCP/IP si ha un host-to-host protocol (in realtà sarebbe un process-to-process), in cui sono implementati i protocolli TCP e UDP. Il primo effettua dei controlli sul flusso e va a rimediare ad alcuni problemi di affidabilità e comunicazione, in quanto i pacchetti possono non arrivare, o arrivare fuori ordine, o sbagliati, o malformati, ecc., mentre il secondo non effettua alcun controllo accettando i pacchetti così come sono, ma è più veloce, leggero e efficiente. Il primo serve principalmente per trasporti affidabili di dati, ad esempio quando si hanno download di file in cui essi devono essere integri, mentre il secondo per comunicazioni non affidabili, ad esempio streaming audio/video, in cui è possibile perdere anche qualche frame ma si riesce a seguire lo stesso.

Per i livelli superiori, non gestiti dai sistemi operativi, si utilizzano socket per le comunicazioni e nel modello ISO/OSI si hanno tre strati che si occupano di ciò, ma nel modello TCP/IP ve n'è uno solo. Tutto quello che riguarda le problematiche relative alla gestione dei flussi, autenticazione utenti, cifratura, conversione dati, compressioni e ciò che riguarda le applicazioni è tutto dato in incarico al programmatore. In questo caso, se si volessero aprire due flussi sincronizzati, le socket devono essere gestite manualmente (a livello di librerie) in quanto non si occupano di questo. Per fare ciò e ricostruire i messaggi (TCP) si deve inserire in ogni segmento un numero progressivo identificativo, in modo che vi possano essere dei controlli sugli arrivi (livello 4), l'indirizzo di rete della destinazione del pacchetto (livello 3) e altre info per rilevare altri eventuali errori.

Di seguito uno schema di confronto tra ISO/OSI e TCP/IP e di alcuni protocolli utilizzati:



Affianco un grafo che rappresenta i collegamenti tra alcuni dei protocolli livello 3 e livello 4 del TCP/IP. Come è possibile intravedere, per ogni applicazione che si vuole implementare (riquadri in alto) si devono utilizzare i protocolli appropriati, ad esempio per le mail si usa SMTP con IMAP e POP implementati sotto TCP, invece per il risolvere il nome dei provider al loro IP corrispondente si usa il DNS (Domain Name Service) che sfrutta UDP, mentre con SNMP (Simple Network Manage Protocol) si utilizza sia UDP che TCP. Questo tipo di grafo viene chiamato anche “grafo a clessidra” in quanto presenta un restringimento al centro, che dimostra come con i protocolli di rete sia possibile aggiungere facilmente nuove tecnologie di rete semplicemente implementando nuovi moduli, in quanto quando il livello IP è realizzato e si forma una nuova tecnologia di comunicazione (es.: un nuovo standard Ethernet), basta sviluppare un nuovo protocollo IP (e casomai anche un protocollo data-link) basato su quella tecnologia per metterlo in funzione col resto dell’architettura. In grafo manca la parte di sicurezza, che come appena detto è formata da altri moduli che sfruttano quelli già esistenti nell’architettura realizzata.

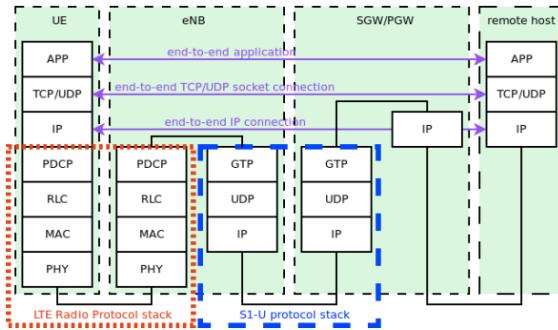


Architettura internet

Per quanto riguarda la sicurezza, appena menzionata, non è stato semplice però realizzare i moduli appena descritti, in quanto le architetture si basavano su modelli aperti e spiegati nei RFC (Request For Comments), documenti pubblici della IETF (Internet Engineering Task Force), leggibili da tutti, che riportano le informazioni di sviluppo gratuite e dicono che chiunque può implementare liberamente l’architettura TCP/IP, al contrario ad esempio del Bluetooth che si deve pagare una licenza. Per questo motivo, nacquero architetture con moduli non comunicanti tra OSI e TCP/IP e che quindi necessitavano di bridge e ripetitori, per “convertire” i messaggi in qualcosa di comunicante con altre architetture (tramite server o proxy).

Ad esempio, per quanto riguarda lo stack della tecnologia 4G (LTE-EPC), il cui obiettivo principale era “fornire una tecnologia di radioaccesso ad alta velocità di trasmissione dei dati, a bassa latenza e ottimizzata per i pacchetti, in grado di supportare distribuzioni flessibili della larghezza di banda.”, è stata progettata per supportare il traffico a commutazione di pacchetto con una mobilità continua e un’ottima qualità del servizio. Nel grafico sotto, è possibile vedere a sinistra il blocco UE (User Equipment) che rappresenta il dispositivo usato, al centro il eNB, ovvero l’antenna/base della cella, e il SGW/PGW (gateway) e a destra l’host. Quando un dispositivo deve comunicare usa la classica architettura vista prima, tuttavia l’antenna e il resto presentano solo il primo livello del TCP/IP e vengono visti come un singolo blocco data-link (scatole rossa e blu del grafico, viste come scheda di rete), i quali si occupano della codifica/decodifica, dei frame della rete wireless, di trovare l’antenna più vicina e con più porte, di gestire il link via radio, l’eventuale passaggio a un’altra antenna,

ecc. Invece, per quanto riguarda il gateway, che potrebbe trovarsi molto distante dal dispositivo e l'antenna, si deve utilizzare un altro mezzo di comunicazione che trasporti le informazioni da entrambe le parti, ovvero il GTP (Gateway Transfer Protocol), che sfrutta UDP e IP per la comunicazione attraverso il mezzo dedicato (fibra, ponti radio, ecc.) che sono però un po' diversi da quelli precedenti, col risultato che il pacchetto che sta viaggiando ha una forma del tipo, dall'esterno verso l'interno degli header: IP (radio), UDP, GTP, IP (del dispositivo iniziale, es.: cellulare), TCP e infine i dati veri e propri, in quanto gli strati intermedi del pacchetto (PDCP, RLC, MAC, PHY) del dispositivo vengono estratti e letti dall'antenna, la quale poi aggiungerà al resto del pacchetto non ancora aperto i propri header GPT, UDP e IP per il resto della comunicazione. Lo stesso procedimento di spacchettamento sarà poi effettuato anche dal gateway, il quale aggiungerà la propria intestazione IP e infine spedirà il tutto all'host.

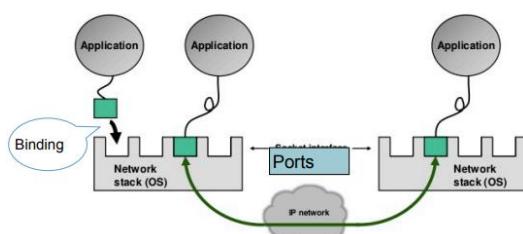
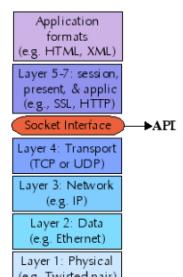


Per quanto riguarda queste reti stratificate, vi sono delle limitazioni: c'è da tenere presente che non tutte le funzionalità possono essere inserite nello stesso strato di architettura differenti e che alcune caratteristiche sono inherentemente cross-layer (quelle che riguardano il modo e la qualità del servizio, ad esempio garantire che i pacchetti arrivino senza ritardi o senza perdite e sicuri). Questo tipo di qualità dei servizi sono difficilmente collocabili in uno specifico strato e può essere causa di alcuni problemi.

Network API

Per chi progetta le reti, i principali clienti/fruitori sono i programmati di applicazioni. Poiché la maggior parte dei protocolli di rete sono implementati nel software e quasi tutti i sistemi informatici implementano i loro protocolli di rete come parte del sistema operativo, quando si parla di interfaccia “esportata dalla rete”, ci si riferisce generalmente all’interfaccia che il sistema operativo fornisce al suo sottosistema di rete. Questa interfaccia è chiamata *Application Programming Interface (API)* di rete.

L’astrazione che viene implementata a questo livello viene chiamata socket. È stata sviluppata dalla distribuzione Berkley di Unix e poi è diventata uniforme e universale, anche se persistono versioni embedded ad-hoc. Le funzionalità e le API offerte a questo livello sono quelle che consentono di creare delle socket e connetterle alla rete per inviare e ricevere dati e poi chiudere la connessione rilasciando la socket. Dal punto di vista del programmatore si intende comunicare con la corrispondente socket dall’altra parte. Essa si astrae completamente da ciò che accade sotto, nascondendo il proprio funzionamento effettivo al programmatore, e rappresenta una struttura dati bidirezionale contenuta all’interno dell’applicazione e per poter comunicare necessita di un collegamento (*bind*) a una porta del dispositivo.



Le socket sono le più diffuse, tuttavia a livello applicativo (piuttosto che a livello di SO) si stanno diffondendo anche altre interfacce, come le web-socket, che sono implementate all'interno dei browser e che utilizzano un protocollo differente chiamato Quick, sviluppato da Google e appartenente al browser stesso. Svolge un compito simile al TCP, ma comunque efficiente: sfrutta l'UDP a cui aggiunge un servizio di controllo aggiuntivo di flusso, paragonabile al TCP, ma in maniera più performante.

Per creare una socket si usa l'omonima chiamata di sistema (*system call*) e si specifica rispettivamente come parametri l'`address_family` (es.: internet ha il IPv6), il tipo (flusso o datagrammi) e facoltativamente il protocollo (lasciando a 0 il valore si lascia fare al sistema operativo). La funzione così descritta risulterà come:

```
int sockfd = socket(address_family, type, protocol);
```

Questa funzione restituisce come valore il descrittore della socket per la nuova connessione appena creata. I tipi di `address_family` supportati sono `PF_INET` che denota quella di Internet, `PF_UNIX` per le pipe di UNIX e `PF_PACKET` per l'accesso diretto alle interfacce di rete, ad esempio bypassando il protocollo dello stack TCP/IP. Mentre per quanto riguarda il tipo si ha `SOCK_STREAM` per lo stream di byte (affidabili, ordinati e orientati alla connessione) e `SOCK_DGRAM` per i messaggi orientati al servizio (UDP).

Di seguito due esempi in cui nel primo caso si utilizza il TCP, mentre nel secondo caso UDP:

```
int sockfd = socket(PF_INET, SOCK_STREAM, 0);
int sockfd = socket(PF_INET, SOCK_DGRAM, 0);
```

Da notare come se si lascia decidere al sistema operativo con quale protocollo comunicare, con buona probabilità verrà scelto il UDP.

Modello Client-Server tramite UDP

Lo scopo è quello di fornire un'API (*Application Programming Interface*) per i programmati, ovvero un insieme di funzionalità che si colloca al di sopra del livello 4 dello stack. Ad esempio tra le funzionalità che devono essere offerte si trova la libreria TLS per rendere le socket più sicure ed essa non è implementata a livello di SO, ma a livello di linguaggio di programmazione (lv. applicativo).

Nel modello Client-Server, si ha un client che invia una richiesta di connessione al server (es.: sottoforma di stringa di caratteri) e il server deve poterla interpretare e rispondere.

Ad esempio, in codice Java, si può implementare un server UDP (comunicazione di tipo datagramma) creando una socket tramite classe `DatagramSocket`, in cui si specifica come parametro in input il numero di porta su cui rispondere. Successivamente si dovrà predisporre due buffer di tipo `byte[]`, di lunghezza prefissata (es.: 1024), in cui riportare i dati da inviare (es.: `sendData`) e quelli da ricevere (es.: `receiveData`). La parte successiva del codice dovrà essere un loop che per prima cosa svuota il buffer dei dati di risposta (`receiveData`), poi crea un nuovo oggetto in cui ricevere il datagramma con la classe `DatagramPacket` che necessita in input rispettivamente del buffer `receiveData` e la sua lunghezza `receiveData.length`, successivamente il server si pone in ascolto tramite metodo `receive()` dell'istanza `DatagramSocket` (attenzione che è un'istruzione bloccante!) creata precedentemente e si fornisce in input `receiveData` come buffer in cui scrivere la risposta e infine si estrapolano le informazioni così ricevute tramite l'istanza `DatagramPacket` creata prima con `getAddress()` si ottiene l'indirizzo IP del mittente, con `getPort()` il suo numero di porta, con `getData()` si ottiene di testo del messaggio di risposta (ovvero il payload, ad esempio in formato `String` o `int`), ecc. e una volta elaborata la richiesta e memorizzata nel buffer `sendData`, il server procede con la creazione di un nuovo `DatagramPacket` con in input rispettivamente `sendData`, `sendData.length` e il numero IP e quello di porta letti prima e infine invia tramite `send()` del socket la sua risposta (array di `byte`). Tutto questo meccanismo che è situato nel loop appena descritto viene svolto infinite volte dai server in ascolto.

Invece, per quanto riguarda la creazione di un client in Java, si deve implementare un lettore di buffer tramite `BufferedReader`, il quale necessita di un `InputStreamReader` per interpretare i dati in input da tastiera, un `DatagramSocket` con facoltativamente un numero di porta (al contrario del server che la necessita obbligatoriamente), impostare l'indirizzo IP (es.: tramite `InetAddress.getByName("localhost")`) e porta del server, creare i due buffer `sendData` e `receiveData` come prima e infine si procede con il loop di istruzioni, in cui si effettua una `readLine()` per prelevare l'input da tastiera (metodo dell'istanza `BufferedReader`), lo si converte in byte e lo si aggiunge dentro a `sendData`, si procede con la creazione di un nuovo `DatagramPacket` in cui specificare `sendData`, la sua lunghezza e indirizzo IP e porta del server, e lo si invia tramite `send()` del socket. Successivamente si crea un nuovo `DatagramPacket` per `receiveData` come prima e si pone il client in attesa tramite `receive()`. Quando la risposta arriva, si leggono i dati (es.: con `getData()` dell'istanza `DatagramPacket`) e si fornisce la risposta in output all'utente. Come per il server questo è un loop che può essere ripetuto all'infinito.

Per i codici completi vedere registrazione S01E04 oppure materiale su Teams/Moodle.

Con questo esempio tuttavia, se il server improvvisamente interrompesse la comunicazione, il client si bloccherebbe perché resterebbe in attesa illimitata, al contrario del server per il quale è normale attendere un client, per cui anche se quest ultimo si disconnettesse non sarebbe un problema perché qualsiasi client può connettersi in un qualsiasi momento. Questo è un piccolo problema del meccanismo UDP implementato in questa maniera. Da notare il fatto che essendo in locale, la perdita dei pacchetti è nulla, mentre se si fosse in una rete locale sarebbe dell'1% circa e in una rete più grande la probabilità di perdita è leggermente più alta, per cui il problema della perdita dei messaggi in UDP in realtà è poco importante.

Inoltre, bisogna sempre scegliere un numero di porta per il server che non sia di quelle standard usate dal SO, in quanto sono dedicate a servizi propri (es.: la 8080, oppure la 7 delle echo, che ritorna il messaggio così come arriva e che è utile per testare la presenza del server).

Modello Client-Server tramite TCP

È possibile realizzare anche modelli Client-Server che sfruttano il TCP, ovvero una connessione stabile tra client e server, con controllo di flusso e gestione di stream di dati maggiore del UDP. Rispetto alla precedente soluzione, però, è più complessa da implementare e richiede anche più lavoro dal punto di vista dei programmati. Per fare ciò, in C, un server usa la system call `bind` per collegare la porta socket:

```
int bind(int socket, struct sockaddr *address, int addr_len);
```

In cui l'indirizzo è una struttura dati composta da indirizzo IP e numero di porta. Mentre per sapere quanto lunga è la lista di attesa su quel socket si usa la `listen`:

```
int listen(int socket, int backlog);
```

E per porre l'endpoint in attesa del prossimo collegamento si usa la chiamata `accept`, che è bloccante:

```
int accept(int socket, struct sockaddr *address, int addr_len);
```

Questa funzione restituisce un nuovo socket che corrisponde alla nuova connessione stabilita e l'argomento dell'indirizzo contiene l'indirizzo del endpoint remoto.

Il client, invece, utilizza una connessione "attiva", tramite la funzione `connect`:

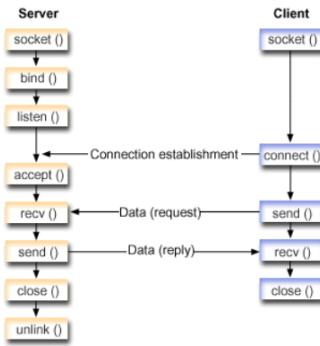
```
int connect (int socket, struct sockaddr *address, int addr_len);
```

Che resta in attesa di una risposta finché il TCP non ha stabilito con successo una connessione e l'applicazione è libera di iniziare a inviare dati (oppure fino a quando non riceve un errore se il server non è disponibile).

Una volta che la connessione è stabilita si usano le funzioni `send` e `receive` (oppure altre operazioni):

```
int send (int socket, char *msg, int msg_len, int flags);
int recv (int socket, char *buff, int buff_len, int flags);
```

Di seguito un'immagine esplicativa del processo completo:



Per quanto riguarda Java, invece, un server può essere implementato tramite `ServerSocket` per creare fisicamente la socket da usare, poi si effettua una `bind()` tramite metodo della medesima classe e in cui si specifica l'indirizzo e la porta di connessione tramite `InetSocketAddress` con in input il numero IP e di porta. Successivamente si crea il loop di servizio in cui al suo interno si attende (attesa bloccante) un nuovo client che si connette, tramite `accept()` della classe `ServerSocket`, che restituisce in output il `Socket` corrispondente. Quando si riceve effettivamente un client si può vedere tramite `getInetAddress()` e `getInetPort()` il suo indirizzo e numero di porta e si può procedere con lo stream di byte. Per prima cosa si crea un flusso di byte di input con `DataInputStream`, che necessita in input di `getInputStream()` di `Socket`, e uno di output con `DataOutputStream`, con in input `getOutputStream()` di `Socket`. A questo punto con un blocco try-catch (per rilevare la disconnessione del client e quindi a chiudere la socket) e un ciclo al suo interno si può procedere con lo scambio di messaggi. Per leggere una linea di testo si usa `readUTF()` dell'istanza `DataInputStream` e per inviare la risposta si usa `writeUTF()` dell'istanza `DataOutputStream`. Queste due istruzioni (e l'elaborazione dei dati in input) vanno collocati nel ciclo interno al blocco try-catch. Per chiudere la connessione (blocco catch) si usa `close()` di `Socket`.

Per quanto riguarda il codice Java del client, invece, si ha un `BufferedReader` per l'input da tastiera (vedere versione UDP sopra), poi si crea un nuovo `Socket` per la comunicazione e si effettua la richiesta di connessione tramite `connect()` di `Socket` che usa un three-way-handshake con pacchetti di servizio. Se la connessione viene accettata, il client procede con la creazione di `DataOutputStream` e `DataInputStream` come visto sopra. Successivamente, in un loop, si ha una `readLine()` dell'istanza `DataInputStream` per prelevare l'input da tastiera e si usa un `writeUTF()` dell'istanza `DataOutputStream` per inviare al server la stringa, mentre con `readUTF()` (che è bloccante) dell'istanza `DataOutputStream` si legge la risposta del server. Per chiudere la connessione e rilasciare il socket si usa `close()` dell'istanza `Socket`.

[Per i codici completi vedere registrazione S01E04 oppure materiale su Teams/Moodle.](#)

In realtà anche questa soluzione ha un leggero problema lato client: se il server interrompesse la connessione, il client se ne accorgerebbe solamente quando deve inviare i dati, in cui vede che la socket non risponde più e quindi solleva un errore. Al contrario il server si accorge subito del problema, chiude la propria socket e si pone nuovamente in attesa di un nuovo client, ma senza alcun errore.

Performance

Un aspetto molto importante della rete è quello della prestazione (*performance*), misurabile in due modi: attraverso la larghezza di banda (*bandwidth*), o più propriamente si dovrebbe parlare di *throughput* (capacità/portata del canale), in cui si quantifica il numero di bit che si riescono a inviare al secondo, oppure tramite ritardo (*delay* oppure *latency*), ovvero, una volta inviati dei dati, quanto tempo intercorre prima che arrivino a destinazione. I due metodi sono in contrapposizione, in quanto di solito più si sfrutta la rete, più aumenta il ritardo di arrivo dei dati, soprattutto quando si hanno reti con commutazione a pacchetto.

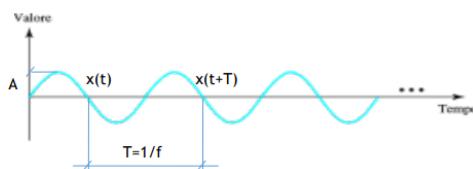
Nel caso dell'informatica, con larghezza di banda si intende erroneamente il throughput (bit al secondo), mentre per ingegneri e fisici significa larghezza della frequenza di banda (Hertz), usata per la comunicazione.

Definizione di bandwidth

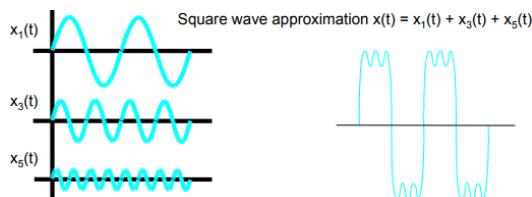
Con frequenza di banda si intende quindi le onde elettromagnetiche che i dispositivi generano per comunicare e tramite modulazione sono in grado di trasportare del segnale. Queste onde sono formate da delle sinusoidi, definite da tre valori:

- ampiezza A , espressa ad esempio in Volt, Pascal, ecc. e rappresenta quanto è alta l'onda;
- frequenza f , espressa ad esempio in Hertz [1/s] e rappresenta quante volte l'onda va su e giù;
- fase φ [f], ovvero quanto l'onda è spostata avanti o indietro

La frequenza può essere scritta anche come $T = 1/f$, ovvero è l'inversa del tempo. La formula della sinusoide è quindi definita come $x(t) = A \sin(2\pi ft + \varphi)$.

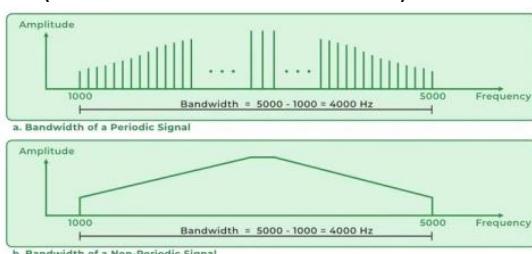


Secondo la definizione di Fourier, ogni segnale $x(t)$ "stazionario" (periodico) può essere visto come la sommatoria di tante sinusoidi eventualmente sfasate: $x(t) = \sum_i x_i(t) = \sum_i A_i \sin(2\pi f_i t + \varphi_i)$. Come è possibile notare dall'immagine sotto, sommando diverse onde sinusoidali si ottiene un'approssimazione di un'onda quadra, infatti per ottenerla si dovrebbero sommare infinite sinusoidi.



Inoltre, Fourier propose di non guardare l'andamento delle sinusoidi nel tempo, ma di studiare la loro rappresentazione in base a delle loro componenti che compongono il segnale. Questo fatto ci porta nello spazio delle frequenze, ovvero a rappresentare il segnale non più come un valore che sale e scende nel tempo secondo una certa forma (onda), ma come sommatoria di tante sinusoidi, ognuna delle quali delinea un certo valore sull'ampiezza (e in realtà anche sulla fase, utilizzando i numeri complessi).

Questo valore rappresenta tutta la sinusoide e può essere raffigurato come un singolo punto in un grafico frequenza-ampiezza (nei grafici talvolta viene disegnato tramite una barra per migliore la sua visibilità). Quando si hanno più sinusoidi e quindi più punti nel grafico si ottiene una sort di campionamento, ovvero una trasformazione da tempo-valore a frequenza-ampiezza chiamato trasformata di Fourier. Tramite questa operazione si possono quindi campionare segnali acustici (file audio musicali come MP3) e anche riconvertirli nel formato originale.



A questo punto si può iniziare a descrivere cosa sia una larghezza di banda: è la parte occupata dal segnale nello spazio delle frequenze, ovvero la differenza tra le frequenze massime e quelle minime con ampiezza non nulla, in quanto se il segnale restasse uniforme e la differenza fosse quindi nulla, non si avrebbe alcuna larghezza di banda (si avrebbe un solo punto e non è molto utile). L'insieme delle frequenze che compongono un segnale periodico è detto spettro delle frequenze.

Definizione di throughput

Per quanto riguarda il throughput, si intende i bit trasmessi in un secondo di tempo, e con bit ci si riferisce a quelli con una particolare larghezza di banda che possono essere considerati come aventi una certa ampiezza (cioè un'estensione temporale).

Latenza

Con latenza si intende il ritardo complessivo dei tempi di propagazione, trasmissione e accodamento, ovvero il tempo di delay introdotto dalla trasmissione dei dati, quindi dal momento in cui sono trasmessi i dati a quanto sono effettivamente utilizzabili. La latenza è data dalla somma dei tempi di:

- propagazione: distanza / velocità del segnale, dipende costanti e leggi fisiche (es.: 3.0×10^8 m/s nel vuoto, 2.3×10^8 m/s in un cavo, 3.0×10^8 m/s in fibra);
- trasmissione: tempo di trasmissione del messaggio (grandezza del messaggio / throughput);
- accodamento (queue): tempo trascorso all'interno del computer, switch e router

In reti multi-hop, questo tempo totale deve essere aggiunto per ogni link e switch attraversato. Inoltre, se si hanno messaggi piccoli, la propagazione è importante, mentre se si hanno tanti byte allora è importante il throughput.

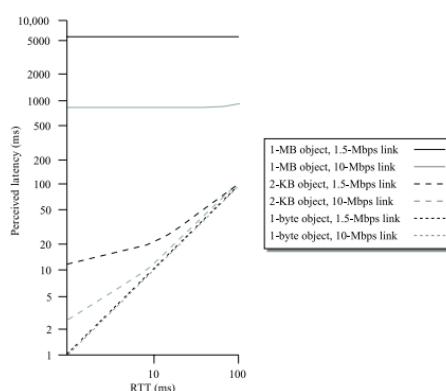
Come è possibile vedere dalle costanti di propagazione in esempio, i cavi sono più veloci della fibra, tuttavia è vero solo se non si considerano gli altri tempi, in quanto il tempo più lungo è dato dal passaggio tra un dispositivo e un altro, e si sa che il cavo necessita di molti più collegamenti rispetto alla fibra (perché a un certo punto il segnale si attenua), per cui alla fine risulta più vantaggiosa la fibra che può percorrere uno spazio maggiore disponendo di un minor numero di dispositivi intermedi, inoltre il suo throughput è maggiore. Questo confronto è esprimibile come fattore di velocità (*Velocity Factor, VF*), ovvero: $VF = \text{velocità della luce nel mezzo} / \text{velocità della luce nel vuoto}$, in cui, più basso è il VF, più lento è il segnale che si propaga, maggiore è il ritardo introdotto dalla propagazione del segnale.

VF (%)	Cable
74-79	Cat-7 twisted pair
77	RG-8/U
67	Optical fiber
65	RG-58A/U
65	Cat-6A twisted pair
64	Cat-5e twisted pair
58.5	Cat-3 twisted pair

Delay vs throughput

L'importanza relativa del throughput e del ritardo dei canali dipende dall'applicazione. Come detto prima, per il trasferimento di file di grandi dimensioni, il throughput è fondamentale, mentre per i messaggi di piccole dimensioni (HTTP, NFS, SMS, ecc.), il ritardo è fondamentale.

Sfortunatamente, è difficile ottimizzare entrambi allo stesso tempo. Per avere un throughput elevato, si deve mantenere i canali pieni, ma poi le code si allungano e quindi il ritardo aumenta. Per avere bassi ritardi, le code devono essere vuote o molto corte, il che implica basso throughput.



I dati non possono essere trattenuti, si deve pensare al canale tra una coppia di processi come ad un tubo vuoto. Il ritardo assimilabile al tempo di propagazione è dato dal delay (lunghezza del tubo) e dal throughput (diametro del tubo). Se ad esempio si avesse un ritardo di 50 ms e throughput di 45 Mbps si avrebbero:

$$50 \times 10^{-3} \text{ secondi} \times 45 \times 10^6 \text{ bit/secondo} = 2.25 \times 10^6 \text{ bit} = 280 \text{ KB}$$

Questo è il volume di dati che può essere contenuto nel “tubo” in quel dato momento, in modo da sfruttarlo a pieno.

Quindi, il prodotto *Delay × Throughput* rappresenta il numero di bit che il mittente deve trasmettere prima che il primo bit arrivi al ricevitore, se il



mittente vuole mantenere la pipe piena. Ci vuole un altro ritardo unidirezionale per ricevere una risposta dal ricevitore. Il tempo per andare e tornare è chiamato Round Trip Time (RTT) e di solito è uguale a $2 * \text{Delay}$ (se il canale è simmetrico). Se il mittente non “riempie il tubo” (cioè invia un intero RTT × prodotto di throughput di dati prima di fermarsi ad aspettare un segnale) non utilizzerà completamente la rete.

Di seguito alcuni esempi, in cui con TP si intende il throughput e con delay si intende RTT:

Link Type	Throughput (Typical)	Distance (Typical)	Round-trip Delay	Delay × TP
Dial-up	56 Kbps	10 km	87 μs	5 bits
Wireless LAN	54 Mbps	50 m	0.33 μs	18 bits
Satellite	45 Mbps	35,000 km	230 ms	10 Mb
Cross-country fiber	10 Gbps	4,000 km	40 ms	400 Mb

Altro aspetto importante è capire quanti dati trasmettere al mittente affinché la ricezione avvenga correttamente e con poco delay. Ad esempio, un’automobile che deve accendersi o alzare un finestrino necessita di inviare segnali di pochi kbyte e di processarli abbastanza velocemente, mentre se deve far partire gli airbag, quei pochi bit di informazione devono essere inviati e processati molto più rapidamente.

Jitter

A volte il ritardo non è un problema, ma lo è la sua variazione (ad esempio, nel settore multimediale, nello streaming audio/video). La varianza del ritardo è chiamata jitter. Tale variazione non è di solito introdotta nei collegamenti (che sono mezzi fisici passivi), ma piuttosto da code variabili negli switch (compresi i router), nelle reti multi-hop e nelle reti di telecomunicazioni.



La situazione ideale è quando il ritardo è costante (e poco!), in quanto se i dati iniziassero rallentare e poi improvvisamente arrivassero tutti in un colpo si rischia di incorrere in un *unbuffering*. Il *buffering* è utile per accumulare dati prima di utilizzarli, tuttavia se il buffer si esaurisce ci si deve fermare per aspettarne altri e nel caso ne arrivassero troppi, alcuni devono essere scartati (si esaurisce la capienza). Dunque, se la varianza è poca, non serve avere buffer tanto grandi e la situazione diventa più facile da gestire. Lo stesso vale ad esempio per lo streaming video, in cui non importa più di tanto avere tanta banda, quanto più avere una connessione con poca varianza e un buffer proporzionato.

Reti di computer

In questo capitolo si analizzeranno in particolare i primi due strati dello stack OSI, ovvero il livello fisico e il data-link e le loro relative problematiche (es.: connessione dei nodi, codifica, framing, errori, wireless, accessi, ecc.). Per prima cosa ci si concentrerà sulle singole linee di connessione tra i nodi.

Link e frequenze

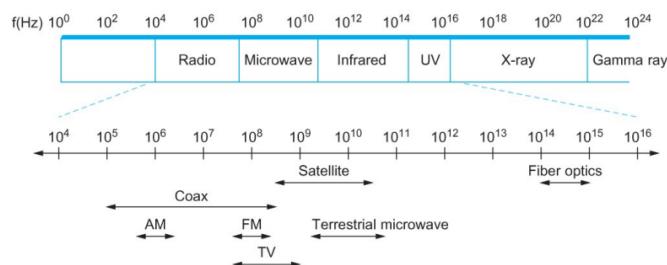
Un collegamento (link) è un qualsiasi mezzo condiviso che permetta di far comunicare direttamente due dispositivi della rete. Questo mezzo può essere fatto in qualsiasi maniera e deve poter far passare dei segnali. I più famosi e utilizzati sono i cavi in rame (es.: doppini, coassiali, ecc.), la fibra ottica e l'etere (es.: wireless).

Ogni mezzo di comunicazione elettromagnetica ha un certo range di frequenze utilizzabili e trasparenti. La frequenza si esprime in Hertz, ovvero la misura con cui oscillano le onde elettromagnetiche. La distanza tra la coppia adiacente di massimi o minimi di un'onda, misurata in metri, è chiamata lunghezza d'onda (*wavelength*). La velocità della luce divisa per la frequenza dà la lunghezza d'onda, ad esempio per la telefonia, un cavo in rame ha frequenze tra i 300Hz e i 3300Hz e la lunghezza d'onda per 300Hz su un cavo in rame è:

$$\text{velocità della luce sul rame} \div \text{frequenza} = \text{circa } 2 \times 10^8 \div 300 = 667 \times 10^3 \text{ metri}$$

È da tenere presente che l'orecchio umano lavora con un range di frequenze tra i 20Hz e i 20kHz, mentre la voce è compresa tra i 20Hz e i 12kHz circa, tuttavia come appena detto il cavo in rame sfrutta delle altre, quindi il segnale deve essere trasformato tramite modulazione. La modulazione consiste nel modificare i segnali in termini di frequenza, ampiezza e fase. Ad esempio, lo spostamento (shift) di un segnale vocale da 0 a 3000 Hz nella gamma di frequenze del collegamento telefonico da 300 a 3300Hz.

Lo spettro elettromagnetico è diviso in frequenze e va da 0Hz (segnale costante e non oscillante) a 10^{24}Hz . Il range di frequenze visibili è una piccola parte compresa tra infrarossi e ultravioletto, mentre le altre non sono visibili, ma potrebbero essere percepibili mediante calore (es.: alcuni infrarossi o le microonde). Le frequenze tra 10^4Hz (onde lunghe, es.: le vecchie radio) e i 10^{16}Hz sono quelle che verranno discusse in questo corso.



Service	Bandwidth (typical)
Dial-up	28–56 kbps
ISDN	64–128 kbps
DSL	128 kbps–100 Mbps
CATV (cable TV)	1–40 Mbps
FTTH (fibre to the home)	50 Mbps–1 Gbps

Per quanto riguarda le frequenze "casalinghe", le vecchie linee telefoniche arrivavano al massimo a 56 kbit/s di throughput, poi si è passati al ISDN che sfruttava al massimo 128 kbit/s (fine anni '90), successivamente si è passati alla DSL che usa mediamente fino a 100Mbit/s, mentre se il cavo è buono e la distanza è corta si può arrivare anche a 2-3Gbit/s. Per quanto riguarda i televisori, quelli via cavo coassiale arrivano a 40Mbit/s. Invece, la fibra ottica arriva anche a 1Gbit/s e oltre. I segnali wireless casalinghi del wi-fi risentono delle condizioni climatiche per via dell'effetto forno a microonde, ovvero la presenza di particelle d'acqua che si muovono, infatti la frequenza di 2,4GHz consente di far vibrare delle molecole di acqua (bipoli) che poi interagiscono con il campo magnetico del forno.

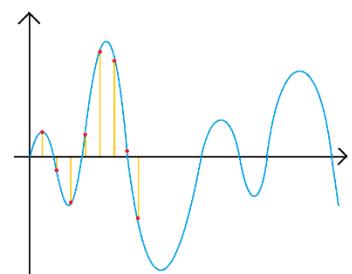
Codifica e simboli

L'inserimento di dati binari in un segnale si chiama codifica (*encoding*). Con simbolo, invece, si intende lo stato in cui può trovarsi un canale, ovvero il tipo di oscillazione dell'onda che produce e a ogni sua variazione corrisponde un simbolo/stato diverso. I simboli variano, inoltre, in base alla tecnologia utilizzata, dalle frequenze, dalla tensione, ecc. Per quanto riguarda la tensione elettrica, di solito si ha 0V o 5V (binaria), che rappresenta due stati diversi del canale. Si possono trasmettere diversi bit in un simbolo e diversi simboli al secondo modulando sia ampiezza che frequenza. Dal punto di vista del grafico è possibile avere diversi punti

sulla stessa barra verticale (stessa frequenza, ma con diversa ampiezza) oppure diversi punti sulla stessa linea orizzontale (diversa frequenza, ma stessa ampiezza) e si effettua una codifica modulando questi parametri.

Teorema di Nyquist-Shannon

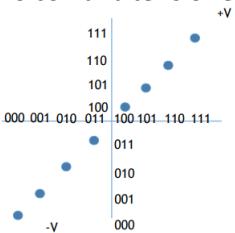
Per quanto riguarda la quantità di dati trasmissibili, dipende dalla banda della frequenza, infatti più simboli si riescono a trasmettere al secondo più veloce diventa la comunicazione. Il numero massimo di simboli al secondo è chiamato *Baud* o *Baud rate* (da non confondere Baud con Hertz!), espresso in *simboli/secondi*. Un importante **teorema** è quello **di Nyquist-Shannon** che dice che un canale con una certa larghezza di banda B (differenza tra frequenza max e min), non è importante su quali frequenze sta lavorando, può trasmettere al massimo il doppio di simboli rispetto alla larghezza di banda, quindi $R = 2B$, in cui R è il numero massimo di baud trasmissibili (es.: 1kHz trasmette al max 2000 simboli al secondo). Questo è dato dal fatto in cui a un certo punto, se i simboli sono troppi, non si riesce più a ricostruire l'informazione (codifica) dovuto al teorema del campionamento: dato un segnale con frequenza f che si vuole campionare, trasmettere su un canale e ricostruire, bisogna prima leggerlo a intervalli regolari e trasmettere i campioni ricavati. Questa lettura va fatta almeno il doppio della frequenza (due campioni per periodo) per ottenere una serie di punti utile a ricostruire l'onda completa.



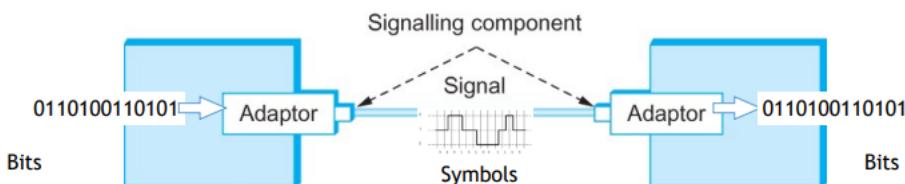
Un'altra dimensione che si deve considerare è quella di quanti bit può portare ogni simbolo. Nel caso si avesse M simboli (alfabeto), dipendenti da segni di modulazione, tecnologia, ecc., si hanno $\log_2(M)$ bit trasportabili in ogni segnale. Ad esempio, se si codificano i bit con una tensione elettrica su un filo, entro un determinato intervallo di tensione, si può dividere questo intervallo in:

- 2 livelli → 2 simboli = 1 bit per simbolo
- 4 livelli → 4 simboli = 2 bit per simbolo
- 128 livelli → 128 simboli = 7 bit per simbolo

La codifica dei bit per livello avviene nel seguente modo: si prende il range $-V$ e $+V$ e lo si divide nel numero di livelli desiderato. Queste suddivisioni rappresentano i possibili livelli di voltaggio (simboli), ognuno dei quali porta un certo numero di bit. La tensione elettrica viene misurata a ogni cambio. Ad esempio con una tensione da 0V a 10V, se si misurano solamente questi due estremi si hanno due livelli e di conseguenza due simboli, mentre se si volessero quattro livelli si avrebbero intervalli di 2,5V e quindi quattro simboli. A ogni livello si assegna una tupla di bit, ad esempio quella più in basso (0V) potrebbe corrispondere a tutti i bit a 0, mentre quella più in alto a una serie di bit tutti a 1 e quelli intermedi sono le possibili combinazioni (es.: con 8 livelli si hanno tuple di bit da 3, ovvero 000, 001, 010, 011, ecc.).

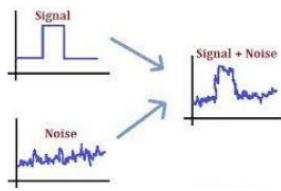


Per trasmettere una tupla di bit, il codificatore segnala il simbolo (cioè imposta la tensione) corrispondente alla tupla data, viceversa per il decodificatore.



Dato che il numero di bit per simbolo è limitato, esistono dei trucchetti per far in modo di aumentare i livelli, quindi di trasportare più dati per simbolo e di conseguenza ogni stato diventa più informativo. Alcune tecnologie consentono di accrescere la grandezza dell'alfabeto di simboli, allargando dunque il numero di bit per simbolo. Questa tecnica consentirebbe di mantenere la stessa banda, ma con una capacità di informazioni

trasportabili maggiore. Il problema è che questa soluzione è utopica: si dovrebbe avere un canale perfetto in grado di rilevare differenze di tensione microscopiche e di non essere sensibile al rumore.



Il rumore (*noise*) è un segnale casuale $N(t)$ che si aggiunge a quello buono $S(t)$ e è causato dal rumore termico dei cavi, dalle distorsioni, dalle interferenze, dai limiti tecnologici sulla precisione dei codificatori e sul potere di discriminazione dei decodificatori, ecc. Se il rumore è troppo forte, un simbolo può essere modificato nel canale e quindi interpretato in modo errato dal decodificatore (ad esempio, un picco può aumentare la tensione abbastanza da far cadere il valore successivo).

Il fattore importante è il rapporto segnale/rumore (*signal-to-noise, S/N*): il rapporto tra la potenza media del segnale S e la potenza media del rumore N . Entrambi sono misurati in Watt (o mW), quindi S/N è un numero puro. Spesso questo rapporto viene espresso in decibel (dB), in questo caso prende il nome di *SNR*:

$$SNR \text{ (in dB)} = 10 * \log_{10}(S/N) \quad [dB]$$

Da notare che i decibel corrispondono a 10 Bel, la cui unità di misura è molto piccola, per cui si prendono i decimi di essa e si moltiplica per 10 il valore. A volte si dice che questo è il “livello di rumore” e determina la “qualità” del canale. Un incremento di 3 dB corrisponde a circa il raddoppio del S/N . Un incremento di 10 dB corrisponde a un aumento del S/N di dieci volte, quindi ad esempio:

- SNR=0 dB significa $S/N = 1$, quindi $S=N$
- SNR=10 dB significa $S/N = 10$
- SNR=20 dB significa $S/N = 100$
- SNR=30 dB significa $S/N = 1000$
- SNR= -20 dB significa $S/N = 0.01$

Teorema di Shannon-Hartley

Il **teorema di Shannon-Hartley** fornisce il limite superiore della capacità di un collegamento in termini di bit al secondo (bps) in funzione dell’SNR del collegamento e della larghezza di banda B disponibile (in Hz):

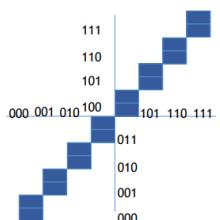
$$C = B * \log_2(1 + S/N) \quad [bps]$$

Come è possibile dimostrare tramite formula, se $S \ll N$ allora il rapporto $S/N \rightarrow 0$ e di conseguenza tutto il resto dell’equazione, mentre col rumore quasi simile al segnale allora si ha del valore. Viceversa se il rumore fosse completamente assente, $S/N \rightarrow +\infty$ quindi tutta l’equazione tende a ∞ , da cui si ritorna alla soluzione utopica di prima: indipendentemente dal valore di B , si potrebbero avere infiniti simboli, quindi infiniti bit.

Conseguenza del teorema: per ottenere una capacità maggiore, è necessario migliorare la larghezza di banda, o la qualità della linea (S/N maggiore), o entrambe le cose.

Ad esempio, per la linea telefonica si ha $B = 3300 - 300 = 3000\text{Hz}$ e S come segnale di potenza (medio), N per la potenza del rumore medio in cui si hanno 30 dB di rumore e quindi $S/N = 10^{30/10} = 1000$, di conseguenza $C = 3000 \times \log_2(1001) = 30\text{kbps}$.

Quindi, in riassunto: la capacità è proporzionale alla larghezza di banda: più grande è B , più veloce è la velocità con cui i segnali possono essere commutati (cioè minore è lo slot di tempo per l’invio di un simbolo); esiste la dipendenza da $\log_2(1 + S/N)$: più grande è S/N , più bit possono essere codificati su un livello di segnale. Es: $N = S/7 \rightarrow$ otto livelli (4 positivi, 4 negativi) possono essere letti dal decodificatore senza interferenze di rumore. Quindi, 3 bit possono essere codificati su ciascun livello (rumore: aree blu).



Esercizi:

La lingua italiana è composta da 45 fonemi e in un linguaggio parlato vengono pronunciati in media 15 fonemi al secondo. A quanto equivale il bit rate (throughput) della trasmissione?

R: $\text{bit per fonema} = \log_2(45) = 5,49$ mentre per il bit rate si ha $5,49 * 15 = 82,5 \text{ bps}$.

E se il canale usato per la trasmissione avesse un SNR di 10dB, quale ampiezza di banda è necessaria?

R: si converte SNR in $S/N = 10^{10/10} = 10$ e prendendo $C = 82,5 \text{ bps}$ calcolato prima, dalla sua formula si ricava $C = B * \log_2(1 + S/N) \rightarrow 82,5 = B * \log_2(11) \Rightarrow B = 82,5 / 3,459 = 23,8 \text{ Hz}$.

Nei sistemi radio DAB e DAB+, i dati sono codificati mediante simboli (OFDM); ogni simbolo porta 3072 bit e dura 1,246 ms. Ogni frame contiene 76 simboli, e i frame sono separati da una pausa di 1,304 ms. (a) A quanto equivale il bit rate grezzo di tale trasmissione? (b) Se il canale usato ha una larghezza di banda di 1536 kHz, quanto dev'essere il rapporto S/N minimo per una trasmissione senza errori?

(DAB = Digital Audio Broadcasting, OFDM = Orthogonal Frequency-Division Multiplexing)

R: (a) ogni frame trasporta $76 * 3072 = 233472 \text{ bit}$ e ha una durata totale di $1,246 * 76 + 1,304 = 96 \text{ ms}$, da cui si ricava la capacità effettiva di $233472 / 96 = 2432 \text{ kbps} = 2,432 \text{ Mbps}$;

(b) dalla formula del bit rate $C = B * \log_2(1 + S/N)$ si ricava $2,432 * 10^6 = 1,536 * 10^6 * \log_2(1 + S/R)$, quindi $2^{2,432/1,536} = 1 + S/R \rightarrow 2^{1,583} = 1 + S/R$ da cui $S/R = 2$ quindi 3 dB.



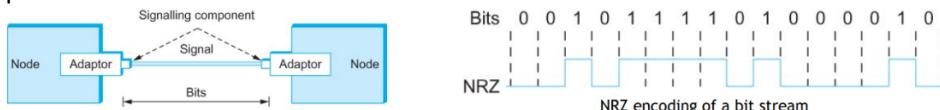
$76 * 3072 = 233472 \text{ bit}$

$1.246 * 76 + 1.304 = 96 \text{ ms}$

Tipi di codifiche

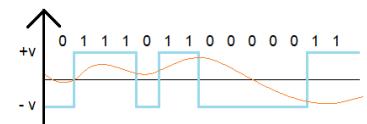
Codifica NRZ

Per quanto riguarda la codifica vera e propria, come detto prima, ogni tupla viene corrispondente a un livello di tensione. Il metodo più naïve è quello di prendere due valori di tensione (due stati) e si assegna loro un corrispondente valore 0 e 1 (es.: luce accesa/spenta, voltaggio 0V o 5V, ecc.). In alternativa esiste la codifica NRZ (Non Return to Zero), ovvero una codifica che va da voltaggi negativi (es.: -5V) a quelli positivi (es.: 5V) senza passare per lo 0.



Problema Baseline Wonder:

Tuttavia questa idea presenta dei problemi. In primo luogo come capire che il segnale è alto o basso in quanto il ricevente misura il voltaggio rispetto allo zero e non è detto che allo zero corrisponda il reale valore 0V, poiché potrebbero accumularsi cariche e sfasare la misurazione (*Baseline Wonder*). Esso si presenta soprattutto in quei casi in cui vi sono molti 0 o 1 consecutivi (vedi immagine) in quanto in realtà il ricevitore mantiene una media dei segnali che ha visto fino a quel momento e utilizza tale media per distinguere tra segnali bassi e alti. Quando un segnale è significativamente più basso della media, è 0, altrimenti è 1. Troppi 0 e 1 consecutivi causano una variazione della media, rendendo difficile il rilevamento;



Problema Clock Recovery:

Inoltre, quando si ha una serie di stesse cifre consecutive vi è anche il problema della sincronizzazione tra velocità del clock di trasmissione e quella di lettura del ricevente (*Clock Recovery*), in quanto nel caso siano sfasati è facile che si legga un intervallo di bit diverso, ovvero si salta un bit se la lettura è più lenta, oppure si legge due volte lo stesso se essa è più veloce. Per consentire il recupero del clock sono necessarie frequenti transizioni da alto a basso o viceversa, e comunque non si è mai sincronizzati con precisione: è difficilissimo, impossibile.

Codifica NRZI

La NRZI (*Non Return to Zero Inverted*) presenta una variante rispetto alla precedente: se il bit da trasmettere è 0, si resta sul valore di tensione attuale in trasmissione, mentre se è 1 allora si inverte il valore di tensione. Per quanto riguarda i simboli, invece, sono associati al cambio di tensione, non al segnale trasmesso.

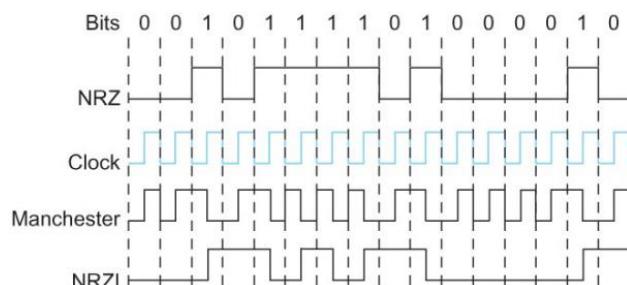
In questo caso, se si è in presenza di una serie di 1 consecutivi, si generano onde quadre perfette che consentono di sincronizzare abbastanza bene il ricevitore, il quale poi è in grado di determinare il cambio di frequenza e prendere il ritmo di trasmissione. Così facendo, anche se arriva una sequenza di 0 non tanto lunga, si riesce a tenere il clock del ricevitore abbastanza preciso, però se è lunga è un problema.

Codifica Manchester

Si ottiene prendendo la codifica NRZ e la si sovrappone (Ex-OR) a una frequenza di clock nascosto. Il clock è una sequenza doppia di bit regolare. La codifica si effettua prendendo l'onda quadra del clock quando si è in presenza di uno 0, altrimenti se è 1 la si prende invertita.

Questa codifica consente di risolvere anche il problema della sequenza di 0 consecutivi, in quanto si ha sempre un'inversione fissa delle onde. Si dice, infatti, auto-clockante. Però ha lo svantaggio che dimezza la capacità del canale, mentre nelle due precedenti la frequenza resta quella (o un po' sfasata per la NRZI). Ciò è dato dal fatto che questa codifica raddoppia la velocità di transizione del segnale sul collegamento, quindi il ricevitore ha la metà del tempo per rilevare ogni impulso del segnale. Bisogna ricordarsi che la velocità con cui il segnale cambia stato è chiamata velocità di trasmissione del collegamento (*baud rate*). Se si considera i livelli come statici, per la Manchester sono necessari 2 stati per rappresentare un bit (basso \rightarrow alto = 0, alto \rightarrow basso = 1), quindi la velocità di trasmissione è la metà della velocità di trasmissione (2 stati = 1 bit), quindi il bit rate è la metà del baud rate.

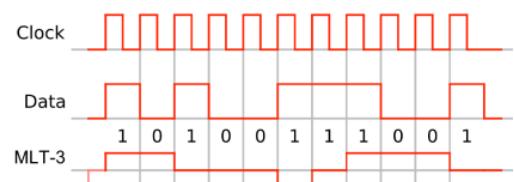
Queste appena viste sono le codifiche più famose e utilizzate. Di quest'ultima esiste anche la Manchester invertita. Di seguito un'immagine che rappresenta le tre codifiche a confronto:



Codifica MLT-3

La sigla sta per MultiLine Transmission con 3 livelli. È simile alla NRZI, solo che ha una terna di bit: -1, 0 e 1. Il terzo livello aggiunto serve per distinguere i voltaggi. Come per la NRZI, se si è in presenza di un bit a 1 si inverte il segnale, se invece si ha 0 si resta lineare. Quando si incontra un 1 si alza il livello di voltaggio e quando si arriva all'estremo +1, al prossimo bit 1 che arriva, si inverte l'ascesa di voltaggio in discesa, fino a quando non si arriva all'estremo -1 e si risale, e così via. Lo 0, invece, mantiene lo stesso livello. Anche in questo caso i simboli sono i cambiamenti di valori.

Quando si hanno lunghe sequenze di 1 si ottiene una sorta di sinusoide, mentre quando si hanno lunghe serie di 0 si ritorna al problema di sincronizzazione.



Codifica 4B/5B

L'idea di questa codifica è di far sparire le lunghe sequenze di 0, dunque il segnale viene pre-codificato prima di essere inviato. In questo modo è possibile sfruttare qualsiasi codifica per l'invio effettivo del segnale. Il trucco è quello di suddividere la sequenza di bit in gruppi di 4 bit e di farli diventare una sequenza di 5 bit,

4 bit
0 1 0 0 | 0 0 1 0 | 1 1 0 1 | 1 0 0 0 | 0 0 0 1
↓ ↓ ↓ ↓ ↓
0 1 0 1 | 1 0 1 0 | 1 1 0 1 | 1 0 0 1 | 0 1 0 1
5 bit

tramite una tabella di conversione fissa. In questo modo ogni codice da 5 bit non presenterà più di uno 0 iniziale, né più di due 0 finali, e così facendo non si ha mai più di tre 0 attaccati, ottenendo un'efficienza dell'80%, a discapito di un delay di 5 bit (il ricevitore ha bisogno di 5 bit per decodificare 4 bit) e di un overhead del 20%.

Le possibili combinazioni da 5 bit sono $2^5 = 32$, di cui 16 sono codici utilizzati per la sequenza di dati (*data sequence*), in quanto rispettano la regola appena enunciata degli 0 iniziali, finali e ternari. Dei restanti 16 che non rispettano i criteri, 8 di essi sono usati per le sequenze di controllo (*control sequence*), ovvero particolari codici interpretati dal ricevitore (es.: 11111 è per il canale quando non usato (*idle*), 00100 per fermarsi (*halt*), 00111 reset, 00000 per indicare che il canale è disconnesso, ecc.).

Data Sequence	Encoded Sequence	Control Sequence	Encoded Sequence
0000	11110	Q (Quiet)	00000
0001	01001	I (Idle)	11111
0010	10100	H (Halt)	00100
0011	10101	J (Start delimiter)	11000
0100	01010	K (Start delimiter)	10001
0101	01011	T (End delimiter)	01101
0110	01110	S (Set)	11001
0111	01111	R (Reset)	00111
1000	10010		
1001	10011		
1010	10110		
1011	10111		
1100	11010		
1101	11011		
1110	11100		
1111	11101		

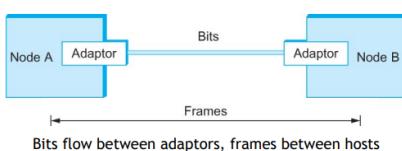
Questa codifica è molto utile per soluzioni come NRZ/NRZI e MP3, le quali hanno ottime performance ma necessitano di assenza di 0 poiché non cambiano il livello di tensione.

Frame

A questo punto è possibile astrarre e passare al secondo strato dell'architettura. I frame sono unità di dati che vengono scambiate tra due interfacce collegate allo stesso mezzo tramite link. A questo punto bisogna implementare dei protocolli che gestiscano un flusso di frame e che rilevino errori (e in caso li correggano).

Quando si è in un canale condiviso da più host, è necessario capire a quale di essi inviare il messaggio, per cui si deve aggiungere qualche altra informazione in più ai bit da trasmettere effettivamente. In questo livello dello stack non ci si cura più dei bit grezzi, ma solamente dei dati da inserire nell'intestazione del pacchetto da inviare. La dimensione del pacchetto di frame, il tipo di informazioni che contengono, ecc. dipende dallo standard utilizzato.

Per quanto riguarda la trasmissione, invece, quando un nodo A desidera trasmettere un frame al nodo B, dice



al suo adattatore di trasmettere un frame dalla memoria del nodo. Ciò comporta l'invio di una sequenza di bit sul link (che può essere punto-punto o condiviso). L'adattatore del nodo B raccoglie quindi la sequenza di bit in arrivo sul collegamento, riconosce il frame corrispondente e lo deposita nella memoria del nodo B. Riconoscere esattamente quale insieme di bit costituisce un frame, cioè determinare dove inizia e dove finisce il frame, è la sfida principale che deve affrontare l'adattatore. Esso deve prima di tutto saper identificare l'intestazione all'interno del frame e capire se effettivamente quel pacchetto è indirizzato a lui o a un altro, dunque deve trovare all'interno dell'header un indirizzo che sia quello suo. Al passo successivo deve controllare che i dati ricevuti siano corretti, in quanto i canali sono soggetti a rumori e altri disturbi, quindi bisogna avere dei meccanismi di controllo e sono inseriti in coda al pacchetto (*trailing*).

Il pacchetto finale sarà quindi composto da header, dati effettivi e trailing. Tutti questi dati per i controlli e interpretazioni sono informazioni aggiuntive ai dati trasmessi che vengono inserite dai vari protocolli e che vengono riconosciute dai medesimi protocolli del ricevente. Di seguito alcuni dei più utilizzati.

Protocollo orientato ai byte

Il protocollo *byte-oriented* è basato su dei byte “sentinella”, ovvero dei simboli/caratteri speciali che indicano l’inizio e la fine dei frame. Questi simboli sono i primi 32 caratteri dell’alfabeto di codici ASCII e vengono utilizzati principalmente quando i contenuti dei pacchetti sono di lunghezza variabile o sono caratteri opzionali.

Esistono diverse versioni tra cui la BISYNC (*Binary Synchronous Communication*), ovvero un protocollo sviluppato dall’IBM (1960) e basato su sentinelle (*sentinel-based*), la DDCMP (*Digital Data Communication Protocol*) basata sul conteggio di byte (*Byte-count based*) e usato dal DECNet, e altre.

BISYNC:

Nella BISYNC, alcune sequenze di byte hanno significati particolari, come ad esempio:

- SYN (*synchronize*, ASCII 22), è un carattere che identifica l’inizio dei frame e che viene spedito doppio
- SOH (*start of header*, ASCII 1) che segnala l’inizio dell’intestazione
- STX (*start of text*, ASCII 2) per l’inizio del testo e ETX (*end of text*, ASCII 3) per la fine
- DLE (*Data Link Escape*, ASCII 16), serve ad esempio nel caso in cui un ETX compare nel body e per evitare che venga interpretato come fine (anomala) del campo testo si fa precedere un DLE (altrimenti il frame risulterebbe corrotto). Questi caratteri extra devono essere aggiunti dal livello di interfaccia / datalink (non viene fatto dai livelli superiori)
- CRC (*Cyclic Redundant Check*) in cui sono contenuti i codici di controllo dei dati (trailer)



Quando i bit vengono convertiti in byte, si procede con l’interpretazione dei dati cercando i due SYN (da cui il nome del protocollo BISYNC) che identificano l’inizio del frame. Una volta localizzati si controlla della presenza del SOH, si leggono poi i byte successivi di intestazione fino al STX, che identifica il testo. Come già detto nell’header ci sono gli indirizzi degli host e altre informazioni che devono essere controllate, per cui si procede con la loro lettura. Una volta terminata si estrae il testo situato tra STX e ETX. Gli ultimi byte (8, 16, 32, ecc.) del CRC, che sono variabili, servono per il controllo dei dati e anche per chiudere il frame, per cui ci si ferma con la lettura dei byte.

Questa soluzione è molto efficace, nonostante introduca un po’ di overhead, infatti per ogni frame bisogna introdurre almeno cinque caratteri in più, che può essere poco o tanto a seconda di quanto grande è il testo da trasmettere (es.: anche un singolo carattere di testo da trasmettere). Per questo motivo, per aumentare l’efficienza si cerca di inviare frame grossi, in modo da ridurre l’overhead.

DDCMP:

Nella DDCMP, si usano campi di lunghezza fissa per cui non serve avere caratteri sentinella. Anche in questo protocollo il preambolo inizia con due SYN, seguiti da un campo classe, dal contatore (*count*) di lunghezza di quanti byte sono contenuti nel corpo del frame, dell’header, dal body e infine dal CRC. Se il count è corrotto si ottiene un errore.



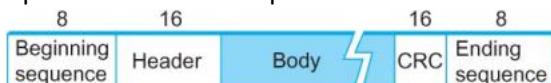
Per identificare dove iniziano e finisco l’intestazione e il body, si hanno dei count interni ai campi che identificano la loro lunghezza totale. In questo modo non si necessita delle sentinelle che creano overhead e quindi si ottengono frame molto più efficienti. Inoltre, dato che sono a lunghezza fissa, talvolta si può evitare di aggiungere i count in quanto si arrangi il protocollo a riconoscere dove iniziano e finiscono i campi (perché

è scritto all'interno del protocollo quale lunghezza obbligatoria devono seguire). Se i dati da trasmettere sono più lunghi della lunghezza fissa, li si possono spezzare su più frame e poi vengono ricomposti dal ricevitore.

Inoltre, questa soluzione, rispetto al BISYNC, non risente del problema dei caratteri speciali che devono essere preceduti da un DLE di escape, in quanto qui non si usano sentinelle come terminatori dei campi. Nel BISYNC questo era un problema nel caso in cui il testo fosse composto da tutti caratteri speciali, per cui si andava a dimezzare la lunghezza utile del body poiché tutti quei char dovevano avere davanti un DLE, quindi alla fine si aveva il doppio dei caratteri da trasportare.

Protocollo orientato ai bit

Similmente a quanto visto coi byte, si può effettuare lo stesso con i singoli bit, ovvero il HDLC (*High Level Data Link Control*) dell'IBM. In questo caso vi sono delle sequenze di bit speciali che identificano l'inizio e la fine del frame, ovvero 01111110. La parte centrale è composta solamente da header, body e CRC.



Sorge subito un problema: cosa succede se nel corpo del testo compare la stessa serie di bit del terminatore? Il protocollo prevede una tecnica chiamata *bit stuffing*, ovvero si rompono le sequenze con più di cinque bit a 1 con degli 0. Il ricevitore sa che se dopo cinque bit a 1 trova uno 0 lo deve scartare a prescindere. Se invece dopo del quinto 1 ne arriva un altro, se viene seguito da uno 0 allora il ricevitore sa che è il terminatore, altrimenti se non è 0 viene interpretato come un errore e viene scartato tutto il frame ricevuto.

Protocollo PPP

Il PPP (*point-to-point protocol*) è un protocollo storico utilizzato (ai tempi dei modem) per fare il framing di dati su linee seriali sulle connessioni Internet e utilizza il HDLC e l'approccio a sentinelle per la trasmissione dei dati. È un protocollo standard che utilizza dei caratteri speciali, chiamati *flag*, per delimitare la lunghezza del frame (es.: 01111110), mentre in mezzo si trovano i campi indirizzo e controllo che sono numeri di default, il protocollo (es.: IP, IPX, ecc.), il payload che di default è di 1500 byte e il checksum per gli eventuali errori.



Il payload è l'unico campo che in questo caso è negoziabile dal punto di vista della lunghezza, ovvero gli host si mettono d'accordo sulla lunghezza di trasmissione e ricezione.

Errori

Al livello datalink, una volta aver riconosciuto i frame, si deve controllare che non vi siano errori (di trasmissione). Alcuni di essi possono essere generati da rumore o da interferenze elettriche, ovvero impulsi elettromagnetici che sfasano i livelli di tensione all'interno di un mezzo, in particolare quando sono di una certa intensità, il cavo in rame diventa un'antenna e assorbe potenza dall'impulso elettromagnetico, col risultato che la codifica non è più la stessa e il decodificatore viene tratto in inganno. Lo stesso fenomeno è percepibile nelle trasmissioni audio in cui si sentono rumori di sottofondo che disturbano la comunicazione. Un fenomeno più raro, invece, è quello del rumore termico.

Il compito di questo strato dell'architettura è quello di filtrare i dati, in modo che ai livelli superiori arrivino solo messaggi corretti e utilizzabili. Esso, quindi, in caso di errore può scartare silentemente il pacchetto senza avvisare i livelli superiori (saranno loro a scegliere poi che fare), oppure notifica il mittente di rimandare il pacchetto, o cerca di correggere il problema (*forward error correction*). Ovviamente, la correzione è limitata a pochi bit o byte e introdurre nel frame meccanismi di correzione aumenta il peso complessivo del pacchetto. Tra i codici di correzione degli errori più conosciuti c'è il codice di Hamming.

Calcolo delle probabilità

Per riconoscere i bit errati, di solito si usano tecniche di tipo probabilistico, ovvero avendo due eventi A e B , la probabilità che avvenga A o B ($P[A \cup B]$) è data dalla probabilità totale (1) a cui si toglie quella in cui non avviene nessuno dei due, quindi la probabilità congiunta di non accadimento di nessuno dei due:

$$P[A \cup B] = 1 - P[\bar{A} \cap \bar{B}] = 1 - P[\bar{A} \cup \bar{B}] = 1 - (1 - p_A)(1 - p_B) = p_A + p_B - p_A p_B \approx p_A + p_B$$

Se questi errori sono rari, dunque bisogna correggere pochi errori (es.: un errore su un pacchetto da 10kb), allora si può provare a correggerlo, altrimenti se la probabilità è più alta diventa sconveniente. Normalmente le linee in fibra ottica presentano errori con una probabilità di uno su un milione o dieci milioni, mentre nel wireless sono uno su diecimila.

Ad esempio, su un pacchetto di $n = 10kb$ (1250 byte) che viaggia su un mezzo con probabilità di avere un singolo bit sbagliato di $p_e \approx 10^{-7}$ (evento indipendente), la probabilità che l'intero pacchetto arrivi corretto è di $1 - P[\text{almeno un bit errato in } 10kb]$, quindi la probabilità di avere almeno un bit errato è di:

$$1 - P[\text{no errori}] = 1 - (1 - p_e)^n = 1 - (1 - 10^{-7})^{10000} = 0.0009995 \approx 10^4 - 10^{-7} = 10^4 * p_e$$

Invece, se si volesse sapere la probabilità di avere almeno due errori ($k = 2$), si avrebbe:

$$\begin{aligned} P[\text{almeno due errori in } 10kb] &\approx (1 - p_e)^{n-k} * (p_e)^k * C(n, k) = \\ &= (1 - 10^{-7})^{9998} * 10^{-14} * 10^4 * \frac{10^4 - 1}{2} \approx 10^{-14} * 5 * 10^7 = 5 * p_e \end{aligned}$$

In cui $C(n, k)$ è la binomiale $C = \frac{n!}{k!(n-k)!}$, cioè il numero di permutazioni di k elementi pescati da n elementi.

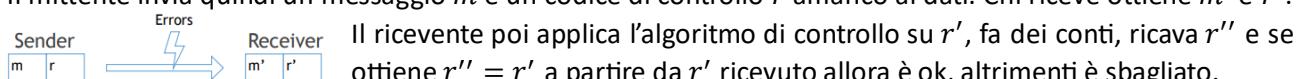
Ovviamente quando si ha a che fare con frame più grandi queste probabilità aumentano, in quanto la lunghezza n determina l'esponente dell'errore $1 - p_e$. Come anche cambiando mezzo fisico di trasmissione la probabilità p_e cambia.

Detezione degli errori

L'idea è quella di aggiungere informazioni ridondanti che consentono di riconoscere l'errore, ovvero il CRC, ma non è l'unico. Uno dei metodi è quello di mandare tutti i bit doppi, in cui se i bit della coppia sono identici bene, altrimenti si scarta il frame, tuttavia non è efficiente.

Di solito la lunghezza del trailing è fissa (k), mentre il body ha lunghezza variabile (n) e molto più grande della precedente ($n \gg k$). Il trailing è il codice di controllo ridondante che non aggiunge alcuna informazione in più al testo, ma fornisce un algoritmo derivato da esso.

Il mittente invia quindi un messaggio m e un codice di controllo r affianco ai dati. Chi riceve ottiene m' e r' .



Parità uni-dimensionale:

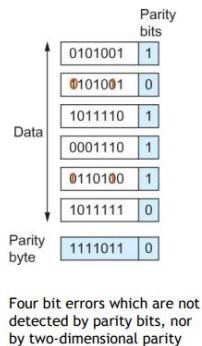
Il *one-dimensional parity*, è un algoritmo di rilevazione degli errori che controlla un bit extra di parità che viene aggiunto a un codice di n bit per bilanciare il numero di 1 presenti nel byte. Ad esempio, se il numero di 1 in una sequenza di 7 bit è dispari, il bit di parità che viene aggiunto sarà 1, altrimenti se essi sono già pari il parity bit sarà 0.

Ogni byte viene controllato separatamente dagli altri e se durante la trasmissione viene invertito uno dei bit del byte (numero di 1 dispari), l'errore viene rilevato. Tuttavia il controllo non è sicuro, poiché se il numero di bit invertiti è pari allora l'errore non viene rilevato correttamente.

Parity bits	
0101001	1
1101001	0
1011110	1
0001110	1
0110100	1
1011111	0

Questo algoritmo molto semplice nacque per il codice ASCII (anni '60), in quale all'inizio usava solamente 7 bit poiché quello extra era riservato al parity bit.

Parità bi-dimensionale:



Il *two-dimensional parity* prende l'algoritmo precedente (parità orizzontale) e ci aggiunge altri bit di controllo per l'intero frame (parità verticale).

Questo sistema dovrebbe essere in grado di rilevare la maggior parte dei bit errati (1 bit, 2 bit, 3 bit e 4 bit o più), in quanto se la singola riga non rilava l'errore doppio (caso precedente), allora lo rileva la colonna. Tuttavia quattro errori non vengono rilevati se sono allineati nella stessa riga e nella stessa colonna.

Questo algoritmo, inoltre, può essere usato per correggere gli errori sul singolo bit.

Internet Checksum Algorithm:

Un altro algoritmo utilizzato per il checksum dei dati è il ICA (*Internet Checksum Algorithm*), che in realtà NON VIENE USATO A LIVELLO DATALINK, ma a livello 3 e 4 dello stack (rete e trasporto). Infatti questo algoritmo si usa per protocolli IP, UDP, TCP, ecc.

Il ICA sfrutta una sorta di parità, solo che rispetto ai precedenti che sommano i bit, questo algoritmo somma interi da 16 bit alla volta. Quindi i dati vengono visti come sequenze di coppie di byte e vengono sommati tra di loro. Quando si arriva in overflow si tronca la somma e si ricomincia. Il risultato finale, complementato a uno, è il *checksum*.

Questo risultato viene spedito assieme al messaggio e il ricevente effettua gli stessi calcoli sui dati ricevuti e compara il suo risultato con quello spedito dal mittente. Se i dati sono corrotti, compreso il checksum, i risultati non combaciano e quindi ci si accorge dell'errore.

Nell'aritmetica di complemento a uno, un intero negativo $-x$ viene rappresentato come il complemento di x (es.: $2 \Leftrightarrow 0010$ se e solo se $-2 \Leftrightarrow 1101$). Quando si sommano numeri in aritmetica di complemento a uno, è necessario aggiungere al risultato un riporto (*carry*) dal bit più significativo. Questo accade perché lo 0 ha due rappresentazioni logiche 00...0 e 11...1, in cui quest'ultima deve essere saltata quando la somma si aggira intorno al dominio rappresentato (motivo per cui, in realtà, i calcolatori usano il complemento a due).

Ad esempio, addizione di -5 e -2 in aritmetica del complemento a uno su interi a 4 bit:

+5 è 0101, quindi -5 è 1010, mentre +2 è 0010, quindi -2 è 1101. Se si somma 1010 e 1101 ignorando il riporto, si ottiene 0111. Poiché c'è un riporto dal bit più significativo, si incrementa il risultato, ottenendo 1000, che è la rappresentazione del complemento a uno di -7, come ci aspetteremmo.

L'algoritmo funziona più o meno nel seguente modo:

```
u_short cksum(u_short *buf, int count) {
    register u_long sum = 0;          \\ 16-bit checksum
    while (count--) {                \\ count 16-bit units
        sum += *buf++;
        if (sum & 0xFFFF0000) { \\ if carry
            sum &= 0xFFFF;      \\ erase carry
            sum++;              \\ increment
        }
    }
    return ~(sum & 0xFFFF);         \\ erase last carry
}
```

Questo algoritmo è molto semplice, buono e robusto, tuttavia non è la soluzione definitiva.

Cyclic Redundancy Check

Un altro algoritmo standard di rilevazione degli errori e che è più usato in quanto universale per la sua enorme flessibilità e efficacia è il CRC (*cyclic redundancy check*), nonostante la sua complicità dal punto di vista matematico. Si usa principalmente per DDCMP, HDLC, CSMA/CD, Token Ring, ecc.

È stato introdotto da Wesley Peterson nel 1961 e rappresenta il modo più compatto per rappresentare tanti controlli diversi, in quanto si possono combinare più livelli di parità e di vari generi all'interno dello stesso checksum o sequenza di bit in maniera modulare.

Questo algoritmo usa l'aritmetica dei polinomi sull'anello dei valori booleani (modulo 2), in cui si rappresenta con i bit i coefficienti numerici dei polinomi e le somme e moltiplicazioni tramite XOR e AND.

Ad esempio, data una stringa 110001, si può usare il seguente polinomio $P(x)$ di grado 5 per rappresentarla:

$$1 * x^5 + 1 * x^4 + 0 * x^3 + 0 * x^2 + 0 * x^1 + 1 * x^0 = x^5 + x^4 + 1$$

Quindi una stringa di bit è un'altra rappresentazione per un polinomio e viceversa. Inoltre, una struttura di k bit ha un grado massimo di $k - 1$. Da qui in avanti si utilizzerà $M(x)$ per indicare un polinomio "messaggio" (in realtà si intenderanno i bit sottostanti) e $C(x)$ un polinomio "generatore di dati".

Aritmetica dei polinomi modulo 2:

I polinomi possono essere sommati, sottratti, divisi e moltiplicati facilmente tramite XOR e AND. Ad esempio se si hanno due polinomi:

$$P_1(x) = \sum_0^n a_i x^i, \quad P_2(x) = \sum_0^n b_i x^i$$

La loro somma è $P_{1+2} = \sum_0^n (a_i + b_i) x^i$, in cui l'operazione eseguita è uno XOR tra i bit, non una semplice somma dei coefficienti. In realtà anche la sottrazione funziona allo stesso modo, infatti si esegue l'OR esclusivo (XOR) su ogni coppia di coefficienti corrispondenti (senza riporto, come per la somma):

$$A(x) + B(x) = A(x) - B(x) = A(x) \text{ XOR } B(x)$$

Questo perché nelle tabelle di verità dello XOR, quando si hanno le stesse cifre si ha 0 e 1 quando diverse. Dunque è indifferente sommare o sottrarre una coppia di polinomi, il risultato è lo stesso.

Per quanto riguarda la moltiplicazione, la si fa normalmente: si moltiplica il primo monomio per il secondo, poi il primo per il terzo, e così via. Solo che quando si effettua la somma, si esegue in realtà uno XOR.

Ad esempio, avendo due polinomi del tipo:

$$P_1(x) = x^3 + x^2 + 1, \quad P_2(x) = x^2 + 1$$

si ottiene come moltiplicazione:

$$P_1(x) * P_2(x) = x^5 + x^3 + x^4 + x^2 + x^2 + 1 = x^5 + x^4 + x^3 + 1 \xrightarrow{\text{in bit}} 11101$$

Anche per la divisione, valgono le stesse proprietà dell'aritmetica polinomiale, ovvero qualsiasi polinomio $A(x)$ può essere diviso da un polinomio divisore $B(x)$ se $A(x)$ è di grado superiore a $B(x)$. Inoltre, qualsiasi polinomio $A(x)$ può essere diviso una volta da un polinomio divisore $B(x)$ se $A(x)$ ha lo stesso grado di $B(x)$: se $\deg(A) = \deg(B)$ allora $A(x)/B(x) = 1 + \text{resto}$, in cui il resto si ottiene sottraendo $B(x)$ da $A(x)$, ovvero $A(x)/B(x) = 1$ con resto $R(x)$ implica $A(x) = 1 - B(x) + R(x)$ e quindi $R(x) = A(x) - B(x)$.

Ad esempio, avendo due polinomi dello stesso grado:

$$P_1(x) = 110101 \xrightarrow{\text{in polinomio}} x^5 + x^4 + x^2 + 1, \quad P_2(x) = 101001 \xrightarrow{\text{in polinomio}} x^5 + x^3 + 1$$

Per ottenere $P_1(x)/P_2(x)$ si deve fare lo XOR logico dei bit, quindi $110101/101001 = 011100$. Da notare che il risultato è di grado inferiore. Convertendo il risultato in polinomio si ha:

$$1 + R(x) = 1 + x^4 + x^3 + x^2$$

Nel caso in cui fosse di grado diverso, invece, si avrebbe un polinomio che ha come grado la differenza del grado del dividendo e del divisore, quindi $\deg(Q) = \deg(A) - \deg(B)$.

Trasmissione dei messaggi tramite polinomi (procedimento teorico):

Dal punto di vista della trasmissione dei dati, bisogna vedere la sequenza di n bit come un polinomio di grado $n - 1$. Si supponga di avere quindi una sequenza di bit da trasmettere in maniera corretta e come controllo di correttezza si decide che deve essere divisibile per un certo periodo prefissato, cioè si sceglie un polinomio generatore C , per cui tutti i messaggi corretti debbano essere perfettamente divisibili per C (resto 0).

Il ricevente, dunque, riceve una sequenza di bit che interpreta come un polinomio molto lungo, la divide per C (che conosce già) e se ottiene 0 come resto accetta il messaggio ricevuto.

Gli errori che si possono verificare, dal punto di vista dei bit, è l'inversione (*flip*) di alcuni di essi. Dal punto di vista dei polinomi, invece, questo corrisponde all'invio di un polinomio $P(x)$ al quale si aggiunge un certo valore polinomiale $E(x)$ che corrisponde all'errore, per cui il ricevente ottiene un $P'(x) = P(x) + E(x)$ che non corrisponde al valore iniziale inviato, cioè $P(x)$. Per accorgersi dell'errore, il ricevente trova degli 1 in corrispondenza dei bit sbagliati (in quanto la somma è uno XOR).

Il ricevitore quando controlla il messaggio ricevuto sa che:

$$R'(x) = \text{resto} \left[\frac{P'(x)}{C(x)} \right] = \text{resto} \left[\frac{P(x) + E(x)}{C(x)} \right] = \text{resto} \left[0 + \frac{E(x)}{C(x)} \right] = \text{resto} \left[\frac{E(x)}{C(x)} \right]$$

in quanto $R(x) = \text{resto} \left[\frac{P(x)}{C(x)} \right] = 0$ è il messaggio corretto. Per cui se si ottiene $R'(x) = 0$ significa che il messaggio è arrivato senza errori, altrimenti se $R'(x) \neq 0$ allora è sbagliato.

Gli unici errori che non vengono rilevati sono quando $E(x)$ è un multiplo di $C(x)$, in quanto il resto è 0, ma significano 32 bit (oppure di più o di meno a seconda del CRC) alterati, il che è poco probabile.

L'unico dato, oltre al messaggio stesso, che sia mittente che destinatario devono conoscere è $C(x)$. Per cui per costruire il CRC, si parte da una certa sequenza di m bit che compone il messaggio con la sua intestazione, ovvero $M(x)$, e si genera un certo $C(x)$ con lunghezza prefissata e che proviene da un frame di lunghezza inferiore agli m bit. Si ottiene quindi un polinomio definito come $P(x) = M(x) * C(x)$, ovvero il pacchetto che poi dovrà essere spedito, e sarà di grado $\deg(M) + \deg(C) = (m - 1) + r = m + r - 1$, cioè le rispettive lunghezze in bit dei due polinomi. Dunque, dati $M(x)$ e $C(x)$ ($\deg(C) = r$), poi si deve trovare un polinomio $R(x)$ con $\deg(R) < r$ (cioè una sequenza R di r bit) da aggiungere ai bit che rappresentano $M(x)$, tale che $M(x) * R(x)$ sia un polinomio $P(x)$ divisibile per $C(x)$. Ciò significa che si deve trovare $R(x)$ tale che

$$P(x) = x^r M(x) + R(x) \quad e \quad P(x) = Q(x)C(x)$$

per qualche $Q(x)$. Cioè:

$$x^r M(x) = Q(x)C(x) - R(x) = Q(x)C(x) + R(x)$$

in cui $R(x)$ è il resto della divisione di $x^r M(x)$ per $C(x)$. Per questo motivo, quando il ricevente effettua la divisione dovrebbe ottenere resto 0 (in assenza di errori).

Trasmissione dei messaggi tramite polinomi (procedimento pratico):

Questo appena visto è il procedimento a livello teorico, mentre a livello pratico si svolge nel seguente modo: l'algoritmo aggiunge r zeri all'estremità di ordine inferiore del frame, in modo che contenga $m + r$ bit e questo dà il polinomio $x^r M(x)$, ovvero $M(x)$ shiftato di r bit; divide poi $x^r M(x)$ per $C(x)$ utilizzando la divisione modulo 2 (vedere procedimento algoritmico più in basso); infine, sottrae il resto $R(x)$ (che è sempre r o meno bit) dalla stringa corrispondente a $x^r M(x)$ usando la sottrazione modulo 2 (immediata: $R(x) \text{ XOR } 0 = R(x)$). Il risultato $P(x) = x^r M(x) - R(x) = x^r M(x) + R(x)$ è il frame con checksum da trasmettere. Da notare che $\text{resto} \left[\frac{P(x)}{C(x)} \right] = 0$.

Il ricevitore riceve $P'(x) = P(x) + E(x)$ e calcola $R'(x) = \text{resto} \left[\frac{P'(x)}{C(x)} \right] = \text{resto} \left[\frac{E(x)}{C(x)} \right]$. Se $R'(x) = 0$, allora scarta r LSB (*least significant bit*), cioè prende $M(x) = x - r P'(x)$; altrimenti si verifica un errore.

Ad esempio:

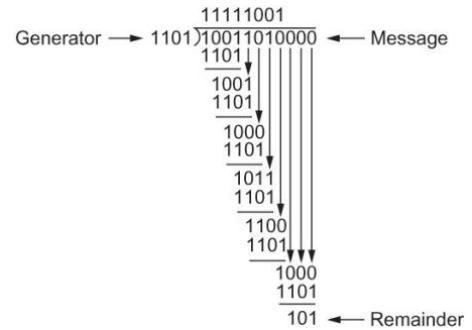
$$M = 10011010 \xrightarrow{\text{in polinomio}} M(x) = x^7 + x^4 + x^3 + x$$

$$C = 1101 \xrightarrow{\text{in polinomio}} C(x) = x^3 + x^2 + 1$$

Dato che $\deg(C(x)) = 3$, si aggiungono tre 0 alla fine di M (quindi il CRC avrà 3 bit).

$$\text{Il resto è } R(x) = x^2 + 1 \xrightarrow{\text{in bit}} R = 101.$$

$$\text{Quindi si avrà: } P(x) = x^3M(x) + R(x) \xrightarrow{\text{in bit}} 10011010101.$$

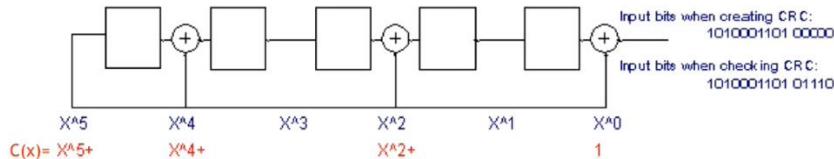


Hardware per il CRC:

Un altro vantaggio di questo algoritmo, oltre all'ottima efficienza, è che può essere implementato facilmente a livello hardware tramite porte `XOR` e `flip-flop`. Ogni adattatore di rete ne ha uno incorporato.

Il circuito è così composto: una volta aver fissato il polinomio generatore (lungo r bit), si costruisce un circuito logico in HW che serve a computare la divisione su polinomi, sia per la generazione del CRC prima di spedire, sia il suo controllo quando si riceve. Il circuito è formato da tanti registri flip-flop per i bit quanti sono i bit del CRC (r) e si mette degli `XOR` nelle posizioni in cui vi sono i bit a 1 del divisore, più uno sull'ultimo bit. Quando arrivano i bit, questi vengono memorizzati e scorsi in avanti e quando viene fatto un clock il bit passa in avanti. Dato che in mezzo ad alcuni bit ci sono degli `XOR`, quello che viene messo dentro a un flip-flop potrebbe essere dato dal bit del flop precedente in `XOR` con quello che esce fuori dall'ultimo. Dopo che tutti i bit sono stati spostati, il contenuto dei registri fornisce il CRC richiesto.

Ad esempio: si ha un messaggio $M = 1010001101$ e $C = 110101$ con $r = 5$, il CRC risultante è $R = 01110$.



Alcune proprietà del CRC:

Sia $P(x)$ il messaggio inviato dal mittente e $P'(x) = P(x) + E(x)$ la stringa ricevuta. $E(x) = x^i$ significa che l' i -esimo bit è stato capovolto nella posizione corrispondente del messaggio $P(x)$. Si sa che $\frac{P(x)}{C(x)}$ lascia un resto di 0, ma se $\frac{E(x)}{C(x)}$ lascia un resto di 0, allora o $E(x) = 0$ o $C(x)$ è fattore di $E(x)$. Quando $C(x)$ è un fattore di $E(x)$ si ha un problema: gli errori passano inosservati.

Se c'è un errore di un solo bit allora si ha che $E(x) = x^i$ determina il bit i in errore. Se $C(x)$ contiene due o più termini, non dividerà mai $E(x) = x^i$, quindi tutti gli errori a singolo bit verranno rilevati. In generale, è possibile dimostrare che una $C(x)$ con determinate proprietà rileva certi tipi di errore. Di seguito i principali.

Se si ha un CRC con $C(x)$ composto da 100...001 con quanti 0 si vuole al centro, con questo polinomio si riesce a rilevare tutti gli errori di un singolo bit, in quanto si riesce a portare avanti nei bit l'errore e quindi lo si rileva. Dunque, per questa proprietà si ha: $C(x) = x^k + x^0$.

Per rilevare gli errori a doppio bit, si necessita di un $C(x)$ che ha un fattore con almeno tre termini.

Il caso più semplice, invece, è quando $C(x) = x + 1 \xrightarrow{\text{in bit}} 11$, in quanto il polinomio è di grado 1 e genera il bit di parità che rileva gli errori dispari. Dunque, $C(x)$ contiene il fattore $x + 1$.

A questo punto resta da individuare qualsiasi errore “burst” (cioè una sequenza di bit di errore consecutivi) per il quale la lunghezza del burst è inferiore a k bit. (È possibile rilevare anche la maggior parte degli errori di burst di lunghezza superiore a k bit). Di seguito l’idea.

Si supponga di avere due polinomi per gli errori, $C_1(x) \neq C_2(x)$ che rilevano errori diversi, e si volesse costruire un polinomio che consenta di rilevare entrambi gli errori. Per fare questo, basta solamente moltiplicare i due polinomi generatori, ovvero $C(x) = C_1(x) * C_2(x)$. Questo funziona perché il polinomio $P(x)$ risulterebbe divisibile per $C(x)$, quindi per entrambi $C_1(x)$ e $C_2(x)$, e di conseguenza quando si effettua il calcolo del resto di $\frac{P(x)+E(x)}{C(x)}$, si ottiene per le stesse proprietà viste prima: resto $\left[\frac{E(x)}{C(x)}\right]$, perché $\frac{P(x)}{C(x)} = 0$.

Dunque, un polinomio $C(x)$ può essere fattorizzato come $C(x) = C_1(x) * \dots * C_n(x)$ se si vogliono garantire più proprietà allo stesso tempo.

Dunque per ogni errore (es.: singolo bit, bit doppio, ecc.) dovuto a qualsiasi tipo (es.: interferenza, rumore, ecc.) è possibile creare un unico polinomio generatore che li individui tutti. Da questo fatto si sono ottenuti dei polinomi generatori che sono diventati standard internazionali:

- **CRC-8:** $C(x) = x^8 + x^2 + x + 1$
- **CRC-10:** $C(x) = x^{10} + x^9 + x^5 + x^4 + x + 1$
- **CRC-12:** $C(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$
- **CRC-16:** $C(x) = x^{16} + x^{15} + x^2 + 1$
- **CRC-CCITT:** $C(x) = x^{16} + x^{12} + x^5 + 1$
- **CRC-32:** $C(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Il CRC-32 è usato per 802.3 (Ethernet), 802.11 (wi-fi), ecc., mentre il PPP usa sia CRC-32 che il CRC-16 con polinomi leggermente differenti.

Altra cosa importante da sapere quando si usa il CRC: esso non dà informazioni su quali bit sono sbagliati, né può correggerli, ma serve solo a rilevare la loro presenza. Servono altri codici per rilevazioni più precise.

Esercizio:

(*Dal testo d’esame “210216.pdf”*):

3) Un certo dispositivo di rete utilizza il CRC FOP-4, il cui polinomio generatore è 10111.

Riceve la sequenza di bit 110101101010. È corretto (e in tal caso, qual è la parte dati) o contiene degli errori?

R: effettuando la divisione, è possibile vedere come è sbagliato (vedi immagine affianco). Da notare che i quattro bit finali della sequenza di bit sono stati staccati in quanto solamente i primi bit sono per i dati.

$$\begin{array}{r}
 11010110 \ 1010 \mid 10111 \\
 10111 \\
 \hline
 11011 \\
 10111 \\
 \hline
 11001 \\
 10111 \\
 \hline
 11100 \\
 10111 \\
 \hline
 1011 \ 1 \\
 1011 \ 1 \\
 \hline
 0010
 \end{array}$$

Trasmissione affidabile

Come già detto, il CRC serve solamente a rilevare degli errori, ma non a correggerli. Tuttavia alcuni codici di correzione sono anche in grado di correggere a mano a mano eventuali errori (*forward error correction*), ma sono più complessi e grandi da trasmettere per cui vengono utilizzati solo nei casi in cui il pacchetto da ricevere non può essere ritrasmesso oppure in presenza di connessioni molto rumorose. Però nella maggior parte dei casi i pacchetti corrotti o errati vengono scartati.

Dato che nella maggior parte di casi si ha a che fare con decine di migliaia di pacchetti e la probabilità di averne uno errato è molto bassa, piuttosto che appesantire il traffico di ogni singolo pacchetto e correggere i loro errori, conviene richiedere un reinvio. È più semplice e economico.

Le connessioni non affidabili (*unreliable*), lascia ai livelli superiori il compito di decidere se farsi reinviare un pacchetto o meno (es.: Ethernet, wi-fi, ecc.). Al contrario, quelle affidabili (*reliable*) devono occuparsi dei frame scartati (es.: PPP). La maggior parte delle volte si implementa il primo.

Le connessioni affidabili, sfruttano principalmente due meccanismi: Acknowledgement e Timeout. Un *acknowledgement* (ACK) è un frame (o messaggio per i livelli superiori) di controllo, che quindi non porta dei dati ma solo informazioni di controllo per informare il mittente dell'avvenuta ricezione o di reinvio del pacchetto. Ad esempio: un mittente invia dei dati a un destinatario, questo risponde con un pacchetto di sola intestazione che informa il primo di avvenuta ricezione o meno del corretto pacchetto.

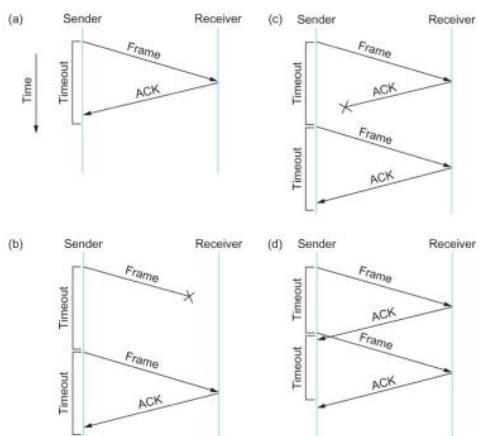
Nel caso in cui il mittente sia in attesa di una risposta dal destinatario e intercorre un certo periodo di tempo senza ricezione di risposta, procede con il reinvio dello stesso pacchetto. Questo tempo trascorso è chiamato *timeout*. Questo meccanismo di acknowledgement e timeout è chiamato *Automatic Repeat reQuest* (ARQ).

Protocollo stop-and-wait

Il meccanismo appena descritto fa parte del protocollo stop-and-wait, in cui ci si pone in ascolto di una risposta dopo l'invio di un messaggio. In questo protocollo esistono quattro diversi scenari:

- Il messaggio ACK viene ricevuto entro il timeout
- Il messaggio originale viene perso, per cui scade il timeout e si reinvia il pacchetto
- Il messaggio originale viene ricevuto, ma l'ACK si perde, per cui si necessita di un nuovo reinvio del pacchetto iniziale, in quanto il mittente non può essere notificato dell'avvenuta ricezione
- Il timeout scade sempre prima della ricezione dell'ACK

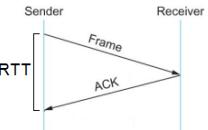
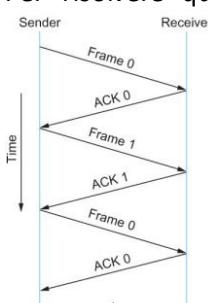
I casi c) e d) sono i più problematici. Nel primo caso il destinatario riceve due volte lo stesso pacchetto per cui si ritrova in una condizione in cui deve capire se l'invio doppio è un fattore voluto dal mittente oppure è perché non è stato ricevuto l'ACK. In realtà anche nel caso d) si ha lo stesso problema dal lato destinatario, solo che si aggiunge un altro dilemma dal punto di vista del mittente, ovvero capire se l'ACK arrivato è del primo o del secondo pacchetto inviato.



Per risolvere questo problema si può semplicemente aggiungere un'informazione extra nell'header (*metadato*) per numerare il frame e distinguere, ad esempio con un flag di un bit. L'ACK di risposta dovrà poi contenere a sua volta il valore del flag del pacchetto ricevuto a cui fa riferimento. Quindi in questo caso il mittente dopo aver inviato un primo frame 0, aspetta il suo ACK con flag 0, dopo aver ricevuto l'ACK passa a inviare il frame 1 e il destinatario risponde con ACK 1 e poi si riparte con lo 0 e poi di nuovo 1 e così via.

In caso di timeout, un frame viene ritrasmesso con lo stesso numero. Quindi, quando il mittente ritrasmette il frame 0, il ricevitore può determinare che sta vedendo una seconda copia del frame 0 piuttosto che la prima copia del frame 1 e quindi può ignorarla (il ricevitore la riconosce comunque, nel caso in cui il primo riconoscimento sia andato perso).

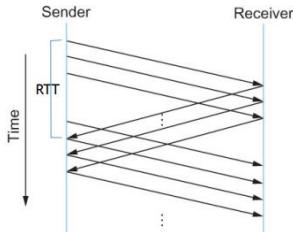
Ad esempio, un collegamento a 1.5 Mbps con 45 ms di RTT (*round trip time* = tempo di andata e ritorno) ha una capacità di $delay \times throughput = 67.5 \text{ Kb} \cong 8 \text{ KB}$, in cui il delay in questo caso è il RTT. Da notare che qui si intende con capacità il riscontro di risposta. Quindi, per sfruttare a pieno il canale servono 8 KB, tuttavia il mittente può inviare solo frame da 1 KB per ogni RTT, dunque esso sfrutta in realtà un ottavo del volume del canale. Infatti,



la sua velocità di invio è di: $bit\ per\ frame \div tempo\ per\ frame = 1024 \times 8 \div 0.045 = 182\ Kbps$. Da qui si può dedurre che più lungo è il RTT, meno si usa il canale, e la colpa è data in parte dal tipo di protocollo utilizzato, in quanto la velocità dipende in maniera preponderante da esso. Infatti, questo protocollo è molto semplice e facile da implementare, tuttavia se non ben ottimizzato fa perdere un sacco della capacità del canale. Inoltre, la capacità di un link è data in linea teorica, poi sta ai protocolli saperla sfruttare al massimo.

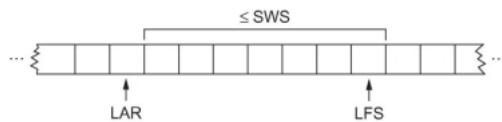
Protocollo sliding window

Per il motivo appena enunciato, esiste un altro tipo di protocollo atto a sfruttare un po' meglio la capacità del canale, ovvero il protocollo della finestra scorrevole (*sliding window*). Tuttavia, il datalink NON prevede meccanismi di recupero, quindi questo protocollo NON È IMPLEMENTATO IN QUESTO LIVELLO DELLO STACK.



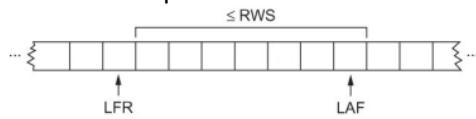
Questo protocollo, utilizzato al livello 4 dello stack (trasporto) e qualche volta nel livello link, è una generalizzazione dello stop-and-wait che consente di mandare più messaggi durante lo stesso RTT in modo da riempire il canale. A questo punto, però, non basterà più avere soltanto un bit per riconoscere i frame. Per questo motivo si aggiungerà un numero di frequenza.

Si supponga, dunque, di avere un buffer di dati dei vari frame da spedire e di avere un numero massimo di frame che possono essere spediti, ma non ancora riscontrati. Con *Sending Window Size* (SWS) si intende la dimensione della finestra, quindi il numero massimo di frame inviati e di cui si sta aspettando un riscontro; con *Last Frame Sent* (LFS) si intende, invece, il numero dell'ultimo frame inviato; con *Last Acknowledgement Received* (LAR), invece, il numero dell'ultimo frame di cui si ha avuto riscontro. A mano a mano che il mittente invia i frame, incrementa il numero di LFS, mentre quando riceve, incrementa il numero di LAR. La cosa importante è che la differenza tra LFS e LAR non superi mai il SWS ($LFS - LAR \leq SWS$). Se questa differenza resta minore, allora il mittente può continuare a inviare dati, mentre in caso contrario deve aspettare. La finestra viene shiftata quando il mittente viene notificato tramite ACK della corretta ricezione.



Si supponga ora di essere nel caso in cui si ha esaurito il numero di invii possibili, si sta aspettando dei riscontri, ma non arriva quello del primo frame inviato. Il mittente legge il numero di ACK e si accorge che non è arrivato il primo, per cui procede con il reinvio del primo frame (con lo stesso numero di frequenza). Il destinatario quindi, vede che ha già ricevuto quel frame, lo scarta e invia l'ACK corrispondente per ridare il riscontro. Lo stesso vale se si esaurisce il timeout di un certo frame.

Come appena anticipato, anche il destinatario ha una sorta di buffer simile a quello del mittente. La sua finestra si chiama *Receiving Window Size* (RWS), ovvero il numero massimo di frame che si possono accettare in tempo, fuori ordine o in ritardo, inoltre ha *Last Acceptable Frame* (LAF) e *Last Frame Received* (LFR), in cui il primo è il numero dell'ultimo frame accettato, mentre il LFR rappresenta l'ultimo numero di frame ricevuto senza buchi (in ordine) e $LAF - LFR$ rappresenta lo spazio ancora accettabile di frame. Anche il ricevente ha lo stesso vincolo sulla differenza che non può superare la dimensione della finestra ($LAF - LFR \leq RWS$), come anche lo shift della finestra avviene solo quando il mittente riceve l'ACK.



Il destinatario, quando riceve un pacchetto, controlla il suo numero di sequenza e se è compreso tra LFR e LAF ($LFR < SeqNum \leq LAF$), allora lo accetta, altrimenti lo scarta. Come abbiamo appena visto, l'ACK che viene inviato porta lo stesso numero del frame ricevuto (ACK ricettivo), mentre in certi casi si usa un ACK cumulativo, cioè quando si rimanda indietro un ACK si tiene in memoria il numero fino al quale tutto è ok e si

tiene il mittente al corrente, in modo da sapere se l'ACK è da reinviare o meno. In questo secondo caso, se un pacchetto non dovesse arrivare al destinatario, esso se ne accorge e non risponde al mittente, il quale si ritrova con i timeout scaduti e procede con un altro invio dei pacchetti mancanti.

Ad esempio, si supponga che LFR = 5, RWS = 4 (cioè che l'ultimo ACK inviato dal ricevitore sia stato per la sequenza numero 5) e LAF = 9. Se arrivassero i frame 7 e 8, verrebbero bufferizzati perché si trovano all'interno della finestra del ricevitore, ma non verrebbe inviato alcun ACK perché il frame 6 deve ancora arrivare. Dunque, i fotogrammi 7 e 8 sono fuori ordine. A questo punto arriva il frame 6 (in ritardo perché è stato perso la prima volta e ha dovuto essere ritrasmesso). Ora il ricevitore riconosce il frame 8, porta LFR a 8 e sposta LAF a 12.

Problemi con questo protocollo:

Mittente e destinatario, in questi casi appena visti, devono tenersi sincronizzati, in modo da sapere quali pacchetti sono stati ricevuti e quali no e agire di conseguenza, restando sempre entrambi dentro alle dimensioni delle loro finestre. Inoltre, solo quando ricevono l'avvenuta ricezione (ACK per mittente e conferma per il destinatario) possono incrementare i loro counter di numero di sequenza.

Tuttavia anche questo sistema ha in realtà una perdita di efficienza, perché comunque bisogna aspettare che gli ACK arrivino, delle volte possono scadere i timeout, bisogna rimandare in caso di errori o scarti, ecc. Un modo per migliorare questa situazione sarebbe quello di fare in modo che mittente e destinatario si dicono cosa non è arrivato e cosa sì, in modo da sapere subito quali frame mancano e reperire alla lacuna il prima possibile. Oppure in alternativa si potrebbe sempre mandare gli ACK, anche se vecchi, in modo da notificare sempre di quali ricezioni sono avvenute (es.: nel caso cumulativo, in cui viene inviato il frame 0, si manda ACK 0, si invia frame 1 che non arriva, frame 2 che arriva, ma il destinatario è fermo allo 0 con l'ok, quindi invia di nuovo un ACK 0 al mittente e questo provvede agli invii mancanti).

Inoltre, diminuendo i dati in transito per via di questi overhead, ritardi e timeout, si sposterebbero meno frequentemente le finestre di invio/ricezione, quindi sia mittente che destinatario sarebbero rallentati, se non del tutto fermi, con la trasmissione dei pacchetti. Il problema si aggrava con gli errori, come detto poco fa, in cui più tempo ci vuole per accorgersi che si è verificata una perdita di pacchetti, più il problema diventa grave.

Per migliorare (parzialmente) la situazione si possono utilizzare diversi ACK in base alla loro utilità/scopo:

- Riconoscimento negativo (*negative acknowledgement*, NAK): il ricevitore invia un NAK per il frame 6 quando arriva il frame 7 (nell'esempio precedente). Tuttavia, questo non è necessario, poiché il meccanismo di timeout del mittente è sufficiente a risolvere la situazione;
- Riconoscimento aggiuntivo (*additional acknowledgement*): il ricevitore invia un ACK aggiuntivo per il frame 5 quando arriva il frame 7 (nell'esempio precedente). Il mittente utilizza l'ACK duplicato come indizio di perdita del frame. Questo è utilizzato nel TCP;
- Riconoscimento selettivo (*selective acknowledgement*): il ricevitore riconosce esattamente i frame che ha ricevuto, piuttosto che il maggior numero di frame. Nell'esempio precedente, il ricevitore riconosce i frame 7 e 8, mentre il mittente sa che il frame 6 è stato perso e quindi può mantenere il canale pieno (al prezzo di una maggiore complessità).

Per quanto riguarda la dimensione dei parametri SWS e RWS, per il primo è relativamente semplice, ovvero è la capacità del canale $SWS = \text{delay} \times \text{throughput}$, quindi bisogna sapere la capacità trasmisiva del canale (bit/secondo) e il RTT, ma nel caso in cui questi fattori fossero ignoti si deve fare una stima impostando alcuni Kbyte e poi ci si adeguà durante la trasmissione; mentre il RWS può essere qualsiasi cosa, in quanto è lo spazio di memoria per le risposte (numeri di ACK).

Tuttavia ci sono dei settaggi da fare, ovvero se $RWS = 1$ significa che i dati devono essere consumati man mano che arrivano, poiché non c'è nessun buffer dal ricevitore per i frame che arrivano fuori ordine, mentre se $RWS = SWS$ il ricevitore può bufferizzare tutti i frame che il mittente invia. Inoltre, se $RWS > SWS$ il buffer del destinatario non viene mai riempito, invece se $RWS < SWS$ alcuni dei frame che il mittente invia non verrebbero memorizzati nel buffer e quindi verrebbero scartati e di conseguenza devono essere reinviati. Infine, il caso $RWS = SWS = 1$ porta al stop-and-wait di ARQ.

Per quanto riguarda i numeri di sequenza dei frame collocati nelle intestazioni, si ha un numero limitato, in quanto il counter è di lunghezza fissa. Inoltre, se il buffer è tanto grande e il numero di bit di conteggio è piccolo, si rischia che dei frame abbiano lo stesso numero di sequenza, pur essendo frame diversi. Per questo motivo, bisogna poter fare un riavvolgimento (*wrap*). È possibile impostare un numero di sequenza maggiore di quello dei frame inviabili: ad esempio nel stop-and-wait era un bit, quindi due numeri di sequenza distinti, mentre per quanto riguarda le finestre scorrevoli, si ha un numero di sequenza massimo di solito definito come $2^n - 1$, dove n è il numero di bit nel campo dell'intestazione, quindi SWS deve esserne più piccolo.

Tuttavia quest'ultima definizione di $maxSeqNum = 2^n - 1$ potrebbe non bastare in alcuni casi, dipende da quanto grande è RWS. Ad esempio, se si avesse un buffer di un byte ($RWS = SWS$), la massima sequenza inviabile è 7, quindi se il mittente inviasse i frame da 0 a 6, il destinatario li ricevesse correttamente e spedisce gli ACK da 0 a 6, ma questi ultimi andassero persi, il mittente ritrasmetterebbe gli stessi pacchetti di prima e il ricevitore li interpreterebbe come corretti, in quanto la sua finestra sta aspettando i frame da 7 a 5 (perché arrivati a 7 si riparte da 0), per cui accetterebbe i valori da 0 a 5, che però non sono corretti.

Per evitare questo problema, bisogna fare in modo che i numeri utili che si hanno nei messaggi siano pochi rispetto a quelli realmente disponibili, ovvero $SWS \leq (maxSeqNum + 1)/2 = 2^{n-1}$, dove n sono i bit per la sequenza di numeri. In questa maniera solo metà sequenza può essere usata per la trasmissione, dopodiché ci si deve fermare e aspettare. Inoltre, si può continuare ad avere $RWS = SWS$. Infatti, riprendendo l'esempio di prima, ora $RWS = SWS = 4$, quindi, quando il mittente invia i frame 0, 1, 2, 3 e si ferma in attesa degli ACK, il ricevitore nel frattempo riceve i frame 0, 1, 2, 3 e invia gli ACK 0, 1, 2, 3 (e si pone in attesa dei frame 4, 5, 6, 7), tuttavia tutti gli ACK vengono persi, quindi il mittente ritrasmette 0, 1, 2, 3, ma non è troppo un problema, in quanto il destinatario che era in attesa di 4, ..., 7 riconosce che i frame 0, 1, 2, 3 appena ricevuti sono copie di quelli vecchi, non li accetta, ma invia nuovamente gli ACK.

Riassunto sul protocollo della finestra scorrevole:

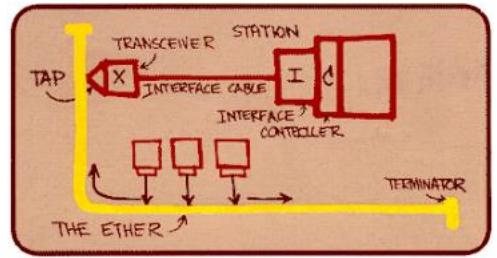
In riassunto questo protocollo serve a tre scopi diversi:

- Affidabilità
- Preservare l'ordine, in quanto ogni frame ha un numero di sequenza e il ricevitore si assicura di non passare un frame al protocollo di livello superiore finché non ha già passato tutti i frame con un numero di sequenza inferiore
- Controllo dei frame (flusso), poiché il ricevitore è in grado di limitare il mittente e si impedisce al mittente di sovraccaricare il ricevitore, trasmettendo più dati di quelli che il ricevitore è in grado di elaborare

Ethernet – IEEE 802.3

L'802.3 è lo standard Ethernet per la comunicazione e è la tecnologia migliore per le LAN dagli ultimi 50 anni. È stato sviluppato a metà degli anni '70 da Bob Metcalfe e altri allo Xerox Palo Alto Research Centers (PARC). Deriva dallo ALOHA, protocollo per pacchetti per reti via radio sviluppato dall'università delle Hawaii (che serviva per comunicare tra isole senza un cavo). Successivamente nel 1978 DEC e Intel si sono uniti alla Xerox per definire uno standard Ethernet a 10 Mbps, da cui poi nel 1985 nacque l'802.3 (e altre varianti in seguito).

Un'altra cosa importante è questo standard garantisce tutt'ora una buona retrocompatibilità con le tecnologie del passato, in quanto sia esso che le sue varianti non si discostano di tanto dal protocollo originariamente idealizzato. Anche per questo motivo è tra i più utilizzati. Invece, vi sono altre tecnologie odierne che sono molto più limitate a livello di retrocompatibilità, ad esempio il Token Ring, il Token Bus, ecc.

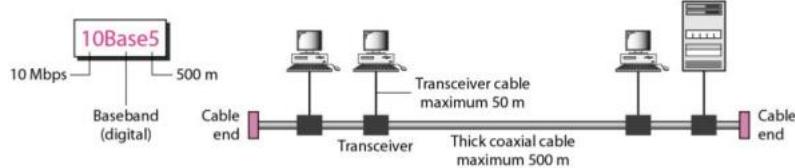


Questo standard è usato al livello 1 e al livello 2 dello stack OSI e definisce come si modifichino i segnali (fisicamente), il mezzo fisico (dimensioni, ecc.) e il protocollo di accesso al mezzo (come si fanno i frame, come li si codifica, cosa fare in caso di frame guasto, ecc.).

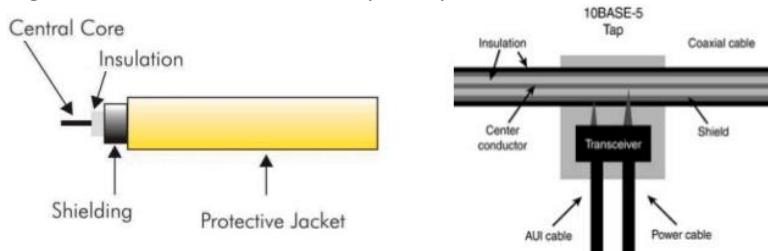
Il protocollo di accesso al mezzo si chiama CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) e consente di regolare l'accesso al mezzo, poiché si ha un insieme di nodi che invia e riceve frame su un collegamento condiviso. *Carrier sense* significa che tutti i nodi possono distinguere tra un collegamento inattivo (*idle*) e uno occupato (*busy*). Accesso multiplo (*multiple access*) significa che ogni nodo può accedere al collegamento e provare a inviare in tempi random dei messaggi (nessun meccanismo di turni o di divisione temporale). Ovviamente, così facendo c'è la possibilità che due o più stazioni inviano contemporaneamente dei messaggi, creando delle "collisioni". Da qui entra in gioco la seconda sigla del protocollo (*collision detection*), ovvero il fatto che ogni nodo sia in ascolto mentre trasmette per essere in grado di rilevare se vi siano state delle collisioni nel canale (si ispira al protocollo ALOHA il quale usava un accesso regolato al satellite per la comunicazione wireless tra le isole hawaiane).

10base5

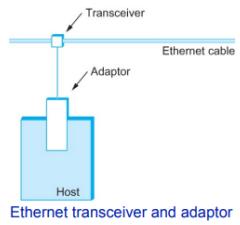
Questa è la primissima versione del protocollo Ethernet e è chiamata così per via della trasmissione da 10Mbps di banda base (digitale) su un cavo da 500m, a cui possono collegarsi al massimo 100 host per ogni singolo segmento di cavo (di 500m ciascuno). Si può quindi immaginare la rete come un unico filo condiviso.



Per la trasmissione fisica si usa un cavo coassiale privo di interferenze elettromagnetiche e che funziona in modo simile a quello di un'antenna televisiva. Esso presenta alle estremità dei terminatori, ovvero dei resistori da 50Ω . Per accedervi elettricamente, ogni stazione si collega sulla linea di trasmissione sfruttando un dispositivo, chiamato *vampire tap*, che fa un buco centrale nel cavo e vi si connette tramite due pin/dentini, uno appoggiato alla maglia del cavo coassiale e uno più in profondità direttamente collegato ai fili interni.



Gli host comunicano tramite un ricetrasmettitore (*transceiver*), un piccolo dispositivo collegato direttamente alla presa, che rileva quando la linea è inattiva e invia il segnale quando l'host sta trasmettendo. Questo dispositivo prende il segnale analogico e lo decodifica riconoscendone i frame. Il ricetrasmettitore è collegato via cavo a un adattatore Ethernet, a cui poi viene connesso l'host concreto. All'interno dell'adattatore si effettuano le creazioni/scompattazioni dei frame, la loro effettiva trasmissione e il riconoscimento di eventuali collisioni e reinvii dei dati. Esso si accorge delle collisioni quando ciò che trasmette è diverso da ciò che ascolta (differenza tra input e output).

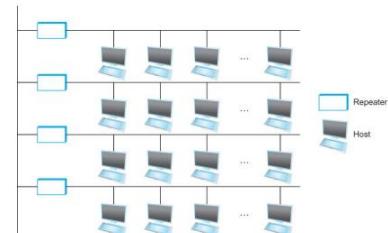
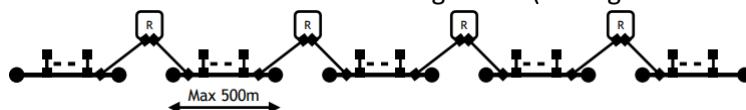


Dal punto di vista elettrico, invece, si ha dei segnali modulati che si propagano lungo il cavo in entrambe le direzioni, con una velocità di $2c/3$ (230000 km/s). I terminali agli estremi del cavo servono ad attutire il

segnale trasmesso. I segnali trasmessi sono codificati tramite codifica Manchester invertita, ovvero 0V per l'assenza di trasmissione (onda quadra che scende e poi risale, al contrario della Manchester normale), mentre $\pm 0.85V$ quando si sta trasmettendo (onda che sale e poi scende). La fine della trasmissione è riconosciuta dalla mancanza di portante, ovvero il *carrier* (non è necessaria una sentinella esplicita). Inoltre, la frequenza delle onde quadre è di 10MHz, poiché il periodo è $\tau = 0.1 * 10^{-6}s = 0.1\mu s$ e la frequenza vale $\frac{1}{\tau} = \frac{1}{0.1*10^{-6}} = 10 * 10^6 Hz = 10MHz$. Il bit rate, che in questo caso corrisponde al Baud rate, è invece 10MΩ, in cui i simboli sono i cambi di tensione (alto→basso è circa 0, basso→alto è circa 1), per cui si hanno 10M simboli, ovvero 10Mbit al secondo.

Tutti questi dispositivi su un segmento da 500m possono essere collegati a altri segmenti tramite ripetitori (*repeater*) di livello 1 che riconoscono una comunicazione in codifica Manchester, la decodificano e la riportano ricodificata nell'altro segmento di cavo. Questa operazione di ritrasmissione non introduce grossi ritardi e non effettua nessun controllo su ciò che viene inviato (si tratta soltanto di quattro transistor).

Lo standard dice che i segmenti possono essere collegati in qualsiasi modo, purché non ci siano loop e non ci siano più di 4 ripetitori tra due host. Una disposizione tipica è quella di un segmento che collega solo i ripetitori (ed eventualmente qualche dispositivo di monitoraggio), chiamato *backbone*, e tutte le stazioni si trovano su ciascun segmento (immagine affianco).



Alcune varianti:

Esistono diverse varianti del cavo coassiale utilizzato per la 10base5, ad esempio:

- il 10base2, detto anche *thinnet*, che è un cavo coassiale più fino (5 mm), corto (200m) e economico, ma mantiene le stesse altre proprietà del 10base5. Questi possono essere concatenati fino a 5 segmenti consecutivi (rispetto ai 4 del 10base5);
- il 10baseT usa coppie di cavi intrecciati (doppini, *twisted pair*) con un massimo di lunghezza di 100m e con codifica 4B/5B o MLT-3 al posto della Manchester. È un po' meno buono del coassiale del 10base5, a meno che non lo si metta all'interno di un cavo schermato e quindi diventa migliore. Tuttavia il metodo di utilizzo è diverso: infatti, questi cavi sono formati da due coppie di doppini (in totale quattro cavi da due fili intrecciati ciascuno) di cui in realtà solo una è usata per connettere due stazioni (solo nella 100baseT e nella 1000baseT si usano tutti), e ogni singolo doppino è full-duplex. Inoltre come per il 10base2 si possono combinare fino a 5 segmenti tramite switch. Questo tipo di cavo è lo standard americano per le linee telefoniche.



Tuttavia esistono altre varianti, che vengono utilizzate ad esempio per canaline o altri posti in cui le necessità di comunicazione differiscono.

Fattori di velocità minima per i cavi di rete

La velocità di propagazione tra un supporto e un altro cambia di tanto in base alla capacità dielettrica degli isolanti (stacco di tensione di un impulso elettrico per propagarsi, quindi il su/giù dell'onda), che dice quanto difficile è stallarli (= mandarli in stallo).

Più è alta la capacità dielettrica dell'isolante (= più sono isolati tra di loro), più è facile. Il vuoto sarebbe l'ideale. Di seguito la formula del fattore velocità (*velocity factor, VF*) di propagazione del segnale:

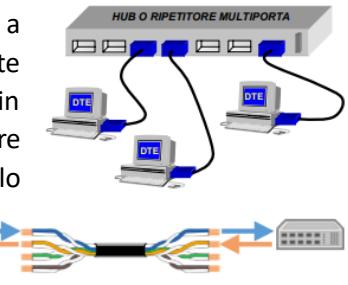
$$\text{Fattore di velocità} = \text{velocità della luce nel mezzo} / \text{velocità della luce nel vuoto}$$

Più basso è il VF, più lento è il segnale che si propaga, maggiore è il ritardo introdotto dalla propagazione del segnale.

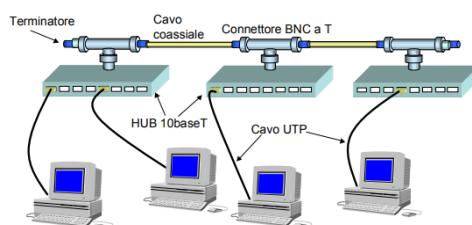
VF (%)	Cable	Ethernet physical layer
74-79	Cat-7 twisted pair	Minimum for 10BASE5
77	RG-8/U	Minimum for 10BASE-FL, 100BASE-FX, ...
67	Optical fiber	Minimum for 10BASE2
65	RG-58A/U	Minimum for 10BASE-T
64	Cat-6A twisted pair	10GBASE-T
64	Cat-5e twisted pair	100BASE-TX, 1000BASE-T
58.5	Cat-3 twisted pair	Minimum for 10BASE-T

10baseT

Le caratteristiche sono state già menzionate poco fa. Questo tipo di cavo serve a connettere in maniera full-duplex due host tra di loro o con altri mediante ripetitore multiplo (multiporta). Questa seconda soluzione è la più adottata, in quanto il ripetitore (di livello 2, *hub*) connette molteplici host e si riesce a formare una rete a stella (*star topology*) efficientemente. Un hub prende un frame e lo replica su tutti gli altri rami/porte a esso collegati (store-and-forward dei frame, controllo CRC, non bit per bit come i ripetitori). Da NON confondere switch e ripetitore multiplo, sono diversi nelle funzionalità (lo switch è più intelligente)!



Reti miste (mixed)



Molto spesso si ha a che fare con reti miste (*mixed*), in cui gli host sono collegati agli hub, che sono a loro volta collegati tra loro da un cavo 10base-5 o -2 (il backbone).

Tuttavia, l'intera rete deve essere intesa come un'unica rete: un frame inviato da qualsiasi host viene inoltrato a tutti i segmenti e a tutti gli host.

Altre varianti

Come già detto prima, l'Ethernet ha una grande versatilità e adattabilità al progresso tecnologico e soprattutto senza essere stravolto. Oggiorno si utilizzano standard 802.3 molto estesi, come ad esempio:

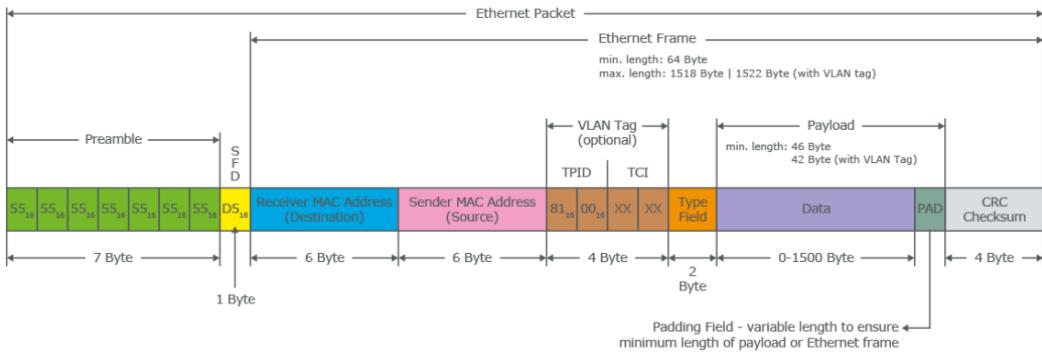
- 100BASE-TX (Fast Ethernet): versione a 100 Mbps su doppino (non disponibile su cavo coassiale)
- 100BASE-FX: lo stesso, su fibra ottica
- 1000BASE-T (Gigabit Ethernet): 1000 Mbps su doppino
- Altre varianti fino a 1 Tb/s sono in fase di sviluppo

Il livello fisico e il protocollo di accesso possono variare, ma il frame è sempre lo stesso (quindi può essere spostato tra diversi segmenti). Il successo di Ethernet ha aperto la strada alla definizione di uno standard simile per le comunicazioni wireless: 802.11, noto come Wi-Fi (mentre l'802.15 (Bluetooth) non è del tutto simile all'802.3).

Frame Ethernet

Formato del pacchetto

I frame Ethernet sono quelli utilizzati nel livello 2 dello stack (datalink) e hanno il seguente formato:



Nell'immagine, il frame Ethernet è la parte appena dopo del preambolo del pacchetto e che prosegue fino al checksum. Il preambolo è una serie di 7 byte che non porta dati, ma serve solamente a iniziare la connessione, e ogni byte ha un valore fisso di $55_{base16} = 01010101_{base2}$, che consente di formare un'onda quadra di sincronizzazione quando il segnale viene codificato tramite codifica Manchester. Al termine del preambolo c'è un byte $D5_{base16} = 11010101_{base2}$ (*Start of Frame Delimiter*, SFD), che serve a identificare l'inizio effettivo del frame da trasportare. Questi primi byte possono essere considerati una sorta di intestazione del pacchetto livello datalink.

Il frame Ethernet a sua volta ha un'intestazione, seguita da un payload e un CRC (trailer). L'header e il CRC hanno dimensione fissa, per cui l'algoritmo che legge la composizione del frame sa come riconoscerli. L'header è composto solitamente dai 14 byte (indirizzo di destinazione, indirizzo di sorgente e tipo), ma può presentare degli ulteriori tag per la VLAN. Il payload ha una lunghezza minima di 46 byte, che se non vengono riempiti si aggiungono dei byte "fasulli" nel campo PAD (*padding field*), e una lunghezza massima di 1500 byte. Il checksum (CRC-32) è formato da 4 byte.

Il campo tipo (*type*) identifica il tipo di payload in base al tipo di protocollo situato al suo interno (es.: IP, ARP) e specifica a quale livello (3 di solito) dello stack inviare il pacchetto.

La fine della trasmissione si riconosce quando il segnale torna a 0 e il conteggio del CRC può terminare. Questo calcolo della lunghezza avviene in fase di ricezione, in cui l'hardware si occupa di contare quanti byte arrivano. Per cui quando vengono registrati gli 0 del termine del segnale si ferma anche il calcolo del CRC. Se gli ultimi byte ricevuti sono tutti 0 si sa che la trasmissione è terminata, altrimenti si prosegue col buffer.

Indirizzi Ethernet

Gli indirizzi identificano le stazioni, ovvero ogni stazione/dispositivo ha un numero univoco a livello globale e viene fissato dal produttore. Con 6 byte sono possibili $2^{48} = 2,8 * 10^{14}$ schede di rete. Bisogna precisare che l'indirizzo non appartiene in realtà all'host, ma al suo adattatore (come per le automobili identificabili non dalla targa ma dal numero di telaio). I primi 3 byte dell'indirizzo identificano il produttore (es.: 8:0:2b è la DEC, Digital Equipment Corporation), mentre i restanti 3 byte sono del prodotto effettivo, il che significa che quando un produttore termina i byte seriali dei prodotti, deve utilizzare un altro numero identificativo per i primi 3 byte. In questo modo si evita il *clash*, ovvero l'utilizzo più volte delle stesse cifre per dispositivi diversi.

Gli indirizzi servono a identificare mittente e destinatario, in particolare se l'adattatore, mentre riceve i byte del frame, si accorge che il frame non è destinato a lui, lo scarta. Tuttavia, c'è un indirizzo particolare che può essere utilizzato: il FF:FF:FF:FF:FF:FF (tutti 1) ovvero l'indirizzo di *broadcast* che viene sempre accettato da tutte le stazioni connesse alla rete e in ascolto. L'adattatore quanto constata che l'indirizzo di destinazione è riferito a lui, controlla il CRC e passa il frame all'host.

Algoritmo di ricezione Ethernet

Con i mezzi condivisi (es.: coassiale, repeater, ecc.), realizzare un sistema broadcast è facilissimo, in quanto tutti i pacchetti inviati sul bus condiviso sono visti da tutti. Per cui con un pacchetto solo si riesce a recapitare il messaggio a tutti i dispositivi.

Il sistema *multicast* funziona in maniera molto simile. Anch'esso sfrutta la comunicazione condivisa, solo che viene identificato da una parte della rete, in particolare è possibile programmare i dispositivi in modo tale che ascoltino anche un certo insieme di indirizzi multicast. Essi si rappresentano con i primi bit dell'indirizzo a 1.

Dunque, in riassunto, un adattatore accetta solamente i pacchetti destinati al suo stesso indirizzo (6 byte), a un indirizzo broadcast e a un certo insieme di indirizzi multicast di cui è stato istruito di riconoscere. Tutti gli altri pacchetti unicast o multicast non appartenenti a esso, vengono scartati.

Quindi, per mandare un pacchetto a una determinata stazione bisogna conoscere il suo indirizzo preciso. Talvolta per riconoscere un dispositivo si invia in broadcast un pacchetto ARP a cui risponderà solamente il dispositivo con l'indirizzo corrispondente (es.: per chiamare una persona ci si fa dare il suo numero di telefono chiedendolo), ma questo argomento si vedrà in seguito.

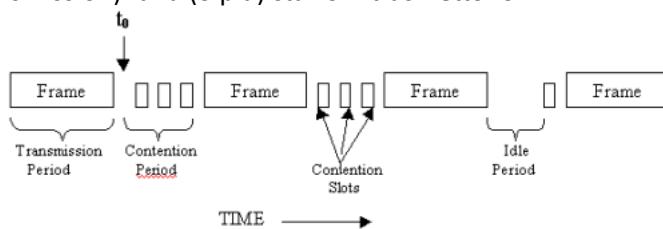
Algoritmo di trasmissione Ethernet

È la parte più complessa di questo meccanismo. L'algoritmo prende il nome di *Media Access Control* (MAC) e viene eseguito completamente in hardware dall'elettronica. Sostanzialmente è un algoritmo che prende una certa sequenza di byte e si occupa di gestire la sua trasmissione codificando i bit.

Una volta realizzata l'intestazione del frame da trasmettere, l'algoritmo pone l'adattatore in ascolto sulla linea e in caso di OV (linea libera) si accinge a far trasmettere il pacchetto. Invece, se la linea è occupata, si sa che il limite superiore del messaggio è di 1500 byte, il che significa che l'adattatore può occupare la linea per un tempo fisso, per cui si aspetta un certo intervallo di tempo (chiamato *inter packet gap*, IPG) e poi si riprova a inviare. Per questo motivo l'algoritmo viene chiamato *1-persistent CSMA/CD*, ovvero l'algoritmo è sicuro che prima o poi invii i dati (non rinuncia alla trasmissione), quindi si ha probabilità 1 di invio in caso di linea *idle*.

Il CSMA/CD ha tre stati:

- Inattività (idle): nessuna stazione ha frame da trasmettere
- Contesa (contention): le stazioni che vogliono trasmettere devono aspettare che la linea sia libera, ovvero non c'è segnale sulla linea per un tempo IPG (l'equivalente di 8 byte=96 bit; 9.6 µs in 10Base-5 e 2). Questo perché in Ethernet non c'è un turno preciso di comunicazione: si prova a trasmettere
- Trasmissione (transmission): una (o più) stazioni trasmettono



Il fatto di provare a trasmettere, può generare collisioni sulla linea, ovvero le onde dei segnali dei messaggi di due o più adattatori che trasmettono in contemporanea si collidono/sovrappongono tra di loro durante la comunicazione attraverso il canale. Questa collisione non rende possibile la decodifica.

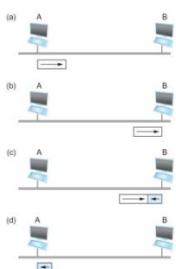


Per questo motivo il protocollo prevede un meccanismo di ricezione delle collisioni (*collision detection*, CD). Quando viene rilevata una collisione (la differenza tra ciò che trasmette e ciò che riceve non è 0), si abortisce la trasmissione e per comunicare l'azione a tutti gli altri dispositivi in ascolto si continua a trasmettere qualche altro bit per riempire il disturbo (*jamming sequence*). Quest'ultima sequenza extra di bit che viene inviata serve a "concludere" il pacchetto che non si è riusciti a inviare, e di solito corrisponde a 32 bit nel caso in cui sia stata inviata solo l'intestazione (64 bit + 32bit = 96 bit totali, chiamati "*runt frame*"). Questo in realtà è se i due host sono vicini, altrimenti se fossero stati più distanti, avrebbero dovuto trasmettere più a lungo, e quindi inviare più bit, prima di rilevare la collisione. Nel caso peggiore, si ha una collisione quando arrivano gli ultimi byte (o bit).

È proprio da quest ultimo fatto menzionato che derivano gli standard di lunghezza dei cavi: se due endpoint fossero tanto distanti tra loro, al termine di una trasmissione il messaggio potrebbe non essere ancora arrivato a destinazione, per cui l'altro endpoint rileva la linea inattiva e inizia a trasmettere, a un certo punto però la collisione avviene e quindi entrambi gli endpoint devono ritrasmettere. Questo accadrebbe di rado se i collegamenti tra dispositivi fossero più brevi.

È da ricordare che un adattatore è in ascolto di collisioni solamente quando sta trasmettendo (altrimenti sarebbe in ascolto per l'arrivo di pacchetti), in quanto è in quel momento che può accorgersi di un altro host che sta inviando un segnale. A maggior ragione, se la linea fosse molto lunga un adattatore termina di ascoltare la collisione al termine del messaggio, ma in realtà questa avviene più tardi, ovvero quando questo sta aspettando un pacchetto in arrivo, per cui non si accorge subito che il segnale in propagazione non è in realtà un messaggio effettivo, ma una collisione, e che quindi la sua trasmissione precedente non è andata a buon fine.

Dunque, si deve avere un cavo di una lunghezza che consenta di poter intercettare un'eventuale collisione anche nel caso pessimo. Per cui il tempo minimo di trasmissione del frame (compreso il tempo di codifica) deve essere maggiore del doppio del tempo di propagazione, ovvero $t_{trasmissione} > 2 * t_{propagazione}$. Il



tempo di propagazione è determinato dal mezzo, quindi dalla fisica.

Questa formula si ricava dal fatto in cui, se un adattatore *A* stesse iniziando a trasmettere in un periodo di tempo *t* (a), il primo bit che arriverà a un adattatore *B* sarà nel tempo *t* + *t_{prop}* (b), per cui se nel frattempo *B* avesse iniziato a trasmettere vi sarebbe una collisione che non appena viene rilevata da *B*, questo invia un frame runt (*jamming*) di 32 bit, tuttavia questo segnale arriverà a *A* solamente nell'istante *t* + 2 * *t_{prop}*, per cui *A* dovrebbe trasmettere 2 * *t_{prop}* dati per essere sicuro di rilevare tutte le possibili collisioni.

Per quanto riguarda lo standard, è previsto un tempo minimo di trasmissione di 51.2 µs, poiché si impone un tempo massimo di propagazione di 25.6 µs. Ad esempio, nel caso di un cavo da 2500 m si ha un tempo di propagazione di 11 µs ($2500 \text{ m} / 2.3 * 10^8 \frac{\text{m}}{\text{s}}$) a cui va sommato il ritardo dei repeater (massimo 4 repeater con 3.7 µs di delay ciascuno). Per avere la certezza che il frame appena inviato non sia entrato in collisione con un altro frame, il trasmettitore può avere bisogno di inviare fino a 512 bit (nel caso di una Ethernet da 10 Mbps), ovvero ogni frame Ethernet deve essere lungo almeno 512 bit (64 byte) perché:

$$14 \text{ byte di intestazione} + 46 \text{ byte di dati} + 4 \text{ byte di CRC}$$

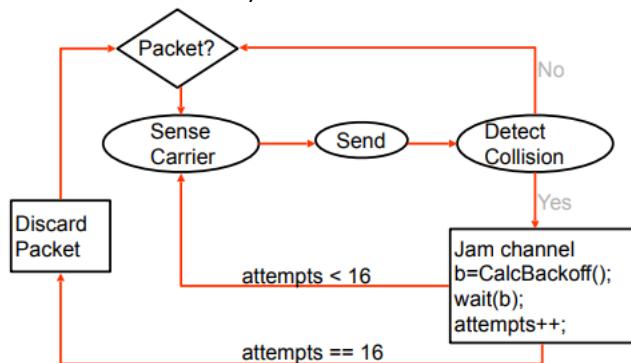
Per questo motivo c'è la necessità di introdurre un eventuale padding nel payload, in quanto serve a raggiungere questa lunghezza minima per una corretta trasmissione e rilevazione di collisioni.

Post collisione – backoff esponenziale

Quando avviene una collisione, gli endpoint producono un breve segnale di jamming e poi aspettano un tempo casuale prima di riprovare a trasmettere. Questo perché se provassero subito dopo a ritrasmettere rincorrerebbero nello stesso problema in quanto invierebbero il messaggio nel medesimo momento.

In realtà, il tempo che i due endpoint devono aspettare è quello dell'IPG più il tempo casuale, di solito abbastanza breve, e che viene scelto in un range di possibili ritardi. Il range di ritardi raddoppia a ogni tentativo da cui il nome di questo meccanismo, ovvero *backoff esponenziale*. Infatti, il tempo di trasmissione di un frame non è deterministico: dipende dal numero di collisioni, cioè da quanti adattatori stanno cercando di trasmettere, cioè dal carico complessivo della rete.

Inoltre, al crescere dell'ampiezza del range di delay diminuisce la probabilità di ulteriori collisioni tra i due endpoint. Dopo n tentativi (il range andrebbe da 0 a $2^n - 1 \mu\text{s}$), con $10 < n < 16$, l'esponente del range resta fisso a 10 e se si verificano più di 16 collisioni consecutive, il pacchetto viene scartato e si segnala un errore. Tuttavia, questa situazione è PRATICAMENTE IMPOSSIBILE, si verificherebbe solo se il cavo avesse qualche problema hardware (es.: terminatore mancante).



In realtà, nelle reti odierne (es.: topologia a stella) e con cavi twisted-pair (o simili), il problema delle collisioni è quasi assente, quanto questo è un dilemma che tende a sussistere nei mezzi condivisi, poiché si ha sullo stesso media una serie di dispositivi connessi. Per questo motivo, anche nelle aziende si cercano soluzioni efficienti e soprattutto che abbiano affidabilità quando ciò che viene utilizzato ha un certo rischio (es.: aerei).

Efficienza della Ethernet

[** LEZIONE S02E7 PARTE 2 SENZA AUDIO **]

L'efficienza è data dal rapporto tra il tempo di trasmissione e il tempo effettivo necessario per trasmettere il frame.

$$\begin{aligned} \text{Efficienza} &= \text{frazione di lungo periodo di trasmissioni riuscite} \\ &= \text{frazione di lungo periodo del canale utilizzato per i dati} \end{aligned}$$

Come già intravisto, il protocollo CSMA/CD non è facile da analizzare, a causa della natura non deterministica e del fatto che non tutte le stazioni si comportano allo stesso modo. Esistono due situazioni semplificabili in:

- Quando solo una stazione trasmette
- Quando tutte le stazioni si comportano allo stesso modo

Quando solo una stazione trasmette:

In questo caso, una sola stazione trasmette sul canale, mentre tutte le altre ricevono. Questa è la situazione di una rete LAN basata su hub o switch: ogni doppino è un dominio di collisione, con un trasmettitore e un ricevitore. In questa situazione, non ci sono collisioni: un frame viene inviato con successo subito dopo l'IPG.

Ad esempio, se si ha:

- IPG = 9,6 μ s = 96 bit = 12 byte
- Preambolo + SFD = 8 byte
- P = lunghezza del payload (fino a 1500 byte)
- Intestazione + CRC = 18 byte

L'efficienza teorica è di: $P/(12 + 8 + P + 18) = P/(P + 38) = 1/(1 + 38/P)$.

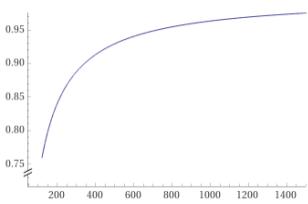


Più grande è la P, più efficiente è il protocollo.

Nel caso peggiore con padding: P=1, ma P effettivo=46 (a causa del padding) \rightarrow efficienza = $1/(46 + 38) = 1/84 = 1,2\%$.

Nel caso peggiore senza padding: P=46 \rightarrow efficienza = $46/(46 + 38) = 54\%$.

Nel caso migliore: P=1500 \rightarrow efficienza = $1500/(1500 + 38) = 97\%$.



Quando si hanno n stazioni che competono:

Si hanno n stazioni che competono su un mezzo condiviso (es.: cavo, hub, ...). In ogni slot, ogni stazione vuole trasmettere un frame (nuovo o ritrasmesso dopo una collisione) con la probabilità p . Si denota con $N * p$ il numero medio di stazioni disposte a trasmettere in ogni slot. In altre parole, Np è il carico globale della rete.

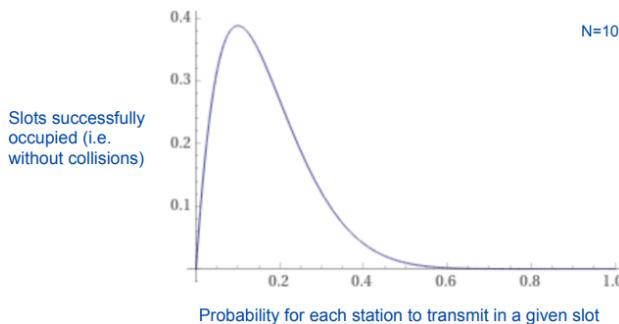
La trasmissione ha successo se per l'intero slot una stazione non rileva collisioni (e quindi tutte le altre stazioni non hanno tentato di trasmettere). In questo caso, l'intero frame viene trasmesso (eventualmente impiegando più tempo di uno slot). In caso di fallimento (inattività o collisione), si tenta lo slot successivo (*1-persistent*).

Se si volesse calcolare qual è la probabilità che uno slot venga utilizzato con successo per la trasmissione di un frame, si avrebbe che:

$$P(\text{slot utilizzato con successo per una trasmissione}) = P(\text{esattamente 1 su } N \text{ trasmette}) = \\ = Np(1 - p)^{N-1}$$

La massima probabilità si ha quando $p = 1/N$, cioè $Np = 1$. Quindi, per evitare collisioni e ottimizzare il throughput globale (cioè massimizzare l'occupazione utile del supporto condiviso), più sono gli host, meno è probabile che trasmettano.

Ad esempio: $P(\text{successo}) = \left(1 - \frac{1}{N}\right)^{N-1} \approx 1/e = 0,37$ (per N grande), quindi al massimo, il 37% degli slot è utilizzato da trasmissioni di successo (per N grande) e il numero medio di tentativi di invio di un frame è $1/(1/e) = e = 2,7$.



Sia t_{trans} = tempo di trasmissione del carico utile (payload) = $\frac{\text{carico utile}}{\text{bitrate}}$, tempo di slot = $2t_{prop}$ e t_{oh} = overhead del frame = preambolo + header + CRC = $\frac{26 \text{ byte}}{\text{bitrate}}$. Il tempo medio di trasmissione (ovvero massimizzazione del throughput, cioè $Np = 1$) è di $IPG + e * 2t_{prop} + t_{trans} + t_{oh}$, perché si deve provare e volte, in media.

$$\text{Quindi: efficienza} = \frac{t_{trans}}{t_{trans} + IPG + t_{oh} + 2et_{prop}} = \frac{1}{1 + (IPG + t_{oh} + 2et_{prop})/t_{trans}}.$$

Per aumentare l'efficienza si può:

- t_{prop} tendente a 0: se il ritardo di propagazione è zero, i nodi in collisione interromperanno immediatamente senza sprecare il canale
- t_{trans} molto grande: quando una stazione afferra il canale, lo manterrà per un tempo molto lungo; quindi il canale svolgerà un lavoro produttivo per la maggior parte del tempo.

Nel caso di Ethernet 10base-5 (10Mbps), tempi di misurazione in byte sono:

- IPG = $9,6\mu\text{s} = 96 \text{ bit} = 12 \text{ byte}$;
- $t_{prop} = 25,6 \mu\text{s} = 256 \text{ bit} = 32 \text{ byte}$;
- $t_{oh} = 8+14+4 = 26 \text{ byte}$;
- P = payload = carico utile (in byte)

Da cui si ottiene: $\text{efficienza} = \frac{1}{1 + (38 + 64e)/P} \approx \frac{1}{1 + 212/P}$.

Se si avesse $P=1500$ byte, si avrebbe un'efficienza del 87,6%. Anche se, una rete Ethernet classica a 10 Mbps a cui accedono molte stazioni equivalenti fornisce in realtà non più di 8,7 Mbps complessivi, da condividere tra tutte le stazioni. Se si avesse, invece, $P=46$ byte si avrebbe un'efficienza del 17,8%.

L'efficienza diventa 1 quando t_{prop} arriva a 0, o P all'infinito, per cui è meglio nelle reti piccole e/o con frame grandi. Questo è uno dei motivi per cui gli standard Ethernet più recenti consentono distanze massime più brevi (gli ultimi standard sono ≤ 35 m) e frame più grandi ("Jumbo frame", fino a 8KB).

Anche se, in realtà, le reti Ethernet funzionano meglio in condizioni di carico leggero, in quanto in condizioni di carico elevato, troppa capacità della rete viene sprecata dalle collisioni. In effetti, la maggior parte delle Ethernet viene utilizzata in modo conservativo, ovvero hanno meno di 200 host collegati: un numero di gran lunga inferiore al massimo di 1024. Inoltre, la maggior parte delle Ethernet è molto più corta di 2500 m, con un ritardo di andata e ritorno di circa 5 μs , molto inferiore al limite di 51,2 μs .

Altro fattore considerevole è che le Ethernet (classiche) sono facili da amministrare e mantenere, poiché non ci sono switch che possono guastarsi, né tabelle di routing e di configurazione da tenere aggiornate (anche se, non è più vero nelle Ethernet commutate...).

Inoltre, è facile aggiungere un nuovo host alla rete e è poco costoso in quanto il cavo è economico e l'unico altro costo è l'adattatore di rete su ogni host, di cui si ha comunque bisogno (anche se non è più vero nelle reti Ethernet commutate...).

Per questi motivi, Ethernet è stata in grado di tenere il passo con il miglioramento della tecnologia e ha avuto un enorme successo, per cui sono state sviluppate molte varianti (deterministiche, es.: industriali, avioniche, ferroviarie, ecc.) e i concorrenti (IEEE 802.4 (*Token Bus*), 802.5 (*Token Ring*), ecc.) sono scomparsi.

Collegamenti wireless

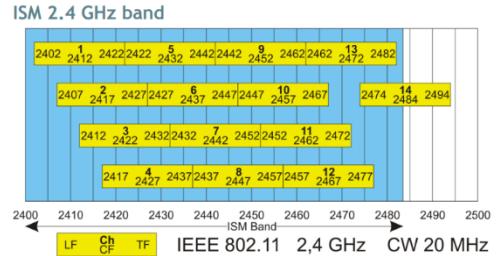
I collegamenti wireless usano segnali elettromagnetici trasmessi nell'etere (es.: onde radio o microonde, segnali ottici infrarossi, ecc.). A differenza dei cavi in rame, in questo caso non si riesce a confinare il segnale, infatti nei mezzi cablati si ha un segnale all'interno di un cavo e da lì non può uscire, mentre con le onde nell'etere si ha molta dispersione ("a sfera") e disturbi. Infatti, una sfida è quella di riuscire in modo efficiente a condividere questo spazio etere senza far interferire indebitamente i segnali l'uno con l'altro e una soluzione è quella di dividere i segnali per tempo, frequenza e spazio (area geografica).

Per questi motivi appena annunciati, la trasmissione via etere è molto più regolamentata e con protocolli rigidi di quella via cavo, specialmente per quanto riguarda l'accesso al mezzo (infatti lo spazio di comunicazione viene concesso dallo Stato, come per l'accesso delle spiagge libere). La violazione di un range di frequenza e/o di interferenza su frequenze non proprie corrisponde a una severa punizione.

A seconda delle frequenze e degli ambiti di applicazione, si hanno degli standard mondiali e nazionali. Ad esempio la radio e la televisione utilizzano frequenze riservate di un certo tipo, diverse ad esempio da quelle militari o governative. Quindi se un'azienda volesse avere delle frequenze per sé dovrebbe farsene concedere dallo Stato, dietro compenso, per un periodo limitato (es.: 10 o 20 anni), dopo del quale ritorna in mano allo Stato (perché non possono essere acquistate poiché risorse nazionali). Talvolta vengono concesse tramite asta. Ogni tanto alcune tecnologie devono migrare di frequenze perché gli standard vengono modificati e quindi può essere che le licenze scadano prima del termine (es.: il digitale terrestre).

Alcuni range di frequenze, invece, sono liberalizzati, ovvero le bande ISM (*Industrial, Scientific, Medical*) in cui si può trasmettere liberamente senza necessità di una licenza (es.: 443 kHz è la frequenza dei vecchi telecomandi per aprire i cancelli di casa).

Nell'immagine affianco c'è l'esempio della banda 2.4 GHz (campo del microonde), che va da 2400 MHz a 2480 MHz. È una banda cosiddetta "rumorosa" per via dell'influenza con l'acqua (principio del forno a microonde, ovvero l'acqua che vibra genera questa frequenza). Questa banda è divisa in 14 canali, in cui ognuno ha uno standard centrale e una frequenza minima e una massima che servono a calcolare la capacità di trasmissione.



Tuttavia, anche queste frequenze liberalizzate hanno una regolamentazione molto rigida. Ad esempio, la potenza di trasmissione è molto limitata e dipende dalla frequenza utilizzata (es.: 100mW max per 2.4 GHz, mentre per le frequenze radio è 15-20 mW). Questa restrizione serve per non creare interferenze con altri segnali, ma anche perché l'intensità del segnale scende col quadrato della distanza quindi anche la portata è molto limitata fisicamente in sé e dipende anche da eventuali ostacoli.

Seconda restrizione è quella sulla tecnica di trasmissione, atta anch'essa a evitare interferenze, in quanto vi potrebbero essere altri dispositivi che vanno uso dello stesso range di frequenza. Dunque, bisogna adottare un meccanismo che minimizzi i *clash* (collisioni/interferenze). Una soluzione sbagliata è quella di scegliere una frequenza e di modulare l'ampiezza del segnale. Tuttavia una signora (attrice) di nome Hedy Lamarr, ideò per le frequenze a uso militare un sistema, che si usa ancora oggi per il Bluetooth, e che sfrutta la tecnica dello *Spread Spectrum*, ovvero di distribuire il segnale su una banda di frequenza più ampia, in modo da ridurre al minimo l'impatto delle interferenze provenienti da altri dispositivi.

Spread Spectrum

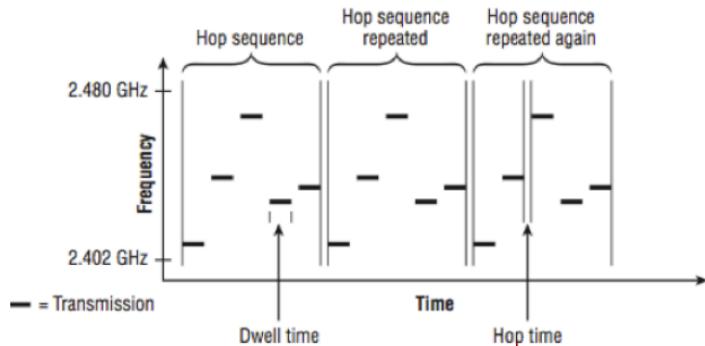
Le tecniche di Spread Spectrum sono tutt'ora utilizzate per il Bluetooth e servono a diminuire le interferenze dei segnali via etere, tuttavia non garantiscono la segretezza dei dati (per quella esistono altre metodologie). Di seguito due diverse tecniche di Spred Spectrum:

Spettro diffuso (*Spread Spectrum*) con salto di frequenza:

Si trasmette il segnale su una sequenza "casuale" di frequenze, ovvero prima si trasmette a una frequenza, poi a una seconda, poi a una terza e così via. In realtà la sequenza di frequenze non è del tutto casuale, ma è calcolata algoritmamente da un generatore di numeri pseudo-random.

Il ricevitore utilizza lo stesso algoritmo del mittente, lo inizializza con lo stesso seme (*seed*) ed è in grado di saltare le frequenze in sincronia con il trasmettitore per ricevere correttamente il fotogramma.

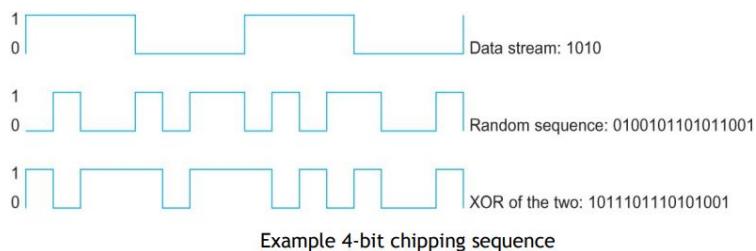
In questo modo la trasmissione viene sparpagliata su più canali di frequenze e si effettua dei salti da uno a un altro, regolata da un ordine/sequenza di “hopping” e dagli intervalli di tempo, a ognuno dei quali si cambia canale. Ovviamente ricevente e trasmittente devono sempre essere sincronizzati sui cambi, per cui a inizio comunicazione si accordano sui numeri di sequenza dei canali e sugli intervalli di tempo. La probabilità di beccare la stessa frequenza (e ampiezza) casuale di un’altra coppia di dispositivi è molto bassa.



Il tempo di permanenza (*dwell time*) è l’intervallo di tempo di utilizzo di una certa frequenza, mentre il piccolo lasso di tempo di sintonizzazione dell’elettronica da un segnale a un altro è il cosiddetto *hop time*. Quest’ultimo crea un piccolo calo di efficienza, in quanto la trasmissione si deve interrompere momentaneamente per cambiare frequenza e risintonizzarsi.

Spettro diffuso (*Spread Spectrum*) a sequenza diretta:

Si rappresenta ogni bit del frame con più bit nel segnale trasmesso. Per ogni bit che il mittente vuole trasmettere, invia in realtà lo *XOR* di quel bit e di n bit casuali, chiamati **sequenza di chipping**. La sequenza di bit casuali è generata da un generatore di numeri pseudo-random noto sia al mittente che al destinatario; pertanto, per recuperare i bit di dati originali, il destinatario deve solo eseguire nuovamente lo *XOR* tra la sequenza ricevuta e la sequenza di chipping. I valori trasmessi, noti come **codice di chipping a n bit**, diffondono il segnale su una banda di frequenza n volte più ampia.



La sequenza di dati e quella casuale sono indipendenti. Da notare, inoltre, come si può vedere dall’immagine, che la sequenza dei dati in realtà di 4 bit, quindi serve anche un meccanismo che “espanda” i bit duplicandoli e di conseguenza il mittente dovrà poi ricavarne quelli effettivi dalla sequenza espansa.

Diverse tecnologie wireless

Le tecnologie wireless si differenziano per una serie di aspetti, tra cui la quantità di larghezza di banda che forniscono e quanto possono essere distanti i nodi di comunicazione. Quattro tecnologie wireless di spicco con caratteristiche e applicazioni diverse sono:

- Wi-Fi (più formalmente noto come 802.11)
- Bluetooth (802.15.1) / ZigBee (802.15.4)
- WiMAX (802.16)
- Wireless cellulare 3G/4G/5G

Ognuna di queste tecnologie è stata creata e viene utilizzata per scopi diversi, ad esempio il Bluetooth è ottimo per connettere dispositivi molto vicini tra loro (es.: telefono e auricolari) e trasmettere con ritardi massimi molto piccoli, quindi ha poca capacità di trasmissione, ma molto affidabile (ottima per il real-time).

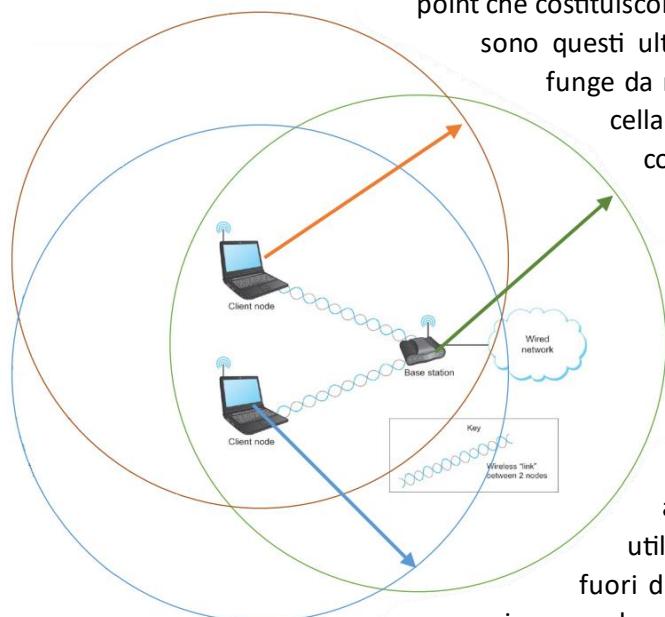
Mentre, per quanto riguarda il WiFi, è nato per sostituire la rete cablata Ethernet e serve di solito per le comunicazioni computer-computer, condivisioni di risorse, accessi a server locali, ecc. che hanno una quantità di dati da trasmettere maggiore del Bluetooth.

I data rate specificati nell'immagine non sono la capacità netta del canale, ma è la velocità dei simboli che viaggiano nel collegamento (nella realtà quei valori sono molto più bassi). Inoltre, c'è da tenere conto che gli ostacoli non sono uguali per tutti i tipi di frequenze, infatti ad esempio le onde più lunghe (es.: onde radio) attraversano più facilmente i muri, mentre quelle più vicine alla luce (es.: infrarossi) fanno molta più fatica e tendono a rimbalzare, oppure ad esempio il WiFi attraversa abbastanza bene i muri, ma non l'acqua, oppure il 5G supera con più difficoltà i muri rispetto al 4G.

Infatti, la portata del segnale è proporzionale alla potenza emessa e alla frequenza utilizzata, quindi a parità di potenza si arriva più lontano con frequenze più basse.

Molti collegamenti wireless attuali sono detti asimmetriche, ovvero quando la comunicazione avviene tra una stazione normale e una particolare (gli endpoint sono di tipi diversi). Ad esempio per il WiFi ci sono gli access point che costituiscono i nodi speciali, mentre nel Bluetooth, in cui non ci

sono questi ultimi, esiste un nodo della rete/comunicazione che funge da master (o base) e che regola il funzionamento della cella Bluetooth (e che quindi fa più lavoro degli altri nodi connessi). Di solito, uno degli endpoint non ha mobilità in quanto ha una connessione cablata a Internet (stazione base).



Altra caratteristica dei collegamenti wireless è che si trasmettono in maniera "sferica", ovvero le onde vanno in tutte le direzioni. Inoltre, i pacchetti inviati nella rete non sono visibili solamente a chi è nella comunicazione (come avveniva per l'Ethernet), ma anche ai dispositivi nelle vicinanze del raggio e che utilizzano le medesime frequenze. Quelli che sono al di fuori della sfera o in prossimità di essa, solitamente non ricevono alcun segnale dal dispositivo o magari ricevono solo parte

di esso, risultando corrotto. Dato che le comunicazioni wireless supportano la mobilità dei nodi, se un dispositivo si dovesse allontanare da un altro con cui sta trasmettendo, la comunicazione si interrompe.

Di solito le comunicazioni sono di tipo punto-punto o punto-multipunto (quando la base trasmette ai singoli client). La comunicazione tra nodi non base (*client*) può essere diretta (come nel WiFi), o instradata attraverso la stazione base (come nei telefoni cellulari e nel Bluetooth).

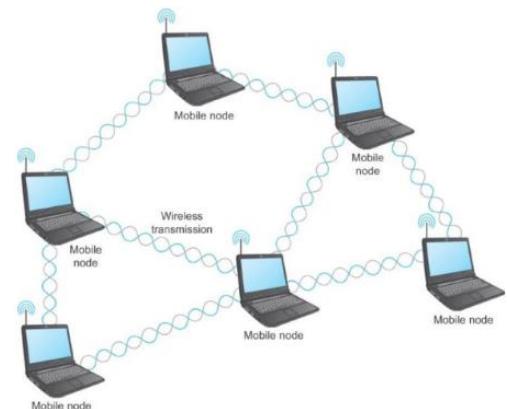
Inoltre, come appena detto le connessioni wireless supportano anche la mobilità dei nodi e vengono forniti tre livelli di mobilità per i client:

- Nessuna mobilità: il ricevitore deve trovarsi in una posizione fissa per ricevere una trasmissione direzionale dalla stazione base (versione iniziale di WiMAX)
- Mobilità all'interno del raggio d'azione di una base (Bluetooth)
- Mobilità tra basi (telefoni cellulari e WiFi)

Quando un client si sposta dal raggio di una stazione, si connette alla prima più vicina (es.: telefono). Tuttavia, in alcuni casi, ad esempio il WIFI, potrebbe non essere sempre possibile.

Alternative alle reti asimmetriche sono le reti a maglie (*mesh network*) o ad-hoc (*ad-hoc network*), in cui i nodi sono alla pari (*peer*) e i messaggi possono essere inoltrati attraverso una catena di nodi peer. Sono utili quando non c'è una stazione base fissa (es.: access point), come nell'IoT, o per raggiungere aree non coperte dalla stazione base, oppure per creare delle reti con specifiche caratteristiche in base alle necessità.

L'immagine in realtà può trarre in inganno, in quanto non vi è una comunicazione diretta tra i dispositivi, mentre nella foto accanto si vuole intendere che sono nel range di trasmissione.



WIFI – IEEE 802.11

Lo standard IEEE 802.11 è quello che si chiama comunemente WIFI e fa parte della famiglia di reti locali (standard 802), ovvero aree geografiche limitate, infatti si usa principalmente per le reti all'interno di edifici, ma non è pensato per fornire servizi di connessione come 3G, 4G o 5G, piuttosto offre servizi come gestione dell'alimentazione, meccanismi di sicurezza, ecc. La sfida principale consiste nel mediare l'accesso a un mezzo di comunicazione condiviso, in questo caso i segnali che si propagano nello spazio.

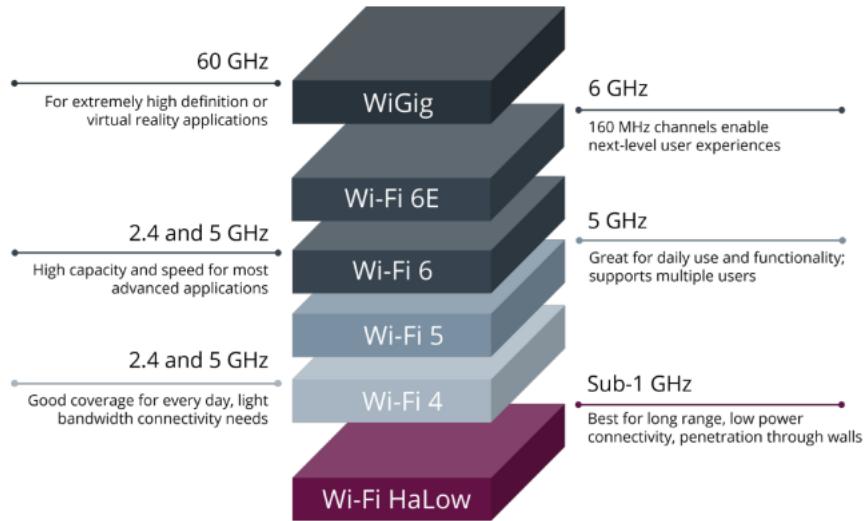
Lo standard 802.11 originale definiva due standard di livello fisico basati su radiofrequenze: uno utilizzava il salto di frequenza su 79 bande di frequenza di 1 MHz (utilizzato anche dal Bluetooth) e l'altro utilizzava la sequenza diretta con una sequenza di chipping a 11 bit (1 bit di dati diventa 11 bit). Entrambi gli standard funzionavano nella banda di 2,4 GHz, esente da licenza, e fornivano fino a 2 Mbps.

Successivamente sono stati aggiunti altri standard di livello fisico, di seguito l'ordine cronologico:

- 802.11b, che utilizza una variante della sequenza diretta e fornisce fino a 11 Mbps
- 802.11a, che fornisce fino a 54 Mbps utilizzando l'OFDM e funziona su una banda di 5-GHz esente da licenze, la cui velocità effettiva disponibile è di circa 20 Mbps
- 802.11g che è retrocompatibile con 802.11b e che utilizza la banda a 2.4 GHz (OFDM), offrendo una velocità massima di 54 Mbps
- 802.11n, che utilizza le bande da 2.4 e 5.4 GHz (OFDM), fornisce fino a 300 Mbps e consente il MIMO (*multiple in, multiple out*), ovvero più antenne in parallelo in entrata e in uscita
- 802.11ac, che utilizza la banda a 5 GHz e fornisce fino a 1300 Mbps e permette il MIMO
- ecc.

L'802.11b è lo standard più vecchio che si può ancora trovare in circolazione e funzionante, ad esempio le console Nintendo DS usavano questa versione per la loro comunicazione a distanza.

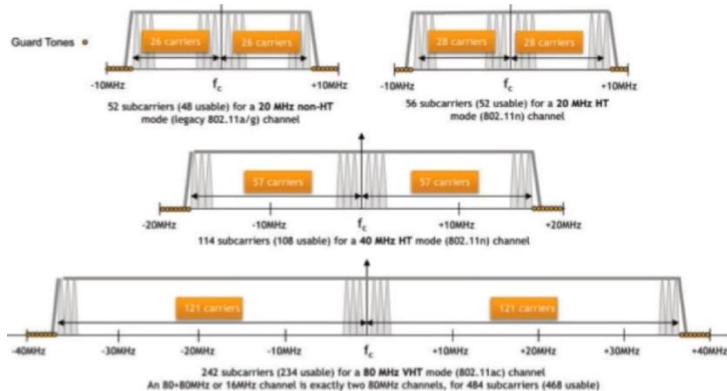
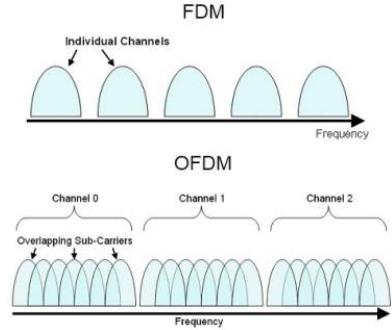
Si noti che la velocità di trasmissione effettiva può variare molto: dipende dalla qualità del segnale (cioè, SNR), dal tasso di codifica, dalla larghezza del canale, ecc. (es.: 802.11n può variare da 6,5 Mbps a 600 Mbps). Inoltre, anche queste velocità di trasmissione sono del supporto wireless condiviso, quindi da condividere tra tutte le stazioni, non per ogni stazione.



OFDM:

L'*Orthogonal Frequency-Division Multiplexing* (OFDM) è uno schema di multiplazione a divisione di frequenza (FDM), in cui un gran numero di segnali a sottoportanti (*sub-carriers*) ortogonali strettamente distanziate viene utilizzato per trasportare dati su diversi flussi o canali di dati paralleli.

Detto in altre parole, è una tecnica che arriva dalla DSL e che serve a codificare segnali digitali su un canale dividendoli in sottoportanti, ognuna



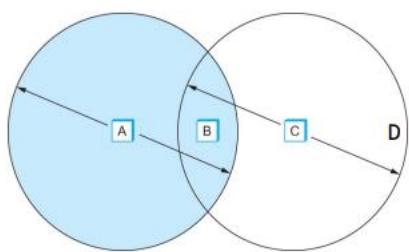
indipendente dalle altre, in modo da sfruttare a pieno la banda data. Nell'immagine sotto, ogni range di frequenze (es.: il primo blocco in alto a sinistra) è diviso in canali (sottoportanti) e f_c denota la frequenza standard centrale. In ognuno dei canali vengono trasmessi i bit, trasmissibili in parallelo, dunque sfruttando tutto il range. Questa codifica e utilizzo dei canali è eseguita a livello elettronico.

IEEE 802.11 – Collision avoidance

Il wireless sembra simile all'Ethernet, tuttavia si ha a che fare con un mezzo trasmissivo invisibile e che non fornisce una visibilità totale fra i nodi, il che crea dei grossi problemi. Si consideri l'esempio in figura, in cui ci sono quattro stazioni, ognuna in grado di inviare e ricevere segnali che raggiungono solo i nodi alla sua immediata sinistra e destra. Ad esempio,

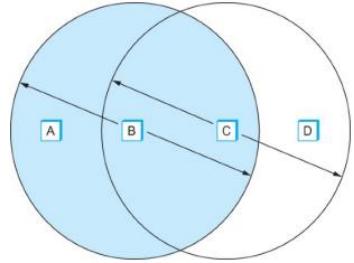
B può scambiare frame con A e C, ma non può raggiungere D, mentre C può raggiungere B e D ma non A. Si supponga che A e C vogliano comunicare contemporaneamente con B e quindi inviano un frame ciascuno. A e C non si conoscono, poiché i loro segnali non si trasmettono così lontano, dunque non sanno che entrambi

vogliono trasmettere alla stessa stazione B. Per cui entrambi iniziano a trasmettere. B inizia a ascoltare il segnale di uno dei due quando gli arriva il segnale dell'altra stazione. Questi due frame si scontrano l'uno con



l'altro (B non riesce più a codificarli perché si sovrappongono), ma, a differenza di Ethernet, né A né C sono consapevoli di questa collisione. Quindi, A e C sono detti nodi nascosti (*hidden nodes*) l'uno rispetto all'altro. In questo caso, il protocollo CSMA/CD non funziona, inoltre, le stazioni possono o ascoltare o trasmettere, ma non possono fare sia trasmissione che ricezione, per cui non è possibile rilevare una collisione durante la trasmissione.

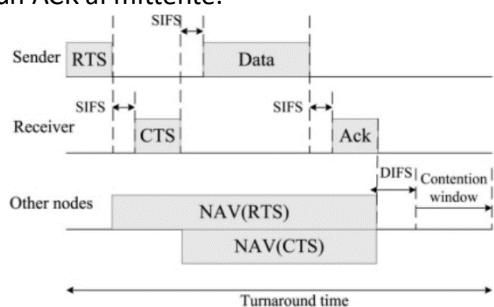
In più c'è un altro problema chiamato del nodo esposto (*exposed node*), ovvero il problema opposto al caso precedente: a un nodo pare di essere sotto una comunicazione che disturba quando in realtà non è vero. Ad esempio, si supponga che B stia inviando ad A. Il nodo C è a conoscenza di questa comunicazione perché sente la trasmissione di B. Sarebbe un errore per C concludere che non può trasmettere a nessuno solo perché può sentire la trasmissione di B. Si supponga quindi che C voglia trasmettere al nodo D. Questo non è un problema, perché la trasmissione di C a D non interferisce con la capacità di A di ricevere da B, poiché anche se una parte delle onde di C si sovrappone a quelle di B, non è un problema per A perché il segnale di C non arriva fin lì, per cui sente solo il segnale di B. Inoltre, B non viene disturbato dal segnale inviato da C.



Un primo passo per risolvere questi problemi è quello di gestire le comunicazioni tramite turni, in cui ogni dispositivo trasmette tempi precisi. Il Bluetooth funziona proprio in questa maniera, mentre il WIFI, che vuole essere un successore di Ethernet, risolve tramite CSMA/CA.

Il CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*) è un algoritmo simile a quello di Ethernet, ma adattato al fatto che non si ha una visione completa della rete. Come per il CSMA/CD, si hanno molti dispositivi connessi alla rete e vi è un meccanismo di ascolto del mezzo condiviso, però in questo caso invece di avere una *Collision Detection*, si ha una *Collision Avoidance* che cerca di fare in modo che non si verifichino clash durante le trasmissioni.

L'idea chiave è quella che, prima di inviare i dati veri e propri, il mittente e il ricevitore si scambiano reciprocamente dei frame di controllo (ausiliari) che servono a verificare se ricevente e trasmettitore sono in una posizione da poter comunicare. Questo scambio informa tutti i nodi vicini che sta per iniziare una trasmissione, in modo che rimangano in silenzio per il tempo necessario. Il mittente trasmette al ricevitore un frame di richiesta di invio (*Request To Send*, RTS). Il frame RTS include un campo, chiamato DURATA (*DURATION*), che indica per quanto tempo il mittente vuole mantenere il mezzo di comunicazione, ovvero il tempo dell'intera trasmissione, fino alla fine dell'ACK. Se il ricevitore ottiene l'RTS correttamente (ovvero senza disturbi di interferenze), allora risponde con un frame *Clear to Send* (CTS, "libero per l'invio"), che rimanda al mittente il campo di lunghezza DURATA e accetta la richiesta di comunicazione. A questo punto vengono inviati i dati veri e propri e che vengono letti e codificati dal ricevitore. Quando il ricevitore ha terminato la sua ricezione, invia un ACK al mittente.



Il SIFS (*Short Inter Frame Space*) è un intervallo di tempo di pausa che intercorre tra la fine di un frame l'inizio di quello successivo e serve per dare il tempo all'elettronica di cambiare modalità (ricezione/trasmissione) e

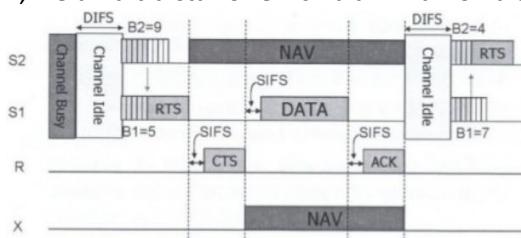
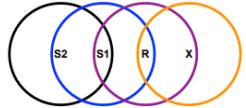
al ricevitore di poter elaborare la risposta fornita dal mittente. Il DIFS, invece, è il tempo che intercorre tra la fine di una comunicazione e l'inizio di una nuova e corrisponde a un SIFS + 2 slot time.

Per quanto riguarda gli altri nodi della rete, quando ricevono l'RTS, vedono il tempo di durata della comunicazione e sanno che fino al termine di essa non possono trasmettere. Questo tempo è chiamato *Network Allocation Vector* (NAV) e risolve il problema dei nodi nascosti. Inoltre, qualsiasi nodo che vede il frame RTS ma non il frame CTS non è abbastanza vicino al ricevitore per interferire con esso e quindi è libero di trasmettere, quindi risolve il problema del nodo esposto. I nodi al di fuori della comunicazione in corso sanno anche a che punto essa si trova in base a quali frame ricevono.

Come è possibile vedere, rispetto alla Ethernet che usava un solo tipo di messaggi, qui se ne tre hanno di nuovi e con scopi differenti. Il frame principale è sempre quello dei dati (con intestazione, payload e CRC), in più vi sono RTS, CTS e ACK che non portano dati veri e propri, ma servono a far funzionare il protocollo, come visto appena sopra.

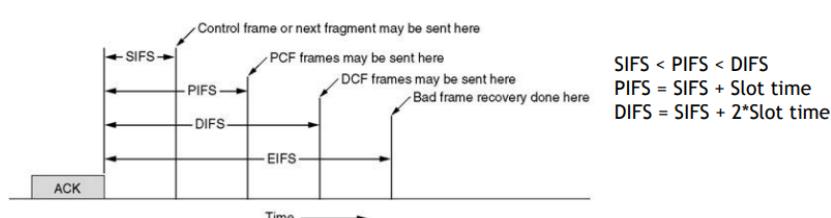
Tuttavia possono sorgere dei problemi: se due o più nodi rilevano un collegamento inattivo e cercano di trasmettere un frame RTS nello stesso momento, i loro frame RTS si scontreranno l'uno con l'altro (nodo nascosto). Ma 802.11 non supporta il rilevamento delle collisioni: il mittente non può percepire la "linea" durante la trasmissione, perché una radio funziona sia in trasmissione che in ricezione. Quindi i mittenti si accorgono della collisione quando non ricevono il frame CTS dopo un certo periodo di tempo. In questo caso, ognuno di loro aspetta un tempo casuale prima di riprovare (1-persistente). Il tempo di attesa di un dato nodo è definito dallo stesso algoritmo di backoff esponenziale utilizzato su Ethernet. Inoltre, vi è l'uso dell'ACK: il ricevitore invia un ACK al mittente dopo aver ricevuto con successo un frame, dunque tutti i nodi devono attendere la fine dell'ACK prima di provare a trasmettere.

Esempio di coordinamento distribuito con quattro stazioni: S1, S2 cercano di trasmettere, R è il ricevitore di S1, X è un'altra stazione vicina a R ma non a S1 e S2.



Cambiando variante dell'802.11 il protocollo resta lo stesso, come anche SIFS < PIFS < DIFS. Le uniche differenze stanno sostanzialmente sui numeri di durata degli intervalli, che sono definiti nei vari standard.

- Slot time: intervallo minimo, usato nel backoff esponenziale
- SIFS (Short Inter Frame Space): tempo utilizzato per elaborare e rispondere a un frame all'interno della stessa transazione
- PIFS (Point Control Function Inter Frame Space): tempo che l'access point deve attendere (e rilevare il mezzo wireless) prima di inviare un RTS.
- DIFS (Distributed Control Function Inter Frame Space): tempo che una stazione deve attendere (e rilevare il mezzo wireless) prima di inviare un RTS.



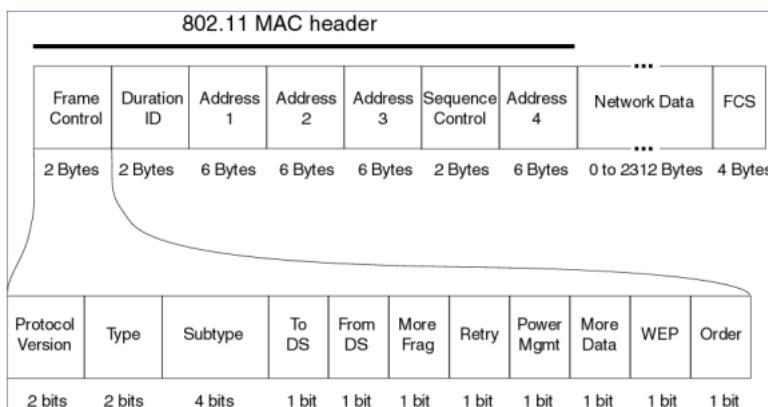
Standard	Slot time	SIFS	PIFS	DIFS
IEEE 802.11-1997 (FHSS)	50	28	78	128
IEEE 802.11-1997 (DSSS)	20	10	30	50
IEEE 802.11b	20	10	30	50
IEEE 802.11a	9	16	25	34
IEEE 802.11g	9 or 20	10	19 or 30	28 or 50
IEEE 802.11n (2.4 GHz)	9 or 20	10	19 or 30	28 or 50
IEEE 802.11n (5 GHz)	9	16	25	34
IEEE 802.11ac	9	16	25	34

All times are in μs

Da precisare il fatto che standard diversi potrebbero non essere compatibili tra di loro per via di slot time differenti (es.: l'802.11b che deve comunicare con un 802.11a non riesce, proprio al livello elettronico). Per cui per metterli in comunicazione si deve rallentare quello più veloce dei due. Lo stesso vale per gli altri tempi.

IEEE 802.11 – Formato dei frame

Come già detto, in questo standard esistono molti frame diversi: DATA, RTS, CTS, ACK o BEACON (usato dall'algoritmo di scansione per trovare l'SSID degli access point), mentre in Ethernet ce n'era solo uno. Essi possono essere identificati osservando i primi due byte (in particolare *bit Type* e *Subtype* del *Frame Control*).



Il frame dell'802.11 è molto simile a quello di Ethernet, ad esempio il campo FCS è il codice di controllo e è identico al CRC-32, tuttavia la grossa differenza sta nell'intestazione. Innanzitutto non ha solo due indirizzi di MAC (mittente e destinatario), ma ne ha quattro; nel campo *type* si specificano i quattro tipi di frame già menzionati; il *To DS* e il *From DS* verranno specificati in seguito; il *power management* serve per mettere in standby una stazione o per risvegiliarla; il *WEP* (*Wide Equivalent Privacy*) si utilizza per la cifratura, in cui se il bit è a 1 significa che i dati trasmessi sono cifrati.

La dimensione totale del *Frame Control* è di 16 bit e viene seguito dal campo *Durata/ID* che contiene la durata residua della transazione (in μs) per l'algoritmo CSMA/CA (il NAV è impostato su questo valore), oppure l'ID della rete (nei frame Beacon). Subito dopo di essi vi sono gli indirizzi di origine e di destinazione, ovvero i soliti indirizzi Ethernet a 48 bit e che seguono la stessa struttura e stessa politica di assegnazione (gli altri indirizzi sono utilizzati nel sistema di distribuzione). Il controllo di sequenza (*sequence control*), che è posto tra il terzo e il quarto indirizzo, è utilizzato per la frammentazione e la deframmentazione di frame di grandi dimensioni. Per quanto riguarda i dati, invece, si hanno da 0 fino a 2312 byte. I restanti byte sono del FCS.

A livello di dimensione totale dei frame dipende dalle informazioni che trasportano. Infatti come appena visto

	bytes	2	2	6	4	
ACK		Frame Control	Duration	Receiver Address	CRC	=14 byte

	bytes	2	2	6	6	4	
RTS		Frame Control	Duration	Receiver Address	Transmitter Address	CRC	=20 byte

	bytes	2	2	6	4	
CTS		Frame Control	Duration	Receiver Address	CRC	=14 byte

il pacchetto DATA presenta un'intestazione di 30 byte e un FCS di 4 byte, per un totale di 34 byte (il payload è variabile), mentre gli altri frame hanno dimensioni molto più ridotte: l'ACK è di 14 byte, il RTS di 20 byte e il CTS di 14 byte. Questi ultimi hanno lunghezza fissa. Come è possibile vedere nell'immagine, i frame ACK e CTS non necessitano di un indirizzo di mittente, ma basta solo quello del destinatario in modo da identificare la stazione a cui sono rivolti i dati, mentre gli altri campi sono identici per tutti e tre i frame, in quanto è importante avere l'intestazione e la durata della comunicazione, nonché fornire un codice di controllo (CRC) per il frame ricevuto.

Quindi, quando si vuole iniziare una comunicazione si ha un alto overhead, poiché per ogni frame DATA che si vuole trasmettere si necessita di (esclusi i dati effettivi) RTS, CTS, ACK e intestazione e FCS del frame DATA, ovvero di $30 + 4 + 14 + 20 + 14 = 82 \text{ byte}$, a cui vanno sommati gli intervalli SIFS e DIFS.

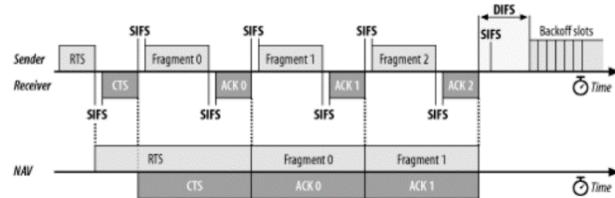
Come per il CSMA/CD, il CSMA/CA è difficile da modellare con precisione e soprattutto i dispositivi, come ad esempio i router, non hanno quasi mai l'efficienza dichiarata, o meglio: riescono a inviare i pacchetti sul mezzo fisico alla velocità dichiarata, ma quelli realmente utili, quindi al netto di tutti gli overhead, sono molti di meno. Ad esempio, si supponga di voler trasmettere un messaggio in 802.11g, a 54 Mbps. In questa situazione, il CSMA/CA aggiunge (almeno):

$$1 \text{ DIFS} + 3 \text{ SIFS} = 4 \text{ SIFS} + 2 \text{ Slot time} = 4 * 10 + 2 * 9 = 58 \mu\text{s}$$

che equivalgono a: $58\mu\text{s} * 54 \text{ Mbps} = 3132 \text{ bit} = 391,5 \text{ byte}$. L'intestazione del frame, il CRC e altri frame inviati hanno lunghezza totale di 82 byte. Dunque, per la trasmissione completa si ha un overhead di 473,5 byte (per ogni transazione).

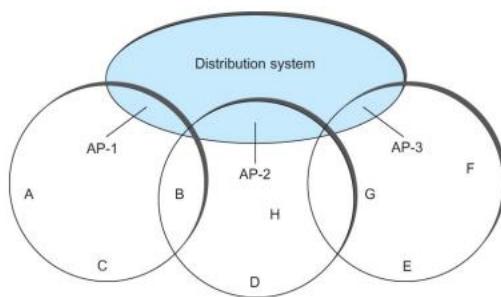
Se il carico utile fosse di 1500 byte, l'efficienza non è superiore a $1500/(1500 + 473,5) = 76\%$, per cui la larghezza di banda netta è $\leq 41\text{Mbps}$, non 54Mbps, mentre con Ethernet l'efficienza raggiungeva il 97%.

Talvolta i dati trasmessi sono spezzettati in più frame (quando il messaggio è >2312 byte), per cui si invia un ACK per ogni pacchetto inviato (senza interrompere ogni volta la comunicazione), facendo aumentare ulteriormente l'overhead. Anche se, in realtà questo metodo è poco usato, poiché è un po' pericolosa in quanto se si dovesse perdere anche solo uno dei pacchetti inviati, si perderebbe l'intera comunicazione e potrebbe essere stata dispendiosa. Questo tipo di molteplici segmenti DATA che vengono inviati prendono il nome di *frammenti*, e tutti, eccetto l'ultimo, hanno il bit *MoreFrag* nell'intestazione impostato a 1.



IEEE 802.11 – Distribuzione

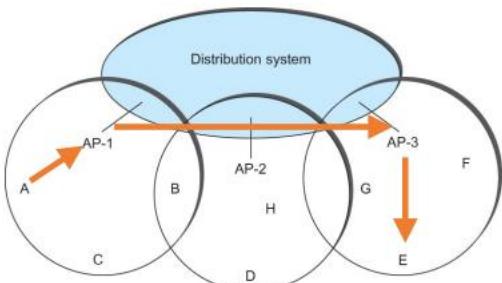
Per quanto riguarda gli access point, normalmente sono collegati da un certo meccanismo di distribuzione (*Distribution System*, DS), ovvero un'infrastruttura di rete cablata di fondo che collega gli AP (tipicamente una LAN Ethernet).



In questo tipo di situazione può succedere che un dispositivo sia collegato a una certa stazione, a sua volta collegata a una rete a posteriori (DS), e che a un certo punto il dispositivo spostandosi si disconnette dall'access point a cui era connesso e si aggancia a un altro. Il compito svolto dal *roaming* è quello di far apparire tutte queste stazioni e questo meccanismo come un unico access point (o rete) condiviso a più sottoreti. In questa maniera si può comunicare con access point differenti, ma che sono in realtà connessi tra di loro. A questo punto entrano in gioco i quattro campi *address* del frame 802.11, ovvero mittente, destinatario, access point del mittente e access point del destinatario, e la loro interpretazione dipende dalle impostazioni dei bit *To DS* e *From DS* nel campo *Control* del frame. È da tenere in considerazione che il mittente originale non è necessariamente lo stesso del nodo di trasmissione più recente (lo stesso vale per il destinatario).

Il caso più semplice è quando mittente e destinatario sono sotto lo stesso access point, in cui entrambi i bit DS sono 0, per cui *Addr1* identifica il nodo di destinazione (es. C) e *Addr2* identifica il nodo di origine (es. A). Nel caso in cui serva attraversare un access point, si inserisce il rispettivo access point in uno dei due campi *address* facoltativi.

Il caso più complesso, invece, è quando entrambi i bit DS sono impostati su 1 (mittente e destinatario di access point differenti), il che indica che il messaggio viene passato da un nodo wireless al sistema di distribuzione e poi dal sistema di distribuzione a un altro nodo wireless. Per cui gli indirizzi utilizzati sono rispettivamente: *Addr1* per la destinazione finale, *Addr2* per il mittente immediato (quello che ha inoltrato il frame dal sistema di distribuzione alla destinazione finale), *Addr3* la destinazione intermedia (quella che ha accettato il frame da un nodo wireless e lo ha inoltrato attraverso il sistema di distribuzione) e *Addr4* identifica la sorgente originale. Nell'esempio in figura: *Addr1=E*, *Addr2=AP-3*, *Addr3=AP-1*, *Addr4=A*.



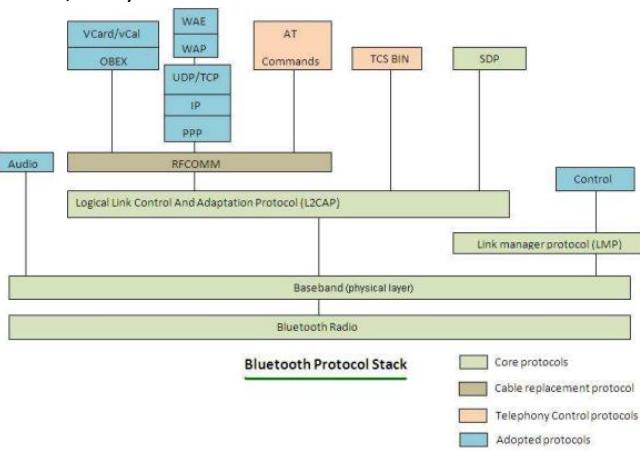
Bluetooth – 802.15

È un protocollo estremamente diffuso del wireless e viene studiato in quanto ha principi completamente diversi dal WIFI, per cui di fronte a qualsiasi tipo di problema si deve sapere che protocolli utilizzare. Rispetto ai due precedenti standard che sono nati per comunicazioni tra computer, questo si è sviluppato in primis per la comunicazione tra dispositivi mobile (es.: per sostituire il cavo degli auricolari).

Il Bluetooth è specificato da un consorzio industriale chiamato *Bluetooth Special Interest Group* (SIG), principalmente Ericsson, e prende il nome da Harald "Blötand" Gormsson, un re vichingo unificatore di Danimarca e Norvegia (958-986).

Questo protocollo viene spesso utilizzato per comunicazioni a brevissimo raggio tra telefoni cellulari, PDA, notebook e altri dispositivi personali o periferici. Opera nella banda esente da licenza a 2,45 GHz, con FHSS, e una portata di 10-50 m, con un consumo energetico "basso" (fino a 1W). I dispositivi di comunicazione appartengono in genere a un individuo o a un gruppo e talvolta sono classificati come *Personal Area Network* (PAN), infatti è nato per sostituire i cavi di connessione per i dispositivi nelle vicinanze (es.: dimensioni di una stanza). Le versioni 1.1 e 1.2 avevano una velocità grezza di 1 Mbps e una velocità netta fino a 723 kbps, mentre la versione 2.0 offre velocità grezza fino a 3 Mbps (velocità netta fino a 2,1 Mbps).

In realtà il Bluetooth non è un protocollo, ma una suite di protocolli, infatti esistono diverse varianti in base agli scopi di applicazione, andando oltre il livello di collegamento per definire protocolli applicativi, chiamati *profili*, per una serie di applicazioni (es.: un profilo per la comunicazione audio, un profilo per la condivisione di file, un profilo per la sincronizzazione di un PDA con un personal computer, un altro profilo consente a un computer mobile di accedere a una rete, tramite un modem simulato (RFCOMM), un altro per lo scambio di vCard, ecc.).



I protocolli *core* (in verde) sono quelli implementati in tutti i sistemi Bluetooth e riguardano gli aspetti fisici, radio e baseband del livello 1 dello stack e poi le logiche link del livello 2, in cui si inizia a intravvedere una sorta di interfaccia. Quello in marrone, l'RFCOMM, simula una seriale come se fosse un modem telefonico seriale, infatti c'è una suite di pacchetti (arancio) per la telefonia e simulano i vecchi comportamenti dei modemi a 56k come "alza la cornetta", "componi il numero", ecc. Di quelli in azzurro, c'è l'OBEX (Object Exchange) su

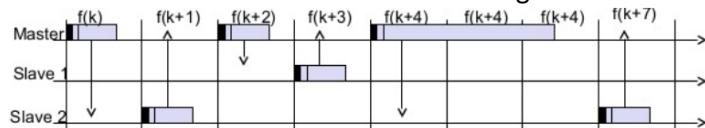
cui è implementato il servizio di scambio dei biglietti da visita (vCard), oppure i moduli TCP/UDP, che tramite seriale RFCOMM, poi protocollo PPP e IP, implementano uno scambio di messaggi ad esempio come il HTTP. Invece, il sistema audio non passa per tutti questi livelli, ma è connesso direttamente sul livello fisico.

Dal punto di vista del TCP/IP, il Bluetooth appare come una sorta di linea seriale su cui sono implementati tutti questi servizi e su cui sono trasmessi dei dati a flusso.

Il Bluetooth fornisce diversi tipi di connessioni:

- Sincrono orientato alla connessione (SCO): simmetrico, a commutazione di circuito, punto a punto. Utilizzato per i dati in tempo reale
- Collegamento asincrono senza connessione (ACL): a commutazione di pacchetto, punto-multipunto, polling master. Utilizzato per trasferimenti di dati

Anche in questo contesto, come nel WIFI, vi sono le celle/stazioni. La configurazione di base della rete Bluetooth è chiamata *piconet* e consiste in un dispositivo *master* (che viene "eletto" all'interno della piconet) e fino a sette dispositivi *slave*. Il dispositivo che fa da master è da NON confondere con l'access point, che svolge un ruolo differente, in quanto il Bluetooth non è nato per questo tipo di esigenza, ma per connettere un gruppo locale e ristretto di dispositivi. Ogni comunicazione avviene a turno tra il master e uno slave, ma gli slave non comunicano direttamente tra di loro, devono sempre passare per il master (il master deve fare il lavoro di tutti). Inoltre, fino ad altri 8 slave possono essere parcheggiati: impostati in uno stato inattivo a basso consumo. La comunicazione non avviene tramite protocollo CSMA/CA, ma in un ordine di tempo definito (*Time Division Multiplexing*), ovvero vengono decisi dei turni per ogni nodo e scanditi in sequenza. L'ordine viene imposto dal master. In questa maniera tutti i dispositivi sono sincronizzati e le collisioni non avvengono.

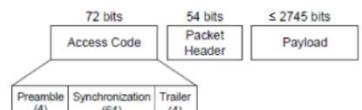


Il tempo viene suddiviso in slot di $625 \mu\text{s}$ e gli slave si sincronizzano sul clock fornito dal master. Negli slot pari (es.: 0, 2, 4, ecc.) parla sempre il master, mentre in quelli dispari può parlare uno degli slave nel suo slot di tempo assegnato, per cui se non lo sfrutta va perso (silenzio di comunicazione). Per questo motivo, questo sistema non è completamente efficiente, in quanto i turni non vengono modificati in base alle esigenze dei dispositivi. Nell'immagine sopra viene mostrato l'esempio in cui lo slave 1 vuole comunicare con lo slave 2.

I frame possono essere inviati anche uno di seguito all'altro senza interrompere la comunicazione al termine dello slot di tempo, l'importante è che siano di numero dispari e massimo 5.

Ogni nodo ha un indirizzo di dispositivo Bluetooth (BD_ADDR). Bluetooth utilizza FHSS su tutti i 79 canali da 1 MHz della banda ISM a 2.4 GHz, con un tempo di salto di $625 \mu\text{s}$, in modo che ogni slot sia su una frequenza diversa (e la radio ha bisogno di tempo per cambiare canale). Nell'immagine sopra, corrisponde ai trattini neri prima della comunicazione, ovvero i frequency hopping, pseudo-random, decisi dal master, che servono a dare il tempo all'elettronica di cambiare modalità. Il dwell time è di $625 \mu\text{s}$. Dunque, ogni comunicazione è situata su una frequenza differente (canale), con salti di hop e trasmissione sincroni.

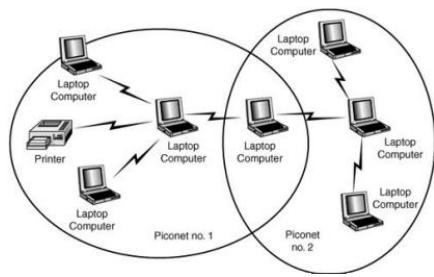
Per quanto riguarda i formati dei pacchetti, la prima parte è formata da un codice di accesso di sincronizzazione, derivato dal master quando la piconet è attiva. A seguire l'intestazione del pacchetto (per ACL), composta da 1/3-FEC, indirizzo MAC, tipo di collegamento, ARQ/SEQ, checksum (HEC). Il payload, invece, in questo caso arriva al massimo a 2745 bit. Dunque se si trasmette un solo pacchetto per slot si hanno 483 bit,



con tre slot 1124 bit e con cinque 2745 bit di dati. Anche se i pacchetti trasmessi sono piccoli, c'è da tenere presente che il protocollo non ha problemi di collisioni (ci sono i turni) e i tempi di consegna dei pacchetti sono deterministicici, per cui si sa che entro un certo periodo di tempo, tutti i dati vengono inviati.

Proprio quest'ultima parte fa del protocollo il suo punto di forza, perché riesce a garantire affidabilità anche a discapito di una trasmissione più lenta. Infatti questa soluzione è stata adottata anche nelle realtà industriali, come la variante ZigBee.

È possibile connettere tra di loro due o più piconet, creando delle *scatternet*, in cui ci sono i master (almeno uno) che fanno da intermediari da una sottorete a un'altra. Il nodo condiviso può fungere da ponte tra le due reti: riceve un frame da un master e lo invia a un altro (eventualmente per essere inoltrato allo slave giusto). I master di ciascuna piconet devono tenere traccia delle posizioni dei nodi, per inoltrare i frame al nodo intermediario corretto. Questa soluzione è molto utile per la costruzione di reti ad hoc (es.: nell'IoT).



BLE (Bluetooth Low Energy)

È stato integrato nel Bluetooth 4.0 (2009) e è destinato a utilizzare una potenza ridotta rispetto al Bluetooth classico: 0.01-0.5W, invece di 1W (con una portata inferiore). Il PHY ha la stessa banda del Bluetooth classico (2.4 GHz) in FHSS, ma con 40 canali da 2MHz ciascuno. La velocità grezza è di 2 Mbps (quella effettiva invece è fino a 1.37 Mbps). Ha un recupero molto più rapido dallo stato di non connessione, infatti è utilizzato in molte nuove applicazioni (es. sanità, sport, sensori, tracciamento dei contatti, ecc.).

Internet

In questo capitolo verrà approfondito il come estendere tutto ciò che è appena stato visto a un livello molto più grande, ovvero non si avrà più un solo mezzo condiviso con qualche dispositivo collegato, ma si avrà una rete molto più estesa con tante sottoreti di nodi.

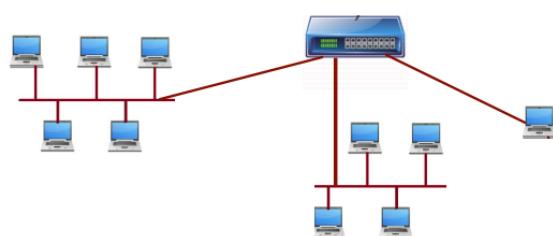
La strategia NON è quella di ingrandire/estendere le reti preesistenti collegando più mezzi fisici condivisi (es.: un cavo Ethernet) tra di loro fino a ottenere qualche kilometro di strada, poiché si genererebbero un sacco di problemi di vario tipo, non solo a livello di costi o di collisioni tra pacchetti e simili.

Per cui, bisogna risolvere i problemi geografici, i problemi delle diverse architetture dei dispositivi, dei diversi mezzi condivisi (es.: dispositivi con cavo e dispositivi wireless).

Switching e Forwarding

Switch

Il primo dispositivo che consente questa comunicazione multi rete è il commutatore (*switch*), ovvero un dispositivo che permette di interconnettere i collegamenti per formare (simulare) un'unica rete di grandi dimensioni. È multi-input e multi-output che trasferisce pacchetti da un ingresso a una o più uscite in modalità full-duplex. Di solito, questi collegamenti sono omogenei: dello stesso tipo, con lo stesso spazio di indirizzi, ma eventualmente con tecnologie diverse (es.: diverse varianti di Ethernet).



Dato che lo switch è collegato a un insieme di link che possono essere di differenti tipi (es.: cavo coassiale, doppino, ecc.), per ciascuno di essi esegue il protocollo di collegamento dati appropriato per comunicare. Il compito principale di uno switch è la commutazione e l'inoltro (*switch and forward*): ricevere i pacchetti in arrivo su uno dei suoi collegamenti e trasmetterli su un altro collegamento. Secondo l'architettura OSI, questa è la funzione principale del livello di rete (3), ma può essere implementata anche al livello 2 (datalink).

La strategia più comune è lo *store-and-forward*, ovvero inoltrare i pacchetti mantenendo il formato dei frame sempre uguale:

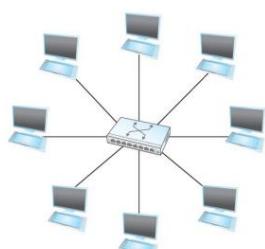
1. Ricevere un intero frame su un certo collegamento
2. Lo si controlla (CRC, ecc.) e se non è valido, lo si lascia cadere (senza preavviso)
3. Utilizzando una tabella interna, sceglie a quale link inoltrare il frame
4. Reinvia il frame al link scelto

(Gli switch più moderni utilizzano una strategia più veloce, chiamata "*cut through*")

Un'abilità degli switch è che "imparano" a riconoscere i dispositivi situati nelle sottoreti (tramite indirizzo che viene fornito nei pacchetti che ascolta). Infatti, lo switch è come un device che appartiene in contemporanea a tutte le sue sottoreti: ascolta i messaggi che arrivano, guarda gli indirizzi e controlla tra quelli mappati dove è collocato il destinatario e se questo fa parte di un'altra sottorete rispetto al mittente, inoltre il pacchetto in essa, altrimenti lo ignora in quanto deriva da un dispositivo già situato in quella rete.

C'è da tenere presente che lo *store-and-forward* ha un costo computazionale, in quanto lo switch deve ascoltare tutto il messaggio, leggerlo e interpretarlo per intero e ritrasmetterlo in un'altra rete. Per questo motivo, alcuni switch moderni per ottimizzare usano il *cut through*, in cui si fermano con la lettura alla sola intestazione per capire chi è il destinatario e poi inoltrano, senza dover leggere tutto il contenuto del frame.

Topologia a stella:



Questo concetto può essere esteso, dando un dominio di collisione separato per ogni singolo calcolatore, dunque si ottiene uno switch con tutti rami con almeno un calcolatore. Questa struttura viene chiamata *topologia a stella* (*star topology*) o *hub-and-spoke*, in cui vi è un hub centrale e tutti altri nodi a esso collegato.

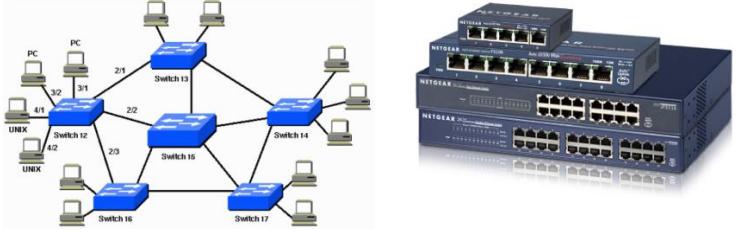
Questo modello è molto utilizzato, soprattutto nelle reti casalinghe, in quanto è molto semplice da realizzare e è abbastanza efficiente in quanto ogni dispositivo ha il suo specifico dominio di collisione per cui non vi sono collisioni durante gli invii dei pacchetti.

Tuttavia, si può verificare un collo di bottiglia, ovvero lo switch poiché è il nodo attraverso il quale dialogano tutti i dispositivi e ha capacità limitate (potenza di calcolo, buffer, ecc.), quindi in caso di troppo traffico potrebbe non essere in grado di gestirlo. I frame che non riesce a gestire solitamente vengono scartati. Altro grande guaio è che se si guasta lo switch, gli altri dispositivi smettono di comunicare (anche se questo è un fenomeno che si verifica di rado, in quanto gli switch sono dispositivi molto affidabili e duraturi).

Inoltre, questo meccanismo consente di adattare le velocità dei dispositivi ad esso collegati, in quanto i device non sono consapevoli della velocità con cui trasmette l'altro endpoint. Infatti, questo meccanismo è comodo per mettere in comunicazione tecnologie differenti, evitando diversi problemi, e è quel dispositivo che si chiama comunemente access point, che in realtà sarebbe un bridge. Questo meccanismo è, inoltre, facilitato dal fatto che gli indirizzi (MAC address) Ethernet e quelli WIFI sono entrambi della stessa famiglia a 6 byte, quindi è facile convertirli da uno all'altro.

Invece, per quanto riguarda i pacchetti, su Ethernet il massimo è 1500 byte, mentre su WIFI è 2346 byte, per cui i dispositivi wireless per evitare problemi provano a inviare messaggi più piccoli.

Altro vantaggio degli switch è che possono comodamente essere collegati tra di loro per formare reti più complesse, in quanto il numero di porte di un singolo switch è limitato. Il risultato è che la rete totale (virtuale) viene vista sempre come una unica, indipendentemente dalla sua struttura fisica. Tuttavia i ritardi e le latenze non sono uniformi.

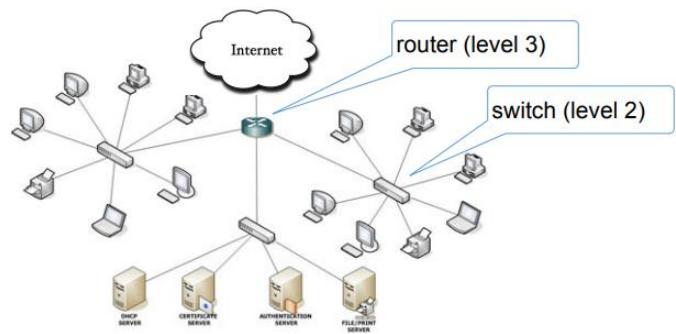


Inoltre, gli switch hanno molto spesso delle porte speciali che consentono di vedere tutte le porte di altri switch a esse collegate come una unica, ad esempio quando si ha una pila di switch e tutti sono collegati tra di loro su queste porte speciali, tutte queste vengono viste come una sola identica. Il risultato è che invece di avere, ad esempio, quattro porte da 100 Mbit l'una, si ha una sola porta che va a 400Mbit. Questa tecnica prende il nome di *trunking* e si usa per fare le dorsali su cui di solito si concentra molto traffico della rete.



Da non confondere con gli switch impilabili, i cui backplane possono essere collegati tra loro tramite collegamenti dedicati e proprietari ad alta velocità. Gli switch impilati agiscono come un'unica unità, non come unità distinte.

Da non confondere queste tecniche con i router, in quanto gli switch sono sostanzialmente invisibili agli host, essi non se ne accorgono nemmeno, tranne per il fatto del leggero delay che viene introdotto per inoltrare i pacchetti. Invece i router sono visibili agli host, si accorgono del nodo intermediario nella comunicazione in quanto le reti sono proprio separate tra loro (è il router che si occupa dell'instradamento dei pacchetti).



È possibile costruire reti di ampia portata geografica. Le MAN/WAN possono essere costruite utilizzando la fibra ottica (esistono switch appositi per collegamenti in fibra ottica) o gli switch ATM, che sono un altro tipo di tecnologia non Ethernet creata appositamente per reti geografiche. Gli ISP utilizzano switch xDSL (DSLAM) per collegare gli utenti finali attraverso le linee telefoniche.

Tutte queste soluzioni appena viste hanno il vantaggio di essere scalabili, l'aggiunta di un nuovo host alla rete collegandolo a uno switch non significa necessariamente che gli host già collegati otterranno prestazioni peggiori dalla rete. Ogni host di una rete commutata ha il proprio collegamento allo switch, quindi è possibile che molti host trasmettano alla massima velocità del collegamento.

Il limite è dato dalla larghezza di banda massima interna e dalla velocità di commutazione (*switching speed* o velocità di inoltro, *forwarding rate*). Negli switch Ethernet, la larghezza di banda interna varia da 10Gbps a ~1M pacchetti al secondo (*pps*) nei dispositivi economici, fino a 10-20 Tbps a decine di Gbps per i dispositivi di fascia alta.

Questo non è vero per le reti a media condivisi. Due host sulla stessa Ethernet non possono trasmettere continuamente alla massima velocità perché ci sono collisioni sul mezzo di trasmissione condiviso. Le collisioni possono essere evitate, ad esempio con tecniche TDM, ma in tal caso ogni host riceve una frazione della larghezza di banda totale del mezzo condiviso.

Approccio switching and forwarding

Si usa principalmente per l'approccio degli switch, ma è in realtà utilizzato anche in altri contesti, ad esempio con i router, che fruttano un meccanismo molto simile, oppure per i servizi di posta elettronica, in cui i messaggi che vengono inviati non sono i soliti pacchetti, ma sono le mail, e concettualmente il processo è il medesimo.

La prima problematica che deve essere affrontata è quella di decidere su che porta mandare fuori un frame che arriva, in quanto uno switch può avere anche 24 porte. L'indirizzo viene letto nell'intestazione del messaggio, tuttavia ci sono tre metodi per capire dove inoltrare il pacchetto:

- *Datagramma* o approccio senza connessione
- *Circuito virtuale* o approccio orientato alla connessione
- *Intradamento alla fonte* (source routing, meno comune)

È importante che tutti gli switch all'interno di una rete devono utilizzare lo stesso algoritmo tra i tre appena menzionati, in quanto sono metodi completamente diversi.

Si assume, inoltre, che ogni host all'interno della rete abbia un indirizzo univoco (al livello 2 si usa il MAC address, mentre al livello 3 quello IP). Esiste un modo per identificare le porte di ingresso e di uscita di ogni switch: tramite numeri (uno per ogni porta) o tramite nomi (dei nodi a cui la porta è collegata). Per semplicità, si usano i numeri.

Approccio senza connessione (datagramma):

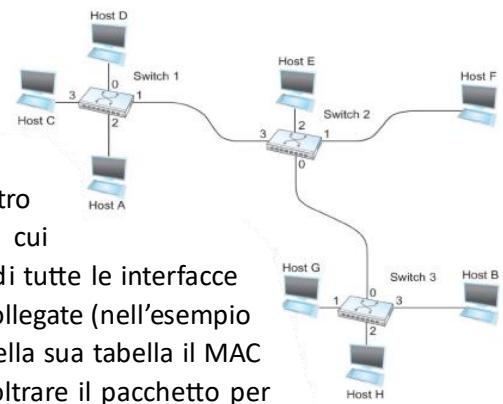
L'idea chiave è che ogni pacchetto viene trattato autonomamente, senza tenere traccia del suo passato, né del suo futuro. Si sa solo che ogni pacchetto contiene informazioni sufficienti per consentire a qualsiasi switch di decidere come farlo arrivare a destinazione e contiene l'indirizzo completo della destinazione. Ogni switch inoltra ogni pacchetto semplicemente osservando la sua intestazione.

Dunque quando un pacchetto arriva in uno switch, questo viene letto e tramite l'indirizzo segnato al suo interno, viene poi inoltrato. Questa semplice procedura viene eseguita su qualsiasi frame in input. Lo switch non segna nessuno stato della comunicazione che è avvenuta.

Esempio:

Per decidere come inoltrare un pacchetto, uno switch consulta una propria *tabella di inoltro* (*forwarding table*, talvolta chiamata erroneamente tabella di routing). Le tabelle di inoltro associano ad ogni destinazione la corrispondente porta su cui

Dest.	Port	inoltrare i pacchetti. Con indirizzi sono intesi quelli di tutte le interfacce della rete, anche quelle che non sono virtualmente collegate (nell'esempio sono gli host da A ad H). Dunque, lo switch segna nella sua tabella il MAC address dell'host e la porta (dello switch) su cui inoltrare il pacchetto per farlo arrivare a destinazione. Affianco, l'immagine della tabella di inoltro per lo switch 2 della rete rappresentata.
A	3	
B	0	
C	3	
D	3	
E	2	
F	1	
G	0	
H	0	



Quando a uno switch arriva un messaggio, controlla la sua tabella e lo inoltra sulla porta corrispondente, poi anche lo switch successivo farà la medesima azione, fino a quando il frame non arriva a destinazione.

Inoltre, è possibile che vi siano più percorsi alternativi per inviare un pacchetto sulla rete.

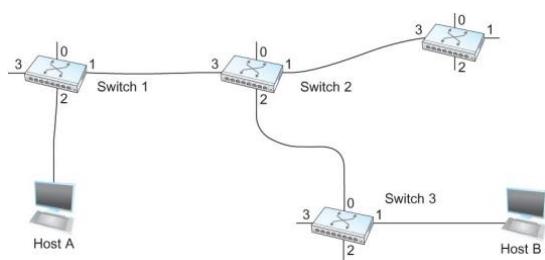
Tra le caratteristiche vantaggiose di questo approccio è che un host può inviare un pacchetto ovunque e in qualsiasi momento, poiché qualsiasi pacchetto che arriva allo switch può essere immediatamente inoltrato (assumendo una tabella di inoltro sia correttamente popolata). Questi semplici passaggi di controllo di indirizzo sulla tabella e il successivo invio non richiedono connessioni prima dell'inoltro e sono autonomi.

Tuttavia quando un host invia un pacchetto, non ha modo di sapere se la rete è in grado di consegnarlo o se l'host di destinazione è attivo e funzionante. Ogni pacchetto viene inoltrato indipendentemente dai pacchetti precedenti che potrebbero essere stati inviati alla stessa destinazione. Pertanto, due pacchetti successivi, ad esempio dall'host A all'host B, possono seguire percorsi completamente diversi. Un guasto a uno switch o a un link potrebbe non avere effetti gravi sulla comunicazione se è possibile trovare un percorso alternativo per aggirare il guasto e aggiornare di conseguenza la tabella di inoltro.

Lo svantaggio di questo approccio, invece, è che bisogna mantenere popolate e aggiornate le tabelle di forwarding e potrebbe non essere così semplice, specialmente in caso di guasti. Gli switch devono automaticamente mandarsi dei pacchetti in modo da sapere a quali host sono connessi, in modo da saper instradare i frame in arrivo. Nel caso di guasto si deve procedere a un aggiornamento immediato delle tabelle.

Approccio con connessione (circuito virtuale):

Sfrutta il concetto di commutazione di circuito virtuale, ovvero una tecnica ampiamente utilizzata, soprattutto in ambito telefonico, per la commutazione di pacchetti, tramite circuito virtuale (VC). Questo modello viene anche chiamato modello orientato alla connessione.



Per prima cosa si crea una connessione virtuale dall'host di origine all'host di destinazione, ovvero si crea uno stato condiviso e mantenuto, programmando gli switch intermedi. A quel punto si può procedere con l'invio dei dati.

La connessione e comunicazione avviene in tre fasi:

1. Impostazione della connessione
2. Trasferimento dei dati
3. Chiusura della connessione (*shutdown*)

Il setup precede la vera e propria comunicazione, ma comunque si tratta di messaggi che vengono scambiati tra gli switch in maniera da impostare/creare le connessioni. Infatti, per stabilire lo "stato di connessione" (che è variabile rispetto alle tabelle di forwarding statiche) in ciascuno degli switch tra gli host di origine e di destinazione, si crea una voce nella "tabella VC" di ogni switch attraverso il quale passa la connessione, talvolta configurata in modo permanente dall'amministratore di rete (*VC permanente*), mentre molto spesso viene stabilita su richiesta tramite segnalazione (*signalling*, SVC):

- Un host può inviare messaggi nella rete per far sì che lo stato venga stabilito.
- Un host può configurare e cancellare tale VC dinamicamente senza il coinvolgimento di un amministratore di rete.

Switch 1	Incoming Interface	Incoming VC	Outgoing Interface	Outgoing VC
	2	5	1	11

Switch 2	Incoming Interface	Incoming VC	Outgoing Interface	Outgoing VC
	3	11	2	7

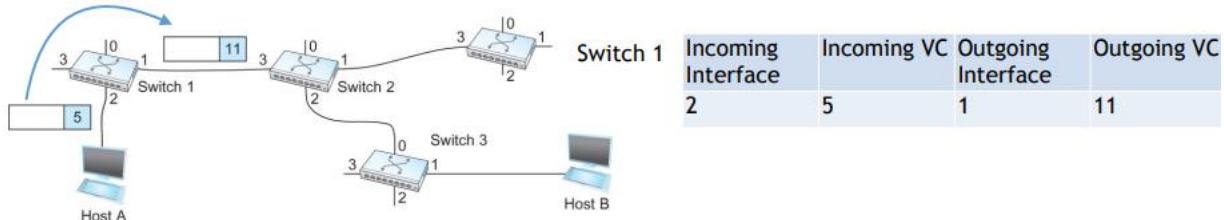
Switch 3	Incoming Interface	Incoming VC	Outgoing Interface	Outgoing VC
	0	7	1	4

Dunque, si procede creando una voce della tabella VC su un singolo switch che contiene:

- un identificatore di circuito virtuale (VCI) che identifica in modo univoco la connessione su questo switch e che sarà riportato nell'intestazione dei pacchetti che appartengono a questa connessione (continua)
- un'interfaccia in entrata in cui i pacchetti per questo VC arrivano allo switch
- un'interfaccia in uscita in cui i pacchetti per questa VC lasciano lo switch
- una VCI (potenzialmente diversa) che verrà utilizzata per i pacchetti in uscita

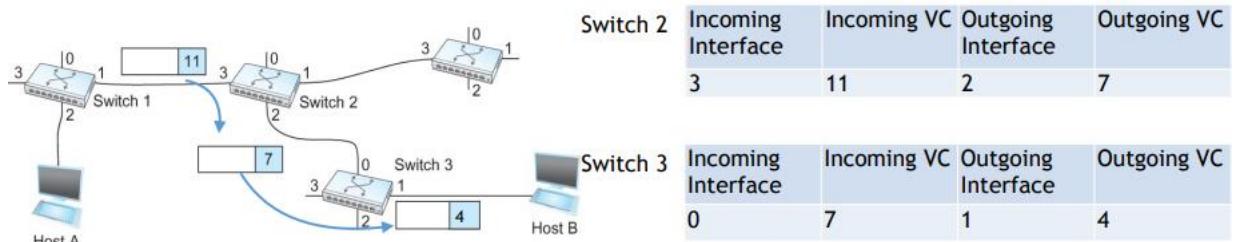
Se un pacchetto arriva sull'interfaccia in entrata designata e contiene il valore VCI designato nell'intestazione, il pacchetto deve essere inviato dall'interfaccia in uscita specificata con il valore VCI in uscita specificato inserito nell'intestazione.

Il circuito virtuale viene identificato da un numero, di solito scelto random. Quindi, ad esempio, uno switch vede arrivare un pacchetto destinato a un VC (es.: *incoming VC 5* nell'immagine) su un'interfaccia (es.: *incoming interface 2*), lui lo inoltra su una certa interfaccia (es.: *outgoing interface 1*) marcandolo con un altro numero (es.: *outgoing VC 11*).



Quindi, per qualsiasi pacchetto che A vuole inviare a B, A inserisce il valore VCI 5 nell'intestazione del pacchetto e lo invia allo switch 1. Lo switch 1 riceve un pacchetto di questo tipo sull'interfaccia 2 e utilizza la combinazione dell'interfaccia e del VCI nell'intestazione del pacchetto per trovare la voce appropriata della tabella VC. La voce della tabella sullo switch 1 indica allo switch di inoltrare il pacchetto dall'interfaccia 1 e di inserire il valore VCI 11 nell'intestazione.

A questo punto lo switch 2 vede arrivare un nuovo messaggio con incoming VC 11 dall'interfaccia 3 e sa che deve inviarlo alla porta 2 con numero di outgoing VC 7.



Questo processo continua finché il pacchetto non arriva all'host B con il valore VCI 4 (dallo switch 3). Per l'host B, questo valore identifica il pacchetto come proveniente dall'host A.

Questi valori random di VC vengono generati dinamicamente dagli switch, prima dell'inoltro dei dati.

(REGISTRAZIONE S03E01 PARTE 2 INTERROTTA)

La combinazione del VCI dei pacchetti ricevuti dallo switch e dell'interfaccia su cui sono stati ricevuti identifica in modo univoco la connessione virtuale. Nello switch possono essere stabilite molte connessioni virtuali contemporaneamente. I valori VCI in entrata e in uscita (*incoming* e *outgoing*) non sono generalmente gli stessi. Il VCI non è un identificatore significativo a livello globale per la connessione, ma ha un significato solo su un determinato collegamento. Infatti, ogni volta che viene creata una nuova connessione, è necessario assegnare un nuovo VCI per quella connessione su ogni link che la connessione attraverserà. Bisogna anche assicurarsi che la VCI scelta su un determinato collegamento non sia attualmente in uso su quel collegamento da una connessione esistente.

Caratteristiche della VC:

Poiché l'host A deve attendere che la richiesta di connessione raggiunga il lato opposto della rete e ritorni indietro prima di poter inviare il primo pacchetto di dati, c'è almeno un RTT di ritardo prima dell'invio dei dati. Mentre la richiesta di connessione contiene l'indirizzo completo dell'host B (che potrebbe essere piuttosto grande, essendo un identificatore globale sulla rete), ogni pacchetto di dati contiene solo un piccolo identificatore, che è unico su un solo collegamento. In questo modo, l'overhead per pacchetto causato dall'intestazione è ridotto rispetto al modello del datagramma. Ad esempio, in ATM, l'intera intestazione è di 5 byte (su un frame di 53 byte).

Se uno switch o un link di una connessione si guasta, la connessione è interrotta e occorre stabilirne una nuova. Anche quella vecchia deve essere eliminata per liberare spazio nella tabella degli switch. La questione di come uno switch decida su quale collegamento inoltrare la richiesta di connessione ha delle analogie con la funzione di un algoritmo di routing.

Buone proprietà del VC:

Quando l'host riceve il via libera per l'invio dei dati, conosce molte cose sulla rete, ad esempio, che esiste davvero un percorso verso il destinatario e che quest'ultimo è disposto a ricevere i dati. È anche possibile allocare risorse al circuito virtuale nel momento in cui viene stabilito. Ad esempio, in X.25, i buffer vengono assegnati a ciascun circuito virtuale quando il circuito viene inizializzato e il circuito viene rifiutato da un dato nodo se non sono disponibili buffer sufficienti.

Confronto con il modello datagramma:

La rete datagramma non ha una fase di creazione della connessione e ogni switch elabora ogni pacchetto in modo indipendente. Ogni pacchetto in arrivo compete con tutti gli altri pacchetti per lo spazio del buffer. Se non ci sono buffer, il pacchetto in arrivo deve essere scartato.

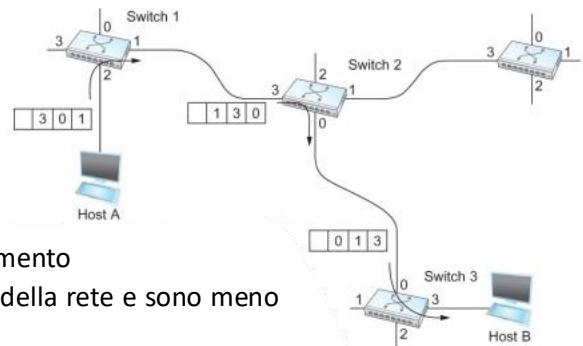
In VC, si potrebbe immaginare di fornire a ogni circuito una diversa qualità del servizio (*QoS*). La rete fornisce all'utente una sorta di garanzia sulle prestazioni. Gli switch accantonano le risorse necessarie per soddisfare questa garanzia, ad esempio, una percentuale della larghezza di banda di ciascun collegamento in uscita, oppure la tolleranza di ritardo su ogni switch.

Gli esempi più diffusi di tecnologie VC sono Frame Relay e ATM. La ATM (*Asynchronous Transfer Mode* = modalità di trasferimento asincrono) presenta una rete a commutazione di pacchetto orientata alla connessione, in cui i pacchetti sono chiamati *celle* e sono di dimensione fissa: 5 byte di intestazione + 48 byte di carico utile. I pacchetti a lunghezza fissa sono più facili da commutare in hardware, inoltre è più semplice da progettare e consentono il parallelismo.

Approccio source routing:

Tutte le informazioni sulla topologia della rete necessarie per passare un pacchetto attraverso la rete sono fornite dall'host di origine (es.: le porte su cui il pacchetto deve essere inoltrato).

Il vantaggio è che si hanno switch estremamente semplici e non è necessaria alcuna connessione. Tuttavia, l'instradamento è a carico della sorgente, che deve conoscere la topologia della rete e sono meno resistenti ai guasti.



Bridge e Switch LAN

In questo caso si parlerà dei bridge specifici presenti dentro agli switch e che servono per le LAN, ovvero gli switch più comuni nel mercato.

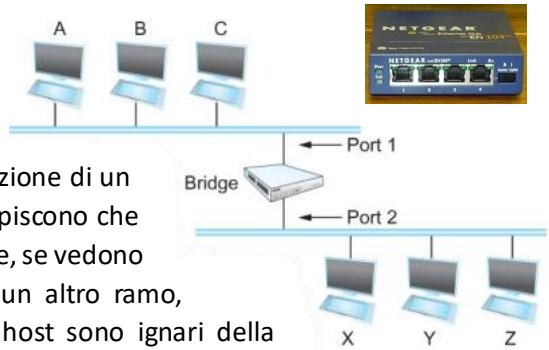
La soluzione più semplice, se si vuole estendere un ramo Ethernet è quello di mettere un *repeater*, ovvero un dispositivo che connette due rami e che prende i bit da una parte e li porta dall'altra. Esso lavora a un livello più basso del layer 2 degli switch, in quanto codifica e decodifica i bit, senza avere condizione di cosa sia un frame o di come correggerlo (non si accorge nemmeno degli errori). Questa soluzione iniziale, tuttavia è stata abbandonata, poiché non si può avere più di quattro ripetitori tra qualsiasi coppia di host e non è consentita dal protocollo una lunghezza totale di 2500 m.

Un'alternativa potrebbe essere quella di interporre un nodo tra le due reti Ethernet e far sì che il nodo inoltri i frame da una Ethernet all'altra. Questo nodo è chiamato *bridge* (= ponte). Un insieme di LAN collegate da uno o più bridge viene solitamente definito una LAN estesa.

La più semplice implementazione adottata su questi device era una strategia di store-and-forwarding con *flooding* (= innondazione), ovvero quando un dispositivo riceve un frame da una sua interfaccia, lo replica a tutte le altre interfacce a esso collegate. Veniva controllata anche la correzione o meno del frame. Questa tecnica appena descritta veniva utilizzata dai bridge *hub* e è una tecnologia ormai obsoleta.

Successivamente ci si è accorti che non è necessario inoltrare a tutti gli host i frame ricevuti da un bridge, ma basta inoltrarli solo ai diretti interessati: per definizione di Ethernet, se c'è un dispositivo collegato e che ha un certo indirizzo, non può trovarsi su altre porte (gli indirizzi sono univoci).

Dunque, i bridge continuano a osservare il flusso di pacchetti che circolano sulla rete e internamente memorizzano dove sono le posizioni dei dispositivi. In questo modo impara su quali porte sono affacciati i diversi device. Quando vedono passare su un certo link un frame che ha l'indirizzo di destinazione di un host presente in quello stesso link, non fanno nulla poiché capiscono che gli host stanno dialogando all'interno dello stesso ramo. Mentre, se vedono passare un frame che ha l'indirizzo di un host che sta su un altro ramo, forwardano il pacchetto riconosciuto nel link apposito. Gli host sono ignari della presenza di questo bridge.

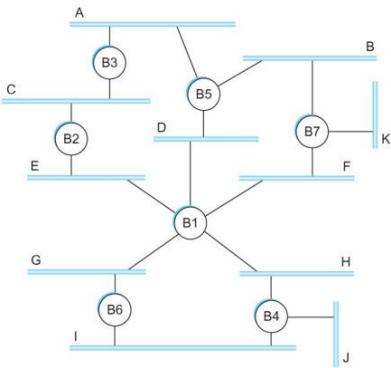


Per sapere dove sono attaccati gli host, esistono due possibilità: programmare il bridge manualmente (admin di rete), ma è una soluzione rognosa e pericolosa (basta poco per sbagliare il settaggio), oppure si fa in modo che gli switch apprendano autonomamente le posizioni (*switch ad autoapprendimento*). La seconda opzione è quella più utilizzata e consiste nel popolare costantemente e automaticamente la tabella con gli indirizzi osservando il traffico sulle interfacce. Il bridge ascolta il traffico di rete e controlla i mittenti delle comunicazioni e a mano a mano si segna chi è presente. Se i destinatari rispondono ai messaggi dei mittenti, allora vengono segnati anch'essi nelle tabelle, in quanto il bridge riconosce un nuovo mittente.

Inoltre, a ogni voce (indirizzo del mittente che viene inserito nella tabella) è associato un timeout. Il bridge scarta la voce dopo un determinato periodo di tempo (tipicamente di qualche minuto), per proteggersi dalla situazione in cui un host viene spostato da una rete all'altra. Invece, se il bridge riceve un frame indirizzato a un host non presente nella tabella, lo inoltra su tutte le altre porte in flooding.

Come già detto in passato, esistono degli indirizzi particolari riservati a delle funzionalità specifiche, ovvero l'indirizzo di *broadcast* e quello di *multicast*. Nel primo caso, il bridge effettua un flooding del messaggio a tutte le sue porte (tranne in quella da cui proviene il frame stesso). Nel secondo caso, invece, si hanno due varianti, che dipendono da come realizzato lo switch:

- uno switch economico implementerebbe un sistema di flooding indipendentemente dal tipo di indirizzo multicast che riceve e lascia che sia l'host a decidere se è interessato o meno. Tuttavia può generare molto traffico indesiderato sulle interfacce;
- gli switch più sofisticati, invece, imparano le posizioni degli host che appartengono al gruppo (livello 3). Questo risultato si ottiene facendo in modo che ogni membro di un certo gruppo G invii un frame all'indirizzo multicast del bridge con G nel campo sorgente.



Questa strategia funziona bene fin tanto che la LAN estesa non abbia un loop al suo interno, in cui i frame possono attraversare la LAN estesa per sempre. Nell'immagine affianco, i bridge B1, B4 e B6 formano un loop, per cui se ad esempio un host in B, volesse trasmettere su K, vi sono diversi percorsi fattibili.

In realtà, per come è stata realizzata l'immagine ogni nodo potrebbe essere (quasi) qualsiasi dispositivo, ad esempio A, B, C, ecc. potrebbero essere dei dominii di collisione, dei cavi coassiali, degli hub, o dei singoli cavi punto-punto, ecc. Infatti, in un contesto reale, una rete può essere formata da device di diversa natura.

Per cui quando si ha un ciclo nella rete, c'è il rischio che il pacchetto continui a girarvi senza riuscire ad uscirne. Sembra un problema da poco, ma nella realtà è abbastanza frequente che vi siano dei loop e Ethernet non mette a disposizione alcun meccanismo per prevenirlo.

Una strategia che risolve in parte questo dilemma è il TTL (*Time To Live*) con cui si imposta un numero limitato di hop che un pacchetto può fare e dopo del quale viene cancellato, ma ritornando al frame Ethernet, nella sua intestazione, non vi è alcun meccanismo del genere.

Una soluzione è quella di eliminare i loop, ma non è sempre possibile e soprattutto in reti molto estese non è possibile determinare a priori se vi siano cicli, oltretutto delle volte tornano utili (es.: in caso di guasto non si lasciano zone scoperte, per cui i cicli rendono in questo caso la rete più robusta). Inoltre, nel corso del tempo la rete può subire modifiche, in cui i dispositivi vengono tolti o aggiunti. Per cui è esclusa questa strada che consiste nel rimuovere i loop.

Un'altra soluzione è quella di "esplorare" le zone della rete, ovvero di fare *network mapping*. Questa tecnica è sia efficace che invisibile. La rete ideale sarebbe quella simile a una sorta di albero, ovvero un grafo connesso aciclico, tuttavia non essendo sempre possibile si usa un algoritmo di network mapping chiamato *Distributed Spanning Tree* (DST).

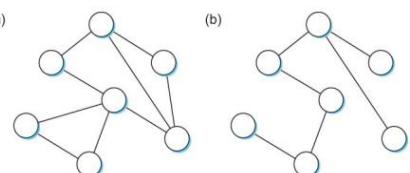
Distributed Spanning Tree

L'idea di base de DST è quella di "disattivare" alcuni collegamenti in modo da avere un grafo senza cicli, il che nella realtà corrisponde nel disattivare alcuni link e porte degli switch in maniera tale da ridurre i collegamenti attivi. Così facendo si ottiene un albero di copertura (*spanning tree*) sovrapposto a quello esistente (l'albero è aciclico).

Uno spanning tree, infatti, è un sottografo del grafo originale che copre tutti i vertici (del tipo LAN semplici), ma senza cicli. Esso mantiene tutti i vertici del grafo originale, ma elimina alcuni spigoli.

Nell'immagine, il grafo a sinistra è quello originale, mentre quello a destra è il risultato dell'algoritmo.

È da tenere presente che la rete originale viene mantenuta, quindi i link NON vengono in realtà scollegati fisicamente, ma solamente a livello software (viene tenuto spento) tramite "dialogo" tra switch.



L'algoritmo di spanning tree corrisponde allo standard IEEE 802.1D (del 2004) e ormai un po' tutti gli switch lo contengono. Questo software è un esempio di algoritmo distribuito che consente il dialogo tra switch in maniera indipendente dal traffico di pacchetti della rete. Con distribuito si intende che non c'è un controllo centrale che decide chi è il master o la radice dell'albero o chi deve attivarsi/disattivarsi, ma è un sistema paritario (*peer*) in cui sono i dispositivi stessi che gestiscono la politica di collegamento (*politica di agreement*). Questo sistema si adeguà, inoltre, a ogni cambio della struttura o aggiunta/rimozione dei dispositivi all'interno della rete. La riconfigurazione è automatica.

Esiste anche una variante chiamata "*Rapid Spanning Tree*", ovvero un protocollo utilizzato da un insieme di bridge per concordare uno spanning tree per una particolare LAN estesa.

Per quanto riguarda il funzionamento dell'algoritmo, ogni bridge ha un identificativo univoco (*Ethernet address*). A ogni switch è associato l'indirizzo più basso che ha (indirizzo visto come numero a 48 bit = 6 byte). Uno dei bridge, quello con l'ID più basso nella rete, viene "eletto" (concordato tra i bridge) la root dell'albero e svolgerà il ruolo di *hub*, ovvero distribuirà i pacchetti che gli arrivano agli altri bridge presenti, mentre questi ultimi computano la distanza (in termini di hop) dal root bridge e annotano quale delle loro porte si trova su questo percorso. In caso di loop, il bridge sceglie la distanza più corta (*shortest path*). Questa porta che viene selezionata come percorso preferito dal ponte verso la radice viene chiamata porta radice (*root port*). Infine, tutti i bridge connessi a una determinata LAN eleggono un singolo bridge designato che sarà responsabile dell'inoltro dei frame verso l'alto e verso il basso.

I bridge raggiungono questo accordo scambiando semplici messaggi di configurazione, chiamati *BPDU* (ovvero particolari tipi di pacchetti Ethernet), tra i vicini immediati (cioè sui segmenti condivisi). Ogni messaggio è una tripla (X, d, Y) dove:

- Y è l'identificatore del bridge che genera il messaggio
- X è l'identificatore del root bridge, in accordo con Y
- d è la distanza da X a Y , espressa in numero di LAN da attraversare.

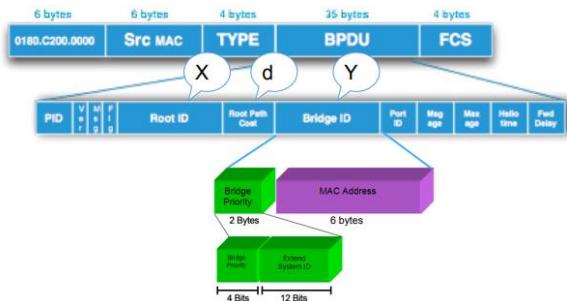
In realtà, questi messaggi sono definiti dall'802.1D e trasportano anche altri dati (come la priorità del bridge, i costi, ecc.) e si trovano all'interno di frame 802.3 con un indirizzo di destinazione speciale.

Ogni bridge ID è composto dall'indirizzo MAC e da una priorità del bridge di 16 bit (di cui solo i primi 4 bit sono la vera priorità, mentre gli altri indicano la VLAN, che non è rilevante in questo caso). La priorità predefinita è:

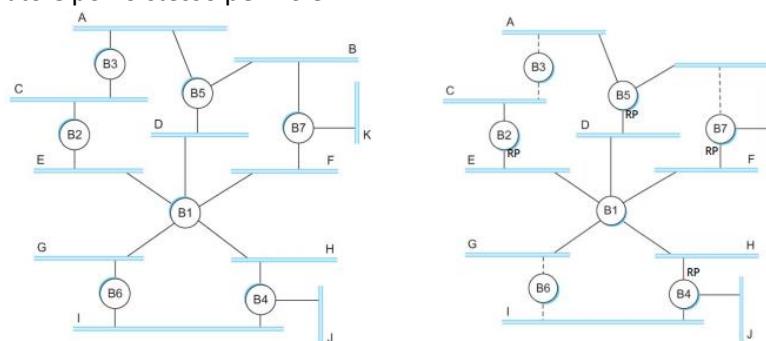
$$32769 = 1000\ 000000000001$$

Può essere modificata dall'amministratore se vuole che un ponte specifico diventi root, ad esempio mettendo:

$$4097 = 0001\ 000000000001$$



Nell'esempio dell'immagine precedente, si potrebbe avere come configurazione DST: B1 come ponte radice, B3 e B5 collegati alla LAN A, ma con B5 come il bridge designato, e B5 e B7 collegati alla LAN B, ma con B5 come il ponte designato e poi lo stesso per B6 e B4.



In questo caso i bridge B3 e B6 che sono completamente scollegati, vengono considerati dei bridge di backup.

Funzionamento pratico dell'algoritmo:

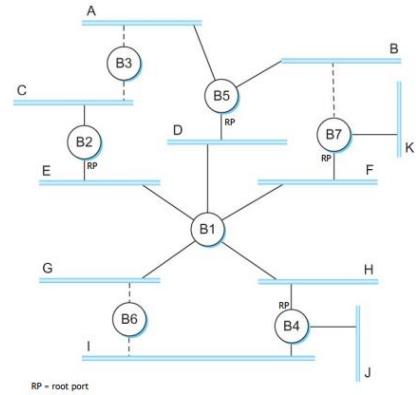
Inizialmente ogni bridge X pensa di essere la radice, quindi invia un messaggio di configurazione $(X, 0, X)$ su ciascuna delle sue porte identificandosi come radice e dando una distanza dalla radice di 0. Alla ricezione di un messaggio di configurazione (Y, d, Z) su una particolare porta, il bridge controlla se il nuovo messaggio è migliore dell'attuale miglior messaggio di configurazione registrato per quella porta. La nuova configurazione è migliore delle informazioni attualmente registrate se:

- identifica una radice con un ID minore, oppure
- identifica una radice con un ID uguale ma con una distanza minore, oppure
- l'ID della radice e la distanza sono uguali, ma il bridge mittente ha un ID più piccolo

Se il nuovo messaggio è migliore di quello attualmente registrato, allora il bridge prima di tutto aggiunge +1 al campo distanza-radice e poi scarta le vecchie informazioni e salva le nuove. A questo punto sa di non essere il bridge root, per cui smette di generare messaggi di configurazione (in quanto è solo il root che li genera) e inoltra i messaggi di configurazione provenienti da altri bridge dopo aver aggiunto +1 al campo di distanza.

Si consideri, una situazione in cui la corrente è stata appena ripristinata nell'edificio che ospita la seguente rete. Tutti i bridge iniziano con la pretesa di essere il root. Si consideri (Y, d, X) un messaggio di configurazione da parte del nodo X in cui afferma di trovarsi a una distanza d dal nodo radice Y .

Ad esempio, si consideri l'attività del nodo B3. Esso riceve $(B2, 0, B2)$ e poiché $2 < 3$, B3 accetta B2 come radice. B3 aggiunge +1 alla distanza annunciata da B2 e invia $(B2, 1, B3)$ a B5. Nel frattempo, B2 accetta B1 come radice perché ha l'ID più basso e invia $(B1, 1, B2)$ a B3. Allo stesso modo B5 accetta B1 come radice e invia $(B1, 1, B5)$ a B3. B3 a sua volta accetta B1 come radice e nota che sia B2 che B5 sono più vicini alla radice di lui. Pertanto, B3 smette di inoltrare messaggi su entrambe le sue interfacce. Questo lascia B3 con entrambe le porte non selezionate.



Anche dopo che il sistema si è stabilizzato, il root bridge continua a inviare periodicamente messaggi di configurazione. Gli altri bridge continuano a inoltrare questi messaggi. Quando un bridge si guasta, gli altri situati a valle di esso non ricevono i messaggi di configurazione. Per cui dopo aver aspettato un determinato periodo di tempo, essi affermano nuovamente di essere il root e l'algoritmo ricomincia.

Limitazioni di questo algoritmo:

Sebbene l'algoritmo sia in grado di riconfigurare lo spanning tree ogni volta che un bridge si guasta, non è in grado di inoltrare i frame su percorsi alternativi per aggirare un bridge congestionato. Oltre al fatto che ogni volta che un bridge si connette/disconnette bisogna rifare tutto il giro di configurazione.

Inoltre, la disabilitazione delle porte negli switch porta a percorsi più lunghi tra due LAN (es.: si consideri il percorso tra le LAN B e K). Questo è un problema per il traffico di pacchetti, in quanto potrebbe congestionarsi su alcuni rami della rete e quando un bridge ha il buffer saturo deve scartare i pacchetti.

Limitazioni dei bridge:

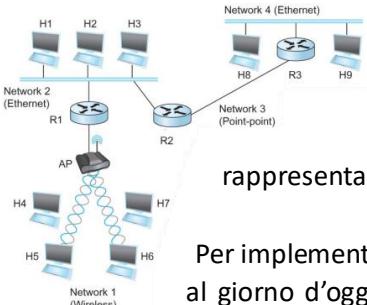
I bridge non sono scalari, ovvero con l'algoritmo spanning tree non si scala (tempo lineare, non sfrutta la gerarchia), come anche il broadcast non scala per via del troppo traffico su larga scala (non se ne ha bisogno).

Inoltre, non si adattano all'eterogeneità, in quanto tutte le parti della rete devono avere lo stesso tipo di indirizzo e caratteristiche simili (es.: il broadcast).

Se due segmenti ammettono MTU (grandezza massima del payload) diverse, ad esempio Ethernet (802.3) a 1500 byte e WiFi (802.11) a 2346 byte, in una direzione, il bridge deve frammentare un frame in due o più frame, che devono essere ricomposti a livello di datalink dal ricevitore (*frammentazione PAF*).

Per superare queste limitazioni, si deve astrarre dai dettagli del datalink e quindi passare al livello 3 (*Internetworking*).

Internetworking



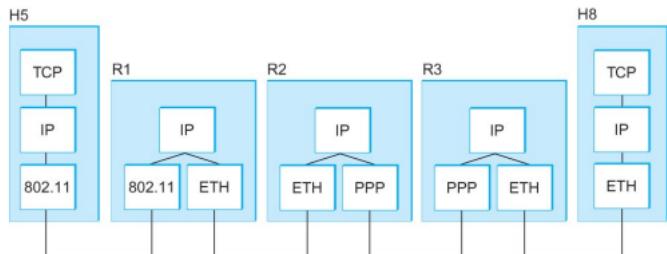
L'internetworking è implementato a livello 3 (Internet) dell'architettura stack e è un insieme arbitrario di reti interconnesse per fornire una sorta di servizio di consegna dei pacchetti da host a host.

Ad esempio nell'immagine affianco si ha una semplice rete Internet in cui *H* rappresenta gli host, *R* i router e *AP* un access point.

Per implementare questa astrazione si ha bisogno di un protocollo di livello 3 e il più utilizzato al giorno d'oggi per questo scopo è l'**IP** (*Internet Protocol*). Esso consente di costruire reti internet scalabili e eterogenee, viene eseguito su tutti i nodi di un insieme di reti e definisce l'infrastruttura che consente a questi nodi e reti di funzionare come un'unica rete Internet logica.

C'è da tenere presente che a questo livello di astrazione, si potrebbero avere infrastrutture molto differenti tra di loro, in quanto in certi casi anche i livelli 1 e 2 dello stack possono differire anche radicalmente tra una tecnologia a un'altra. Ma non solo: anche gli algoritmi di indirizzamento dei pacchetti potrebbero essere ineguali tra di loro. Ad esempio, si potrebbe avere una rete come quella nell'immagine sopra, che dispone sia di Ethernet e di wireless, come anche potrebbero esserci delle fibre ottiche o altri metti trasmissivi che lavorano con protocolli livello 2 diversi da quelli utilizzati da Ethernet e wireless (che hanno un funzionamento in realtà abbastanza simile).

Per fare in modo che i pacchetti riescano ad arrivare in reti eterogenee tra loro, si usano dei particolari dispositivi chiamati *intradicatori (router)*. Essi sembrano delle sorte di switch, solo che invece che funzionare a livello 2, questi commutatori sono appositi per il livello 3. Per fare in modo che funzionino con qualsiasi tipo di rete, essi possiedono diverse interfacce, ad esempio per la rete in figura sopra, i router R1, R2, R3 hanno interfacce per protocolli Ethernet, wireless (802.11) e PPP (*point-to-point*). Al contrario gli switch visti finora avevano interfacce tutte uguali. Dunque, quando un router riceve un pacchetto di un certo tipo di protocollo e deve inviarlo a un dispositivo che ne utilizza un altro, deve de-capsulare il payload da un certo livello 2 e poi incapsularlo in un altro (es.: in figura R1 sfrutta le interfacce 802.11 e Ethernet, entrambe livello 2). Per poter effettuare questo passaggio utilizza il protocollo IP, con cui converte il messaggio in formato adatto alla tecnologia utilizzata. Per cui, a questo livello, anche gli indirizzi degli host vengono convertiti in indirizzi IP di tipo IPv4 o IPv6, in modo da uniformarli indipendentemente dalla tecnologia utilizzata.



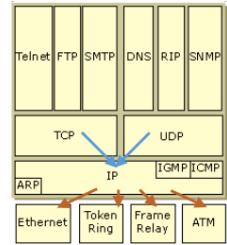
Per effettuare le conversioni tra le diverse interfacce, si deve ricevere dei pacchetti, leggere la loro intestazione e scomporla (si dovrà in realtà controllare anche l'eventuale incorrettezza). Poi, una volta prese le informazioni rilevanti, tra cui il payload, si deve creare una nuova intestazione per l'interfaccia dell'altro dispositivo che utilizza una tecnologia differente e aggiungervi il payload con tutte le sue altre informazioni. A questo punto si inoltra il messaggio.

Il livello 3 inoltre, mette a disposizione alcune API per i livelli superiori dello stack, tra cui il servizio di inoltro (*packet forwarding*) che indipendentemente da dove sia l'host e da quale tecnologia utilizzi, cerca di far arrivare a destinazione il messaggio (es.: come per i telefoni che non si sa a priori dove sia la persona né che tipo di telefono abbia, ma il servizio cerca di funzionare lo stesso). Tuttavia non viene garantita totalmente la ricezione del destinatario. Altri protocolli e servizi implementati a questo livello sono: ICMP, ARP, RARP, ecc.

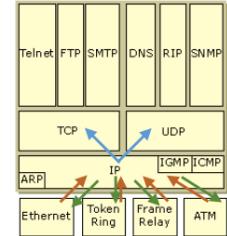
Un pacchetto (talvolta chiamato “datagramma”) può arrivare al livello IP in due modi:

- quando un livello superiore (es.: TCP o UDP) richiede di inviare alcuni dati (*payload*) a un altro host:
 1. questi dati vengono incapsulati in un datagramma IP
 2. il datagramma viene inviato al nodo successivo utilizzando un protocollo di collegamento dati appropriato.

In questo modo, un'applicazione può inviare/trasmettere dati attraverso qualsiasi interfaccia sottostante, senza saperlo. Inoltre, è compito di IP sapere a quale interfaccia passare il datagramma



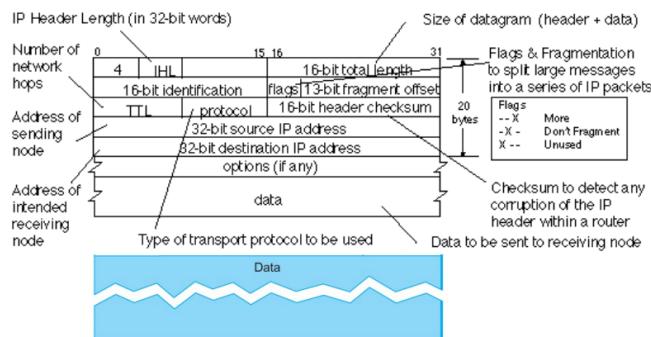
- quando un pacchetto viene ricevuto da un livello inferiore (da un’interfaccia):
 - se è indirizzato all’host locale, l’intestazione IP viene spachettata e il payload viene consegnato al protocollo di trasporto corretto
 - se è indirizzato a un altro host, allora può essere inviato utilizzando un’interfaccia, cioè utilizzando un livello di datalink sottostante (se l’host è istruito a inoltrare questi pacchetti)



Anche in questo caso, come per gli switch si adopera con una strategia di *store-and-forward*, ma, a differenza degli switch di livello 2, si astrae dalla tecnologia del collegamento sottostante.

Il modello di consegna dei pacchetti è un modello senza connessione (no circuiti virtuali), quindi consegna *best-effort* (servizio inaffidabile). Dunque, i pacchetti possono andare persi, essere consegnati in ordine sparso, duplicati o ritardati. Si segue uno schema di indirizzamento globale, che fornisce un modo per identificare tutti gli host della rete Internet (astratti dagli indirizzi di livello 2 (MAC) sottostanti).

Formato dei pacchetti



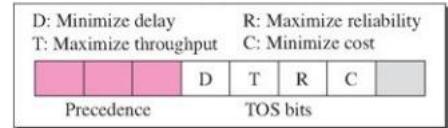
Ogni parola (*word*) è formata da 4 bit e l’intestazione è sempre di almeno 1 byte. Tra i parametri presenti dei pacchetti datagram IPv4 vi sono rispettivamente:

- Versione (4 byte): attualmente 4 per IPv4 (o 6 per IPv6)
- Header length (4): numero di parole a 32 bit nell’intestazione (quindi la lunghezza reale è $4H.\text{len.}$)
- TOS (8): tipo di servizio (*Type Of Service*)
- Lunghezza (16): lunghezza totale del pacchetto, compresa l’intestazione, in byte
- Identificazione (16): usato dalla frammentazione
- Flag (3) e Offset (13): usati dalla frammentazione, in cui se il bit di flag è 0 deve essere 0, se è 1 significa non frammentare (DF) e se è 2 vuol dire che ci sono più frammenti (MF)
- TTL (8): numero di salti (*hop* tra router) che questo datagramma può percorrere prima di essere scartato (per evitare loop)
- Protocollo (8): chiave demux (TCP=6, UDP=17, gli altri sono in figura affianco)
- Checksum (16): solo dell’header e corrisponde allo stesso CRC già visto
- Indirizzi di destinazione (DestAddr) e sorgente (SrcAddr) di 32 bit ciascuno

Protocol	Value
ICMP	1
IGMP	2
TCP	6
UDP	17
OSPF	89

Tra i parametri più importanti vi è il TOS (Tipo di servizio, da 8 bit) e che dovrebbe essere usato per specificare come trattare il datagramma. Serve quando le interfacce che si stanno implementando hanno determinate caratteristiche, ad esempio '*lenta e economica*', '*veloce e costosa*', '*velocissima e economica ma inaffidabile*', ecc. per cui si deve specificare quale usare. È nato per fornire una priorità in caso di congestione (buffer pieno), poiché la rete che gestisce i datagrammi deve avere la precedenza. Il tipo di servizio richiesto è:

- 0000: normale (predefinito, il router sceglie)
- 1000 (*D*): minimizzare il ritardo
- 0100 (*T*): massimizzare il throughput
- 0010 (*R*): massimizzare l'affidabilità
- 0001 (*C*): minimizzare il costo

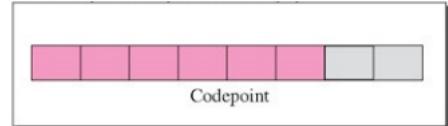


Il "costo" è dipendente dalla politica del router e potrebbe corrispondere a costi di €/Kbyte, oppure energetici, o di CPU, memoria, ecc.

Di solito il TOS viene usato per pacchetti speciali di controllo che devono avere la priorità e quindi essere passati avanti nella coda. Se non si specifica nulla (0000), si lascia il compito di decidere il tipo di servizio al router e quelli più economici inoltrano incuranti del tipo di link che hanno, mentre quelli più sofisticati controllano il pacchetto come è strutturato (di solito sbirciano nel payload, non solo nell'header) e cercano di interpretare il tipo di servizio che dovrebbero avere e glielo forniscono (es.: se trovano un TELNET o un http cercano di minimizzare il delay). Ogni protocollo di trasporto ha un servizio predefinito associato: TELNET, FTP (com), SMTP (com, usa il servizio *D*), SNMP (con servizio *R*), FTP (dati), SMTP (dati, con servizio *T*), ecc.

Questo appena visto, in realtà, è il TOS originale previsto per le prime versioni di IPv4, mentre oggigiorno di forniscono servizi differenziati (*Differentiated services*, DSCP). Quindi, sono di conseguenza cambiati i codici (quelli appena visti):

- Se 3 bit più a destra = 000: i 3 bit più a sinistra sono interpretati come priorità, come nel TOS;
- Se 3 bit più a destra ≠ 000: i 6 bit più a sinistra definiscono 64 servizi:
 - xxxx0: servizi standard IETF
 - xxxx11: servizi definiti dalle autorità locali
 - xxxx01: uso temporaneo, sperimentale



Con standard IETF si intende quelli ufficiali come ad esempio il default, minimizzare il delay, massimizzare il throughput, mentre con "autorità locali" ci si riferisce a "categorie" (es.: con un certo codice si rappresentano tutti i pacchetti degli studenti) e si fornisce un servizio diverso in base a come sono stati definiti.

Frammentazione e ricomposizione

Al livello 3 si ha a che fare con la frammentazione dei pacchetti e la loro ricomposizione. Questo perché il livello 3 mira ad astrarsi rispetto alla tecnologia di collegamento sottostante e una delle dipendenze tra una tecnologia e un'altra è la dimensione del frame, ovvero la dimensione di quanti dati è possibile inserire in una singola unità di trasmissione a livello 2 (es.: Ethernet ha un payload di massimo 1500 byte, mentre nel PPP viene negoziata durante l'handshake). Questo valore è chiamato MTU (*Maximum Transmission Unit*). Se un datagramma non può essere contenuto in un frame, si hanno due possibili scenari:

- Se il flag *Don't Fragment* è a 1, è sufficiente abbandonare il datagramma e notificarlo al mittente (usando ICMP).
- Se DF=0, il router può procedere alla frammentazione.

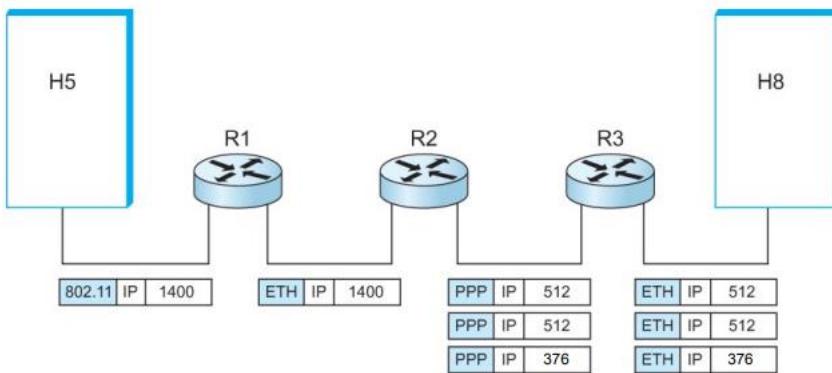
Network/protocol	MTU (bytes)
Hyperchannel	65535
Token Ring	17914
FDDI	4352
WiFi	2346
Ethernet	1500
X.25	576
PPP	Negotiated

I datagrammi vengono frammentati a livello di rete, in modo trasparente per i livelli superiori, i quali vedono solo i datagrammi interi.

La frammentazione avviene in un router solo quando riceve un datagramma che vuole inoltrare su una rete con $MTU < \text{lunghezza del datagramma}$ (se fosse minore non servirebbe). Tutti i frammenti hanno lo stesso identificatore nel campo *Ident*. I frammenti sono pacchetti autonomi: viaggiano indipendentemente l'uno dall'altro e possono essere frammentati ulteriormente. La dimensione di ogni frammento è scelta dal router, in modo tale che $\text{intestazione} + \text{carico utile} \leq MTU$. In generale, la dimensione del payload è scelta come il più grande multiplo di 8 (byte) inferiore a $MTU - \text{lunghezza dell'intestazione}$. Ad esempio: se $MTU = 536$, header=20 byte, allora $536 - 20 = 516$, e il più grande multiplo di 8 meno di 516 è 512. Quindi ogni pacchetto è lungo 512 + 20 byte.

Il riassemblaggio viene effettuato dall'host ricevente. Quando arriva un frammento di un nuovo identificatore, viene allocato un buffer (e associato a quell'identificatore). Il payload di ogni frammento viene collocato in questo buffer in base al suo offset (se un frammento arriva due volte, l'ultimo arrivato sovrascrive il precedente). Quando tutti i frammenti sono arrivati, il buffer contenente il payload ricostruito viene passato ai livelli superiori. Qui non vi è nessun recupero per i frammenti mancanti: se ne mancano ancora alcuni dopo il timeout ($\sim 15\text{-}60$ secondi), tutti i dati vengono scartati e viene inviato un messaggio ICMP al mittente.

Di seguito un'immagine dei datagrammi IP che attraversano la sequenza di reti fisiche (il payload è da considerarsi senza header, quindi nel conteggio vanno aggiunti 20 byte, es.: 1420 byte per Ethernet):



(a)	Start of header
	Ident = x 0 Offset = 0
	Rest of header
	1400 data bytes

(b)	Start of header
	Ident = x 1 Offset = 0
	Rest of header
	512 data bytes

Start of header
Ident = x 1 Offset = 64
Rest of header
512 data bytes

Start of header
Ident = x 0 Offset = 128
Rest of header
376 data bytes

Nell'esempio, i campi dell'header utilizzati nella frammentazione IP sono di due tipi:

- (a) pacchetto non frammentato;
- (b) pacchetti frammentati corrispondenti. Si noti che:

$$\text{campo offset} = \frac{\text{offset reale}}{8}$$

(es.: $64 = \frac{512}{8}$), quindi in ogni frammento (tranne l'ultimo) la lunghezza del payload è un multiplo di 8. Questo perché l'intestazione ha spazio molto ridotto per cui si devono utilizzare numeri piccoli (dunque non si parte da 512 che sarebbe la reale dimensione).

Da notare, inoltre, come il pacchetto non frammentato ha sempre offset e flag di *More Fragment* entrambi a 0, mentre per quello PPP che è stato frammentato ha il flag di multi-pacchetto a 1 (tranne l'ultimo) e come offset il numero di offset, ovvero in che posizione all'interno della serie di frammenti si colloca quella serie di dati.

Nel caso in cui durante la trasmissione un pacchetto arrivasse in ritardo e nel frattempo scade il delay, il ricevente scarterebbe tutti i frame memorizzati fino a quel momento. Quando poi arriva il pacchetto che si era perso, il ricevente vede che è di una serie (*More Fragment* a 1 oppure $\text{offset} > 0$) e quindi alloca un nuovo buffer e resta in attesa di altri pacchetti della stessa serie. Tuttavia, quando il mittente finisce di mandare i pacchetti, il ricevente non può ricomporre il messaggio originale perché mancano i primi frame, quindi scarta nuovamente la serie di pacchetti. Questo meccanismo ha questo possibile svantaggio del doppio scarto.

Indirizzi globali

Quando si lavora con i protocolli livello 3, si ha a che fare con il concetto di *indirizzi globali*, ovvero degli indirizzi (IPv4 o IPv6) unici a livello globale e che identificano i dispositivi connessi alla rete. Un host può avere molti indirizzi, ma un indirizzo non può essere assegnato a più di un host.

Network number	Host number
----------------	-------------

Gli indirizzi globali sono anche gerarchici, cioè l'indirizzo è diviso in due parti: la prima indica la rete, assegnata in modo univoco a una singola entità (*Autonomous System*) dalle autorità centrali, mentre la seconda indica l'host effettivo all'interno della rete. Ogni sistema autonomo può assegnare gli indirizzi all'interno delle proprie reti come preferisce. In IPv4 si utilizzano 32 bit con la notazione a punti (es.: 10.3.2.4, 128.96.33.81, 192.12.69.77, ecc.).

Questo meccanismo consente di identificare le controparti che dialogano a questo livello in maniera omogenea, astratta e indipendente dalla tecnologia utilizzata. Infatti, i router sono in grado di instradare i pacchetti tramite identificazione dell'indirizzo della rete, ovvero in base a un indirizzo globale. Al contrario del MAC address che può essere locale ed è indifferente in quale luogo, a livello mondiale, esso sia collegato, un indirizzo globale una volta assegnato è riconoscibile univocamente da chiunque, dunque cambiando di collocazione un dispositivo, cambia anche il suo indirizzo (assegnazione dinamica).

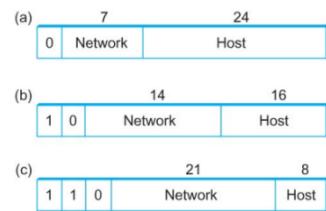
A questo punto sorge un nuovo problema: come distinguere il numero di rete e il numero di host. Se si utilizzasse un meccanismo simile alle tabelle di inoltro degli switch, si avrebbe una rete che funziona solo a livello teorico, ma non pratico in quanto non è scalabile. Infatti, si supponga ad esempio di avere un miliardo di host connessi e per ognuno di essi avere un indirizzo di 4 byte, per memorizzare tutti gli indirizzi servirebbero 4 Gbyte (di RAM) per ogni router (per la sola tabella, a cui si somma anche il tempo di scansione, quindi si creerebbe un delay molto grande, oltre al fatto che non è scalabile). Inoltre, i router non hanno tutta questa memoria, perché sono tecnologie molto vecchie con al massimo qualche Kbyte.

Per questi motivi è stata adottata una strategia che consenta la memorizzazione di più indirizzi sotto a un'unica rete e quindi entra in gioco la struttura gerarchica degli indirizzi IP e il fatto di dividerli in due parti. Quindi nel caso di 4 byte si avrebbero $4 \text{ byte} \times 8 = 32 \text{ bit}$ dunque 2^{32} combinazioni di indirizzi, di cui 2^{16} sono identificare per la rete e 2^{16} l'host, ovvero 65.536 entry ciascuna. In questa maniera con qualche Kbyte si possono memorizzare moltissimi indirizzi di rete (non serve memorizzare la parte degli host, in quanto si arrangiano i router a sapere dove instradare i pacchetti).

Tuttavia si può fare di meglio: invece che usare solo due livelli per la memorizzazione degli indirizzi, se ne possono utilizzare di più, dividendo le varie reti in sottoreti identificabili univocamente all'interno della rete più grande a cui appartengono (gerarchia). In questo modo, gli indirizzi IPv4 restano di 32 bit però è possibile realizzare una classificazione in maniera tale da individuare le varie sottoreti.

Originariamente, si erano definite tre classi di reti riconoscibili dai primi bit:

- Classe A: indirizzi che iniziano con 0, i cui 7 bit successivi sono interpretati come numero di rete e i 24 bit rimanenti sono il numero di host all'interno della rete.
- Classe B: indirizzi che iniziano con 10, i cui successivi 14 bit sono interpretati come numero di rete e i restanti 16 bit sono il numero di host all'interno della rete
- Classe C: indirizzi che iniziano con 110, i cui 21 bit successivi sono interpretati come numero di rete e gli 8 bit rimanenti sono il numero di host all'interno della rete.



Da ricordare il fatto che gli indirizzi che iniziano con 111 non sono utilizzati per l'indirizzamento degli host: infatti, la classe D (1110) è per il multicast e la classe E (1111) è inutilizzata.

Inoltre, all'interno di una rete, non è possibile utilizzare due indirizzi speciali per gli host: quello con tutti "0" nella parte host (il primo generabile), poiché identifica la rete (es.: 158.110.0.0), e quello con tutti "1" nella parte host (ultimo generabile), in quanto riservato al broadcast all'interno della rete (es.: 158.110.255.255).

Tutti gli altri indirizzi rimanenti possono essere assegnati agli host (quelli da 158.110.0.1 a 158.110.255.254).

Per quanto riguarda il numero di host per ogni rete (indirizzi *unicast*), si ha che:

- Classe A: $2^7 = 128$ reti possibili, con $2^{24} - 2 = 16777214$ host ciascuna
- Classe B: $2^{14} = 16384$ reti possibili, con $2^{16} - 2 = 65534$ host ciascuna
- Classe C: $2^{21} = 2097152$ reti possibili, con $2^8 - 2 = 254$ host ciascuna

Complessivamente: $2,1 \cdot 10^6$ reti (in realtà meno, a causa delle reti "speciali") e limite teorico di host possibili di $3753869056 \cong 3,75 \cdot 10^9$.

Tuttavia come è possibile vedere dai numeri, questa soluzione andava bene solo in passato (circa fino agli anni '80), poiché si hanno "soltanto" circa 3 miliardi di host possibili, mentre nel mondo oggigiorno vi sono molti più dispositivi (es.: telefoni, computer, dispositivi IoT, ecc.). Per questo motivo si sta passando da IPv4 a IPv6, il quale consente di poter creare molti più indirizzi.

IP Datagram Forwarding

Per quanto riguarda il meccanismo di inoltro, è molto simile a quello degli switch di livello 2, solo che cambia il modo di leggere l'indirizzo di destinazione (perché come già detto le tabelle di inoltro degli switch non sono scalabili e serve tanta memoria, anche se esistono switch di livello 3 con memorizzazione di tutti gli host presenti, ma vale solo per le reti di dimensioni relativamente piccole e non a quelle globali).

L'inoltro avviene per rete di destinazione, cioè la rete a cui l'indirizzo di destinazione appartiene, di cui la tabella di inoltro mappa il numero di rete nel salto successivo.

Ogni router deve adottare una strategia semplice e efficace per inoltrare correttamente i pacchetti, senza perdita di dati, con poco delay e pochi calcoli per capire dove instradare (poiché i router sono macchine semplici e economiche, con poche risorse, ma veloci e duraturi).

Dunque, ogni router (*nodo*) legge l'indirizzo di destinazione, che contiene l'indirizzo della rete di destinazione, e se il nodo è direttamente connesso alla rete di destinazione, inoltra il pacchetto all'host, altrimenti inoltra il pacchetto a un router intermedio che dovrebbe sapere come gestirlo (*next hop*). Ogni host ha un router predefinito (*gateway*) e mantiene una tabella di inoltro: per questo è necessario un algoritmo.

Affianco, l'esempio della tabella di routing per il router R2.

Nelle tabelle reali, la destinazione (*NetworkNum*) è l'indirizzo di rete (es.: 158.110.0.0, 158.111.0.0, ecc.).

Quindi a ogni rete locale deve essere assegnato un indirizzo di rete diverso.

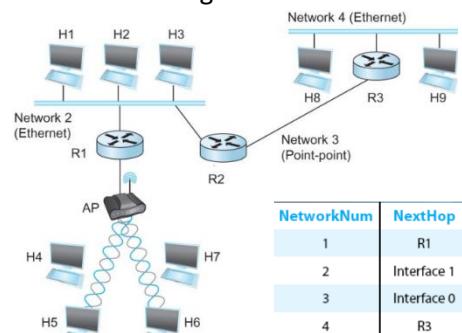
Gli host di diverse LAN hanno indirizzi di reti diverse.

Di seguito l'algoritmo (omettendo i problemi di TTL):

```
if (NetworkNum of destination == NetworkNum of one of my interfaces) then
    directly deliver packet to destination over that interface
else if (NetworkNum of destination is in my forwarding table) then
    deliver packet to NextHop router
else if there is a default router
    deliver packet to default router
else drop packet
```

Per un host con una sola interfaccia e un solo router predefinito nella sua tabella di inoltro, questo si semplifica in:

```
if (NetworkNum of destination == my NetworkNum) then
    deliver packet to destination directly
else
    deliver packet to default router
```



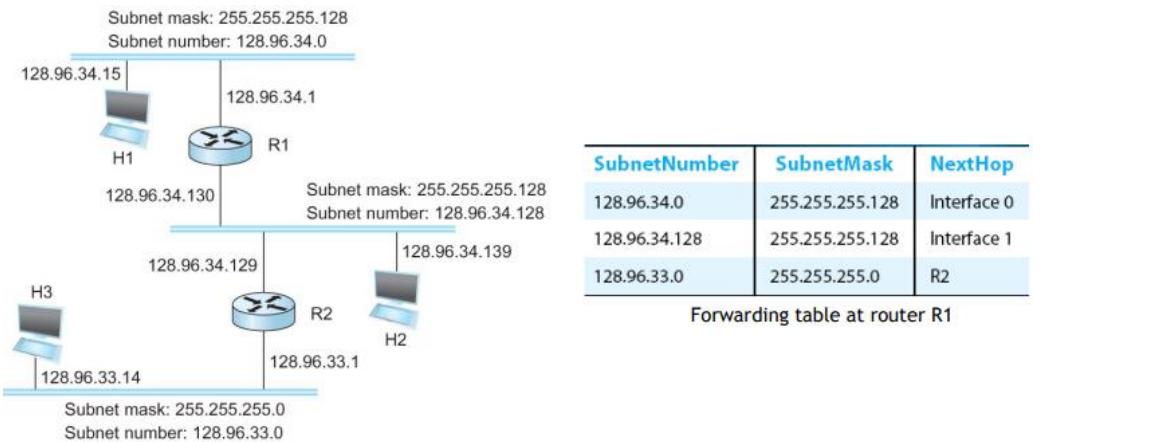
Come appena premesso, in questo algoritmo proposto non sono stati gestiti i cicli, ovvero il controllo sui TTL dei pacchetti. Infatti, ogni router decremente di 1 il TTL del pacchetto e se vede che è a 0 viene scartato (questo è uno dei problemi per cui un pacchetto può non arrivare a destinazione). Di solito si imposta un TTL di 255 (ovvero 2^8 , un 1 byte), ovvero il diametro massimo della rete, in quanto sono gli hop massimi che un pacchetto può effettuare, quindi i router che attraversa.

Subnetting

Tematica centrale del livello 3 è quello delle sottoreti (*subnet*) in quanto la rete presenta una struttura gerarchica. Talvolta la suddivisione per classi (A, B, C, ecc.) può non bastare, poiché ad esempio su una rete di classe B possono esserci tanti router appartenenti a essa, per cui si necessita di suddividere ulteriormente in altre sottoreti.

Infatti, gli indirizzi vanno scelti con cura in base al numero di host, ad esempio per una rete universitaria o aziendale, che possono essere molto grandi, è necessario utilizzare la classe B. Talvolta non è possibile sapere a priori quanti host saranno collegati in contemporanea (es.: studenti di una scuola che si connettono), tuttavia si cerca di stimare il numero più adatto.

Un meccanismo che consente la suddivisione in più sottoreti è quello delle maschere di sottorete (*subnet mask*). In realtà, le maschere di sottorete definiscono una partizione variabile della parte host degli indirizzi di classe A e B, in base alle reti fisiche. Le sottoreti sono visibili solo all'interno della rete (cioè del sistema autonomo). Di seguito un esempio:



Nella prima sottorete 128.96.34.0 si ha una subnet mask di 255.255.255.128 poiché si destinano gli ultimi 7 bit agli host, per cui i loro indirizzi possono andare da 128.96.34.1 a 128.96.34.126 (128.96.34.127 è per il broadcast). Infatti, l'host computer ha come indirizzo 128.96.34.15, ovvero uno a caso di quelli disponibili, come anche l'interfaccia del router è collegata alla 128.96.34.1 (poteva essere una qualsiasi, ma di solito per semplicità e memonica si sceglie la prima disponibile, in quanto la 128.96.34.0 è della rete).

La seconda rete che il router R1 vede è la 128.96.34.128, che presenta la stessa subnet mask della rete precedente. Questa cosa è possibile in quanto l'indirizzo presenta comunque tutti 0 alla fine del proprio ultimo byte, ovvero sotto alla subnet mask. In questo secondo caso il numero di host è lo stesso di prima, solo che vanno da 128.96.34.129 a 128.96.34.254 (+ quello di broadcast). Infatti, le porte dei calcolatori sono 128.96.34.129 e 128.96.34.130 per i due router e 128.96.34.139 per il computer.

La terza rete (128.96.33.0), invece, non è direttamente collegata al router R1. In questo caso, però, la subnet mask è di 255.255.255.0 per cui gli indirizzi possibili per gli host sono 254 (256 – 2 che sono rete e broadcast). Infatti, le *netmask* si devono creare in base alla classe e al numero di host che ci si prefissa di avere, l'importante è che non ci siano sovrapposizioni. In questo caso si occupa tutto l'ultimo byte dell'indirizzo. Lo stesso meccanismo lo si può replicare all'interno della sottorete per dividerla ulteriormente in due (o più), ottenendo la stessa subnet mask delle due reti precedenti, poiché si va a occupare un bit per lo split.

Per quanto riguarda il router R1, di cui si può vedere la sua *forwarding table*, può inviare direttamente i pacchetti nelle due sottoreti a esso collegare (128.96.34.0 e 128.96.34.128), tuttavia se volesse inviare un pacchetto alla sottorete 128.96.33.0, dovrebbe inviare il messaggio al router R2. Per cui esso invia il pacchetto all'indirizzo 128.96.34.129, ovvero all'interfaccia di R2, e poi si occuperà quest ultimo del forward da parte di R1 al rispettivo host di quella sottorete. A sua volta R2 avrà una propria tabella di inoltro.

L'algoritmo di forwarding è:

```
D = destination IP address
for each entry <SubnetNum, SubnetMask, NextHop>
    D1 = SubnetMask & D           // bitwise "and"
    if D1 = SubnetNum
        if NextHop is an interface
            deliver datagram directly to destination and return
        else
            deliver datagram to NextHop (a router)
    if no entry matches
        drop datagram
```

L'operazione logica AND è intesa bit a bit, ovvero mette a 0 tutti i bit che sono a 0 della netmask. Quindi prende ogni entry e confronta la sua subnet mask con l'indirizzo di destinazione e se il risultato corrisponde al numero di subnet allora forwarda il pacchetto (quindi o lo inoltra direttamente alla rete oppure delega a un altro router il compito di forward), altrimenti continua a controllare fino a quando non lo trova o lo deve scartare.

Secondo questa logica dell'algoritmo, c'è un indirizzo che combacia/funziona (*match*) sempre: il 0.0.0.0 della subnet mask, poiché qualsiasi indirizzo venga utilizzato il risultato è sempre 0. Quindi, utilizzerebbe un router predefinito se non corrisponde a nulla.

Inoltre, altre cose particolari di questo meccanismo sono che non è necessario che tutti gli 1 nella maschera di sottorete siano contigui, il che può portare alla creazione di sottoreti strane. In più, come è già stato detto, è possibile inserire più sottoreti in una rete fisica. Esse vengono decise internamente dall'amministratore di rete e non sono visibili dal resto di Internet e di solito vengono assegnate in base a separazioni logiche (es.: uffici diversi, filiali, ecc.) e fisiche diverse).

Esercizio (DA NON CONSIDERARE) [LEZIONE SO305 PARTE 1]:

Dato l'indirizzo IP della rete "Eduroam" dell'università 158.110.239.57, la subnet mask è 255.255.240.0 (?), di cui 240 $\xrightarrow{\text{in binario}} 11110000$ e 239 $\xrightarrow{\text{in binario}} 11101111$. Effettuando una AND tra i due si ottiene:

$$11110000 \text{ AND } 11101111 = 11100000 \xrightarrow{\text{in decimale}} 224$$

Quindi la rete UNIUD è 158.110.224.0 con subnet mask 255.255.240.0 (?) e $4096 - 2 = 4094$ indirizzi assegnabili per gli host.

Indirizzamento senza classi

Esiste anche una tecnica di subnetting che sostanzialmente fa l'operazione inversa rispetto a quella delle classi, ovvero l'instradamento interdominio senza classi (*Classless Inter-Domain Routing*). L'idea è che se il subnetting consiste nel suddividere le reti in categorie (classi), in questa si cerca di raggruppare reti diverse e contigue in una unica. Ad esempio se si prendessero due reti di classe C, 200.7.9.0 con netmask 255.255.255.0 e 200.7.1.0 con stessa subnet mask, e le si fondessero in una si otterebbe un indirizzo 200.7.0.0 con subnet mask 255.255.254.0, in cui il bit meno significativo diventa uno spazio di indirizzamento e al posto di avere 7 bit di indirizzamento per gli host, se ne hanno 9, per cui la rete non è più vista come due separate, ma come una sola, il cui primo indirizzo è 200.7.0.1, mentre l'ultimo è 200.7.0.255 (broadcast).

Questo meccanismo lo si può anche applicare a più di due sottoreti contighe, ad esempio se si prendessero quattro reti contigue, si dovrebbero dedicare 2 bit e non 1 come prima alla parte di indirizzamento (quindi alla parte host) ottenendo una netmask 255.255.252.0, poiché si hanno questi 2 bit aggiuntivi di suddivisione, e 1024 indirizzi disponibili. Infatti, si riesce a “assemblare” reti più piccole per ottenere reti intermedie che non corrispondono esattamente a una specifica classe, ma sono una via di mezzo tra due, modellando quindi la dimensione della rete fino a adattarla a quella necessaria a un utente (col sistema a classi invece si sceglie la prima configurazione che riesce a contenere il numero di host).

In questo modo si aumenta l’efficienza di assegnazione degli indirizzi e si risolvono due problemi di scalabilità in Internet: la crescita delle tabelle di instradamento della dorsale, in quanto è necessario memorizzare un numero sempre maggiore di numeri di rete, e il potenziale esaurimento dello spazio degli indirizzi a 32 bit.

Con efficienza nell’assegnazione degli indirizzi si intende che a causa della struttura degli indirizzi IP con indirizzi di classe A, B e C, si costringe a distribuire lo spazio degli indirizzi di rete in parti fisse di tre dimensioni molto diverse (es.: una rete con due host ha bisogno di un indirizzo di classe C che ha efficienza di assegnazione $2/254 = 0,78\%$, mentre una rete con 256 host ha bisogno di un indirizzo di classe B che ha efficienza di assegnazione dell’indirizzo di $256/65535 = 0,39\%$). Mentre nel caso *classless* si possono ridurre gli sprechi sugli indirizzi (che si rischia di non usare) ottimizzandone l’efficienza complessiva.

In realtà, le reti odierne hanno anche un’altra problematica: al giorno d’oggi non si possono più assegnare classi B o A se non si dimostra che se ne ha realmente bisogno. Questo perché una classe B utilizza spazi di 65535 indirizzi per gli host e se non se ne utilizza almeno la metà non ha molto senso assegnare tale indirizzo così grande. Inoltre, non è sempre possibile effettuare un reclamo per farsi dare uno spazio/classe di indirizzi inferiore se ci si accorge che in realtà se ne usano molti di meno.

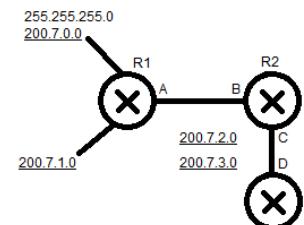
Per cui una soluzione può essere quella di dare invece un numero adeguato di indirizzi di classe C, quindi per qualsiasi *Autonomous System* (AS) con almeno 256 host, si può garantire un utilizzo dello spazio degli indirizzi di almeno il 50%.

Tuttavia anche questa soluzione ha un problema: l’eccessivo fabbisogno di memoria nei router. Infatti, se a un singolo AS sono assegnati, per esempio, 16 numeri di rete di classe C, ogni router della dorsale Internet ha bisogno di 16 voci nelle sue tabelle di routing per quell’AS, anche se il percorso verso ognuna di queste reti è lo stesso. Se invece si avesse assegnato una rete di classe B all’AS, le stesse informazioni di routing possono essere memorizzate in un’unica voce, con un’efficienza di $16 \times 255 / 65536 = 6,2\%$.

Infatti, c’è da tenere conto che a livello globale esistono molteplici reti di classe C (anche 2 milioni), per cui la memorizzazione complessiva di tutte queste è eccessiva (2 milioni di entry accedute per ogni singolo pacchetto). A maggior ragione il sistema di suddivisione per classi risulterebbe insostenibile, poiché i router sarebbero molto appesantiti dalla ricerca degli indirizzi su una così vasta tabella.

Per cui per i router sarebbe meglio aggregare le reti, in modo da ridurre le entry della tabella, anche se le reti non fossero aggregabili fisicamente.

Ad esempio, nella figura accanto si ha un router R1 collegato alla rete 200.7.0.0 con subnet mask 255.255.255.0 e alla 200.7.1.0 con stessa subnet mask, che possono essere due reti qualsiasi (della stessa azienda o meno). Si supponga di dover inviare un messaggio alla rete 200.7.0.0 (oppure l’altra, la tabella è uguale) con subnet mask 255.255.255.0 e con next hop di indirizzo A. Nonostante le due reti siano distinte, per il router, la configurazione della tabella è identica (o comunque simile).



Finché le reti sono solo due non vi sono molti problemi, tuttavia se le reti fossero ad esempio due mila, si avrebbero due mila entry identiche nella tabella. Per cui una soluzione è quella di aggregare le reti sotto all’interfaccia B. Si supponga di avere altre due reti: 200.7.2.0 e 200.7.3.0 con stesse subnet mask di

prima. Il router R2 a questo punto vedrebbe al next hop R1 la rete aggregata appena descritta e un'altra di indirizzo 200.7.2.0 con subnet mask 255.255.254.0 al next hop C. Nella tabella di instradamento di R3, invece, si ha 200.7.0.0 con subnet mask 255.255.252.0 con next hop D. Questo perché le reti sono state collassate in una sola, in modo da ridurre lo spazio di memoria anche se a discapito di un'assegnazione in maniera gerarchica. [L'immagine potrebbe non essere come quella rappresentata alla lavagna a lezione a causa di una limitata visione dalla webcam]

Per questo motivo risulta conveniente raggruppare le reti e abbandonare completamente il concetto di classi. Ci pensano le subnet mask a dire quanti bit sono dedicati alla rete e quanti agli host. Per cui si può parlare di *Classless Inter-Domain Routing* (CIDR), che cerca di bilanciare il desiderio di ridurre al minimo il numero di rotte che un router deve conoscere con la necessità di distribuire gli indirizzi in modo efficiente. Per questo motivo il CIDR utilizza percorsi aggregati, in particolare utilizza una singola voce nella tabella di inoltro per indicare al router come raggiungere molte reti diverse.

Ad esempio si consideri un AS con 16 numeri di rete di classe C. Invece di distribuire i 16 indirizzi a caso, si distribuiscono in un blocco di indirizzi di classe C contigui. Si supponga di assegnare i numeri di rete di classe C da 192.4.16.0 a 192.4.31.0. Si può osservare che i primi 20 bit di tutti gli indirizzi in questo intervallo sono uguali (11000000 00000100 0001). Dunque si ha appena creato un numero di rete a 20 bit (che si colloca tra il numero di rete di classe B e quello di classe C).

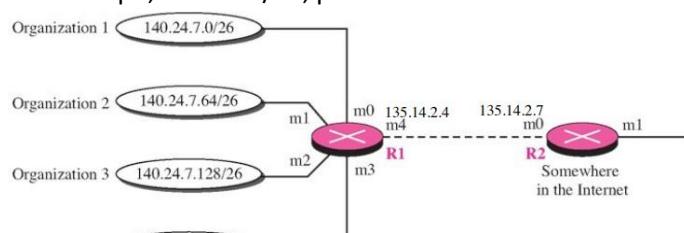
A questo punto si richiede di distribuire blocchi di indirizzi di classe C che condividono un prefisso comune. La convenzione prevede di inserire un /X dopo il prefisso, dove X è la lunghezza del prefisso in bit. Quindi, ad esempio, il prefisso a 20 bit per tutte le reti da 192.4.16.0 a 192.4.31.0 è rappresentato come 192.4.16/20. Al contrario, se si volesse rappresentare un singolo numero di rete di classe C, lungo 24 bit, lo si scriverebbe come 192.4.16.0/24 (o semplicemente 192.4.16).

(Nell'esempio precedente le reti sarebbero del tipo 200.7.0.0/24)

Nei server, ad esempio, per vedere le tabelle di routing si scrive su terminare di comando `route`, mentre per vedere nello specifico le interfacce si digita `ipconfig`, eventualmente specificando anche quale.

Per quanto riguarda come si comportano i protocolli di routing con gli indirizzi senza classe, essi devono capire che il numero di rete può essere di qualsiasi lunghezza, per cui si rappresenta il numero di rete con una singola coppia <lunghezza, valore>. Tutti i router devono comprendere l'indirizzamento CIDR e il meccanismo di inoltro IP presuppone che sia in grado di trovare il numero di rete in un pacchetto e quindi di cercarlo nella tabella di inoltro. Questo presupposto deve essere modificato nel caso di CIDR, in quanto con CIDR significa che i prefissi possono essere di qualsiasi lunghezza, da 2 a 32 bit.

Di seguito un altro esempio aggregazione di instradamenti, in cui il router R1 è collegato a quattro sottoreti di diverso tipo, tutte da /26, per cui la loro subnet mask è di 255.255.255.192 (in quanto fino ai tre 255 si ha /24 a cui si sommano 2 bit per cui 128+64).



Mask	Network address	Next-hop address	Interface
/26	140.24.7.0	-----	m0
/26	140.24.7.64	-----	m1
/26	140.24.7.128	-----	m2
/26	140.24.7.192	-----	m3
/0	0.0.0.0	135.14.2.7	m4

Mask	Network address	Next-hop address	Interface
/24	140.24.7.0	135.14.2.4	m0
/0	0.0.0.0	Default	m1

Routing table for R2

Per quanto riguarda il router R1 ha la tabella completa in quanto deve sapere dove inviare i pacchetti per ogni singolo indirizzo, mentre R2 vede solo una rete aggregata.

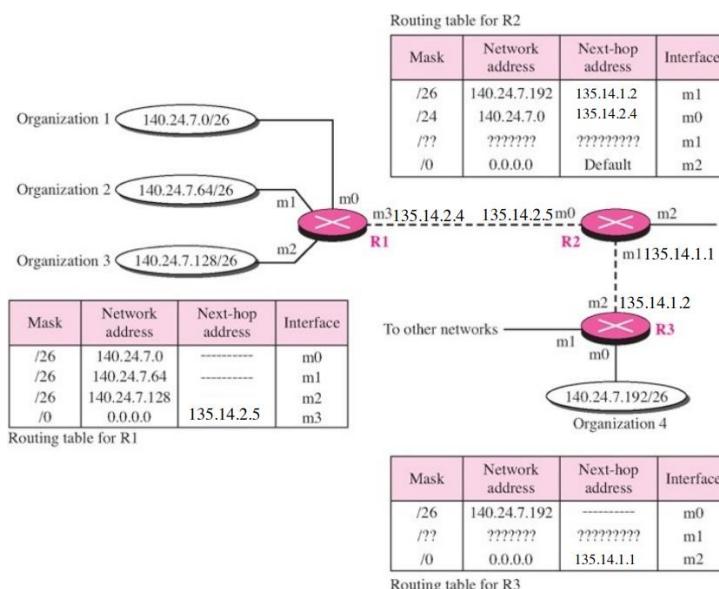
Da notare che i due router hanno degli indirizzi di default, per cui se arriva un messaggio il cui indirizzo IP non corrisponde a nessuno di quelli rilevati, allora si inoltra a quello di default (non è un obbligatorio che il router lo abbia, e neanche NON è da confondere col *destination unreachable*).

IP forwarding rivisitato

Arrivati a questo punto è possibile effettuare una modifica all'algoritmo di forwarding, in particolare è possibile che i prefissi nelle tabelle di inoltro si sovrappongano, come anche che alcuni indirizzi possano corrispondere a più di un prefisso. Ad esempio, nella tabella di inoltro di un router si potrebbe trovare 140.24.7.0/24 e 140.24.7.192/26 in quest'ordine e se un pacchetto è destinato a 140.24.7.200 corrisponde chiaramente a entrambi i prefissi.

La maggior parte dei router sceglie seguendo il principio della "corrispondenza più lunga" (*longest match forwarding*), ovvero la sequenza che corrisponde (*match*) e che ha il prefisso più lungo (140.24.7.192/26 in questo caso). Dunque, i router che adottano questa tecnica prediligono le reti più piccole (e tecnicamente è più probabile che sia quella più precisa/corretta).

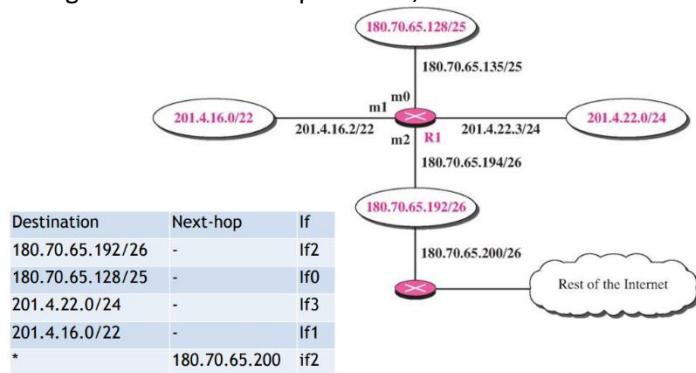
Altri, invece, eseguono una scansione sequenziale della tabella di inoltro e applicano la prima regola di corrispondenza. In questo caso, un pacchetto destinato a 140.24.7.200 corrisponderebbe a 140.24.7.0/24 e non a 140.24.7.192/26.



Affianco lo stesso esempio appena sopra, visto con la variante della longest match forwarding. In questo caso si hanno tre router. Il primo (R1) ha tutti gli indirizzi elencati e un default al router R2, come anche il router R3 ha l'indirizzo del R2 e il default. Quello più complesso è il caso del router R2, in quanto vede una situazione particolare: esso potrebbe elencare tutte le destinazioni che vede (es.: 140.24.7.0/26, 140.24.7.64/26, ecc.), tuttavia la sua tabella diventerebbe troppo lunga, per cui se le quattro reti venissero aggregate, ottenendo una rete di indirizzo 140.24.7.0/24, ci si ritroverebbe con un problema se ad esempio arrivasse da R1

un pacchetto con indirizzo 140.24.7.200 perché R1 applicando la regola entrerebbe nel default, quindi inoltra in R2, quest'ultimo applica la regola e vede che il match è sulla porta *m0* (poiché non si possono mettere due o più porte dunque si sceglie questa, per esempio) per cui inoltra a R1 e così in loop. Per cui questa strategia non può funzionare e R2 è costretto a cambiare la sua politica di controllo e applica quindi il longest match forwarding. A questo punto può aggregare tutte le reti di R1 sotto all'indirizzo di prima 140.24.7.0/24 e mantenere quello di R3 come sarebbe normalmente, quindi 140.24.7.192/26. Se ad esempio R1 inviasse il pacchetto di prima di indirizzo 140.24.7.200, esso arriverebbe a destinazione correttamente in R3 poiché la longest match forwarding dà priorità alla regola del prefisso più grande (più preciso).

Di seguito un altro esempio simile, in cui si hanno due reti del tipo 180.70.65.192/26 e 180.70.65.128/25, in



cui la prima è sottorete dell'altra. In questo caso non è possibile applicare la longest match forwarding, in quanto i pacchetti verrebbero sempre inoltrati verso l'esterno in quanto indirizzo 180.70.65.192/26 ha un prefisso più grande, quindi gli host nelle reti superiori resterebbero esclusi. In questo caso R1 deve applicare come regola quella di controllare se il match su 180.70.65.192/26 funziona e in caso negativo controlla le altre reti (oppure default).

Esercizi:

(Scheda del 2020-01-27):

- 7) Si considerino le seguenti reti: (A) 192.168.4.0/22; (B) 192.168.4.0/23; (C) 192.168.3.0/24. (a) A quali di queste reti appartiene l'indirizzo 192.168.6.10? (b) Quali di queste reti sono sottorete di 192.168.0.0/22? (c) Quanti indirizzi utili contiene la rete A?

R: (a) si applica la subnet mask e si cerca di capire in quale vi è una corrispondenza, quindi nel caso di (A) si prendono i primi 22 bit e si vede se si ottiene l'indirizzo richiesto, in caso negativo si prende (B) e si effettua lo stesso controllo ecc. In questo caso la corrispondenza si ha solo per l'indirizzo (A).

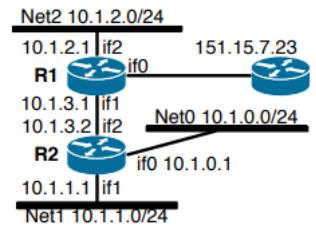
(b) In questo caso si deve provare a togliere/considerare solo i primi 22 bit di rete e vedere quali di essi dà 192.168.0.0 (tramite confronto binario). In questo caso è solo la C.

(c) Si prendono i 32 bit totali della rete e vi si tolgono i 22 della sottorete e i 2 bit di indirizzo rete e quello di broadcast, ottenendo $2^{32-22} - 2 = 1022$.

(Scheda del 2020-06-22):

- 7) Si dia la tabella di inoltro del router R2 della rete in figura:

Net/CIDR	if	next hop	ma si può omettere una entry:
10.1.0.0/24	if0	-	Network/CIDR
10.1.1.0/24	if1	-	10.1.0.0/24
10.1.2.0/24	if2	10.1.3.1	10.1.1.0/24
10.1.3.0/24	if2	-	10.1.3.0/24
/	if2	10.1.3.1	*/*



Da notare che il 151.15.7.23 è considerato come un gateway globale. Le reti sull'interfaccia if0 e if1 possono essere gestite direttamente in quanto sono collegate direttamente con R2, mentre quelle sopra possono essere gestite da R1 per cui si configura con 10.1.2.0/24. Infine, c'è la rete tra i due router R1 e R2 per cui si assegna anche l'indirizzo della rete 10.1.3.0/24 (anche se non sembra essere una rete legittima). In realtà il default può inglobare anche la rete 10.1.2.0/24 e R2 può continuare a funzionare correttamente.

(Scheda del 2020-09-07):

- 7) Si considerino le seguenti reti: A: 184.85.16.0/22; B: 184.85.16.0/24; C: 184.85.16.0/26. (a) Quale è la rete con il maggior numero di indirizzi disponibile? (b) A quale/i di queste reti appartiene l'indirizzo 184.85.19.15? (c) E l'indirizzo 184.85.16.200?

R: (a) La risposta è molto veloce, ovvero la A, poiché ha il numero più piccolo.

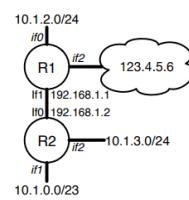
(b) Si deve guardare i primi n bit e li si confronta con la rete dell'indirizzo dato. In questo caso il match avviene su A soltanto.

(c) Dato che la trasformazione in binario comprende due reti, esse sono A e B.

(Scheda del 2020-07-20):

- 7) I router in figura hanno le tabelle qui sotto, e applicano le regole di inoltro in ordine. (a) Le tabelle sono corrette? ossia, gli host di tutte le reti sono raggiungibili? (b) Se no, si indichi la/e regola/e errata/e, e si suggerisca una possibile correzione.

R1:	Net/CIDR	if	next hop	R2:	Net/CIDR	if	next hop
	10.1.0.0/22	if1	192.168.1.2		10.1.0.0/23	if1	-
	10.1.2.0/24	if0	-		10.1.3.0/24	if2	10.1.3.1
	192.168.1.0/24	if1	-		192.168.1.0/24	if0	-
	/	if2	10.1.3.1		*/*	if0	192.168.1.1



R: (a) No, la 10.1.2.0/24 è sottorete di 10.1.0.0/22, per cui non è raggiungibile con la tecnica del controllo in ordine perché la prima regola di R1 susssume anche quella rete ed è errato. (Inoltre il default gateway di R1 deve essere 123.4.5.6, e in R2 il next hop per 10.1.3.0/24 non ci deve essere perché è una consegna diretta). (b) Le tabelle corrette sono:

R1:	Net/CIDR	if	next hop	R2:	Net/CIDR	if	next hop	
	10.1.2.0/24	if0	-		10.1.0.0/23	if1	-	Per cui basta invertire l'ordine degli indirizzi di rete
	10.1.0.0/22	if1	192.168.1.2		10.1.3.0/24	if2	-	degli indirizzi di rete
	192.168.1.0/24	if1	-		192.168.1.0/24	if0	-	10.1.2.0/24 e 10.1.0.0/22 per
*	*	if2	123.4.5.6	*	*	if0	192.168.1.1	ottenere un forwarding corretto. R2 è corretto.

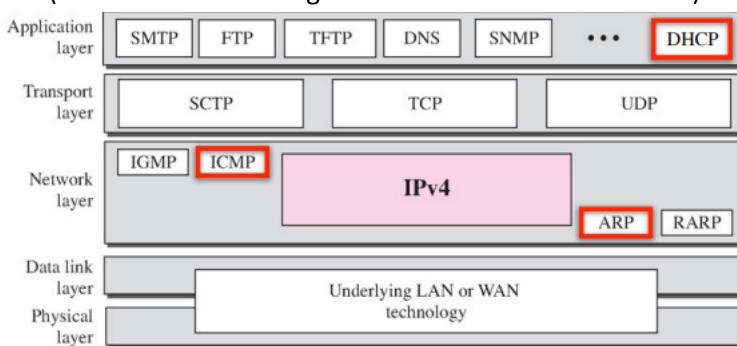
Indirizzi IP per usi speciali (RFC 5735)

- 0.0.0.0/8: indirizzo non instradabile, non valido. Utilizzato per indicare “qualsiasi indirizzo IP”
- 10.0.0.0/8: utilizzato per le reti private, quindi non instradabile su Internet, ovvero si usa per reti private scollegate dal mondo (va bene per test)
- 127.0.0.0/8: *loopback* (cioè “me stesso”)
- 169.254.0.0/16: autoconfigurazione su un collegamento locale quando il DHCP non è disponibile (e/o quando non si sa che indirizzo usare e quindi si spara un numero a caso in questo range)
- 172.16.0.0/12, 192.168.0.0/16: utilizzati per reti private, non instradabili su Internet ([come 10.0.0.0](#))
- 224.0.0.0/4: Classe D: Indirizzi multicast
- 240.0.0.0/4: Classe E: riservato per usi futuri (mai utilizzato...)

Protocolli IP ausiliari (ancillary)

Prima di affrontare nuove tematiche e problematiche che riguardano i pacchetti, è buona norma analizzare l'ecosistema di IP, ovvero capire come funzionano i protocolli ausiliari (*ancillary*). Essi sono protocolli di supporto/aiuto di IP e che servono a farlo funzionare (senza di questi non si riuscirebbe nemmeno a inviare i pacchetti).

Esistono di tipi diversi, ad esempio ARP, ICMP, DHCP, ecc. e alcuni di essi risiedono nel livello 3 dello stack, altri invece (es.: DHCP) stanno più in alto. Quelli del livello 3 sono collocati lì perché usufruiscono direttamente di servizi di livello sottostante oppure perché sfruttano il protocollo IP, ma sono servizi che non appartengono direttamente ai livelli che stanno più in alto nello stack (es.: ICMP). Il DHCP è un protocollo livello applicativo, però che in realtà serve a risolvere i problemi di assegnazione degli indirizzi IP. Per cui questi protocolli ausiliari fanno sì che IP possa funzionare nella sua interezza e in maniera efficace e gestiscono ciò che IP non sarebbe in grado di gestire da solo (es.: creazione e configurazione delle tabelle di inoltro).



Protocollo di traduzione degli indirizzi (ARP)

Per quanto riguarda l'inoltro dei pacchetti, il router deve sapere dove inviarli, indipendentemente dalla tecnologia utilizzata e sfruttando i link a esso collegati. Esso deve saper associare ad esempio un indirizzo Ethernet (livello 2) a un indirizzo IP (livello 3), semplicemente riconoscendo gli indirizzi all'interno dei frame. Per cui mappa gli indirizzi IP logici in indirizzi fisici e li inserisce in una tabella ARP (*Address Resolution Protocol* = protocollo di risoluzione degli indirizzi).

In quella tabella saranno presenti gli indirizzi logici IP e quelli fisici MAC a cui fanno riferimento. In realtà, non c'è alcuna relazione tra MAC address e indirizzo IP, quindi un router deve avere una riga della tabella per ogni host a esso collegato (direttamente) per sapere dove inviare i pacchetti. Gli indirizzi memorizzati possono essere di un host di destinazione oppure di un router intermedio (next hop).

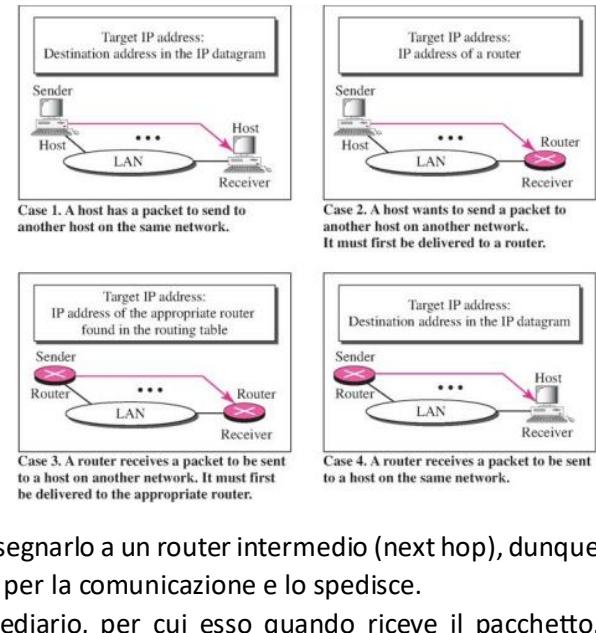
Infatti, si hanno quattro scenari possibili quando si inoltra e/o riceve pacchetti (figura a lato), in cui la comunicazione avviene: tra host e host, tra host e router, tra router e router oppure tra router e host.

Nel caso 1, l'indirizzo IP viene incapsulato in un frame Ethernet che ha come MAC address quello del destinatario vero e proprio e poi viene spedito sulla rete locale e si occupano gli switch di farlo arrivare a destinazione.

Nel caso 2, invece, il pacchetto non è di un host della rete locale, per cui si deve utilizzare almeno un router intermedio e quindi si utilizzerà l'indirizzo IP del gateway per fare in modo che il messaggio esca dalla rete locale. Dunque, il pacchetto IP dovrà essere incapsulato in un frame il cui MAC address è quello dell'interfaccia del gateway.

Nel caso 3, un router riceve un pacchetto e deve controllare dove inviarlo e tramite la sua tabella di inoltro sa che per avvicinare il messaggio al destinatario deve consegnarlo a un router intermedio (next hop), dunque delega a esso la consegna. Per cui crea un frame apposito per la comunicazione e lo spedisce.

Nel caso 4, il messaggio arriva all'ultimo router intermedio, per cui esso quando riceve il pacchetto, controlla la propria tabella e si accorge che il destinatario è nella sua sottorete, per cui glielo consegna direttamente.

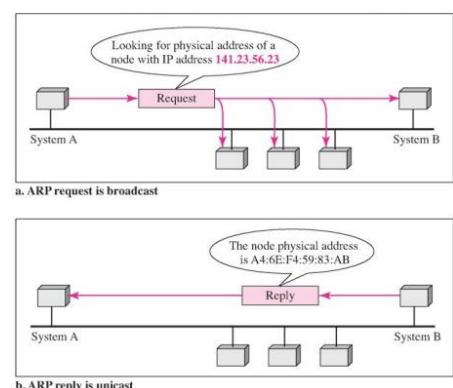


Quando viene consegnato un pacchetto, al suo interno vi è un indirizzo MAC di destinazione, per cui il modulo IP che sta al livello 3 deve sapere su quale interfaccia inoltrare il pacchetto all'interno della rete, quindi guarda il MAC address nella sua tabella.

Logic Addr	Physical Address
192.44.80.1	FF:6E:A0:13:6C:A4
192.44.80.2	67:4A:6D:1A:B4:33
192.44.80.3	1A:27:44:8F:C2:14
192.44.80.4	1F:56:AB:C3:12:34
192.44.80.5	F2:65:29:04:72:60
192.44.80.6	2E:70:0E:A2:53:66
...	...

Le entry della tabella possono essere gestite anche manualmente, ma è scomodo, dunque si fa solo in caso di reti piccole. Infatti di solito si lascia la gestione automatica gestita dal protocollo apposito, ovvero ARP (*Address Resolution Protocol*). Esso si occupa di gestire la tabella dei collegamenti tra indirizzi IP e fisici.

Quando arriva un nuovo pacchetto si controllano le entry della tabella: se l'indirizzo è già mappato si sa dove su quale interfaccia inoltrare, altrimenti se non è presente si prova a chiedere nella rete locale in broadcast. I pacchetti che vengono inoltrati su tutte le interfacce del router sostanzialmente chiedono chi ha un certo indirizzo IP. Il messaggio di richiesta in broadcast contiene nel suo payload un pacchetto ARP che ha l'indirizzo fisico del router mittente (che porge la domanda) e l'indirizzo da cercare. Ogni nodo della rete locale confronta l'indirizzo con quelli che ha in tabella e se non lo possiedono ignorano la richiesta ARP, altrimenti se trovano una corrispondenza rispondono (in unicast direttamente al mittente con un altro ARP). A quel punto il router mittente sa dove inoltrare il pacchetto.



Questa procedura è molto ricorrente, in quanto c'è un timeout (es.: 5 minuti, ma è modificabile dall'utente) in cui se un pacchetto non viene inviato o ricevuto per un certo indirizzo, esso viene eliminato dalla tabella di inoltro, per cui alla comunicazione successiva verso quell'indirizzo si dovrà chiedere chi lo possiede.

Per visualizzare la tabella ARP di un computer, è possibile utilizzare il comando `arp -a` su terminale e subito compare una serie di configurazioni che mostrano gli indirizzi logici e fisici e su quale interfaccia sono collegati (e anche il tipo, es.: interfaccia Ethernet). Un'informazione che non viene esplicitamente fornita è quella della maschera di subnet, che non sempre è intuitibile (es.: per la rete 158.110.97.0 che ha un broadcast di indirizzo 158.110.97.255 si è indotti a pensare che la subnet mask sia 255.255.255.0, ma potrebbe non essere così). Se si esegue più volte il comando dopo qualche minuto, ci si può accorgere del fatto che alcuni indirizzi spariscano: i loro timer sono scaduti e quindi sono stati rimossi dalla tabella.

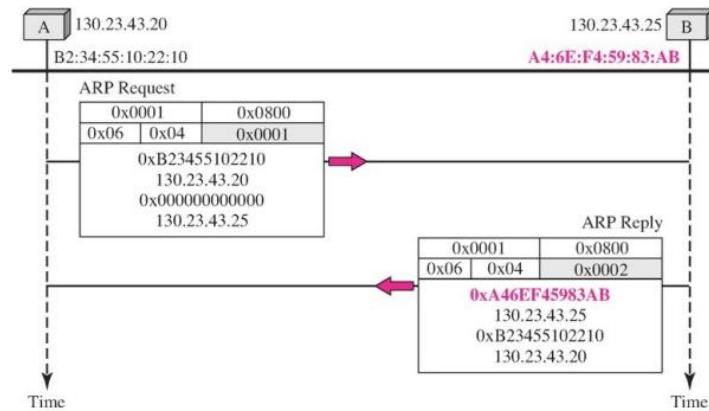
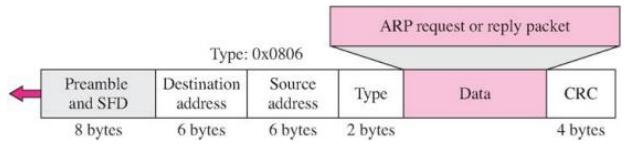
Formato del pacchetto ARP

0	8	16	31		
Hardware type=1	ProtocolType = 0x0800				
HLen=48	PLen=32	Operation			
SourceHardwareAddr (bytes 0-3)					
SourceHardwareAddr (bytes 4-5)	SourceProtocolAddr (bytes 0-1)				
SourceProtocolAddr (bytes 2-3)	TargetHardwareAddr (bytes 0-1)				
TargetHardwareAddr (bytes 2-5)					
TargetProtocolAddr (bytes 0-3)					

Accanto un'immagine dei vari campi di un pacchetto ARP. Con *HardwareType* si intende il tipo di rete fisica (es.: Ethernet), con *ProtocolType* il tipo di protocollo di livello superiore (es.: IP), con *HLEN* e *PLEN* la lunghezza degli indirizzi fisici e di protocollo, mentre con il campo *operation* ci si riferisce alla richiesta o alla risposta. A seguire vi sono una serie di indirizzi fisici/protocollo e sorgente/target. Infine, c'è *TargetHardwareAddr* che è vuoto nella richiesta. È da ricordare il fatto che

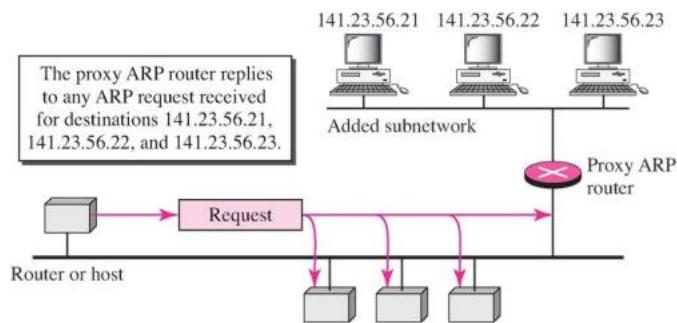
interfacce diverse possono avere tipi, indirizzi e tecnologie differenti. Inoltre, i formati dei pacchetti IPv4 e IPv6 sono un po' diversi tra loro.

I pacchetti ARP non utilizzano IP (ovviamente) e sono incapsulati direttamente nei frame del datalink (es.: Ethernet, qui affianco). Sono composti da preamble, da due indirizzi, dal tipo (in questo caso essendo un ARP ha il type 0x0806), dai dati della richiesta/risposta e dal CRC.



ARP proxy

Un ARP proxy è un host che rappresenta altri host (spesso intere reti) e che risponde con il proprio MAC alle richieste ARP degli host "nascosti". Esso inoltra i frame che riceve alle destinazioni corrette, come uno switch (ma cambiando l'indirizzo MAC).



Configurazioni degli host e DHCP

Precedentemente si è visto il caso degli indirizzi Ethernet che sono configurati in rete dal produttore e sono unici, mentre per quanto riguarda gli indirizzi IP devono essere unici in una determinata rete Internet, ma devono anche riflettere la struttura della rete Internet.

La maggior parte dei sistemi operativi host fornisce un modo per configurare manualmente le informazioni IP per l'host. Tuttavia vi sono degli svantaggi nella configurazione manuale: è necessario molto lavoro per configurare tutti gli host di una rete di grandi dimensioni e il processo di configurazione è soggetto a errori, per cui è necessario un processo di configurazione automatizzato.

Gli indirizzi IP fissi sono vantaggiosi solo nel caso in cui si hanno macchine (es.: server) che devono essere accedute da remoto e/o svolgono un servizio per cui devono sempre essere raggiungibili staticamente. Di solito in una rete si hanno pochi indirizzi con queste necessità. Mentre per quanto riguarda altri dispositivi (es.: telefoni, computer personali, ecc.) è meglio avere un meccanismo in cui anche se ci si sposta da una rete, non si debba aspettare una configurazione manuale per avere un IP, per cui si deve avere un meccanismo non solo automatizzato, ma anche dinamico.

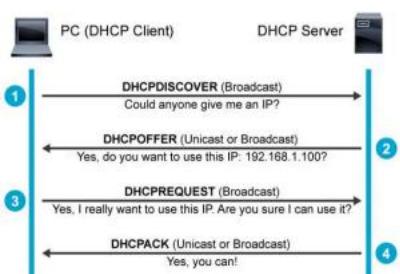
Per fare ciò, una volta si usava il protocollo RARP (*Reverse ARP*), in cui un host dice quale è il suo MAC address e si fa dare un indirizzo IP (quindi l'opposto dell'ARP in cui si chiede chi ha un certo host), oppure il protocollo BOOTP.

Invece, oggi si utilizza un protocollo di livello 7 chiamato DHCP (*Dynamic Host Configuration Protocol*), il quale necessita di un server (almeno uno per un dominio amministrativo) che tenga traccia degli indirizzi IP assegnati al client (*pool di indirizzi*), dunque di un sistema che memorizzi lo stato della rete. Il server DHCP è responsabile della fornitura di informazioni di configurazione agli host e assegna gli indirizzi su richiesta. Dunque quando un host necessita di un indirizzo interroga il server e questo gliene fornisce uno in maniera statica (es.: in base agli indirizzi MAC) oppure dinamica ("lease" limitati nel tempo).

L'opzione di scelta di un indirizzo IP statico basato su MAC address è comodo nelle casistiche di dispositivi come stampanti, server o calcolatori, ecc. in cui si vuole sempre avere lo stesso indirizzo quando sono disponibili e connessi alla rete.

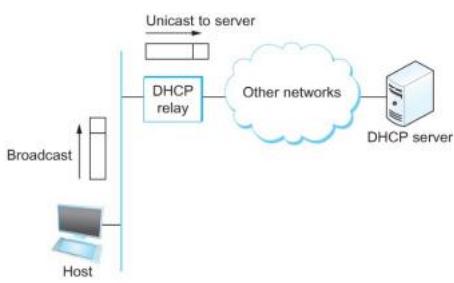
Il meccanismo di assegnazione funziona nel seguente modo. L'host (chiamato *supplicant* in questo caso) appena avviato o collegato invia un messaggio DHCP di DISCOVER con un indirizzo IP di destinazione broadcast, quindi 255.255.255.255 ("broadcast limitato", ignorato dai router), mentre per quanto riguarda l'indirizzo IP di origine è irrilevante. Il messaggio contiene anche l'indirizzo MAC dell'host di origine. Il pacchetto in realtà è una sorta di richiesta di presenza del server DHCP, che potrebbe anche non esserci in quella sottorete.

Se esiste un server DHCP, esso risponde con un pacchetto che contiene l'indirizzo IP offerto e lo trasmette in broadcast o lo consegna direttamente solo all'host di origine (utilizzando il suo MAC, non il suo IP



poiché non ha ancora un IP). Il client poi chiede conferma al server e in caso positivo, il client DHCP imposta l'indirizzo IP locale (e altri parametri, come gateway, subnet mask, router, DNS, ecc.) sull'host.

Se invece non è presente alcun server DHCP, e quindi dopo un certo intervallo di tempo la richiesta del client scade, esso si inventa un indirizzo IP della famiglia/classe 169.254.0.0/16 (indirizzo speciale).



Un server DHCP può servire più reti contemporaneamente, anche se non direttamente collegate. Per ogni rete è necessario un agente di relay DHCP che ascolti le richieste DHCP. Gli agenti di relay DHCP non assegnano gli indirizzi; semplicemente inviano i messaggi al server DHCP e attendono la risposta, da inviare nelle reti locali.

Il *lease* (assegnazione dell'IP) dell'indirizzo avviene dopo un certo periodo di tempo, che può essere scelto dall'amministratore di sistema. Tuttavia se un cliente ha bisogno di più tempo deve rinnovare il tempo mandando una richiesta al server DHCP e può non essere accettata.

Come già detto, il server DHCP deve mantenere lo stato della rete (anche molto vasta) e ricordarsi a quali host ha dato certi indirizzi IP. Si supponga che questo si spenga (o venga resettato) e poi venga riaccesso. A quel punto non ricordandosi a chi aveva assegnato cosa e con quale timer di scadenza, potrebbe essere che due o più host si trovino con indirizzi uguali. A quel punto succede un disastro (e può accedere realmente!).

Protocollo di segnalazione e gestione della rete (ICMP)

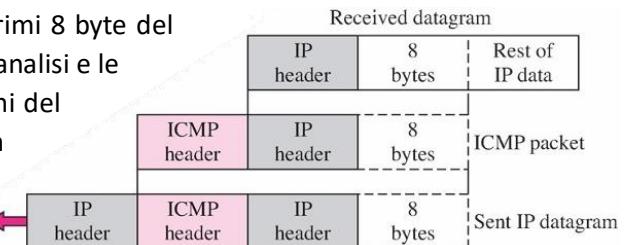
Il protocollo ICMP (*Internet Control Message Protocol*), definisce un insieme di messaggi di errore e di notifica che vengono rispediti all'host di origine ogni volta che:

- un router o un host non è in grado di elaborare un datagramma IP con successo (es.: host di destinazione irraggiungibile per un guasto del collegamento/nodo, processo di riassemblaggio fallito magari per colpa dei timeout scaduti, TTL raggiunto a 0 e quindi il datagramma è stato scartato, il checksum dell'intestazione IP è fallito, ecc.);
- oppure quando esiste un percorso migliore per quella destinazione (es.: dal router all'host di origine, chiamato *ICMP-Redirect*).

In tutti i casi, viene notificato solo l'host sorgente.

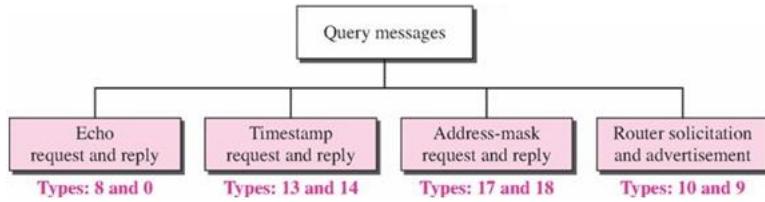
Tutti questi sono in realtà delle sorte di messaggi di diagnostiche e di controllo e verifica del sistema/rete. Essi non trasportano né dati, né informazioni di configurazione, ma solamente dati di feedback della rete (es.: sullo stato, sul funzionamento, ecc.). In realtà ICMP è sopra IP, ovvero è incapsulato al suo interno.

La parte iniziale del datagramma rifiutato (header + primi 8 byte del pacchetto) è inclusa nel messaggio ICMP, per facilitare l'analisi e le contromisure. Quindi, si prendono le prime informazioni del messaggio da scartare e le si inseriscono come dati di un pacchetto ICMP, a sua volta incapsulato in un pacchetto IP. Questo lavoro può essere svolto da un router intermedio qualsiasi.



Tipi di messaggi di richiesta ICMP

Esistono diversi tipi di messaggi di richiesta ICMP, tra cui *echo* ("ping"), che verifica la raggiungibilità di un host, oppure il *timestamp* per misurare il tempo di andata e ritorno (RTT), il *mask discovery* e il *router discovery* per scoprire quale maschera di subnet e quale router vi sono (in realtà sono implementati da DHCP).



Ping:

Servizio maggiormente utilizzato. Invia diverse richieste ICMP *Echo* e calcola il tempo di andata e ritorno (stima), quindi il parametro RTT (*Round Trip Time*). È molto utile per capire se un host è raggiungibile. Di seguito una sua variante.

```

└── ping www.google.com
PING www.google.com (216.58.198.36): 56 data bytes
64 bytes from 216.58.198.36: icmp_seq=0 ttl=54 time=25.934 ms
64 bytes from 216.58.198.36: icmp_seq=1 ttl=54 time=46.239 ms
64 bytes from 216.58.198.36: icmp_seq=2 ttl=54 time=25.844 ms
64 bytes from 216.58.198.36: icmp_seq=3 ttl=54 time=25.701 ms
64 bytes from 216.58.198.36: icmp_seq=4 ttl=54 time=27.354 ms
64 bytes from 216.58.198.36: icmp_seq=5 ttl=54 time=46.450 ms
64 bytes from 216.58.198.36: icmp_seq=6 ttl=54 time=29.715 ms
64 bytes from 216.58.198.36: icmp_seq=7 ttl=54 time=26.358 ms
64 bytes from 216.58.198.36: icmp_seq=8 ttl=54 time=26.286 ms
64 bytes from 216.58.198.36: icmp_seq=9 ttl=54 time=26.120 ms
64 bytes from 216.58.198.36: icmp_seq=10 ttl=54 time=27.245 ms
^C
--- www.google.com ping statistics ---
11 packets transmitted, 11 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 25.701/30.288/46.450/7.646 ms

```

Traceroute:

Serve per scoprire il percorso che fanno i pacchetti verso un host (o comunque uno dei possibili percorsi). Utilizza i messaggi di errore ICMP e il TTL, in particolare:

- Invia un primo messaggio con TTL=1: il primo router lo droppa e invia al mittente un errore ICMP, in questo modo il mittente scopre l'indirizzo del primo router.
- Invia un secondo messaggio con TTL=2: il primo router decrements il TTL e lo inoltra al router successivo, mentre il secondo router lo elimina e invia al mittente un errore ICMP, così il mittente scopre l'indirizzo del secondo router
- ecc.

```

prompt> traceroute fhda.edu
traceroute to fhda.edu (153.18.8.1), 30 hops max, 38 byte packets
1Dcore.fhda.edu      (153.18.31.254)  0.995 ms 0.899 ms 0.878 ms
2Dbackup.fhda.edu    (153.18.251.4)   1.039 ms 1.064 ms 1.083 ms
3iptoe.fhda.edu      (153.18.8.1)     1.797 ms 1.642 ms 1.757 ms

```

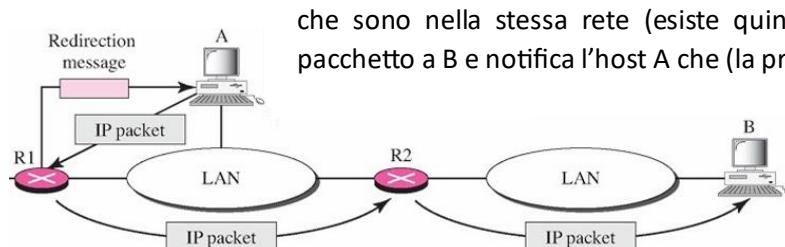
In realtà questo servizio non è affidabilissimo, perché per natura propria delle reti di commutazione a pacchetto, i messaggi inviati possono non fare sempre lo stesso percorso (in realtà nemmeno quelli della stessa serie). Per cui la strada descritta dall'invio dei ICMP del protocollo potrebbe non essere attendibile al 100%, ma come ipotesi va più che bene. Infatti, talvolta si può perdere per strada qualche pacchetto.

Questa tecnica non funziona per l'host di destinazione, infatti anche se TTL=0 il pacchetto è arrivato, ma non viene inviato alcun errore ICMP. Invece per quanto riguarda l'ultimo passaggio, viene inviato un pacchetto UDP sulla porta 1, che non è valida, quindi genera un errore ICMP di "porta non raggiungibile".

Esiste un easteregg scrivendo su terminale traceroute -m 50 bad.horse, in cui i sistemisti della rete hanno preso un certo numero di macchine virtuali e le hanno usate come indirizzi DNS per scrivere come percorso il lyrics della canzone "Bad Horse".

ICMP-Redirect:

Questo non è un errore, ma è un utilizzo di ICMP come informativa utile. La situazione è la seguente: si ha ad esempio una rete in cui vi è un host A su una rete locale con due router e si vuole inviare un pacchetto all'host B. Guardando l'immagine sotto viene da pensare di passare per R2 per inviare il pacchetto a B, tuttavia A potrebbe essere configurato con R1 come gateway. R1 quando riceve un pacchetto da A verso B, si accorge



che sono nella stessa rete (esiste quindi un percorso migliore), per cui invia il pacchetto a B e notifica l'host A che (la prossima volta) può inviarlo direttamente lui, senza passare per R1. Di conseguenza A riconfigura le proprie tabelle di inoltro (quindi adesso per inviare a B ci sarà un riferimento nella tabella che dice di passare per R2).

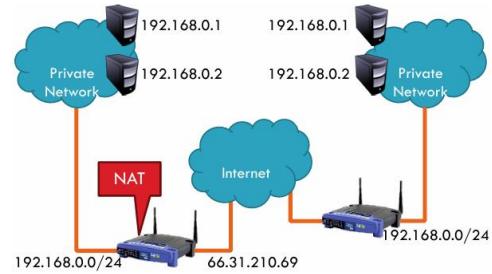
Reti IP private e NAT

La tecnologia dei protocolli IP (ping, traceroute, ecc.) non si utilizza soltanto per le reti connesse a Internet: si possono anche costruire reti locali (anche staccate da Internet). Questa tecnica è molto usata ad esempio nel contesto di impianti industriali o dei trasporti (es.: non è possibile “pingare” lo scambio di un certo treno).

Dunque l’idea è quella di creare una serie di IP privati separati dal resto della rete per il routing interno. Per fare questo si utilizza un router speciale per fare da ponte tra la LAN e la WAN. Come visto precedentemente, per questi scopi si devono adoperare indirizzi speciali che sono privati, non unici globalmente e di solito vengono presi da intervalli IP non instradabili. Dei tipi di intervalli IP privati possono essere:

- 10.0.0.0/8, ovvero 10.0.0.1 - 10.255.255.255
- 172.16.0.0/12, ovvero 172.16.0.1 - 172.31.255.255
- 192.168.0.0/16, ovvero 192.168.0.1 - 192.168.255.255

Il vantaggio di usare IP privati è che è possibile slegarsi dalle rigide regole degli IP pubblici (es.: gerarchia, assegnamento, ecc.), non serve fare richieste a nessuno (sono private!), si può riciclare degli indirizzi limitati, ecc. Infatti, così facendo si hanno molteplici duplicazioni degli stessi indirizzi a livello globale, per cui a questo punto si deve introdurre un meccanismo di riconoscimento delle varie reti e di instradamento dei pacchetti.



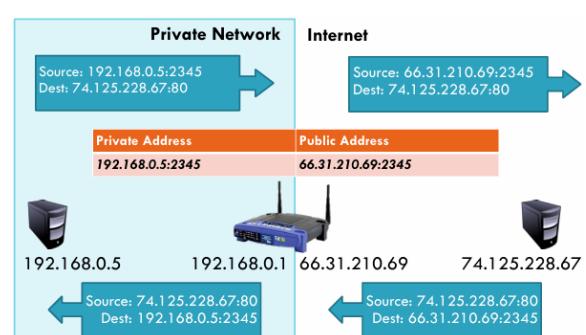
Quando ci si connette a una rete, il server DHCP assegna al dispositivo un indirizzo IP univoco all’interno della sua sottorete (non importa che non lo sia al livello globale). Idem accade con la telefonia, in cui il provider telefonico assegna un indirizzo IP in un certo range privato al dispositivo mobile.

I pacchetti che provengono da questi host, non essendo più con indirizzi univoci, non possono essere instradati nella rete perché sono ambigui. I router normali delle reti Internet, quando vedono passare dei pacchetti con indirizzi mittente e destinatario con IP compresi nei range specificati sopra, li scartano.

Per cui si usa un dispositivo di traduzione degli indirizzi, ovvero il NAT (*Network Address Translation*), che viene implementato da un router/dispositivo speciale di livello 3, situato al confine di una rete privata, e che trasforma gli indirizzi di un range privato in indirizzi di un range pubblico cambiando anche le intestazioni dei pacchetti inviati. La connettività fornita non è continua e il router NAT può anche sostituire i numeri di porta TCP/UDP (*Port Mapping*). Inoltre, esso deve mantenere una tabella dei flussi attivi: i pacchetti in uscita inizializzano una entry della tabella, mentre quelli in arrivo vengono riscritti in base alla tabella.

Funzionamento NAT di base (“cono pieno” = “full cone”)

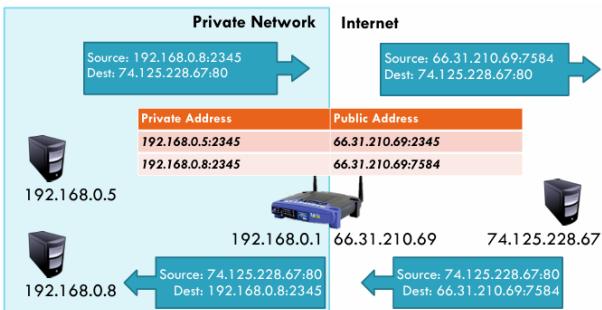
Ad esempio, si supponga di avere una rete privata comunicante con una rete Internet tramite un router. Un host della rete privata vuole inviare un messaggio a un server connesso sulla rete pubblica. Il pacchetto che esso invia al suo gateway (es.: 192.168.0.1) contiene il vero indirizzo di destinazione e il suo vero indirizzo IP (del mittente). Il router a questo punto sostituisce l’indirizzo privato con quello pubblico, lo segna su una sua tabella interna e spedisce un nuovo pacchetto col contenuto di quello precedente, ma con una nuova intestazione con l’indirizzo mappato (quello del router). Il messaggio poi arriva al server che risponde inviando un nuovo pacchetto col suo indirizzo sul campo *source* (mittente) e con indirizzo di destinazione quello del router che gli ha inviato il pacchetto (eventualmente con un



numero di porta diverso). Quando il messaggio arriva al router, esso guarda la sua tabella di inoltro e crea un nuovo pacchetto con l'indirizzo di destinazione dell'host mittente di prima e che si era segnato sulla tabella e lo invia. A quel punto l'host della rete privata riceve la risposta del server.

Funzionamento NAT di base (“cono pieno” con PAT)

Nel caso in cui vi fosse più di un host che vuole utilizzare la stessa porta sorgente (intesa la propria porta), il router mapperà su porte differenti i diversi indirizzi IP (con ‘porte differenti’ si intendono le porte del router).



Il meccanismo di inoltro e scambio degli indirizzi privati/pubblici è identico a prima, tranne per il fatto che quando i pacchetti tornano indietro, il router dovrà cambiare non solo l'indirizzo del mittente, ma anche la porta verso cui inoltrare il datagramma. Questo nuovo meccanismo si chiama NAT PAT (*Port Address Translation*), molto spesso si chiama NAT tutto quanto.

Applicazioni dei NAT

Il NAT consente la condivisione di un IP pubblico tra più indirizzi privati, in questo modo si possono avere pochi indirizzi pubblici unici e tanti indiritti privati ripetibili.

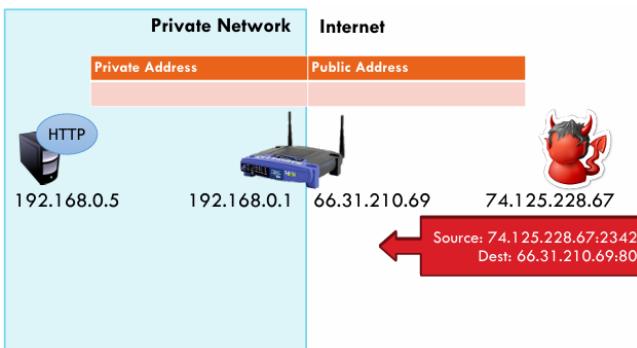
Inoltre, consente la migrazione tra ISP, ad esempio se si ha una rete con un certo numero di indirizzi privati, si è indipendenti dagli indirizzi pubblici, quindi si può configurare tutta la rete con indirizzi privati (non importa chi degli host è un indirizzo pubblico) e se si cambia provider, non serve riconfigurare tutti gli indirizzi in quanto restano quelli, a eccezione fatta per il router (indirizzo pubblico).

Infine, si può usare il NAT per fare bilanciamento del carico (*load balancing*), ovvero quando arrivano le richieste, un router potrebbe forwardarle a più di un server. Esternamente appare come un unico indirizzo, tuttavia nella rete c'è questo dispositivo *load balancer* che guarda lo stato del pool dei server e trasforma la richiesta in arrivo in una richiesta per il server più “scarico”, ovvero meno oberato. La risposta viene poi inviata normalmente come se fosse stato un unico indirizzo (quello che il mittente si aspettava).



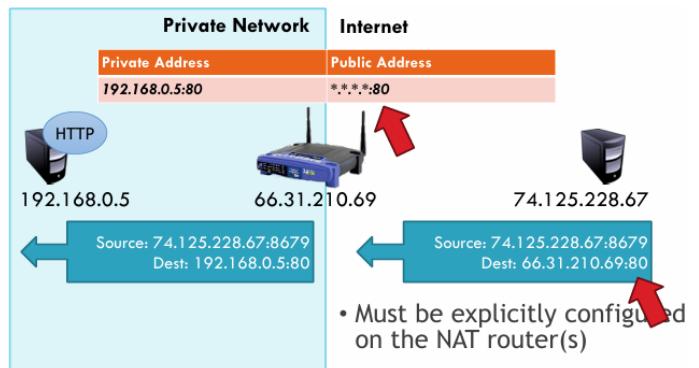
“Firewall” naturale e port forwarding

Il NAT viene anche utilizzato per motivi di sicurezza, o meglio per limitare la visibilità degli host, anche se non è una grande misura di sicurezza. Infatti, un host dietro a un NAT non è che non viene visto, il fatto è che il NAT non fa passare indirizzi da pubblico a host privato (i pacchetti con indirizzo di destinazione di un host privato vengono scartati).



Tuttavia un host pubblico può spedire un pacchetto a un router a cavallo di una rete privata. Il router riconosce il messaggio, cerca il relativo indirizzo privato nella tabella e non lo trova: il pacchetto viene scartato. Questo perché la tabella interna non ha registrato alcun passaggio da indirizzo privato a uno pubblico che corrisponde a quella porta. Infatti il NAT mappa solamente da privato verso il pubblico e non viceversa.

Tuttavia è possibile configurare (esplicitamente) il router in maniera tale che conceda l'accesso a pacchetti che arrivano a una certa porta per un certo indirizzo, anche se arrivano da un indirizzo pubblico verso uno privato. Questa tecnica si chiama *port forwarding* e fa in modo che nella tabella interna al router compaiano delle entry che consentano l'accesso a un certo indirizzo privato configurato esplicitamente. Questo meccanismo però crea dei buchi di sicurezza in quanto apre l'accesso all'esterno, anche se in maniera limitata (es.: osservando il traffico di pacchetti è possibile vedere quale indirizzo di una certa macchina è uscito dalla rete e simulando un pacchetto che esce fuori da dentro alla rete e che usa come IP l'indirizzo di una terza macchina interna, questo finisce nel NAT e lo configura aggiungendo una entry in maniera da potersi poi garantire l'accesso).



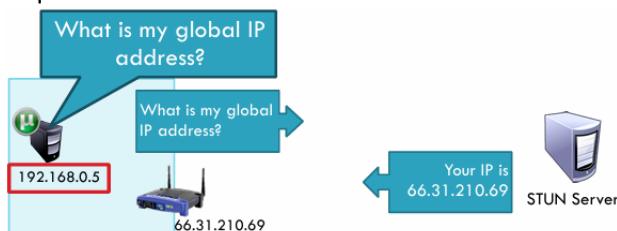
Hole Punching (STUN)

Esiste un altro problema: come abilitare la connettività attraverso i NAT? Se due reti hanno il NAT, si escludono l'accesso a vicenda, poiché i pacchetti verrebbero riconosciuti come locali oppure si rischia che non abbiano il corretto accesso dall'esterno verso l'interno. Una possibile soluzione per la comunicazione potrebbe essere quella di utilizzare l'indirizzo IP del router esterno alla rete NAT destinataria e quest'ultimo tramite port forwarding conceda l'accesso a qualche macchina locale. Quindi almeno una delle due reti che faccia il port forwarding, tuttavia non è sempre garantita come cosa (più che altro per sicurezza interna).

Altrimenti c'è un'altra soluzione chiamata *hole punching*, ovvero una tecnica che consiste nel creare una sorta di buco nel firewall del NAT. Essa ha possiede due protocolli: lo STUN e il TURN.

Lo STUN (*Session Traversal Utilities for NAT* = utilità di attraversamento della sessione per NAT) risiede in un server esterno alla rete e serve per scoprire il proprio indirizzo IP pubblico (perché le macchine dentro a una rete locale non sanno con quale indirizzo IP possono uscire). In una rete vi possono essere più NAT annidati (es.: in un'azienda ci potrebbe essere un NAT per ufficio, poi uno complessivo dell'azienda, ecc.), per cui può essere che l'indirizzo IP locale venga cambiato più volte, per cui a maggior ragione un host può necessitare di sapere il suo indirizzo IP pubblico. Ad esempio anche i container di Docker creano una sorta di NAT con un range di indirizzi per cui può essere utile sapere l'indirizzo pubblico.

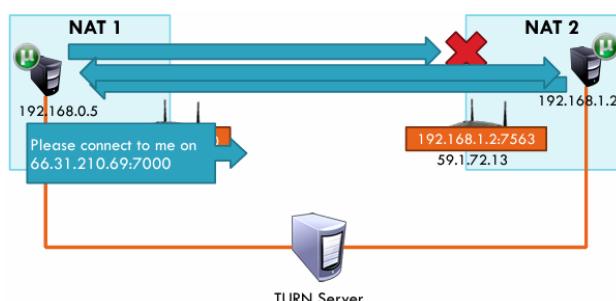
Per funzionare, il server STUN che viene interrogato, sfrutta una terza parte per fare l'eco dell'indirizzo IP globale da cui arriva la richiesta (l'ultimo router). Questa funzionalità è stata creata per le macchine, non per far sapere a un essere umano quale sia il suo IP.



Questo protocollo viene utilizzato anche per sondare i NAT/firewall simmetrici, ovvero quando si è un peer dentro a una NAT simmetrica e l'altra rete con cui si vuole comunicare è pubblica o con port forwarding (o comunque un NAT parziale e non totale, perché altrimenti non si avrebbero le tabelle di inoltro popolate). In questo modo si riesce a sapere il proprio indirizzo IP pubblico e a passarlo a un'altra NAT. Tuttavia nemmeno questo basta per l'intento: se non si ha un port forwarding apposito si rischia che all'invio del pacchetto esso non riesca a passare al di là del router perché non è stata popolata la entry della tabella, per cui per risolvere bisogna avere un altro server, chiamato TURN.

Hole Punching (TURN)

Il server TURN (*Traversal Using Relays around NAT*) fa da forwarder e contemporaneamente consente di popolare un po' le tabelle (gli host che vi si collegano è come se si "registrassero"). Esso è pubblico, per cui è accessibile da tutte le reti. Gli host che comunicano con esso non si conoscono tra di loro, tuttavia magari grazie a uno STUN sono riusciti a scoprire i loro IP pubblici uno dell'altro. I tentativi di comunicazione diretti tra una rete e un'altra, tuttavia, fanno sì che le tabelle dei router si popolino, però il router del destinatario rifiuta la connessione esterna, per cui la rete NAT mittente non riceve la risposta. A questo punto si passa a utilizzare il server TURN, per cui il mittente invia al server una richiesta di inoltrare un pacchetto alla rete dell'host destinatario e questo lo forwarda dall'altra parte, per cui il pacchetto riesce a entrare nella rete NAT del destinatario in quanto prima erano state inviate le richieste al server. Arrivati a questo punto, il destinatario può comunicare direttamente con la rete del mittente in quanto le tabelle dei router sono state popolate correttamente.



Un esempio pratico di questo meccanismo è quello utilizzato dai NAS, per cui un NAS locale diventa accessibile all'esterno (es.: un NAS casalingo accessibile anche da mobile in giro).

Preoccupazioni relative al NAT

Il NAT, tuttavia, crea delle piccole complicazioni, specialmente per quanto riguarda le performance e la scalabilità, per via delle trasformazioni, intestazioni, calcolo dei checksum, ecc. per ogni pacchetto e anche per tenere lo stato del flusso, di conseguenza serve anche memoria.

Inoltre, per stabilire la connessione serve la richiesta degli IP pubblici, la comunicazione con un server, ecc. per cui si aumenta anche il traffico dei pacchetti supplementari alla comunicazione effettiva. Questo comporta anche a una leggera perdita di astrazione.

Si aggiunge poi anche una rottura della connettività Internet end-to-end. È da tenere conto che il NAT nasce per creare delle reti private scollegate da Internet, per cui il fatto che sia stata aggiunta una connessione alla rete Internet esistente ha comportato a delle modifiche, soprattutto per quanto riguarda gli indirizzi speciali che non possono essere instradati direttamente e quando si hanno host di NAT diverse e annidate.

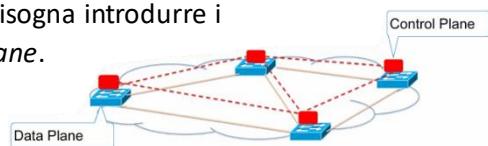
Nel complesso, il NAT è un "male necessario": quasi tutte le reti create negli ultimi 20 anni si trovano dietro uno o più NAT e non sarebbe necessario se avessimo un numero sufficiente di indirizzi IP pubblici. Inoltre, la separazione della rete può essere ottenuta utilizzando i firewall.

Routing

Routing vs forwarding

A questo punto resta da vedere la parte relativa all'instradamento (*routing*) vero e proprio, compresa la costruzione della tabella di inoltro. Per capire questo fenomeno bisogna introdurre i due layer/livelli della rete Internet, ovvero i *data plane* e i *control plane*.

I primi appartengono agli switch e sono per l'inoltro dei pacchetti, mentre i secondi ai router e sono algoritmi di instradamento.



I data plane sono quindi algoritmi per il forwarding dei pacchetti, ovvero il fatto di poterli inviare verso la loro destinazione, mentre i secondi, i control plane, servono per sapere dove mandarli effettivamente (routing = instradamento), infatti regolano il funzionamento dei data plane. Questo meccanismo risulta essere ben visibile all'interno di una rete, specialmente nel caso delle SDN (*Socket Defined Network*). In realtà ci sarebbe anche un terzo piano layer di gestione (management) di tutto questo meccanismo, in particolare il control plane, in quanto è il responsabile degli instradamenti.

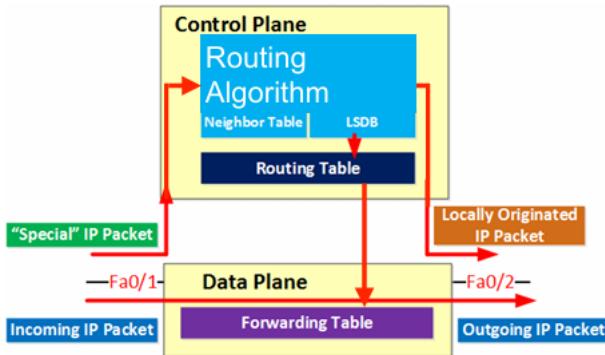
Altre differenze tra gli algoritmi di inoltro e quelli di instradamento sono:

- Inoltro:
 - seleziona una porta di uscita per ciascun pacchetto, in base all'indirizzo di destinazione e alla tabella di inoltro
 - deve essere semplice e veloce
 - possibilmente implementato in hardware
- Instradamento:
 - processo di costruzione della tabella di routing, che è precursore della tabella di forwarding
 - viene eseguito come processo in background (nello spazio utente)
 - può essere complesso

Quindi per quanto riguarda il forwarding, è un meccanismo che controlla gli indirizzi dei pacchetti in arrivo e tramite un'apposita tabella di inoltro sa su quale interfaccia deve essere mandato fuori un pacchetto per fare in modo che arrivi alla corretta destinazione. Per cui gli algoritmi devono essere molto efficienti e semplici. Invece, per quanto riguarda il livello di controllo del routing, esso ha l'incarico di costruire le tabelle di forwarding o, più precisamente, mantiene delle tabelle di informazioni extra da cui si derivano le tabelle di inoltro. Per questo motivo vengono definite precursori delle tabelle di inoltro, ma anche se sono simili sono in realtà molto diverse tra loro e soprattutto cambiano in base all'algoritmo di routing applicato (algoritmi diversi applicano metodologie e strutture dati diverse, anche se le tabelle finali di forwarding sono simili).

Il fatto che gli algoritmi di routing sono molto complessi, fa sì che debbano essere eseguiti una volta ogni tanto (es.: una volta al minuto o ogni 10 secondi), ma non periodicamente, piuttosto quando accade qualche evento del tipo un cambio di linea, aggiunta/rimozione di un router, cambio di carico della rete, ecc.

Infatti i router eseguono due compiti principali: forwarding dei pacchetti (semplice e veloce) e la costruzione delle tabelle di instradamento, da cui poi ricavare le tabelle di forwarding (complesso). È più o meno ciò che accade quando si guida un'auto: prima si pensa a dove si deve andare, da A a B, e poi ci si mette in marcia, per cui si deve sapere quando cambiare marcia, rallentare/accelerare, svoltare, ecc. Stessa cosa vale per control plane e data plane, il primo decide come i dati debbano essere trasportati, il secondo deve solo inviarli.



Per creare la tabella di routing, costruita dagli algoritmi di routing, si utilizzano le informazioni ricevute da altri nodi della medesima rete. Infatti gli algoritmi si parlano tra di loro attraverso i router e i pacchetti IP (sono simili agli altri, solo che possiedono dei dati di controllo). I router quindi dialogano tra di loro in maniera distribuita/concorrente in modo da tenersi aggiornati sugli eventi che accadono all'interno della rete e se serve aggiornano lo stato e le tabelle di routing (e di conseguenza anche quelle di forwarding).

Altra cosa molto interessante è che i router tengono traccia anche delle distanze, ovvero il “costo” per poter raggiungere un destinatario. Anche questo parametro deve sempre essere aggiornato e può cambiare le politiche di inoltro e instradamento in base al percorso che l'algoritmo dichiara più efficiente. Inoltre, possono essere calcolate diverse metriche/misure e costi, es.: percorso minimo, percorso più “economico”, ecc.

Ad esempio, si supponga di avere una tabella di routing e una di forwarding.

- **(a) tabella di routing:** utilizzata dal piano di controllo, contiene dati di livello 3, con informazioni su costi, distanze, ecc.
- **(b) tabella di inoltro:** utilizzata dal piano dati, contiene i dati per l'inoltro dei pacchetti: quale interfaccia, indirizzo del prossimo hop, ecc.

(a) Prefix/CIDR	Cost	Next Hop
18/8	3	171.69.245.10
151.85/16	4	195.84.38.5

(b) Prefix/CIDR	Interface	Next Hop
18/8	if0	84.3f.96.43:ad:6c
151.85/16	if2	6a:5c:04:22:5d:90

Ad esempio la rete 18/8 di (a) necessita di 171.69.245.10 come next hop, che ha un costo 3, per cui può risultare più efficiente della seconda sotto che ha un costo maggiore.

Algoritmi di routing

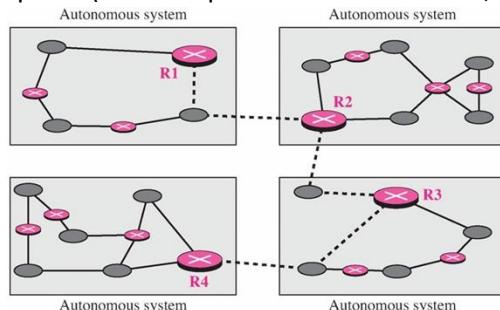
Inoltre, con questi meccanismi appena visti, è possibile vedere la rete come grafi (pesati e tipicamente non orientati, anche se in alcuni casi possono esserlo), in cui i nodi sono i router collegati alle reti (di base), i link sono le connessioni/collegamenti o archi del grafo e i costi (es.: velocità, latenza, costi, tasso di perdita, ecc.) sono i “pesi” degli archi. Il problema di base dell'instradamento è trovare il percorso col costo più basso tra due nodi qualsiasi, dove il costo di un percorso è uguale alla somma dei costi di tutti gli archi che lo compongono.

Infatti, i router possono avere più tabelle di instradamento in base al tipo di servizio richiesto (es.: una tabella per i percorsi più affidabili, ma dispendiosi, oppure quelli più veloci per due punti molto distanti, ecc.).

Tuttavia esistono degli svantaggi, infatti è difficile avere un algoritmo unico per tutta Internet oltre al fatto che vi sono problemi di scalabilità, indipendentemente da quale algoritmo di Spanning Tree, o di percorso minimo, che si vuole applicare (mappare una rete grande potrebbe essere così tanto dispendioso che per quando la si ha mappata tutta nel frattempo potrebbe essere già cambiata).

Inoltre, domini amministrativi diversi possono scegliere politiche di instradamento diverse (es.: un router potrebbe calcolare tutti i pesi per ritardo effettuato e un altro per massima velocità di trasmissione), per cui è meglio separare in due livelli:

- all'interno dei sistemi autonomi (AS):
 - ogni amministratore è responsabile degli algoritmi di routing
 - reti non troppo grandi → algoritmi più semplici (chiamati intra-domain, o protocolli interni)
- tra sistemi autonomi (AS):
 - problemi di raggiungibilità (*reachability*) piuttosto che di efficienza
 - algoritmi più complessi (chiamati protocolli inter-domain, o esterni)



Finché si è in una AS (intra-domain) si devono seguire le regole interne, ma poi quando si esce da essa, si deve sottostare alle regole delle altre AS o di quelli tra AS e AS (inter-domain). Con regole si intende come sono costruite le tabelle di inoltro. Questa situazione è simile a guidare un'auto a sinistra o a destra (es.: EU e UK).

I protocolli di inter-domain sono i più difficili da configurare (es.: ICP4) e anche un singolo errore lì dentro può creare seri problemi catastrofici di comunicazione. Sono i talloni d'Achille per i sistemisti.

Per una rete semplice, invece, si può calcolare tutti i percorsi più brevi e caricarli in una memoria non volatile su ogni nodo. Questo approccio statico presenta, però, diversi difetti: non gestisce i guasti dei nodi o dei collegamenti, non considera l'aggiunta di nuovi nodi o collegamenti e implica che i costi dei bordi non possano cambiare.

Quindi, la soluzione è di adottare un protocollo distribuito e dinamico per la costruzione di ogni tabella di instradamento, in cui i router dialogano tra di loro e aggiornano le tabelle in base agli eventi succeduti nella rete. Per fare ciò, i router devono adottare gli stessi algoritmi e strategie in modo da potersi accordare. Tutti gli algoritmi utilizzati dai router possono essere classificati in tre classi principali di protocolli:

- Vettore di distanza (distance vector): utilizzato nei protocolli interni come RIP
- Stato di collegamento (link state): utilizzato in protocolli interni come OSPF e IS-IS
- Vettore di percorso (path vector): utilizzato in protocolli esterni come BGP

Per quanto riguarda quelli interni, il RIP è un po' obsoleto in questo periodo, tuttavia potrebbe trovarsi in utilizzo qualche sua variante, in quanto è ancora valido. Quello più utilizzato oggigiorno è il link state.

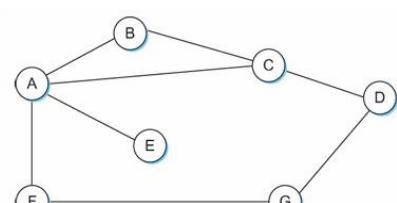
Invece, per l'esterno si usano solamente algoritmi basati sul percorso, poiché essendo per reti molto ampie ci si accontenta di trovare un percorso valido (le ottimizzazioni sono difficili come già detto).

Vettore di distanza (distance vector)

Questo è l'algoritmo più semplice e con costi di implementazioni bassi. L'ipotesi di partenza è che ogni nodo conosca il costo del collegamento con ciascuno dei suoi vicini direttamente connessi. Ogni nodo costruisce un array monodimensionale (vettore) contenente le "distanze" (costi) da tutti gli altri nodi e distribuisce tale vettore ai suoi vicini immediati.

Ad esempio, si supponga di avere la seguente rete con la seguente tabella con le distanze dei nodi immediati (vicini), in cui ogni riga è il vettore di partenza. Col simbolo ∞ si

Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	∞	1	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0



intende che si sa che esiste un nodo ma non si sa quanto disti dal nodo di partenza. Nessun nodo per il momento vede tutta la tabella, ma solo la propria riga.

Destination	Cost	NextHop
B	1	B
C	1	C
D	∞	—
E	1	E
F	1	F
G	∞	—

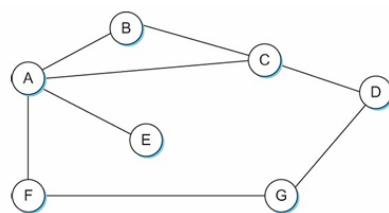
Prendendo in considerazione ad esempio il nodo A, si può costruire la sua tabella di routing iniziale (figura a sinistra). Per i nodi di cui conosce le distanze sa anche qual è il next hop, mentre di quelli non direttamente a esso collegati non può avere attualmente un'informazione di dove mandare i pacchetti (*visione parziale e soggettiva della rete*).

Tramite uno scambio ripetuto di pacchetti relativi

alle distanze tra i router, a un certo punto A riesce ad avere una tabella completa per il forwarding (immagine a destra). Lo stesso varrà per gli altri router, per cui ognuno si ritroverà alla fine con una propria tabella completa della rete con cui sono in comunicazione.

Di seguito quella completa della rete (*visione globale*):

Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0



Destination	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Algoritmo di Bellman-Ford

L'algoritmo di routing del distance vector è talvolta chiamato algoritmo di *Bellman-Ford* e funziona come di seguito. Ogni T secondi ogni router invia il proprio vettore ai vicini. Quando un router riceve un vettore da un router vicino, aggiorna la sua tabella in base alle nuove informazioni, in cui per ogni voce:

- se si ha un percorso migliore di quello già noto, la voce viene sostituita
- se il percorso migliore passa ancora per il router vicino, il costo viene aggiornato

Questo algoritmo si stabilizza in $O(|N| * |L|)$, dove N=nodi e L=collegamenti.

Per quanto riguarda l'algoritmo effettivo si ha che:

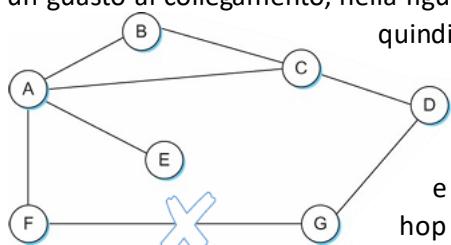
- V = vettore ricevuto dal vicino R
- T = la propria tabella di routing locale
- costo(R) = costo del collegamento verso R

Per cui l'algoritmo di aggiornamento quando si riceve un vettore V da R è:

```
for each entry (D, c, N) ∈ T:
    c' = V(D) + costo(R)
    if R = N then
        replace (D, c, N) in T with (D, c', N)
    else if c' < c then
        replace (D, c, N) in T with (D, c', R)
```

I router regolarmente si scambiano messaggi di configurazione, per cui se si rileva un problema si è in grado di farne fronte. Tuttavia esistono dei casi in cui si commettono degli errori. Ad esempio quando un nodo rileva un guasto al collegamento, nella figura è il link F-G, si ha: F rileva che il collegamento con G si è interrotto,

quindi imposta la distanza da G all'infinito e invia l'aggiornamento ad A, che a sua volta imposta la distanza da G all'infinito poiché utilizza F per raggiungere G. Il nodo A poi riceve un aggiornamento periodico da C con un percorso a 2 hop verso G, per cui imposta la distanza da G a 3 e invia l'aggiornamento a F, in quale decide che può raggiungere G in 4 hop attraverso A.



Altre circostanze leggermente diverse potrebbero impedire la stabilizzazione della rete, ad esempio se il link A-E va in tilt, nel successivo ciclo di aggiornamenti, A pubblicizza una distanza infinita da E, mentre B e C pubblicizzano una distanza di 2 da E. A seconda dell'esatta tempistica degli eventi, potrebbe accadere quanto segue:

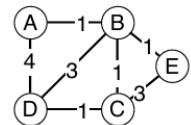
- il nodo B, dopo aver appreso che E può essere raggiunto in 2 hop da C, conclude che può raggiungere il nodo E in 3 hop e lo pubblicizza ad A;
- il nodo A conclude che può raggiungere E in 4 hop e lo pubblicizza a C;
- il nodo C conclude che può raggiungere E in 5 hop;
- e così via, fino all'infinito!

Questo problema nasce dal fatto che i router non sanno in maniera sicura se gli altri router hanno assegnato un certo peso ai link perché hanno un collegamento effettivo o se quel numero deriva da uno scambio di messaggi, per cui essi si fidano e incrementano ogni volta il numero di hop.

Esercizi:

(Scheda del 2021-06-22):

9) I router della rete a lato utilizzano un algoritmo di routing basato sul vettore delle distanze. (a) Si dia la tabella di routing iniziale di E. (b) Si dia la tabella di routing finale di E (ossia dopo la stabilizzazione).



R: (a) per la tabella iniziale basta vedere con quali nodi E comunica direttamente, mentre per (b) quella finale si sommano le distanze più corte da E verso ognuno degli altri nodi. Per quanto riguarda i next hop, nel caso iniziale si prendono le lettere dei vicini, mentre nei passaggi intermedi e quindi nella configurazione finale si considerano i percorsi più brevi e si selezionano le lettere dei vicini del primo link diretto tra E ed il suo vicino che porta a quel percorso breve. Si ottiene quindi:

(a)			(b)		
dest	d	n.h.	dest	d	n.h.
B	1	B	A	2	B
C	3	C	B	1	B
E	0	-	C	2	B
			D	3	B
			E	0	-

(NON DELL'ESERCIZIO: da notare che nel caso in cui i sensori che monitorano la linea, rilevassero che un link fosse magari più libero di un altro o più congestionato, allora essi cambiano i pesi di quel link, con la conseguenza che le tabelle di routing debbano essere riaggiorionate. Ad esempio, se B-D restando vuoto passasse da peso 3 a peso 1 e B-C essendo più utilizzato passasse da 1 a 3, le tabelle di routing cambierebbero in modo da favorire B-D piuttosto che B-C)

Problema del conteggio all'infinito

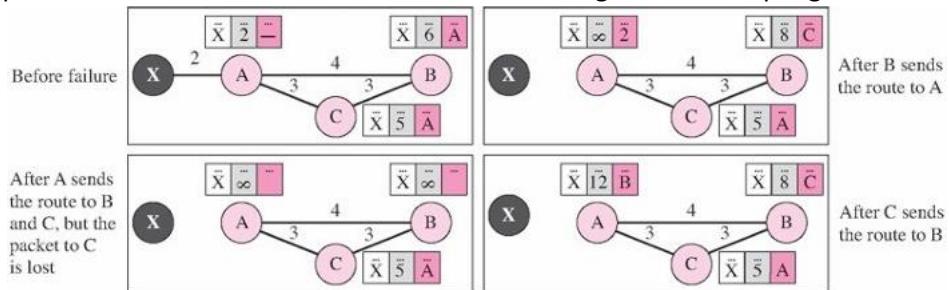
Come visto poco fa, ci sono dei casi in cui il numero del peso del link può crescere all'infinito quando così non è. Per risolvere il problema si potrebbe utilizzare un numero relativamente piccolo come approssimazione dell'infinito. Ad esempio, il numero massimo di hop per attraversare una certa rete non sarà mai superiore a 16. Questo meccanismo è utilizzato nelle implementazioni reali (RIP). Quando il numero raggiunge il limite, si smette di incrementarlo e lo si considera come ∞ . Tuttavia è rischioso poiché non funziona se la rete supera il limite e di conseguenza va bene solo per reti limitate, come gli AS.

Una tecnica per migliorare il tempo di stabilizzazione dell'instradamento è chiamata orizzonte diviso (*split horizon*). Quando un nodo invia un aggiornamento del routing ai suoi vicini, non invia a questi ultimi le rotte che ha appreso da ciascun vicino. Ad esempio, se B ha la rotta (E, 2, A) nella sua tabella, sa che deve averla

appresa da A e quindi ogni volta che B invia un aggiornamento di routing ad A, non include la rotta (E, 2) in tale aggiornamento. Tuttavia, anche questa tecnica ha un problema: lo split horizon non consente a un nodo di sapere se i vicini ricevono i suoi messaggi.

In una versione migliorata di split horizon, chiamata split horizon con veleno inverso (*split horizon with poison reverse*), ad esempio B invia effettivamente il percorso inverso ad A, ma inserisce un'informazione negativa nel percorso per garantire che A non utilizzi B per arrivare a E (es.: B invia il percorso (E, ∞) ad A).

Inoltre, le tecniche di split horizon non risolvono il problema con loop di 3 o più nodi. Per questo motivo, di solito nelle implementazioni reali si usa un “infinito finito”. Di seguito un esempio grafico:

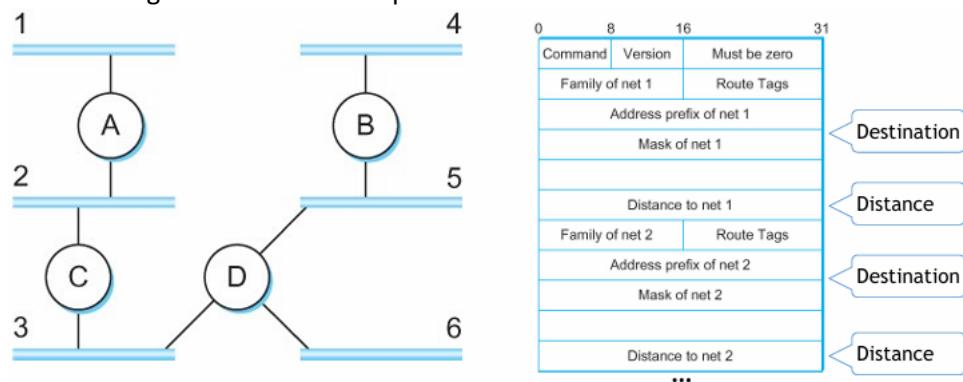


In questo caso in immagine A si accorge che X non è più raggiungibile, per cui informa B che imposta il valore a ∞ , in contemporanea però anche C notifica B, per cui B vede che C ha assegnato un peso per X inferiore, dunque aggiorna la entry con la nuova configurazione data da C. Adesso B crede di essere a distanza 8 da X e non più ∞ , quindi dice ad A che ha un nuovo percorso definito e A si aggiorna di conseguenza. Questi aggiornamenti però sono fasulli...

Protocollo di informazione di routing (RIP)

L'implementazione principale per effettuare un controllo sullo split horizon è il protocollo RIP (*Routing Information Protocol*), attualmente in versione 2. Esso è implementato come protocollo a livello applicazione e viene eseguito su UDP (porta 520), sebbene possa essere implementato direttamente su IP.

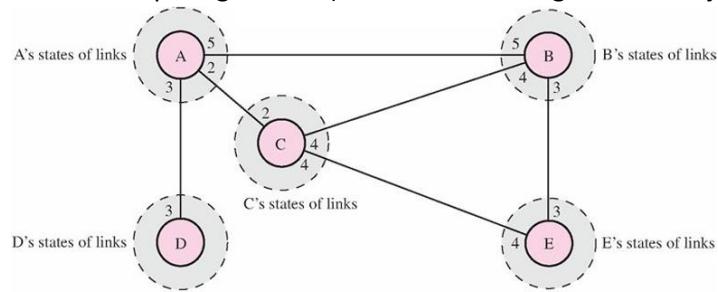
Era una soluzione comune per l'instradamento intra-domain e era implementato su sistemi Unix da routed(8). Adotta la tecnica del vettore di distanza con infinito finito: le destinazioni finali sono gli indirizzi di rete, i nodi sono router direttamente connessi alle reti, il next hop è l'indirizzo di un router, o nullo se la destinazione è nella stessa rete, ogni hop si conta +1 e il valore limite per l'infinito è 16 (quindi funziona su reti con diametro ≤ 15). Di seguito un'immagine del formato del pacchetto.



Stato dei collegamenti (link state)

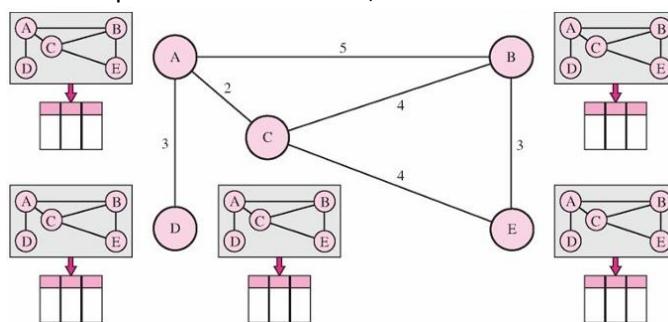
Dato che le soluzioni appena viste non sono sempre fattibili o mostrano delle debolezze in alcuni casi, si è preferito adottare un approccio completamente diverso: ogni nodo costruisce una visione completa della rete (nodi e collegamenti). Nodi diversi possono avere viste diverse e ognuno di essi controlla lo stato delle sue

connessioni dirette e informa gli altri di ogni cambiamento. Quando un nodo ha una visione completa della rete, può calcolare i percorsi minimi per ogni nodo (es.: utilizzando l'algoritmo di Dijkstra).



Questa strategia mostra già uno svantaggio: serve ampia potenza di calcolo e memoria. Per questo motivo in passato non si era riusciti a utilizzare una tecnica simile, mentre con i router odierni risulta essere già più fattibile (per le sole reti di piccole dimensioni). Per questa ragione, oggigiorno è più probabile utilizzare un link state che non un RIP.

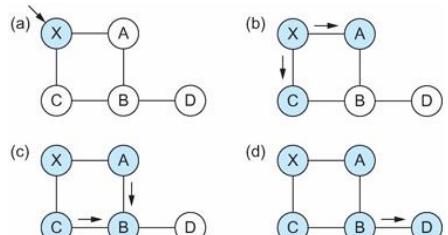
Per questo approccio ci sono due fasi principali: la prima consiste nel mantenere una visione globale del sistema, ovvero ogni nodo deve acquisire le informazioni, mentre nella seconda si calcolano i percorsi minimi.



La strategia che si può utilizzare è quella di informare periodicamente tutti i nodi (non solo i vicini) delle informazioni sui collegamenti direttamente connessi (non l'intera tabella di routing). Per fare ciò si usa un pacchetto di stato del collegamento (LSP, *Link State Packet*). Esso contiene: id del nodo che l'ha creato, il costo del collegamento con ciascun vicino direttamente connesso, il numero di sequenza (SEQNO) e il tempo di vita (TTL). I pacchetti sono propagati tramite un flusso affidabile (*Reliable Flooding*): esso memorizza l'LSP più recente da ciascun nodo (tramite timestamp), inoltra l'LSP a tutti i nodi tranne quello che l'ha inviato, genera periodicamente nuovi LSP (incrementando il SEQNO), avvia SEQNO a 0 al riavvio e decremente il TTL di ogni LSP memorizzato (scartandolo quando TTL=0).

Quindi quando si ha un flooding affidabile si ha una situazione del tipo:

- (a) LSP arriva al nodo X;
- (b) X invia l'LSP ad A e C;
- (c) A e C inviano l'LSP a B (ma non a X);
- (d) il flooding è completo



Instrandamento sul percorso più breve (Dijkstra)

Per calcolare il percorso più breve tra i nodi si usa l'algoritmo di Dijkstra, che ipotizza pesi dei collegamenti non negativi. Si segna con N l'insieme dei nodi del grafo, mentre con $l(i, j)$ il costo non negativo associato all'arco tra i nodi $i, j \in N$ e $l(i, j) = \alpha$ se nessun arco collega i e j . Inoltre, sia $s \in N$ il nodo iniziale che esegue l'algoritmo per trovare i percorsi più brevi per tutti gli altri nodi in N . Vengono utilizzate anche altre due variabili: M per insieme dei nodi finora incorporati dall'algoritmo e $C(n)$ per il costo del percorso da s a ciascun nodo n .

L'algoritmo risulta quindi così:

```

M = {s}
    for each n in N - {s}
        C(n) = l(s, n)
while (N ≠ M)
    M = M ∪ {w} such that C(w) is the minimum for all w in (N-M)
    For each n in (N-M)
        C(n) = MIN (C(n), C(w) + l(w, n))

```

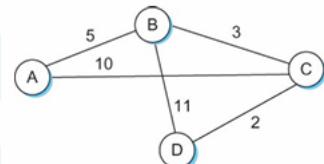
Con complessità: $O(|l| + |N|\log(|N|))$.

In pratica, ogni switch/router calcola la propria tabella di routing direttamente dagli LSP raccolti, utilizzando una realizzazione dell'algoritmo di Dijkstra chiamata algoritmo di ricerca in avanti (*forward search algorithm*). In particolare, ogni switch/router mantiene due elenchi, noti come Tentativo (*Tentative*) e Confermato (*Confirmed*). Ciascuna di queste liste contiene un insieme di voci della forma (*Destinazione, Costo, Next hop*).

L'algoritmo esegue i seguenti passaggi (immagine più esplicativa più in basso):

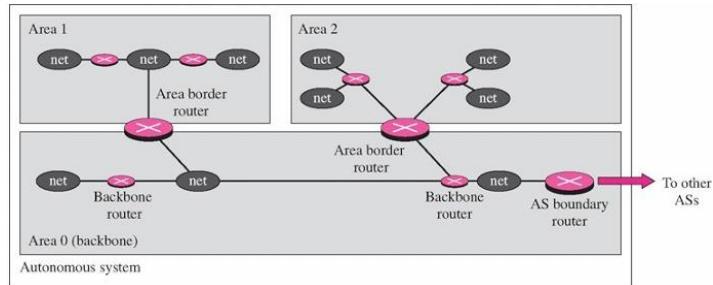
1. Inizializza la lista *Confirmed* con una voce per sé stesso con un costo pari a 0
2. Per il nodo appena aggiunto alla lista *Confirmed* nel passo precedente, chiamato nodo *Next*, seleziona il suo LSP
3. Per ogni vicino (*Neighbor*) di *Next*, calcola il costo (*Cost*) per raggiungere questo *Neighbor* come somma del costo da sé stesso a *Next* e da *Next* a *Neighbor*.
 - a. Se il *Neighbor* non è attualmente presente né nell'elenco *Confirmed* né in *Tentative*, aggiunge (*Neighbor, Cost, Next hop*) all'elenco *Tentative*, dove *Next hop* è la direzione da seguire per raggiungere il *Next*.
 - b. Se il *Neighbor* è attualmente nell'elenco *Tentative* e il *Cost* è inferiore al costo attualmente elencato per il *Neighbor*, allora sostituire la voce attuale con (*Neighbor, Cost, Next hop*) dove *Next hop* è la direzione in cui si va per raggiungere il *Next*.
4. Se l'elenco dei tentativi è vuoto, ci si ferma. Altrimenti, si sceglie dall'elenco *Tentative* la voce con il costo più basso e la si sposta nell'elenco *Confirmed* e si torna al passo 2.

Step	Confirmed	Tentative	Comments
1	(D,0,-)		Since D is the only new member of the confirmed list, look at its LSP.
2	(D,0,-)	(B,11,B) (C,2,C)	D's LSP says we can reach B through B at cost 11, which is better than anything else on either list, so put it on Tentative list; same for C.
3	(D,0,-) (C,2,C)	(B,11,B)	Put lowest-cost member of Tentative (C) onto Confirmed list. Next, examine LSP of newly confirmed member (C).
4	(D,0,-) (C,2,C)	(B,5,C) (A,12,C)	Cost to reach B through C is 5, so replace (B,11,B). C's LSP tells us that we can reach A at cost 12.
5	(D,0,-) (C,2,C) (B,5,C)	(A,12,C)	Move lowest-cost member of Tentative (B) to Confirmed, then look at its LSP.
6	(D,0,-) (C,2,C) (B,5,C)	(A,10,C)	Since we can reach A at cost 5 through B, replace the Tentative entry.
7	(D,0,-) (C,2,C) (B,5,C) (A,10,C)		Move lowest-cost member of Tentative (A) to Confirmed, and we are all done.



Protocollo Open Shortest Path First (OSPF)

Il protocollo Open Shortest Path First (OSPF) è implementato su unix da gated(8) direttamente su IP (senza UDP), con autenticazione per evitare attacchi di reindirizzamento: un host può reindirizzare il traffico inondando di false informazioni. Adotta l'approccio Link State, in cui l'AS è diviso in aree e backbone. Ogni area è collegata alla dorsale o ad altre aree da un router di confine (*border router*). Questo protocollo e il IS-IS, che funziona in modo simile, sono i più usati nei sistemi intra-dominio al giorno d'oggi.



L'amministratore può definire i costi dei collegamenti in vari modi (velocità, costo, carico, latenza, affidabilità, ecc.). L'OSPF consente agli LSP di avere diversi tipi di costi, anche per gli stessi collegamenti, quindi un router può mantenere diverse tabelle di routing (e di conseguenza diverse tabelle di inoltro), una per ogni tipo di servizio previsto e di conseguenza può fare scelte di inoltro diverse per i diversi servizi. La scelta della tabella da applicare viene decisa in base a:

- TOS/DSCP (raro)
- QoS, priorità (se supportata)
- tag di flusso e priorità in IPv6
- Rilevamento automatico del tipo di flusso (*traffic shaping*)

In realtà, i router più costosi e seri applicano una tecnica di machine learning per capire qual è il tipo di flusso utilizzato (*flow detection*) e quindi sapere quale tipo di tabella utilizzare (es.: un SSH lo si può riconoscere non solo dalla porta utilizzata, ma anche dalla forma del pacchetto, dalla quantità di dati, ecc.). In questo modo sfrutta il *traffic shaping* per capire cosa sta circolando nel flusso.

I formati dei pacchetti che OSPF utilizza sono i seguenti:

OSPF Header Format			OSPF Link State Advertisement		
0	8	16	31		
Version	Type	Message length	LS Age	Options	Type=1
			Link-state ID		
		SourceAddr	Advertising router		
		Areald	LS sequence number		
	Checksum	Authentication type	LS checksum	Length	
		Authentication	0	Flags	Number of links
			Link ID		
			Link data		
	Link type	Num_TOS	Metric		
			Optional TOS information		
			More links		

Una cosa interessante e che in RIP non c'è, è che non viene effettuato alcun controllo per evitare pacchetti duplicati o alterati. Questo è perché OSPF fornisce un meccanismo di autenticazione. Questo è importante perché è possibile simulare l'arrivo di un pacchetto, quando questo però è fasullo (generato da un *demon*), invece OSPF è in grado di stabilire se un pacchetto è autentico o meno.

Infatti, i protocolli visti finora sono particolarmente sensibili a ciò che viene inviato, ad esempio se un router malevolo guarda il contenuto di un pacchetto prima di inviarlo, potrebbe cambiare il contenuto, aggiungere la nuova intestazione, ricalcolare il checksum e inviare il pacchetto e quest ultimo essere perfettamente legittimo (del resto i router cambiano già di loro natura le intestazioni). Oppure un router potrebbe avere delle tabelle di instradamento particolari, che fanno passare obbligatoriamente il flusso di pacchetti per un router/dispositivo malevolo e che quindi può infettare la rete guardando i pacchetti che gli arrivano.

Dunque OSPF mette a disposizione un meccanismo di autenticazione, in cui si inserisce una sorta di "firma" nel messaggio inviato. È comunque da ricordare che l'informazione che circola è pubblica e può benissimo essere vista da chiunque sbirci nel traffico della rete. Una delle soluzioni che viene adottata è quella delle chiavi condivise, in cui un router, quando riceve un pacchetto lo valida con la chiave che possiede per vedere se è autentico o meno (in caso negativo lo scarta).

Internetworking avanzato

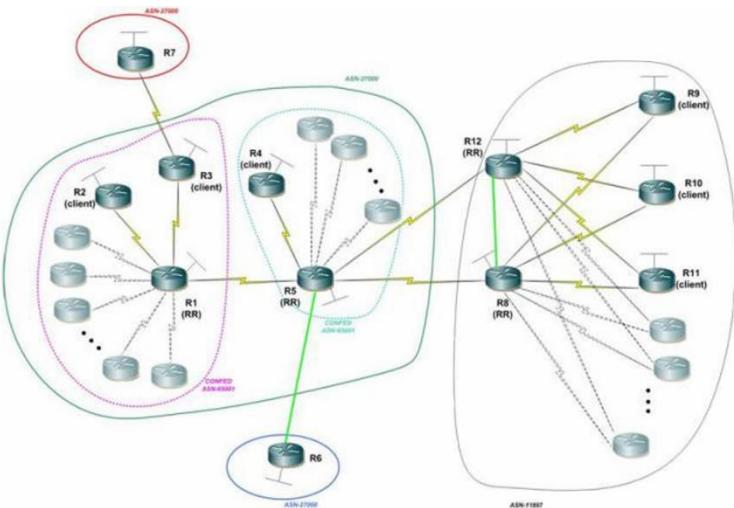
In questo capitolo si affronteranno le tematiche avanzate del capitolo precedente, ovvero il routing inter-dominio, l'IPv6, ecc.

Instrandamento inter-dominio

Internet è organizzata in sistemi autonomi (AS), ciascuno dei quali è sotto il controllo di una singola entità amministrativa. Il sistema autonomo (AS) corrisponde a un dominio amministrativo (es.: università, azienda, rete dorsale, ecc.), mentre l'entità amministrativa è quella che assegna le regole politiche e omogenee da far rispettare all'interno della rete affinché essa funzioni correttamente (es.: tutti i router devono essere di un certo tipo, oppure tutti devono usare un certo protocollo come RIP o altri, oppure si misurano i ritardi o i throughput, ecc.).

La rete interna di un'azienda potrebbe essere un singolo AS, così come la rete di un singolo fornitore di servizi Internet. Quindi un AS, visto come un insieme di router e reti locali (in figura le zone racchiuse dalle linee colorate), è una sottorete di Internet.

Ad esempio, le AS possono anche essere i singoli provider di un certo servizio che inglobano le sottoreti dei clienti del servizio offerto (es.: la rete rosa) e che a sua volta sono inglobate e messe in comunicazione con reti nazionali o internazionali più grandi di livelli superiori (es.: la rete verde).



Il GARR (*Gruppo per l'Armonizzazione della Rete della Ricerca*) è il consorzio italiano di interconnessione delle reti di ricerca, istruzione e cultura. È una sorta di provider per reti di ricerca e è parallelo a altre reti nazionali e internazionali. Nell'immagine affianco, in corrispondenza dei pallini bianchi vi possono essere anche trenta o cinquanta o più router per connettere reti differenti e decine di migliaia di host diversi.



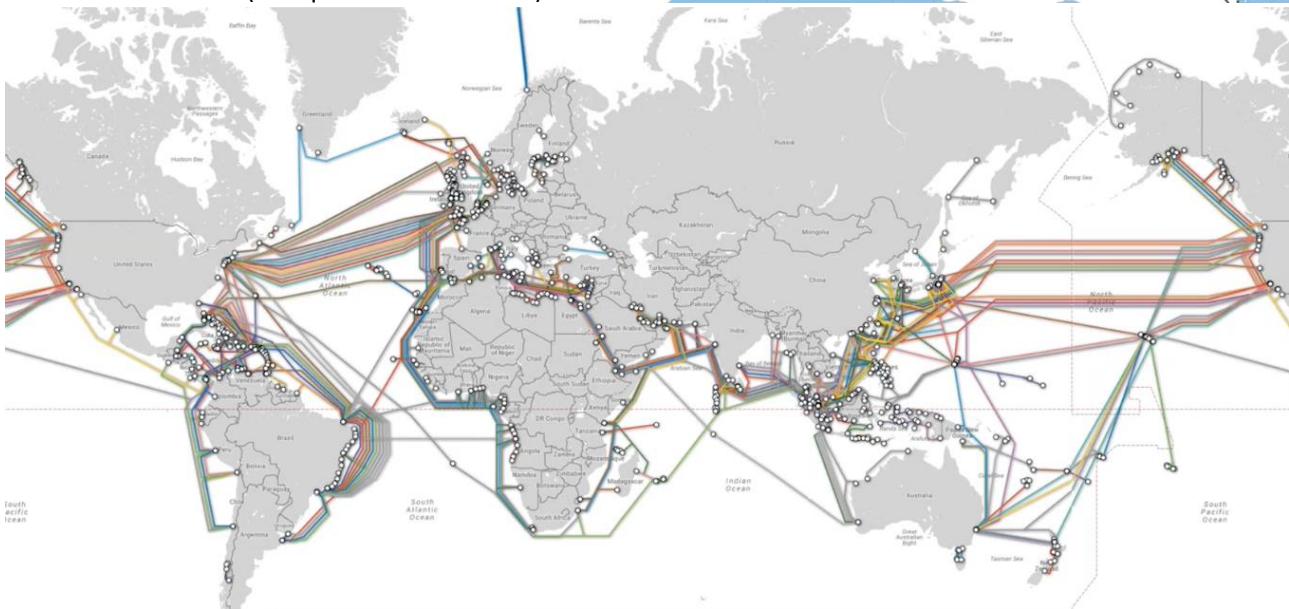
Le linee blu sono i PoP (*point of presence*), ovvero quei punti di accesso a cui ci si collega con linee noleggiate. Se ci si collega al sito della GARR è possibile vedere in tempo reale le prestazioni e le caratteristiche di tali punti e linee, nonché il carico del traffico (es.: di un certo collegamento si può visualizzare quanti link dedicati ha e con quali performance, se vi sono link di backup e quanto traffico stanno facendo in quel momento).

Per quanto riguarda i link Géant, NaMeX, Cogent, ecc. sono i collegamenti verso le reti internazionali o a altre reti nazionali simili a questa italiana. Essi servono a uscire dalle AS e a fare in modo di poter visualizzare ad esempio siti web di altre nazioni o continenti.

Di seguito delle immagini che rappresentano le reti europee e internazionali. Da notare come il GARR riguardi solo l'Italia e rappresenti una piccolissima parte delle reti intercontinentali e mondiali.

Affianco la rete europea, di cui il GARR è stilizzato ai soli nodi e linee più rilevanti. Sotto, invece, la rete mondiale. Da notare il link (NaMeX) da Roma verso l'esterno dell'Italia fino in America attraverso l'oceano e che costituisce il punto di accesso/uscita dei pacchetti da fuori/dentro l'Italia.

È da ricordare che questa è solo la struttura internet per la reti di ricerca, studio e cultura e che è parallela a altre reti mondiali (es.: quella commerciale).



Propagazione del percorso (Route Propagation)

Per quanto riguarda il routing, a grandi dimensioni non è possibile utilizzare i protocolli e le tecniche viste finora, in quanto è praticamente impossibile mettere d'accordo tutti i router su una metrica di distanza (costi) e sulle decisioni di instradamento dei pacchetti, che non sono necessariamente sempre guidate da motivi di ottimizzazione dei costi, ma molto spesso sono ragioni "politiche" (= *policies*, es.: una rete deve essere evitata, mentre un'altra no) e i protocolli appena visti tengono conto solo in parte di questi fattori.

Per cui un'idea è quella di fornire un modo aggiuntivo per aggregare gerarchicamente le informazioni di routing in una rete Internet di grandi dimensioni. In questo modo si migliora la scalabilità. Si divide, quindi, il problema dell'instradamento in due parti:

- instradamento all'interno (***within***) di un singolo sistema autonomo
- instradamento tra (***between***) sistemi autonomi

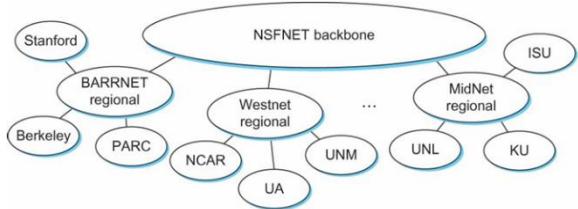
Nel primo caso i router sono *peer* e hanno stesse regole e protocolli, nel secondo si devono creare delle gerarchie e domini, da cui si ottiene anche un altro nome per i sistemi autonomi (AS) in Internet: "domini di routing" (*routing domains*). La gerarchia di propagazione dei percorsi a due livelli segue due protocolli:

- Protocollo di instradamento interdominio (standard per tutta Internet)
- Protocollo di routing intradominio (ogni AS sceglie il proprio)

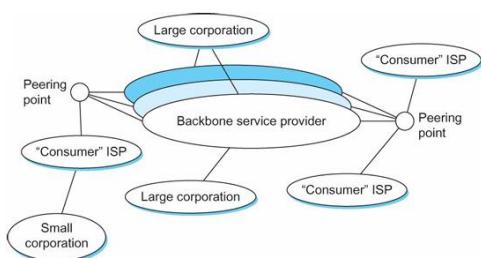
EGP e BGP

Originariamente (quindi nel 1990 circa) il sistema Internet aveva una struttura ad albero, in cui vi era come root una grande rete backbone a cui poi si collegavano reti minori regionali o nazionali e poi nei livelli inferiori vi erano altre zone locali.

La situazione è simile a quella del GARR, in cui il medesimo fa da backbone e i PoP sono le reti inferiori locali a cui si collegano le singole aziende/università/enti di ricerca.



Questo tipo di struttura, tuttavia è rigida e andava bene finché Internet era piccola (sistema “accademico”), in cui vi era un’unica dorsale e i AS sono collegati solo come genitori e figli e non come peer, uno dei motivi che non ha permesso che la topologia diventasse generale col passare degli anni. Inoltre, i protocolli utilizzati al suo interno erano protocolli di routing inter-dominio, come il *Protocollo Exterior Gateway* (EGP).



Per cui la soluzione è stata quella di creare tanti provider di tipo backbone (es.: governativo, universitario, commerciale, ecc.) connessi tra loro, ottenendo una sorta di struttura a grafo. Da ricordare che vi possono essere anche provider minori (consumer ISP) che sfruttano delle backbone per offrire un servizio ai propri host. Inoltre ci si può collegare a più backbone o provider (maggiore affidabilità).

Per quanto riguarda il protocollo utilizzato, in questo caso si ha il protocollo di gateway di confine (BGP, *Border Gateway Protocol*). Esso presuppone che Internet sia un insieme arbitrariamente interconnesso di AS. Infatti, l’Internet di oggi consiste in un’interconnessione di più reti dorsali (di solito sono chiamate reti di fornitori di servizi e sono gestite da aziende private piuttosto che dal governo). I siti sono connessi l’uno all’altro in modi arbitrari.

Alcune grandi aziende si collegano direttamente a una o più reti dorsali, mentre altre si collegano a fornitori di servizi più piccoli, non dorsali. Molti fornitori di servizi esistono principalmente per fornire servizi ai “consumatori” (individui con PC nelle loro case), e questi fornitori devono connettersi ai fornitori di backbone. Spesso molti provider si organizzano per interconnettersi tra loro in un unico “punto di peering”.

Il protocollo BGP non ha lo scopo di ottimizzare i percorsi, piuttosto ha quello di trovare i percorsi (privi di loop). Infatti, il protocollo BGP-4 per i gateway di confine, è attualmente in funzione. Si parte dall’ipotesi che Internet sia un insieme arbitrariamente interconnesso di AS. Si definisce traffico locale (*local traffic*) quello che ha origine o termina su nodi all’interno di un AS e traffico di transito (*transit traffic*) quello che attraversa un AS. A questo punto si può classificare gli AS in tre tipi:

- **AS stub:** un AS che ha una sola connessione con un altro AS; un AS di questo tipo trasporta solo traffico locale (la *small corporation* nella figura sopra).
- **AS multi-homed:** un AS che ha connessioni con più di un altro AS e a cui, però, non interessa trasportare traffico di transito (la *large corporation* in alto nella figura): semplicemente sfrutta uno o più link per motivi propri (es.: per affidabilità, efficienza, ecc.). Quindi, queste AS non sono provider, ma magari aziende.
- **Transit AS:** un AS che ha connessioni con più di un altro AS ed è progettato per trasportare sia il traffico di transito che quello locale (*backbone provider*). Infatti, il traffico che gira (entra/esce) in questi AS non sempre riguarda il traffico di questi AS stessi, ma si serve di essi per passare da una parte a un’altra della rete.

Come appena detto, BGP non ottimizza i percorsi, ma li trova soltanto e fa attenzione che non contengano cicli, poiché a queste dimensioni della rete conta la raggiungibilità piuttosto che l'ottimizzazione (già a fatica si ottiene la prima). Trovare un percorso che si avvicini a quello ottimale è considerato un grande risultato. Infatti molti dei titoli di BGP sono tratti da questo.

Infatti, gli obiettivi di BGP sono sostanzialmente:

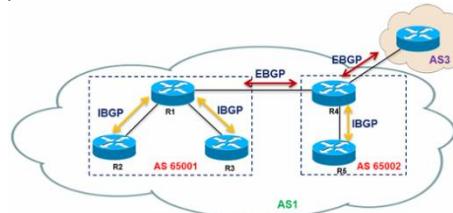
- Scalabilità: un router della dorsale Internet deve essere in grado di inoltrare qualsiasi pacchetto destinato a qualsiasi punto della rete. Dunque, deve avere una tabella di routing che fornisca una corrispondenza per qualsiasi indirizzo IP valido;
 - Natura autonoma dei domini: è impossibile calcolare costi di percorso significativi per un percorso che attraversa più AS, soprattutto per il fatto che se si ha un costo di 1000 attraverso un provider potrebbe implicare un ottimo percorso per esso, ma per un altro potrebbe essere inaccettabile;
 - Problemi di fiducia (trust): il fornitore A potrebbe non essere disposto a credere a certe pubblicità del fornitore B, ad esempio se si hanno due server di streaming e uno promette un delay più basso dell'altro si potrebbe essere più invogliati a sceglierlo piuttosto dell'altro, ma magari è solo una trovata di marketing e in realtà è più inaffidabile.

Ogni AS ha uno o più router speciali chiamati *speaker BGP* (= portavoce). Uno speaker BGP fa da portavoce ai suoi peer (altri router speaker BGP), ovvero comunica agli altri portavoce a esso collegati quali sono le reti locali interne (a quel AS), e eventualmente le reti non sue (AS di transito), e fornisce quindi informazioni sul percorso (es.: essi si dicono con chi possono comunicare in modo da “avere un’idea della rete” e sapere a chi inoltrare i pacchetti per farli arrivare a destinazione). Ad esempio, nell’immagine sotto i router portavoce potrebbero essere R1 e R4.

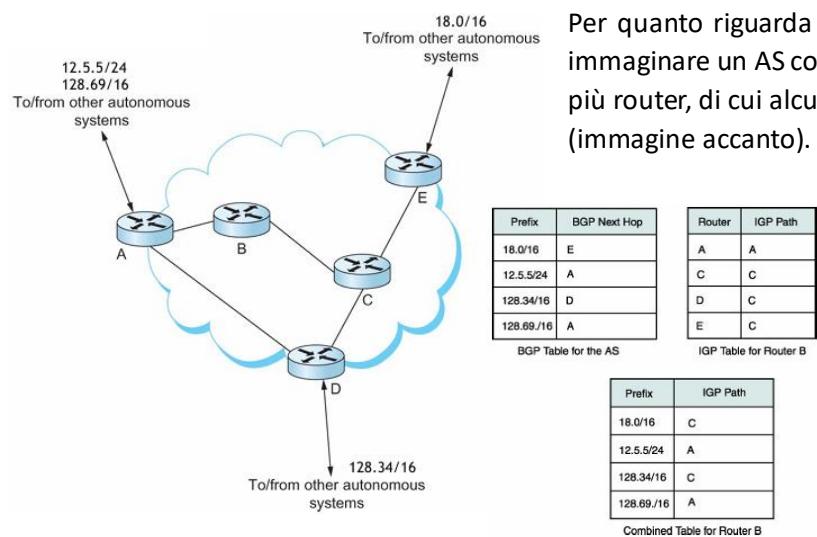
Oltre agli speaker BGP, un AS ha uno o più gateway di frontiera, che non devono necessariamente coincidere con gli speaker. I gateway di frontiera sono i router attraverso i quali i pacchetti entrano ed escono dal AS.

Esistono due versioni del protocollo:

- BGP esterno (*eBGP*): utilizzato tra router appartenenti a diversi AS
 - BGP interno (*iBGP*): utilizzato tra router all'interno dello stesso AS



Integrazione del routing inter-dominio e intra-dominio



Per quanto riguarda il funzionamento vero e proprio si può immaginare un AS come una “nuvoletta”, in cui sono connessi più router, di cui alcuni di confine e comunicanti con l'esterno (immagine accanto).

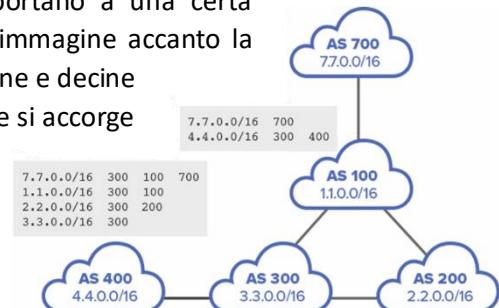
Tutti i router eseguono iBGP e un protocollo di routing intra-dominio (es.: RIP, OSPF, ecc.).

I router di confine (A, D, E) eseguono anche eBGP verso altri AS. Esiste una tabella di routing BGP valida per l'intero AS, le cui informazioni vengono poi propagate ai router interni tramite iBGP (in immagine).

BGP funziona propagando i percorsi (*path*) completi e che portano a una certa destinazione, quindi hanno sempre delle reti definite (es.: nell'immagine accanto la 7.7.0.0/16). Dunque in un sistema completo si possono avere decine e decine di router, con decine di reti e decine di migliaia di host e non ce ne si accorge in quanto la rete sembra tutta omogenea. Gli speaker BGP comunicano questi percorsi completi (es.: il router "AS 100" può comunicare agli altri di sapere come inoltrare i pacchetti alla rete 7.7.0.0/16, oppure sa che per arrivare alla 4.4.0.0/16 deve passare per i router "AS 300" e "AS 400").

Inoltre, il BGP non appartiene a nessuna delle due classi principali di protocolli di routing (vettori di distanza e protocolli a stato di link). BGP pubblicizza percorsi completi come liste enumerate di AS per raggiungere una particolare rete.

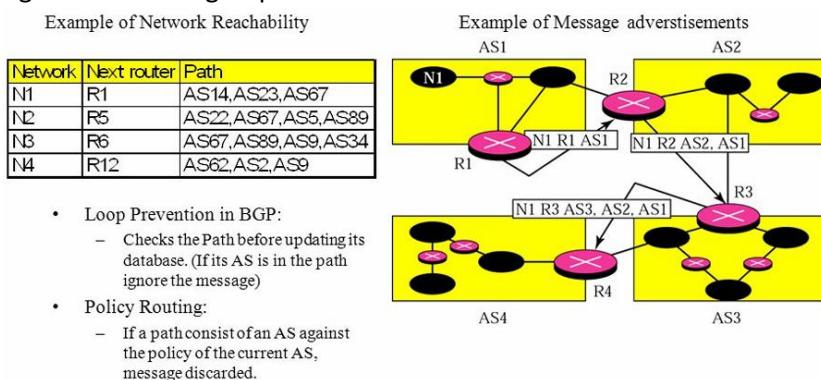
Le regole di inoltro e di accettazione dei pacchetti dipendono da come sono configurati gli speaker BGP, in quanto sta a loro sapere se accettare certi tipi di pacchetti (o con certi IP host) o meno.



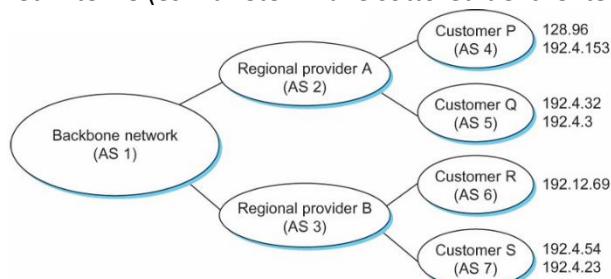
Prevenzione dei loop BGP e policy routing

In realtà i router BGP non scartano i pacchetti solo in base alle proprie regole interne, ma anzi, quando essi ne ricevono uno di configurazione per i percorsi, analizzano tutto il path e decidono se scartare il pacchetto o meno per evitare loop. Per sapere se è in presenza di un loop controlla che non ci sia il proprio AS lungo il percorso comunicato dall'altro speaker. Dunque, un router BGP ignora un percorso se: il suo AS compare in esso (per evitare i loop tra AS), oppure se viola qualche politica dell'AS.

Ad esempio, se si hanno quattro AS come in figura e un router R1 volesse comunicare che è in collegamento con la rete N1, lo fa attraverso R2, il quale sa di essere il "*forwarder router*" per la rete N1 poiché ne è messo in comunicazione. Per cui R2 comunica a R3 di essere in comunicazione con la rete N1 utilizzando il link che attraversa la propria rete interna AS2 e il router R1. A sua volta R3 comunica a R4 che è in comunicazione con N1 tramite rete interna, AS2 e AS1 (tramite rispettivi router). Tutto questo viene eseguito controllando prima le proprie *policy* e regole interne a ogni speaker BGP.



Di seguito, invece, si ha un esempio di una rete BGP gerarchica, in cui ci sono dei provider che hanno delle reti interne (es.: la rete A ha le sottoreti del cliente P 128.96/16 e 192.4.153/24) e sono comunicanti tra loro tramite una backbone.



In questo esempio, il cliente P invia al proprio provider le informazioni sui percorsi che possiede, ovvero una sottorete 128.96/16 con "AS 4" e una sottorete 192.4.153/24 con "AS 4". Il provider A (AS 2) aggiunge le due entry alla propria tabella e inoltra le informazioni alla backbone.

Da notare che nei casi appena visti, e come già detto, i router che fanno da speaker NON forniscono informazioni su come entrare in una rete, né instradano direttamente loro stessi i pacchetti (non sono router di confine per far attraversare il traffico di pacchetti, o comunque non lo fanno se non espressamente incaricati di tale lavoro), quindi vi possono essere altri router incaricati di inoltrare il traffico di dati effettivi, mentre gli speaker aggiornano solo le tabelle di inoltro.

Problemi di BGP

È importante che i numeri di AS trasportati in BGP devono essere univoci. Ad esempio, nell'immagine precedente, l'AS 2 può riconoscersi nel percorso AS dell'esempio solo se nessun altro AS si identifica nello stesso modo. I numeri AS (ASN) sono numeri a 32 bit assegnati da un'autorità centrale (IANA e registri Internet regionali, RFC 4893) e sono circa 60.000 AS, al momento. Essi sono i nodi della rete che eseguono eBGP.

IP di nuova generazione (IPv6)

È un protocollo di rete alternativo a IPv4. Tutto quanto visto finora, continua a valere anche per IPv6, in quanto la grande differenza sta nel fatto che gli indirizzi IP non sono più da 4 byte, ma da 6 byte.

Tra le ragioni che hanno portato a questa variante c'è in primis il fatto che gli indirizzi IPv4 sono terminati. In fatti nel 2019 è stato assegnato da RIPE (l'ente che assegna gli indirizzi IP univoci a livello europeo), l'ultimo indirizzo IPv4 /22 (nel 2012 era stato assegnato l'ultimo /8). Questo significa che non è più possibile scalare orizzontalmente la rete, quindi a livello di numero di nodi.

Per cui è stato ridefinito il concetto di IP, utilizzando indirizzi a 128 bit, che forniscono uno spazio di indirizzamento dell'ordine di 3×10^{38} (a titolo di confronto: la superficie terrestre è di $5.1 \times 10^{14} m^2 = 5.1 \times 10^{20} mm^2$, quindi 0.58×10^{18} indirizzi per ogni mm^2).

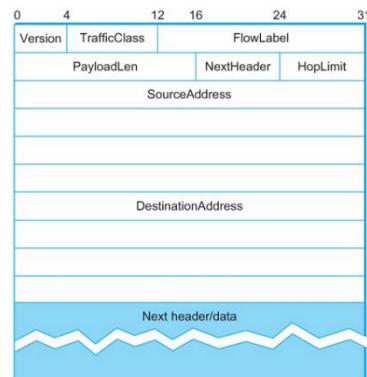
Inoltre, dato che le cose da modificare sarebbero state tante, si è sfruttata l'occasione per aggiungere in IPv6 servizi e funzionalità che IPv4 non aveva: multicast, servizio in tempo reale, autenticazione e sicurezza (un dispositivo IPv4 non era obbligato a fornire servizi di sicurezza, mentre IPv6 lo è, altrimenti non è conforme), autoconfigurazione, frammentazione end-to-end, funzionalità di routing migliorate (compreso il supporto per gli host mobili, ovvero staccare da una rete un dispositivo, spostarsi e riagganciarlo altrove, mantenendo lo stesso IPv6 di prima senza dover riconfigurare), estensibilità, ecc.

In IPv6 l'indirizzamento/routing è senza classi (simile a CIDR) e si usa la notazione: x:x:x:x:x:x:x:x (x = numero esadecimale a 16 bit). Gli 0 contigui sono compressi (es.: 47CD::A456:0124 oppure ::::1 per indicare sé stessi), mentre gli indirizzi IPv4 sono compatibili con IPv6 (es.: nella dicitura ::128.42.1.87). L'assegnazione degli indirizzi è basata sul provider (come per l'IPv4) oppure geografico (es.: l'Europa ne ha in un certo range e alcuni di essi sono per l'Italia).

Formato del pacchetto

Per quanto riguarda i pacchetti utilizzati in IPv6, l'intestazione "base" è di 40 byte e si hanno 64 Kbyte di carico utile (*payload*, con estensioni). Le intestazioni di estensione (ordine fisso, lunghezza per lo più fissa) hanno frammentazione, instradamento della sorgente, autenticazione e sicurezza e molti altri. In realtà l'header di IPv6 è più semplice di quello di IPv4, ma perché è stato pensato per essere modulare, dunque togliendo informazioni e spostandole come parametri extra (intestazione di estensione).

I primi 4 bit sono per la versione (come per IPv4), mentre gli 8 bit successivi (*trafficClass*) servono per un meccanismo simile al ToS, il *flowLabel* serve a identificare i pacchetti che fanno parte dello stesso flusso, il *payloadLen* è



per la lunghezza del pacchetto (max 64 Kbyte), il *nextHeader* è una sorta di puntatore che dice dopo di un certo numero di bit si ha un altro header (o se è 0 non c'è) e il *hopLimit* è il TTL.

Un'informazione che in IPv4 c'era, ma nell'intestazione base di IPv6 no è ad esempio il campo di frammentazione, che viene aggiunto nell'header aggiuntivo. In quest ultimo si trovano anche le opzioni sulla sicurezza o sul routing prefissato (che strada deve prendere il pacchetto, impostata manualmente).

Priorità

Il campo *trafficClass* definisce la priorità di un datagramma rispetto ad altri della stessa origine, ovvero è un suggerimento che si dà a un router per poter gestire meglio il pacchetto. È utile nel caso in cui alcuni datagrammi debbano essere abbandonati a causa di congestioni. I valori possibili sono:

- 0 = Traffico generico (valore di default)
- 1 = Traffico di background (es.: NNTP), per traffici asincroni non prioritari (es.: backup o aggiornamento SW mentre si stanno eseguendo altri servizi in foreground come l'update di un SO)
- 2 = Traffico senza attesa (es.: SMTP), si usa per le e-mail in cui anche se un messaggio arriva 5 minuti più tardi non è troppo un problema (di solito)
- 3 e 5 = riservato
- 4 = Traffico con attesa (es.: FTP, HTTP), quando si sta aspettando la risposta di un servizio in tempi ragionevoli
- 6 = Traffico interattivo (es.: TELNET, SSH)
- 7 = Traffico di controllo (es.: OSPF, RIP, SNMP, ecc.), serve per controllare ad esempio lo stato degli switch (es.: per gestire congestioni)
- 8-15 = Traffico non dipendente dalla congestione (non deve essere interrotto), ordinato per ridondanza (es.: audio/video) e sono realtime (es.: dati di sensori), quindi con alta priorità

Flow label

È un numero casuale a 20 bit, assegnato dalla sorgente. I datagrammi appartenenti allo stesso flusso sono etichettati con lo stesso valore. Questa etichetta può essere utilizzata dai router per implementare un servizio coerente a tutti i datagrammi dello stesso flusso (es.: stesso instradamento, oppure stressa qualità del servizio per il soft real-time, quindi l'utilizzo delle risorse riservate, che precedentemente erano riservate per mezzo di altri protocolli come RSVP, ecc.). Questo servizio è molto simile ai circuiti virtuali e è compito dei router capire come gestire al meglio i pacchetti in base al servizio richiesto. Di solito si usa all'interno del AS e potrebbe implicare la riconfigurazione dei router appartenenti allo stesso AS.

Transizione da IPv4 a IPv6

IPv4 e IPv6 non sono compatibili. In realtà IPv4 dovrebbe essere sostituito da IPv6, anche se questo è molto difficile: troppi host da aggiornare, non può essere fatto in una volta sola, non può essere forzato "per legge", ecc. Per cui esistono tre tecniche per supportare la transizione:

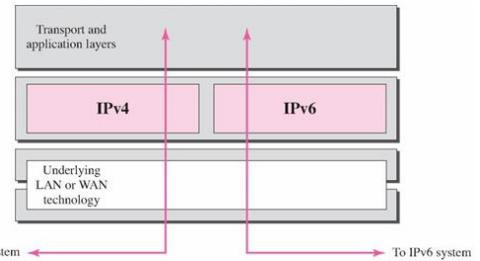
- Doppio stack di protocollo (*dual protocol stack*)
- Tunneling
- Traduzione dell'intestazione

Di seguito sono descritti brevemente.

Doppio stack di protocollo (*dual protocol stack*):

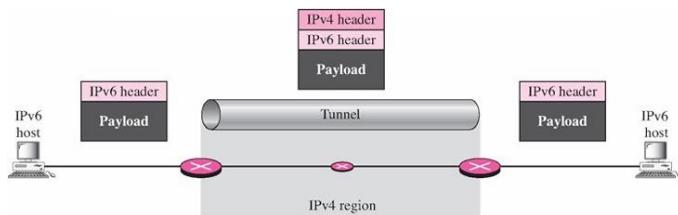
Un host (o un router) può implementare entrambi i protocolli (IPv4 e IPv6). Questo è utilizzato nella maggior parte dei sistemi operativi moderni grazie alla sua versatilità (IPv4 e IPv6 sono come se fossero binari paralleli, non sono compatibili).

La decisione su quale dei due utilizzare viene presa quando si deve stabilire una connessione. Inoltre, si usano due protocolli a livello 3 in quanto l'host appartiene a due reti internet separate. Le applicazioni devono gestire esplicitamente entrambi gli indirizzi, il che comporta più lavoro per gli sviluppatori e i distributori (es.: vedere Apache httpd.conf).



Tunneling:

Se due host usano entrambi IPv6 e devono attraversare un certo numero di router IPv4, è possibile creare "tunnel" o "ponte" da una regione IPv6 a un'altra, incapsulando i datagrammi IPv6 in datagrammi IPv4. Questo incapsulamento è dato dal fatto che gli indirizzi IPv6 sono pochi e isolati in mezzo a una marea di IPv4

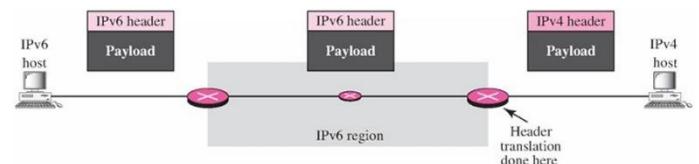


e se un router IPv4 vede un pacchetto IPv6, non lo riconosce e lo scarta. Dunque il payload è un intero pacchetto IPv6 che viene fatto passare per il "tunner" e il destinatario in questo caso diventa il router di destinazione (della rete dell'host) e non più l'host stesso. Questo approccio è gestito

in modo trasparente dai router e gli host possono comunicare in IPv6, a discapito di perdere alcuni vantaggi di IPv6 (vengono persi nella regione IPv4), in più si crea un maggiore overhead e frammentazione (i pacchetti IPv6 sono più grandi e vengono incapsulati ulteriormente).

Traduzione dell'intestazione:

Permette di trasmettere un datagramma IPv6 a un host solo IPv4. Il router di destinazione sostituisce l'header IPv6 con l'"equivalente" IPv4 e viceversa. Simile al NAT, ma la traduzione avviene tra indirizzi e intestazioni IPv4 e IPv6. Per fare ciò si usa un bridge (livello 3).



Internet Multicast

Questa funzionalità era presente anche in IPv4, solo che essendo un po' difficile da implementare non viene quasi mai utilizzata. Come già spiegato, con multicast si intende la possibilità di inviare un messaggio a un gruppo di host riceventi, senza conoscerli o specificarli.

Invece, con la sigla uno-a-molti (*one-to-many*) ci si riferisce a un multicast specifico della sorgente (SSM), quindi un host ricevente specifica un gruppo multicast in cui viene inviato un host specifico (es.: trasmissione di stazioni radio, notizie, prezzi delle azioni, aggiornamenti software a più host, ecc.).

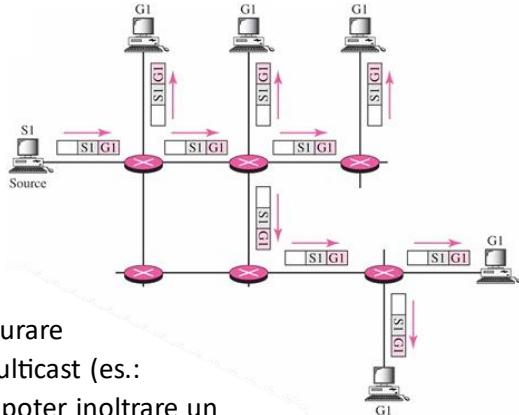
Mentre con molti-a-molti (*many-to-many*) si intende un multicast di qualsiasi sorgente (ASM), ad esempio le teleconferenze multimediali, i giochi online multigiocatore, le simulazioni distribuite, ecc.

Multicast in IP

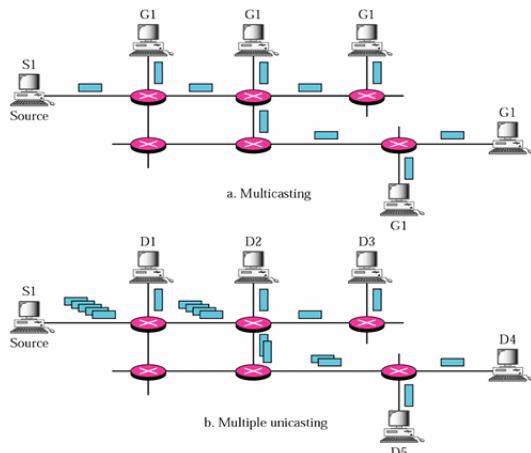
Utilizzando il multicast IP, è possibile inviare lo stesso pacchetto a ogni membro del gruppo. Dunque un host invia una singola copia del pacchetto, indirizzato all'indirizzo multicast del gruppo e poi di esso verrà data una copia per ogni singolo host membro del gruppo.

L'host sorgente non ha bisogno di conoscere l'indirizzo IP unicast di ciascun membro, in quanto i pacchetti vengono duplicati dai router lungo il percorso, quando necessario.

Tuttavia, se la rete fosse di grandi dimensioni, si dovrebbe configurare tanti router in modo che sappiano dove siano i vari indirizzi multicast (es.: nell'immagine accanto sono stati configurati cinque router per poter inoltrare un pacchetto a tutti gli host G1), il che diventa impossibile se la rete fosse a livello globale.



Tuttavia se non si avesse il supporto del multicast, una sorgente dovrebbe inviare un pacchetto separato con gli stessi dati a ciascun membro del gruppo e questa ridondanza consumerebbe più larghezza di banda. Inoltre il traffico ridondante non sarebbe distribuito in modo uniforme, ma si concentrerebbe vicino all'host mittente. In più, la sorgente dovrebbe tenere traccia dell'indirizzo IP di ogni membro del gruppo (potrebbero essere tanti!) e un gruppo può essere dinamico. Per cui utilizzare solo unicast sarebbe ancora di più uno svantaggio.

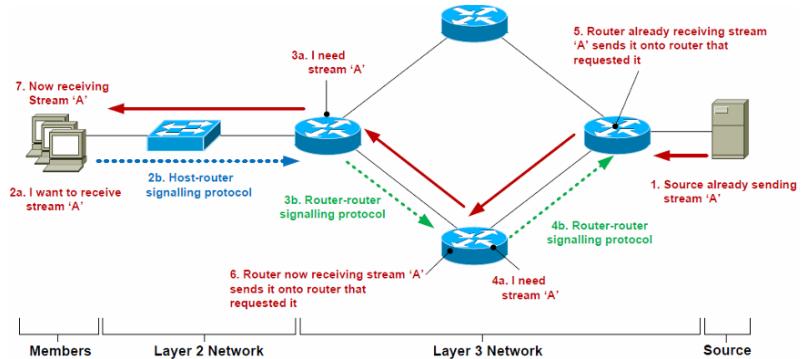


Per fortuna, IP fornisce un modello multicast molti-a-molti (a livello IP) basato su gruppi multicast. Ogni gruppo ha il proprio indirizzo IP multicast nella classe D (da 224.0.0.0 a 239.255.255.255). Gli indirizzi dei gruppi sono assegnati in diversi modi:

- 224.0.0.0/24: l'uso è definito dalla IANA, ma è limitato alla rete locale.;
- 224.0.1.0/24: gruppi globali, assegnati staticamente dalla IANA (es.: 224.0.1.1 è NTP)
- Affittando dinamicamente per il tempo di una sessione, utilizzando il protocollo SAP/SDP (es.: per sessioni di videogiochi online multiplayer)
- ...

Per quanto riguarda la gestione dei gruppi multicast, un host può entrare e uscire dai gruppi come anche un host può far parte di più gruppi. Per cui un host segnala il suo desiderio di entrare o uscire da un gruppo multicast comunicando con il suo router *locale* utilizzando uno speciale protocollo. In IPv4 si usava il protocollo di gestione dei gruppi Internet (IGMP), mentre in IPv6 c'è il *Multicast Listener Discovery* (MLD). Il router ha la responsabilità di far sì che il multicast si comporti correttamente nei confronti dell'host.

Si ipotizzi di avere una certa sorgente che sta già inviando un flusso di pacchetti a un indirizzo multicast e un host effettua la richiesta di unione a tale gruppo multicast per cui invia un pacchetto IGMP al router locale. Questo router inoltra la richiesta con l'indirizzo IP del multicast che l'host necessita agli altri router vicini, fino ad arrivare a un router intermedio che fa da streaming di tale sorgente. A quel punto i pacchetti della sorgente vengono inoltrati anche al nuovo host tramite i router intermedi e l'host può a sua volta inviare pacchetti al multicast a cui si è appena unito. Tutto questo avviene a insaputa della sorgente.



Se a un certo punto l'host non volesse più ricevere pacchetti dal multicast, invia una richiesta ICMP al proprio router locale e questo la fa arrivare al router intermedio che fa da sorgente al flusso di pacchetti. Quest ultimo capisce che non serve più inviare il flusso in quella direzione e smette di inviarne su quel ramo della rete.

[Nella registrazione S04E03 c'è un esempio di chat in UDP che usa il multicast, in cui si usa la `setsockopt` (...) per inviare la richiesta ICMP, `recvfrom` (...) per aspettare un messaggio dalla socket (bloccante), `sendto` (...) per inviare un buffer di dati sulla socket, ...]

La portata dei messaggi può essere definita scegliendo un TTL appropriato. Le questioni di sicurezza (es.: segretezza, integrità, autenticazione, controllo degli accessi, ecc.) non sono gestite a questo livello (possono essere gestite a livello di sessione/applicazione).

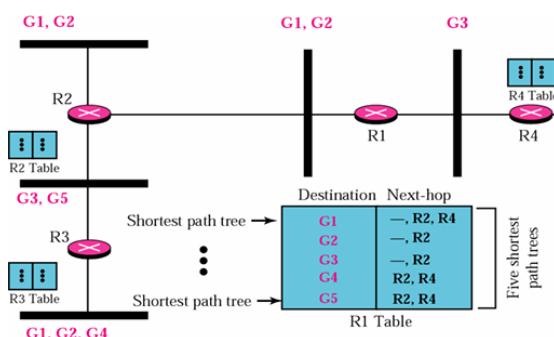
Le tabelle di inoltro unicast (*unicast forwarding tables*) di un router indicano, per qualsiasi indirizzo IP, quale link utilizzare per inoltrare il pacchetto unicast. Per supportare il multicast, un router deve disporre inoltre di tabelle di inoltro multicast (*multicast forwarding tables*) che indicano, in base all'indirizzo di destinazione multicast, quali collegamenti utilizzare per inoltrare il pacchetto multicast. Le tabelle di inoltro multicast specificano collettivamente una serie di alberi, uno per ogni gruppo, e sono chiamati alberi di distribuzione multicast (*multicast distribution trees*). L'instradamento multicast (*multicast routing*) è il processo con cui vengono determinati gli alberi di distribuzione multicast.

Attualmente non ci sono dei criteri ben precisi su come costruire e popolare le tabelle di inoltro multicast in maniera efficiente e scalabile a livello globale (rispetto all'unicast sono più complessi). Per cui nel tempo sono stati proposti molti protocolli di routing (es.: DVMRP, RPB, PIM, ecc.) e due grandi famiglie di alberi: quelli basati sulla sorgente (*source-based trees*) e quelli condivisi dal gruppo (*group-shared trees*).

Routing multicast basato sulla sorgente (source-based)

Nell'approccio ad albero basato sulla sorgente, ogni router deve avere un albero del percorso più breve per ogni gruppo. Quindi ogni router si costruisce una propria visione della rete tramite una tabella di inoltro, che in realtà non contiene i percorsi completi, ma presenta solo i router necessari al forward (*next hop*). Infatti, in questo contesto si ha a che fare coi soli router direttamente coinvolti, ovvero quelli che stanno servendo delle reti all'interno delle quali ci sono degli host appartenenti al gruppo multicast.

Questo è dato dal fatto che, mentre per l'instradamento unicast si necessita del path completo per sapere quale host univoco servire, qui si ha un gruppo generico.



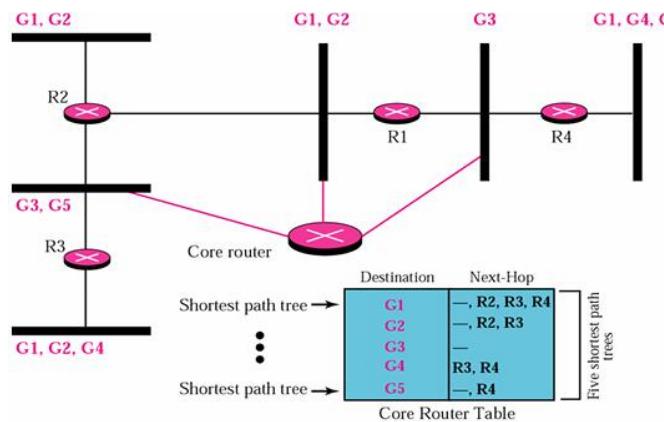
Nell'esempio in figura, si ha la tabella di inoltro multicast del router R1 e per ogni gruppo si segna quali router (*next hop*) attraversare per far raggiungere al pacchetto la sua destinazione host (dove c'è un “—” si intende la rete locale).

Instradamento multicast ad albero condiviso per gruppi (group-shared trees)

Nell'approccio group-shared tree, solo un router (chiamato *core* o *rendezvous* router) ha un albero del percorso più breve per ogni gruppo ed è coinvolto nel multicasting. Esso tiene traccia di tutto il traffico di pacchetti e non serve che se ne occupino gli altri.

Per effettuare una richiesta di inserimento nel gruppo o per mandare un messaggio in multicast, si deve mandare un pacchetto in unicast al router core/rendezvous e poi questo si occupa di gestire la richiesta.

Soltanamente questo tipo di soluzione è più facile da gestire, ma non è sempre scalabile.



Protocollo di instradamento multicast a vettore di distanza (DVMRP)

Il primo protocollo di routing multicast implementato (RFC 1075, novembre 1988) è il *Distance-Vector Multicast Routing Protocol* (DVMRP). Estende i protocolli di routing distance vector (= vettori di distanza, già visti) per il routing unicast e segue la strategia "*Flood and prune*": inonda la rete di traffico multicast e nel frattempo pota i rami non interessati al traffico.

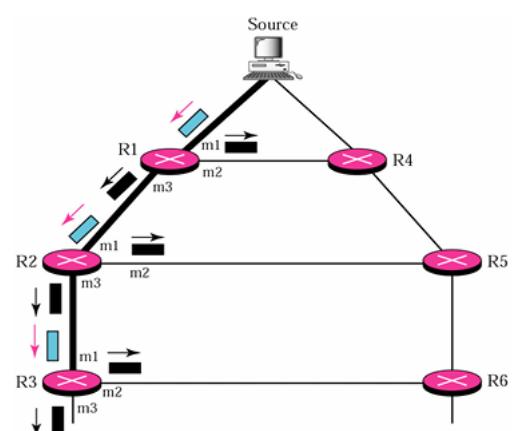
Questo protocollo dunque, sfrutta i vettori di distanza (già forniti/creati) e vi aggiunge sopra delle informazioni ausiliarie per il "flood and prune". Quando si crea un gruppo multicast, per prima cosa si "innonda" la rete inoltrando a tutti gli host della AS il traffico di pacchetti (simile al broadcast) e poi si procede col tagliare (escludere) i "rami secchi", ovvero i rami della rete che non sono interessati al traffico generato (essi rispondono esplicitamente di non esserne interessati). Ciò che resta e che viene mantenuto è il vero albero che interessa al protocollo.

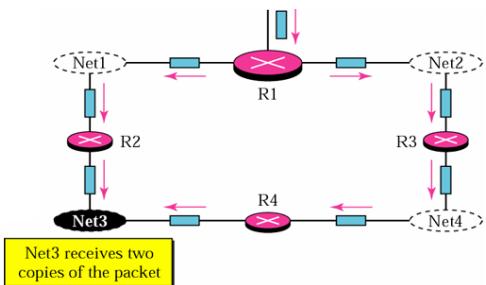
Flooding:

Per quanto riguarda il flooding vero e proprio, si cerca di inviare i messaggi in giusta quantità, senza esagerare, tramite una tecnica chiamata inondazione del percorso inverso (*Reverse Path Flooding*), il cui scopo è quello di inviare dalla sorgente un certo quantitativo di pacchetti in modo tale da arrivare a tutti i nodi e una volta sola (per questo motivo si è appena detto di non si deve esagerare: un flooding qualsiasi ripeterebbe i path).

Utilizzando gli algoritmi preesistenti, già visti con RIP per l'unicast, si sa quali sono i next hop da passare per arrivare a tutti i nodi della rete, per cui si sfruttano le tabelle di instradamento questi.

Ogni router quindi sa se deve inoltrare il pacchetto o meno, in base a quale indirizzo e porta sono stati utilizzati per fargli arrivare il pacchetto. Per cui esso guarda da dove è arrivato il pacchetto e se vede che arriva dal percorso che lui farebbe per arrivare alla sorgente (percorso più breve dalla sorgente), allora procede con l'inoltro su tutte le altre porte (esclusa quella del pacchetto appena arrivato), altrimenti non forwarda nulla. In questo modo, non ci sono loop nel processo di flooding.





Tuttavia, le reti con più di un router possono ricevere pacchetti broadcast duplicati (come nell'immagine affianco). Per eliminare i pacchetti duplicati si fissa un router “genitore” (relativo alla sorgente) per ogni LAN e si lascia che solo i genitori inoltrino i pacchetti. I router genitori sono quelli con il percorso più breve verso la sorgente (che conoscono tramite il distance vector). Nel caso affianco, R2 è il genitore per Net3. Si rompono poi i legami scegliendo il router con TTL minore o con l'indirizzo IP più piccolo nel caso di pareggio del TTL (questo è simile al protocollo Spanning tree nelle LAN commutate). Questa strategia, chiamata *Reverse Path Broadcast*, garantisce che ogni LAN riceva esattamente una copia di ogni pacchetto (senza perdita di pacchetti), lungo un albero del percorso più breve basato sulla sorgente.

Prune:

Con la fase di pruning, si “potano” (= *prune*) le reti che non hanno host interessati al gruppo multicast. Per cui si escludono dei rami in modo da non espandere il flooding più del necessario. Questa fase si effettua in due fasi.

Nella prima fase si determina se la LAN è una foglia dell’albero di distribuzione con nessun membro nel multicast (non ci sono host interessati). Una LAN è una foglia se il suo genitore è l’unico router della LAN. Gli host della LAN che vogliono partecipare a un multicast devono notificarlo al router utilizzando IGMP (periodicamente altrimenti vengono rimossi dopo un timeout). In questo modo, il router sa se la LAN ha o non ha membri nel multicast.

Nella seconda fase si propaga l’informazione di “nessun membro del multicast qui”. Quindi si aumenta ogni voce <Destinazione, Costo> nei vettori di distanza inviati ai vicini con l’elenco dei gruppi per i quali questa rete è interessata a ricevere pacchetti multicast. In questo modo, ogni vicino sa se deve considerare questo router nel percorso inverso di flooding/broadcast, osservando i gruppi a cui è interessato.

Tuttavia, includere sempre l’elenco dei gruppi interessati nei vettori di distanza può essere pesante e inutile (es.: nel caso in cui una LAN sia interessata a diversi gruppi, ma nessun host stia trasmettendo su questi gruppi). Invece, il router aggiunge ai vettori di distanza l’elenco dei gruppi a cui NON è interessato, e solo quando l’indirizzo multicast diventa attivo (qualcuno trasmette).

Quindi, in riassunto, quando inizia una trasmissione multicast sul gruppo:

- all’inizio si inonda tutta la rete, tramite RPF/RPB, costruendo un albero di distribuzione che copre praticamente tutta la rete (perché nessuno dice ancora di non essere interessato a G);
- poi l’albero viene potato, ovvero i router non interessati al multicast iniziano a comunicare di non trasmettere più a loro;

Se un host di una rete potata/tagliata si unisce al gruppo multicast in un secondo momento, notifica il suo router locale utilizzando IGMP e il router inizia ad accettare il traffico dai suoi vicini (innesto = *grafting*). Nel complesso, il DVMRP funziona bene su piccola scala (intra-dominio) e meno su grande scala a causa di questo continuo flooding e pruning.

Multicast indipendente dal protocollo (PIM)

Questo protocollo nasce in seguito al precedente e è indipendente dalla costruzione della topologia di rete (da risolvere con un protocollo diverso), infatti è basato su RIP. Il *Protocol Independent Multicast* (PIM) ha due modalità diverse:

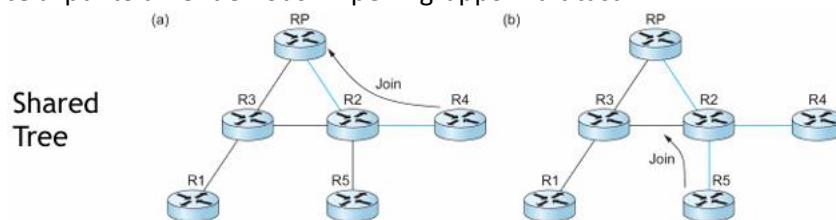
- PIM-DM (*dense mode*, RFC 3973, [2005](#)): utilizzato quando molti host sono origine di dati rispetto ai router (es.: all’interno di una LAN o di un AS)

- PIM-SM (*sparse mode* = sparsa/rada, RFC 2362, [1998](#)): utilizzato quando il multicast coinvolge pochi nodi della rete rispetto al numero di router (es.: pochi host in Internet o reti sconnesse)

Questo protocollo è in grado di passare da una versione a un'altra, adattandosi alla rete.

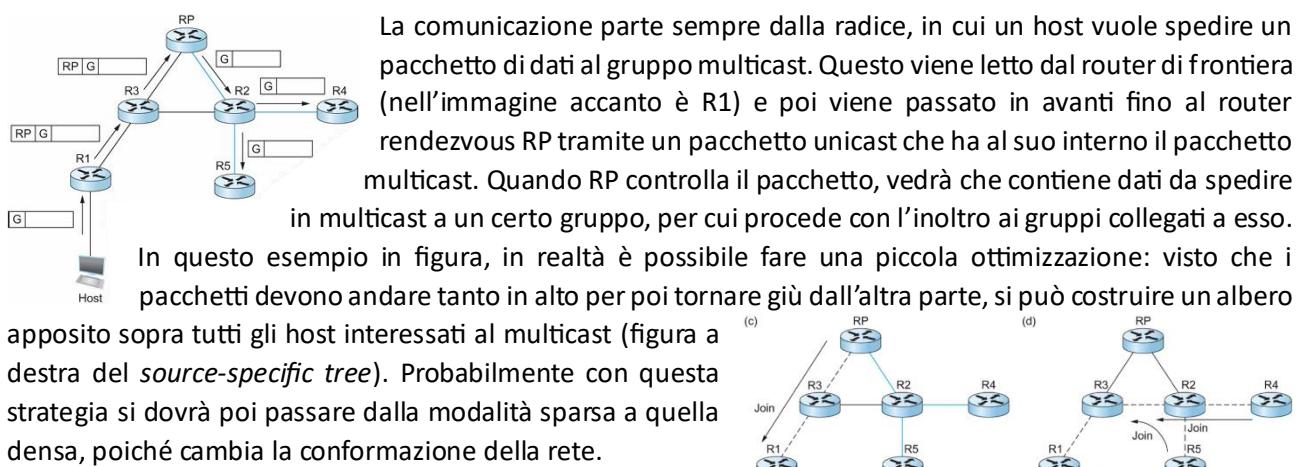
La modalità densa è simile al DVMRP, ma è indipendente dal protocollo di routing unicast sottostante (dversamente dal DVMRP, che è legato ai protocolli distance vector, questo si slega dai vettori). Utilizza alberi basati sulla sorgente, in cui ogni nodo (router) conserva la propria copia degli alberi di distribuzione multicast, e la strategia flood-and-prune, in cui si usa RPF per distribuire il traffico e pruning/grafing per gestire gli alberi. Funziona bene quando il gruppo è piccolo e non deve essere attraversato da troppi router.

La modalità sparsa (più vecchia della precedente) può essere scalata su grandi gruppi, che sono in realtà scarsi su Internet (es.: è utilizzato per l'IPTV). Per ogni gruppo, viene costruito un albero condiviso di gruppo e in ogni AS viene eletto un router “rendezvous”. I router (a seguito di una richiesta IGMP da parte di qualche host locale) possono unirsi agli alberi condivisi inviando una richiesta di “join” (unicast) al loro punto di rendezvous. Durante l'attraversamento dei router, il messaggio di “join” al gruppo multicast crea un albero di distribuzione multicast, con radice al punto di rendezvous RP per il gruppo multicast.



Ogni router analizza il messaggio e aggiunge alla propria tabella la regola per inoltrare verso il basso il traffico del gruppo multicast lungo l'interfaccia da cui proviene il messaggio di “join”. Se il router non partecipa già all'albero (a), allora inoltra la richiesta di “join” verso RP e segna l'interfaccia corrispondente come l'unica da cui può provenire il traffico; altrimenti (b) non fa nulla.

In questa maniera i router “imparano” la strada da far seguire ai pacchetti multicast per fare in modo che arrivino a destinazione. Essi segnano nelle proprie tabelle di inoltro il percorso, fino a quando non scade la richiesta dell'host di restare nel gruppo multicast. Così quando si aggrega un altro host al flusso multicast, i router hanno già la strada segnata.



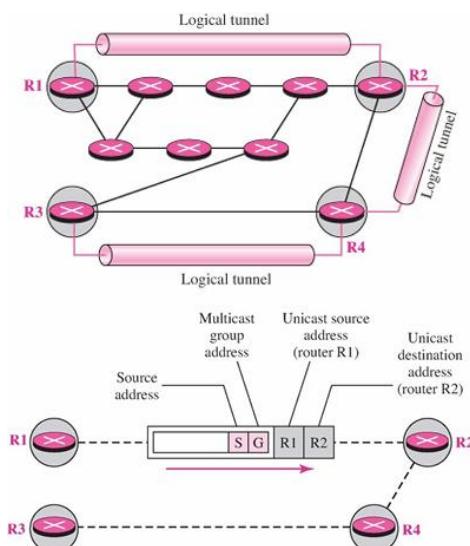
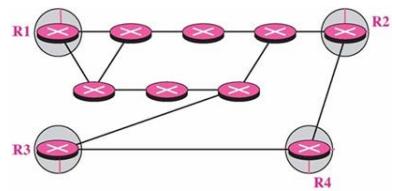
Riassumendo, una volta creato l'albero condiviso:

1. L'host invia un pacchetto al gruppo multicast sulla sua LAN
2. Viene ricevuto dal router progettato (qui R1), oltre che da qualsiasi altro host locale
3. Il DR lo sintonizza verso l'RP, incapsulato in un normale pacchetto IP unicast
4. Il RP riceve il pacchetto, lo apre e lo inoltra lungo l'albero condiviso (qui a R2, poi a R4 e R5)

Si noti che si può assumere che i router tra DR e RP non partecipino all'albero (potrebbero non implementare affatto il multicast, come spesso accade su Internet); solo il DR deve sapere qual è l'RP.

Backbone multicast (MBONE)

In realtà ad oggi, solo una piccola parte dei router implementa il routing multicast. Gli amministratori non sono propensi ad ammettere il traffico multicast a causa del loro costo nei router e soprattutto molto spesso questi router non sono contigui, cioè sono collegati con regioni che non sono multicast-aware. Infatti, nell'esempio in figura accanto, R1, R2, R3 e R4 sono router multicast-aware, ma tutti gli altri e nelle reti reali non lo sono (si limitano a far cadere tutto il traffico multicast).



Una soluzione "temporanea" può essere quella di creare dei tunnel unicast tra i router multicast, attraversando i router non multicast. I tunnel sono collegamenti logici e agiscono come una dorsale per il multicast, chiamata MBONE (*multicast backbone*). I pacchetti multicast sono incapsulati all'interno di normali pacchetti IP unicast e possono passare attraverso i router multicast-unaware come di consueto. Quindi, i protocolli di routing multicast possono essere eseguiti sull'MBONE.

In realtà, MBONE (e il multicast in generale) non sono molto diffusi, più che altro per problemi di accounting, gestione del traffico, traffico sui provider, si aggiunge più carico sui router, ecc. Storicamente, il più grande evento mai trasmesso in diretta è stato il concerto dei Rolling Stones a Dallas, 1994 (50.000 clienti).

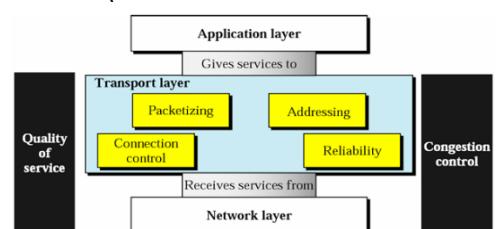
Inoltre, MBONE è obsoleto: sta per essere sostituito da IPv6 con PIMv6. Tuttavia il multicast viene comunque adottato all'interno di singole organizzazioni (a livello di LAN e AS), ad esempio per il trasporto di contenuti IPTV all'interno di un dominio di hotel/ospedale/provider ecc.

Protocolli end-to-end

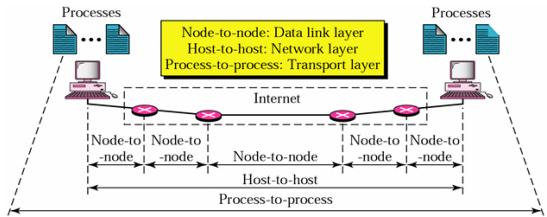
I protocolli end-to-end sono dei protocolli a livello di trasporto e riguardano il *Simple Demultiplexer* (UDP) e il *Reliable Byte Stream* (TCP). Nel livello 3 si è visto come poter inviare pacchetti da un host a un altro situati in punti qualsiasi di una rete molto grande (astraiendosi da dove e come essi siano collegati fisicamente), tuttavia non si è approfondito come evitare di perdere pacchetti, come riordinarli, come inviare duplicati del messaggio, limitare le dimensioni dei messaggi, ecc. Per cui al livello 4 si cerca di gestire queste lacune.

Livello di trasporto

In questo livello si sviluppano algoritmi che trasformino le proprietà desiderabili della rete sottostante nell'alto livello di servizio richiesto dai programmi applicativi. Alcune funzionalità (come il QoS e il controllo della congestione) non si adattano a un singolo livello, ma richiedono piuttosto la collaborazione tra i livelli (*cross-layer*). In alcuni casi, invece, non si ha un solo programma che gira, per cui si deve gestire anche un carico di lavoro maggiore e diversificato (es.: multitask, thread, ecc.). Anche la tematica della sicurezza ha a che fare con questo livello (in realtà essendo QoS sono più di uno).



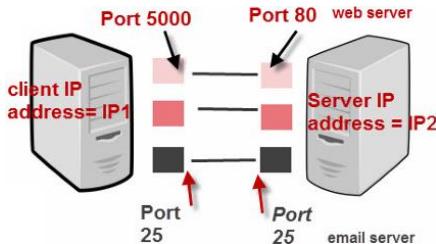
Come è possibile vedere dall'immagine sopra, i compiti principali del livello di trasporto sono la pacchettizzazione (trasformazione in pacchetti e invio tramite flusso), l'indirizzamento (a quale processo ci si sta rivolgendo e indipendentemente dalla piattaforma), l'affidabilità (dati che arrivano in ordine, senza duplicati, ecc.) e il controllo sulla connessione (stato tra chi invia e chi riceve, nonché il controllo di flusso).



I protocolli end-to-end implementano le comunicazioni da processo a processo (all'interno degli host) e possono essere orientati alla connessione o senza connessione. Molti endpoint all'interno di un nodo fanno sì che sia necessario un altro livello di indirizzamento per identificarli.

Indirizzamento end-to-end in TCP/IP

Il modo per verificare e identificare i processi a livello di Internet è attraverso il conetto di porta. Il numero di porta è un numero compreso tra 0 e 65535 (16 bit) e sono da considerarsi come porte fisiche, ovvero dispositivi fisicamente collegati alla macchina. In ogni connessione, sono coinvolti due numeri di porta, uno



per ogni processo sui due host in comunicazione (non è necessario che siano gli stessi numeri), oltre, agli indirizzi degli host su cui i processi sono in esecuzione. Per cui se si hanno molti processi su un host, si necessita di molte porte sullo stesso indirizzo di rete.

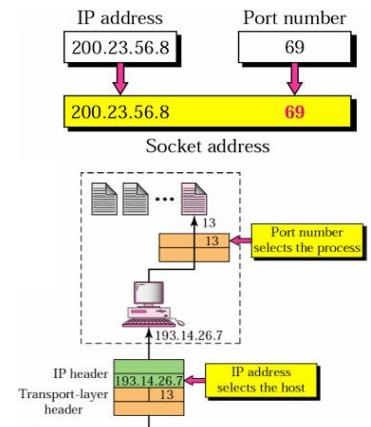
Per identificare univocamente un processo, un'idea poco furba può essere quella di fornire al numero di porta lo stesso numero del processo, ma questo sarebbe possibile solo se tutti i dispositivi del mondo si mettessero d'accordo sulla numerazione dei processi. Dunque, non è fattibile, ma in seguito si vedrà una tecnica più facile.

Socket

Un endpoint di comunicazione tra due host è chiamato socket e è creato tramite opportune chiamate di sistema. La socket è quella che si solito si scrive come *indirizzo_host:numero_porta* (es.: 200.6.0.12:86), infatti il suo indirizzo è la coppia <IP, porta>. L'indirizzo IP identifica l'host, mentre il numero di porta identifica il processo all'interno dell'host.

Due processi, per comunicare, devono avere ciascuno un socket e utilizzare lo stesso protocollo di trasporto. In ogni istante, una comunicazione è completamente identificata dalla tupla: < $IP_A, port_A, IP_B, port_B, protocollo$ >.

È da tenere presente che quando si vede un numero del tipo *IP:porta* non è da intendersi come indirizzo dell'host, ma è l'indirizzo di una specifica porta su quell'host che presumibilmente è utilizzata da un processo su quell'host.

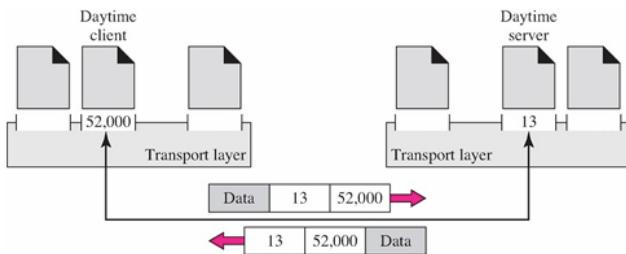


Modello client/server

Spesso due processi comunicano secondo il modello client/server. Le due entità sono di seguito descritte:

- Server: è un processo che offre un servizio (es.: l'accesso a una risorsa), è sempre in esecuzione (da quando viene offerto il servizio) e è in attesa delle richieste del cliente (ruolo *passivo*)
- Cliente: è un processo desideroso di utilizzare il servizio, può essere attivo solo quando necessario e invia richieste al server (ruolo *attivo*)

Da notare che i client e i server sono processi, non host.

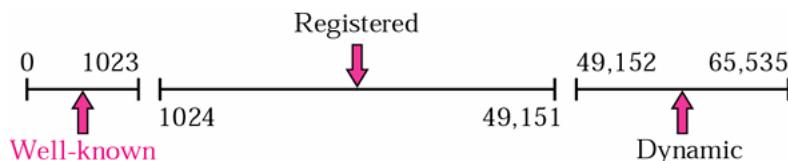


Per iniziare la comunicazione, il client deve essere in grado di conoscere la porta del server, mentre il server è in grado di conoscere l'indirizzo del client quando viene contattato. Spesso il server utilizza una porta ben nota (es.: il server Web usa la porta 80, il server e-mail la 25, ecc.), mentre il client utilizza porte dinamiche (effimere) scelte in modo casuale e possibilmente solo per ogni comunicazione.

Per questa ragione, la IANA ha stabilito degli intervalli di porte:

- Porte note: utilizzate dai server ufficiali. Possono essere utilizzate solo da processi di livello/diritti amministratore (garantiscono una forma limitata di autenticazione del server, es.: per SSH)
- Porte registrate: possono avere un uso standard, ma non limitato ai processi di amministrazione (es.: i DBMS usano questi tipi di porte, oppure la 3306 è di MySQL, ecc.)
- Dinamiche: libere per tutti, di solito assegnate dal kernel ai client che non specificano alcuna porta

Per vedere su terminale delle configurazioni delle porte si esegue il comando `less /etc/services` (oppure vedere RFC 1700).



Tipi di comunicazione

Esistono diversi tipi di comunicazione implementabili con le socket e tra le classificazioni principali ci sono:

- Orientata alla connessione (*connection oriented*): è necessario stabilire una connessione prima di scambiare dati, il che è più costoso, in quanto si deve creare e mantenere uno stato ricco di informazioni lungo tutta la durata della comunicazione. I datagrammi appartengono a un'unica connessione, quindi sono più controllati e di conseguenza più affidabili (ma è più lento il setup). Nel modello TCP/IP è implementato da TCP (e SCTP, che è un'evoluzione del TCP, ma viene usato pochissimo)
- Senza connessione (*connectionless*): in questo caso si va a replicare col minimo sforzo il modello della rete del livello 3 sottostante, quindi ogni messaggio (datagramma) viene inviato senza l'apertura di una connessione esplicita, ottenendo una risposta più rapida (infatti, è adatto a comunicazione soft-realtime, in cui è più importante la reattività dell'affidabilità). Ogni datagramma è indipendente e solitamente è inaffidabile (stesso tipo di servizio di IP). Nel modello TCP/IP è implementato da UDP

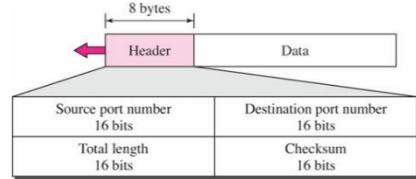
Altrimenti altri tipi di comunicazioni (non completamente) implementati, sono:

- Senza connessione affidabile (*reliable connectionless*): è teoricamente possibile, ma in realtà è raramente implementato. Può essere simulato utilizzando TCP o aggiungendo controlli a UDP. I pacchetti sono affidabili e controllati (senza duplicati, ecc.). Nel modello TCP/IP è implementato tramite RUDP o RDP (non molto diffuso)
- Richiesta/risposta (*request/reply*): ogni richiesta del client è seguita da una corrispondente risposta del server (es.: richiesta di una pagina HTTP e risposta dal server, oppure richiesta di un file e risposta). Ogni richiesta/risposta è indipendente dalle altre e la richiesta può essere rifiutata. È una tecnica molto diffusa nelle architetture di comunicazione attuali (circa 90%). Nel modello TCP/IP è implementato nei protocolli a livello di applicazione (es.: RPC, RMI, SOAP, REST, ecc.)

|| UDP

Il protocollo UDP (*User Datagram Protocol*) è definito anche come *demultiplexer semplice*. Esso estende il servizio di consegna da host a host della rete sottostante in un servizio di comunicazione da processo a processo e aggiunge un livello di demultiplexing che consente a più processi applicativi su ciascun host di condividere la rete. Non vi è alcuna garanzia oltre a quelle di IP (a parte un po' di buffering).

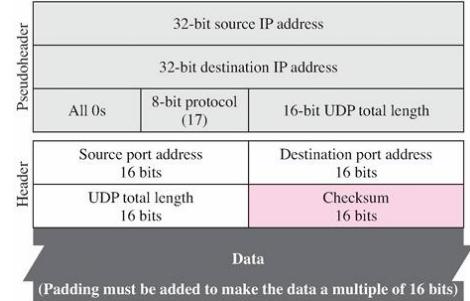
Il formato del datagramma UDP è quello nell'immagine accanto, in cui vi è un semplice header e poi il payload coi dati. La lunghezza totale può essere dedotta dall'intestazione della rete, ma è meglio che sia presente anche in questo formato.



|| Calcolo del checksum

Il checksum, se presente (cioè non nullo), è calcolato su *intestazione UDP + dati + intestazione pseudo-IP*. L'intestazione pseudo-IP è un "riassunto" dell'intestazione IP vera e propria, contenente solo: indirizzi IP, protocollo (numero 17 per UDP), lunghezza totale del datagramma IP, mentre altri dati (es.: frammentazione, TTL, ecc.) sono omessi. Esso identifica la comunicazione, ovvero la quintupla di *indirizzo + porta* e serve a identificare la comunicazione. Questo header in realtà non viene trasmesso: viene usato soltanto per il calcolo del checksum, quindi chi invia il pacchetto (datagramma), per calcolare il checksum utilizza questi dati, ma poi non li trasmette. Quindi, nella realtà viene trasmessa l'intestazione vera (di 20 byte).

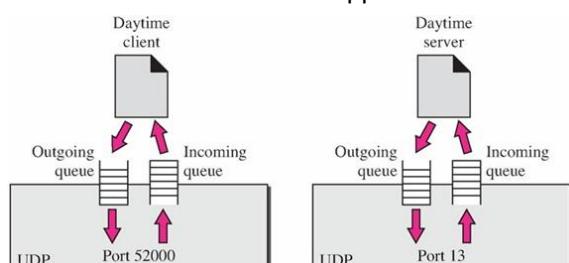
Se durante la connessione venisse cambiato l'indirizzo IP del mittente, bisognerebbe ricalcolare il checksum (pacchetto IP e UDP).



|| API per i servizi UDP

Si supponga che un pacchetto UDP venga accettato. Esso viene quindi inserito in appositi buffer, infatti è possibile immaginare il modello UDP come delle sorte di code di datagrammi. I buffer utilizzati sono uno di ingresso e uno di uscita, in modo da poter far arrivare i datagrammi, processarli e poterne costruire uno di nuovo per rispondere/inoltrare.

Infatti, quando è legata a una porta, ogni socket ha una coda di ingresso e di uscita, mantenuta nel kernel. Le code sono accessibili dalle applicazioni chiamando system call come `sendmsg()` o `recvmsg()`. Quando si

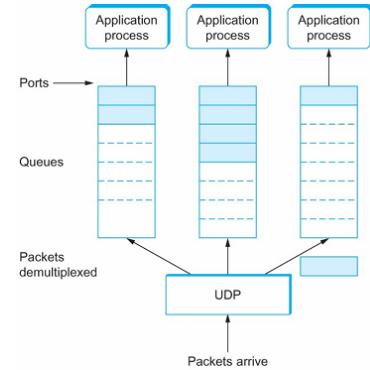


esegue la `recvmsg()`, se nella coda di input è presente almeno un datagramma, questo viene spostato su buffer interni per essere processato, altrimenti se è vuota l'host resta bloccato in attesa di un nuovo datagramma (infatti la `recvmsg()` è bloccante). Per questa ragione entra in gioco il concetto di sincronizzazione tra client e server. La `sendmsg()`, invece, non è bloccante, quindi non appena serve inviare un nuovo datagramma, l'operazione viene soddisfatta subito senza ostacolare altre funzioni.

Quando a un server arriva una nuova comunicazione da un nuovo client, esso processa subito la risposta e la restituisce nella medesima porta letta nel pacchetto in arrivato.

Demultiplexer semplice

Quando arrivano all'host, i datagrammi vengono controllati e inseriti nella coda associata alla porta indicata. I processi applicativi consumano i pacchetti in ordine FIFO e un intero pacchetto alla volta. La dimensione delle code non è scritta da nessuna parte, per cui potrebbero anche essere limitate. Infatti, se una coda non avesse spazio libero, il datagramma verrebbe scartato silenziosamente, come anche quando la porta non è associata a una coda (cioè nessun processo sta utilizzando la porta), il datagramma viene scartato. I datagrammi provenienti da sorgenti diverse e diretti alla stessa porta possono essere intercalati.



Anche per queste ragioni scritte appena sopra, potrebbe essere che un pacchetto non arrivi a destinazione. UDP non ha controlli di flusso, per cui in caso ad esempio di streaming audio/video, se il buffer si saturasse poiché la sorgente invia più pacchetti di quelli che possono essere contenuti nel buffer del client, essi devono essere scartati e eventualmente richiesti nuovamente dal destinatario (i controlli su errori/fallimenti o duplicati sono gestiti all'interno del processo manualmente, NON dal protocollo UDP in sé).

Inoltre, i datagrammi non sono numerati e non appartengono a una sessione/connessione, per cui sono consegnati in modo indipendente. Non vi è nessun controllo di errore, dunque il mittente non viene avvisato se i datagrammi vengono persi o se il destinatario è congestionato (casomai il destinatario richiede per conto suo delle informazioni mancanti). Vi è infine il solo incapsulamento in IP, eventualmente con frammentazione.

UDP, quindi, è orientato alle transazioni (*transaction-oriented*), adatto a semplici protocolli di richiesta-risposta come il *Domain Name System* (DNS) o il *Network Time Protocol* (NTP), è semplice, stateless, buono per il bootstrap o per altri scopi senza uno stack di protocollo completo (es.: DHCP e TFTP) e adatto a un numero molto elevato di client, come nelle applicazioni di streaming multimediale (es.: IPTV), applicazioni in tempo reale (es.: *Voice over IP* o giochi online, dove può essere meglio perdere un solo datagramma, piuttosto che ritardare la consegna dei dati ai processi in attesa di quelli mancanti, ecc.), comunicazione unidirezionale (es.: per comunicazioni broadcast e multicast, dove non è possibile avere uno stato con un gruppo di host, come la scoperta di servizi e la condivisione di informazioni), implementazione di specifici protocolli di trasporto end-to-end nello spazio utente (es.: QUIC di Google, RPC di Sun).

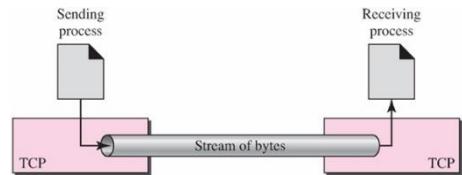
Alcuni servizi di UDP sono poco utilizzati per la loro scarsa utilità, come *Echo* e *Daytime*, mentre altri come *DNS* per ad esempio siti web, oppure *Bootps* e *Bootpc* per fare il bootstrap via rete (es.: host stateless, quindi senza sistema operativo a bordo, come per l'aggiornamento dei firmware dei router, che hanno solo la RAM), in cui si chiede al momento dell'accensione se qualcuno ha un sistema operativo da fornire e se nella LAN esiste un dispositivo che ha un servizio per quella macchina glielo trasferisce (viene memorizzato in RAM). Il NTP si usa per la sincronizzazione degli orologi dei dispositivi (es.: PC o telefoni) tramite un server apposito.

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	Bootps	Server port to download bootstrap information
68	Bootpc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

TCP

È stato implementato nel 1974 dalla IEEE e a differenza di UDP, il protocollo di controllo della trasmissione (*Transport Control Protocol*, TCP) offre servizi su flusso di byte affidabili (*reliable byte stream*), affidabilità e orientamento alla connessione.

Costruisce una sorta di imbuto/pipe tra un processo e un altro per instaurare una connessione. Questa astrazione garantisce che i dati vengano inviati tutti, in ordine e senza ripetizioni. TCP gestisce da sé tutte le problematiche relative al recupero di eventuali dati andati persi e di ordinamento dei segmenti dei dati, di cui IP (simile a UDP) non si occupava. In realtà una socket ha due pipe, una di andata e una di ritorno, quindi è bidirezionale.

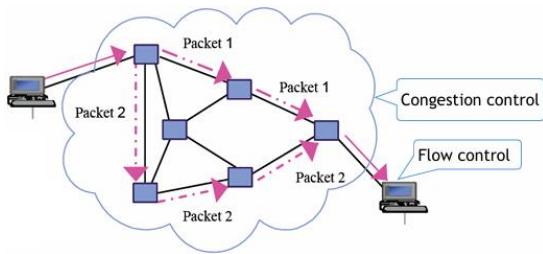


Controllo del flusso vs controllo della congestione

TCP, quindi si occupa di:

- Controllo del flusso: impedisce al mittente di superare la capacità del ricevitore (*buffer saturo*) e era una tecnica presente, in parte, nel livello 2 (*protocollo della finestra scorrevole*) anche se il controllo vero e proprio viene delegato e gestito ai/dai livelli superiori.
- Controllo della congestione: consiste nell'impedire che una quantità eccessiva di dati venga immessa nella rete, causando così un sovraccarico degli switch o dei link. La congestione è quando un router viene raggiunto da diversi flussi di pacchetti e che non riesce a gestire, con la conseguenza che è costretto a dover scartare i pacchetti in ingresso. La congestione è all'oscuro di mittente e destinatario, in quanto solitamente riguarda i router intermedi nel percorso

Il TCP deve affrontare entrambi i problemi e di conseguenza, la larghezza di banda e i ritardi end-to-end sono definiti dal destinatario e dal link/switch più lento lungo il percorso dalla sorgente alla destinazione.



Il controllo di flusso è relativamente facile da gestire, in quanto basta implementare un meccanismo di avviso di riempimento imminente della coda da parte del destinatario verso il mittente. Mentre la congestione è più difficile, in quanto implica il sapere lo stato della rete, quando in realtà non si conosce la rete (non si sanno quali sono i router intermedi).

Problemi end-to-end

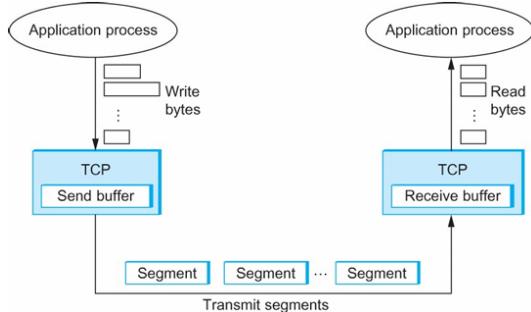
Il cuore del TCP è l'algoritmo della *finestra scorrevole*, i cui problemi da affrontare sono dati dal fatto che TCP funziona su Internet piuttosto che su un collegamento punto-punto: TCP supporta connessioni logiche tra processi in esecuzione su due computer diversi in Internet e possono avere tempi di RTT molto diversi (da 10 µs a 10 s in ordine di grandezza di sei) e con pacchetti possono essere non ordinati (di solito questo non accade su un singolo collegamento).

Per il controllo del flusso, il TCP ha bisogno di un meccanismo che permetta a ciascun lato di una connessione di sapere quali risorse l'altro lato è in grado di applicare alla connessione (upgrade della finestra scorrevole). Mentre, per il controllo della congestione, il TCP ha bisogno di un meccanismo che consenta al lato mittente di conoscere la capacità della rete.

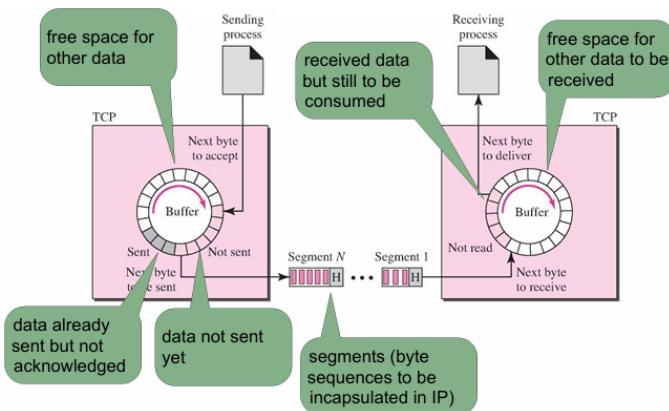
Segmento TCP

Il TCP è un protocollo orientato ai byte (*byte-oriented*), il che significa che il mittente scrive byte in una connessione TCP e il destinatario legge byte dalla connessione TCP (c'è perdita di struttura, poiché i dati vengono presi dal buffer e trattati come un fluido, quindi trasformati in byte e non importa in che quantità o dimensione di blocchi essi arrivino, ma vengono ricomposti com'erano). Inoltre, non vengono mantenuti i confini dei messaggi, cosa che altri protocolli fanno (es.: SCTP).

Sebbene il “flusso di byte” descriva il servizio che il TCP offre ai processi applicativi, il TCP non trasmette di per sé i singoli byte su Internet. Sull’host di origine, il TCP bufferizza una quantità di byte dal processo di invio sufficiente a riempire un pacchetto di dimensioni ragionevoli e poi invia questo pacchetto al suo peer sull’host di destinazione. Poi, sull’host di destinazione, TCP svuota il contenuto del pacchetto in un buffer di ricezione, dal quale il processo ricevente legge byte a suo piacimento. I pacchetti scambiati tra i peer TCP sono chiamati segmenti (payload), ovvero una fetta di byte.



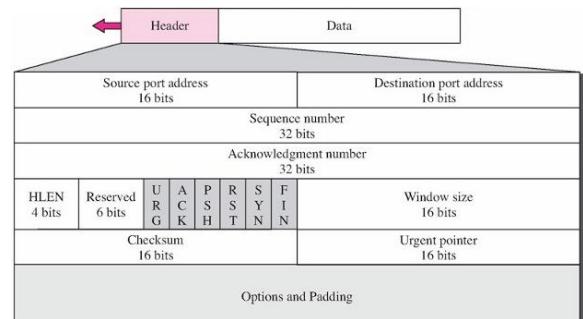
Il protocollo TCP può essere visto come un modello produttore-consumatore, in cui si hanno dei buffer “suddivisi in sezioni”, come in immagine accanto. Il buffer *Sent*, quello in uscita, ha tre colorazioni: bianco per



la parte libera in cui l’applicazione può scrivere i dati, rosa per i dati scritti nel buffer e che non sono ancora stati spediti (scritti dall’applicazione e TCP deve inviarli) e grigio per i dati inviati e di cui non si ha ancora avuto un riscontro (il destinatario non ha ancora notificato del loro arrivo tramite ACK, per cui si tiene questo backup). Il buffer *Not read*, quello in entrata, invece, ha solo due colorazioni: bianco per la parte libera e rosa per la parte di dati ricevuta e che non è ancora stata utilizzata.

Per sapere dove si è arrivati con la lettura/scrittura e con la spedizione/ricezione, si usano dei puntatori. Quindi il mittente, prende una parte libera del proprio buffer e inizia a scriverci e man mano che invia si segna in grigio quali byte non hanno ancora ricevuto un riscontro. Quando il riscontro positivo arriva lo spazio del buffer occupato da questi byte viene liberato. Il destinatario a sua volta riempie i byte liberi del proprio buffer e notifica il mittente di ciò che ha ricevuto (o non) e se ha ancora spazio libero per ricevere (in caso lo informa di rallentare/velocizzare l’invio). A mano a mano che i dati vengono riempiti/svuotati si spostano i puntatori.

Nell’intestazione, di 20 byte (di default), sono segnati gli indirizzi e le rispettive porte, seguiti da numero di sequenza, ovvero dove devono essere collocati i dati del payload all’interno della sequenza, poi c’è il campo acknowledgment per l’altra direzione in modo da informare il destinatario di fin dove il mittente è stato notificato della ricezione (tecnica chiamata *piggybacking*, in cui invece che inviare pacchetti separati della stessa cosa o di qualcosa di simile, li si mettono assieme in un unico pacchetto, in questo caso ACK e dati effettivi), il HLEN per la lunghezza dell’header (in word, di solito 5), poi c’è una serie di bit di controllo (flag SYN, FIN, RESET, PUSH, URG e ACK), di seguito la grandezza della finestra (memoria disponibile del buffer di ricezione, per il controllo di flusso, altro piggybacking), il checksum (stesso di UDP: header TCP + dati + pseudo-header) e altre informazioni di controllo.



I flag SYN e FIN sono utilizzati rispettivamente per stabilire e terminare una connessione TCP, mentre quello ACK viene impostato ogni volta che il campo *Acknowledgment* è valido (bit a 1), che implica che il destinatario deve prestarvi attenzione. Il flag URG indica che questo segmento contiene dati urgenti. Quando questo flag è impostato, il campo *UrgPtr* indica dove iniziano i dati non urgenti contenuti in questo segmento. I dati urgenti sono contenuti nella parte anteriore del corpo del segmento, fino a un valore di *UrgPtr* compreso nel segmento. Il flag PUSH indica che il mittente ha invocato l'operazione push, che indica al lato ricevente di TCP di notificare il processo ricevente di questo fatto. Il flag RESET indica che il destinatario si è confuso, ha ricevuto un segmento che non si aspettava di ricevere e quindi vuole interrompere la connessione. Questa confusione può essere determinata dal fatto che TCP è stateful e c'è bisogno di sincronizzare gli endpoint.

Diagramma di stato di TCP

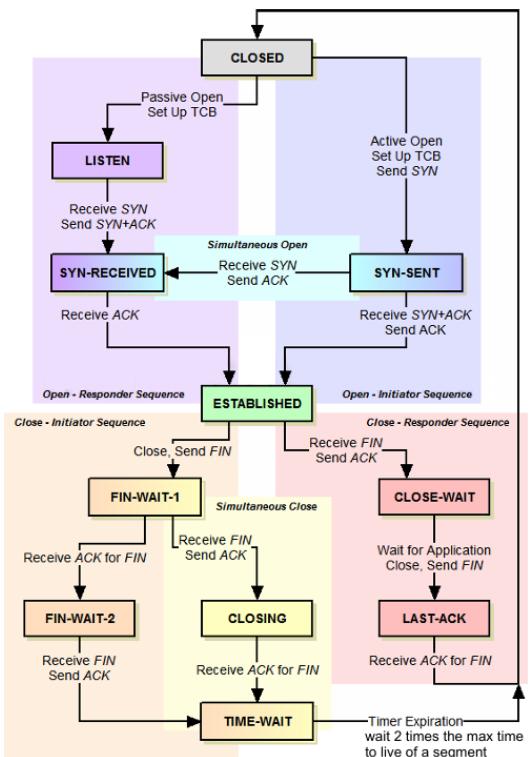
Una comunicazione orientata alla connessione deve mantenere alcune informazioni sui dati, sui segmenti mancanti, ecc. (TCP è stateful). Ogni socket TCP è in uno stato le cui transizioni sono causate da eventi quali: l'esecuzione di comandi a livello di applicazione sul socket, come `accept()`, `connect()`, `close()`, oppure dalla ricezione di segmenti con flag appropriati. Le transizioni possono causare l'invio di segmenti con flag appropriati. Le socket devono essere in stati coerenti tra di loro.

TCP può essere quindi visto come un automa o una macchina a stati, in cui gli stati (finiti) e le transizioni di stato di un socket TCP sono definiti dal protocollo per mezzo di un DFA (in realtà una macchina di Mealy). Ogni transizione è innescata da un evento e può produrre la trasmissione di un segmento (con opportuni flag). Si articola in tre fasi:

1. [Handshaking: creazione della connessione](#)
2. [Trasferimento dei dati: quando i dati fluiscono](#)
3. [Chiusura: fine della connessione](#)

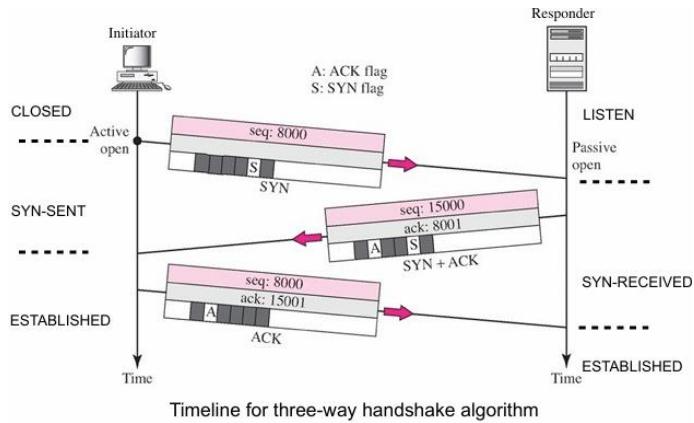
Una connessione può rimanere in ESTABLISHED per un tempo illimitato (anche per sempre).

Nello schema accanto, gli stati sono i blocchetti e le frecce le transizioni che avvengono tra gli stati. La parte alta è quella relativa all'handshaking (il mettersi d'accordo prima dell'invio dei dati veri e propri), quella verde è lo stato in cui la connessione è stabilita e è possibile inoltrare i messaggi (può durare un tempo illimitato), infine la parte bassa è quella di chiusura e che serve a ripristinare lo stato iniziale.



Fase 1 (handshake):

Prima di inviare qualsiasi dato, due processi devono stabilire una connessione. Un processo (il *responder* o server) attende una connessione, eseguendo una funzione TCP adatta per l'apertura passiva (es.: su Unix è la `accept()` bloccante). L'altro processo (l'*initiator* o client) inizia la connessione eseguendo l'apertura attiva (es.: su Unix è la `connect()`, `time-out system call`). A questo punto, i due livelli TCP eseguono il *three-way handshake*:



1. l'host in apertura passiva (server) è in attesa nello stato LISTEN
2. quando il client esegue l'apertura attiva (esso si trova in stato CLOSED), invia il primo segmento (SYN) che ha un numero di sequenza iniziale (e nessun carico utile) spostandosi nello stato SYN-SENT
3. il server risponde con SYN+ACK: riconosce il numero di sequenza del client e imposta il suo numero di sequenza (anche in questo caso nessun payload) cambiando di stato in SYN-RECEIVED
4. il client risponde con ACK: riconosce il numero di sequenza del server (ancora senza payload) e questo punto passa allo stato ESTABLISHED
5. quando il server riceve l'ACK passa anch'esso allo stato ESTABLISHED, per cui la connessione è stabilita e i segmenti che trasportano payload di dati reali possono essere inviati in entrambe le direzioni

Ci sono casi in cui entrambi gli endpoint eseguono una richiesta di connessione attiva. In questo caso, uno dei due client, nel vedersi arrivare un segmento SYN, potrebbe decidere di rifiutare la richiesta per non cambiare "la propria natura" e quindi invia un RESET. L'altro endpoint accetta e si passa dallo stato SYN-SENT allo stato SYN-RECEIVED. I due endpoint si scambiano quindi due segmenti ACK e passano entrambi allo stato ESTABLISHED. Questo rarissimo cambio di procedura prende il nome di *Simultaneous Open*, poiché entrambi i client effettuano una open. In realtà, un client non sa se un altro endpoint (presumibilmente server) è in apertura passiva finché non tenta di connettersi e se non lo è (caso appena visto), riceve un segmento RESET.

I segmenti che vengono inviati presentano un numero random di sequenza. I segmenti ACK riportano lo stesso numero del campo sequenza dei SYN a cui fanno riferimento, ma incrementato di uno, in modo da poterli riconoscere.

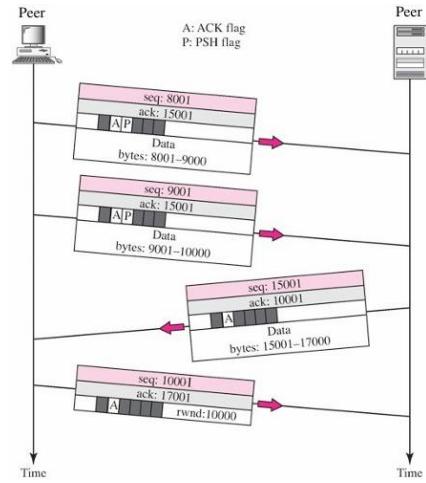
Da terminale è possibile scrivere `netstat -a | less` (oppure col parametro `-na` al posto di `-a`) per aprire una pipe in cui vengono visualizzate tutte le socket che sono aperte con il loro stato. Dove compare il '*' si intende l'indirizzo della propria macchina e presenta accanto il numero di porta, mentre le colonne *Recv-Q* e *Send-Q* rappresentano i numeri di byte in coda rispettivamente di ricezione e di invio.

Questa rappresentazione non è live, ma è un'istantanea di quello che si vede nel momento in cui si preme il tasto 'invio'. Con Wireshark si riuscirebbe a vedere in tempo reale il traffico di pacchetti (*sniffing*) e si potrebbe anche analizzarne il contenuto e vedere i pacchetti frammentati riassemblati (vedi registrazione S05E03).

Fase 2 (trasferimento dei dati):

Entrambe le parti sono quindi in ESTABLISHED. Ogni segmento trasporta dati in una direzione e la conferma dei dati nella direzione opposta (piggyback). Come già detto, i due flussi sono indipendenti.

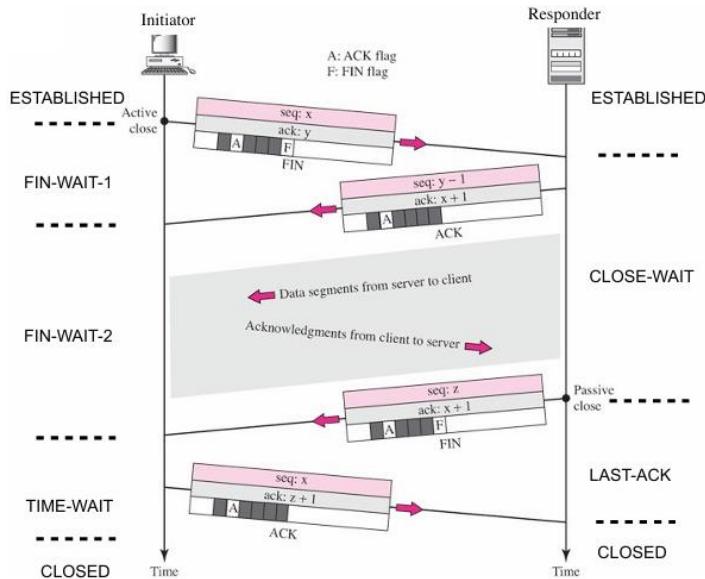
Come numero sequenza di partenza in questo caso si ha 8001 (vedi esempio in figura). Si supponga che il pacchetto sia di lunghezza 1 Kbyte circa, quindi va dal byte 8001 al byte 9000 (il primo segmento che il client invia). Successivamente (senza aspettare l'ACK di risposta) viene inviato anche un segmento che va dal byte 9001 al 10000. Entrambi hanno l'ACK con valore 15001 (valore dell'handshake, vedere immagine precedente). A questo punto arriva la risposta dal server, che conferma la ricezione dei pacchetti tramite un ACK (con lunghezza dal byte 15001 al 17000) con numero di sequenza di 15001 e numero di ACK 10001. Questo segmento appena inviato contiene il numero cumulativo dei pacchetti fino a quel momento ricevuti. Quando il client riceve l'ACK sa che può iniziare a liberare i dati di backup e invia un ACK che conferma di aver ricevuto la risposta precedente. Il flag PUSH che è stato utilizzato dal client nei primi due segmenti indica al server di notificare la ricezione e lo forza a tenerlo aggiornato sulle sequenze di byte che riceve, nonché di inviare i dati più in fretta, senza creare backup ("data pushing", assenza di dati). Nota importante: l'invio multiplo di dati, prima o poi viene interrotto se allo scadere del timeout non arriva un ACK che ne confermi la ricezione (il mittente "si stufa di aspettare").



Come si può notare, il TCP è strettamente sequenziale. A volte è utile consegnare i dati fuori sequenza (es.: segnalazioni urgenti, comandi di interruzione, ecc.). L'applicazione può chiedere una consegna urgente durante un invio, che se accettata, il TCP del mittente colloca i dati urgenti all'inizio del segmento successivo, attivando il bit URG e imposta il puntatore urgente di conseguenza. Il TCP del destinatario trova URG=1, separa i dati urgenti da quelli standard e consegna immediatamente i dati urgenti, anche se sono fuori ordine.

Fase 3 (chiusura):

Dopo qualche tempo, uno dei due host può richiedere di chiudere la connessione. La chiusura è causata dalla system call `close` su una delle due parti, che si chiamerà *initiator* (non necessariamente l'initiator della fase di handshake, in quanto qualsiasi parte può iniziare la sequenza di chiusura). L'altra parte risponde eseguendo la sequenza di risposta. Nel raro caso in cui entrambe le parti inizino contemporaneamente la sequenza initiator, si ha la chiusura simultanea.



Un host (initiator) avvia la procedura di chiusura, inviando un segmento con FIN=1, e passando allo stato FIN-WAIT-1. Questo segmento può contenere l'ultimo payload di dati, dopo del quale nessun altro dato può essere inviato dall'host. L'altro host (responder) conferma all'initiator l'avvenuto invio di ACK, si porta nello stato CLOSE-WAIT e notifica la chiusura all'applicazione (cioè la fine del file). L'initiator si porta in FIN-WAIT-2.

L'applicazione sul responder può ancora inviare dati all'altro host, che continuerà a confermarli. Alla fine, l'applicazione sul responder chiude il socket, quindi quest ultimo invia un FIN all'initiator e passa allo stato LAST-ACK. L'initiator riceve FIN, invia il suo ACK e si sposta in TIME-WAIT. Il responder chiude il socket quando riceve l'ACK.

Chi ha iniziato la chiusura (initiator) e che a questo punto si trova in TIME-WAIT, ha un certo tempo prima di passare allo stato di CLOSED, questo per evitare di tornare troppo presto nello stato iniziale. Il pericolo è che:

- se l'host non passasse per lo stato TIME-WAIT potrebbe essere che esso chiuda la connessione e poi magari il suo ACK vada perso, con la conseguenza che, dato che una volta chiusa la connessione gli host non ricordano più nulla di essa, si riapre una vecchia connessione (*reincarnation*) di cui si sono perse delle informazioni (l'initiator le ha perse);
- potrebbero esserci ancora vecchi pacchetti in giro per la rete, di cui non è chiaro a quale connessione si riferisca e si rischia che interferiscano con altre comunicazioni degli host che prima erano interessati (e che hanno già perso i ricordi della connessione precedente)

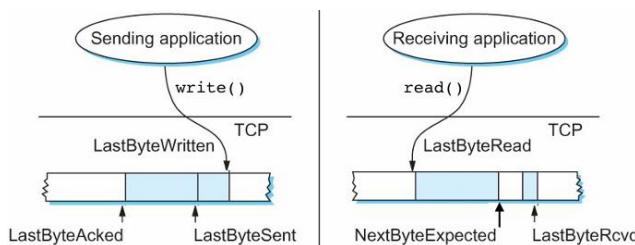
Per cui si importa un tempo di attesa, chiamato *Maximum Segment Lifetime* (MSL) di durata di circa 1 o 2 minuti.

Rivisitazione della finestra scorrevole

La variante TCP dell'algoritmo della finestra scorrevole ha diversi scopi:

- garantisce la consegna affidabile dei dati
- assicura che i dati siano consegnati in ordine
- impone il controllo del flusso tra il mittente e il destinatario

In questo caso non si parla ancora del concetto di congestione.



Nella figura accanto c'è un esempio di finestra scorrevole in cui è rappresentata solo una delle due direzioni della comunicazione (bidirezionale, per cui simmetriche). I due buffer sono da immaginare come buffer scorrevoli continui, le cui aree bianche sono quelle ancora libere, mentre quelle colorate sono quelle che contengono dati utili (dati spediti di cui si aspetta l'ACK e dati non ancora spediti) e hanno dei puntatori che ne segnano l'inizio e la fine.

I puntatori lato mittente sono: *LastByteWritten*, ovvero quello che punta all'ultimo byte che è stato scritto/prodotto dall'applicazione; *LastByteAcked*, cioè dove finiscono i byte liberi di cui si ha già avuto un riscontro dal destinatario (-1 in quanto sono quelli precedenti) e dove iniziano quelli usati; *LastByteSent* è quello che tiene traccia di fino a dove i dati sono stati inviati (e di cui non si ha ancora un riscontro ACK). Essi sono in ordine di grandezza: $LastByteAcked \leq LastByteSent \leq LastByteWritten$. Ogni volta che il destinatario informa di quali dati ha ricevuto, si spostano in avanti i puntatori.

I puntatori lato ricevente (buffer di input, non di read) sono: *LastByteRead* è l'ultimo byte che è stato letto (con una `read()`) dall'applicazione; *NextByteExpected* è il valore che il destinatario invia al mittente nel campo *Acknowledgment* dell'intestazione TCP e rappresenta i dati fino a dove il destinatario può usare (visto che i dati possono non essere contigui, deve aspettare quelli intermedi tra *NextByteExpected* e *LastByteRcvd*); *LastByteRcvd* è l'ultimo byte che il ricevente ha ricevuto (ma non segnala con un ACK poiché potrebbe non aver ancora ricevuto dei segmenti precedenti). I byte del ricevitore potrebbero non essere contigui, poiché dipende da quando arrivano i segmenti dalla rete, per cui potrebbero essere arrivati fuori ordine o andati persi. L'ordine di grandezza dei valori dei puntatori è: $LastByteRead < NextByteExpected \leq LastByteRcvd + 1$.

Per non superare il buffer del ricevitore si deve avere: $LastByteRcvd - LastByteRead \leq MaxRcvBuffer$. Pertanto, il ricevitore può ancora accettare questa quantità di dati, infatti:

$$\text{AdvertisedWindow} = MaxRcvBuffer - ((NextByteExpected - 1) - LastByteRead)$$

Questo è il valore che il ricevitore invia al mittente nel campo *Window* dell'intestazione TCP. Il mittente, d'altra parte, non deve mai inviare al destinatario più dati di quanti ne possa immagazzinare nel suo buffer, cioè:

$$LastByteSent - LastByteAcked \leq AdvertisedWindow$$

Questo include i dati già inviati ma non ancora riconosciuti, quindi il limite effettivo dei dati che possiamo ancora inviare è:

$$\text{Effective Window} = AdvertisedWindow - (LastByteSent - LastByteAcked)$$

$$= AdvertisedWindow - (LastByteSent - (Acknowledgment - 1))$$

$$= AdvertisedWindow - (LastByteSent + 1 - Acknowledgment)$$

dove *Acknowledgment* è il valore più aggiornato ricevuto dal mittente e *AdvertisedWindow* è intesa l'ultima.

Inoltre, poiché il processo di invio dell'applicazione non può superare il buffer di invio, deve essere sempre:

$$LastByteWritten - LastByteAcked \leq MaxSendBuffer$$

Ciò significa che se il processo di invio tenta di scrivere y byte, ma

$$(LastByteWritten - LastByteAcked) + y > MaxSendBuffer$$

L'operazione di "scrittura" viene bloccata fino a quando non si libera spazio nel buffer di invio (cioè fino a quando non vengono inviati dati sufficienti al destinatario e da questo riconosciuti). In questo modo, riducendo la finestra a 0, il processo ricevente può rallentare, o addirittura interrompere, la trasmissione dal processo mittente.

Il mittente può ricominciare a trasmettere quando la finestra si riapre, cioè quando il processo di ricezione consuma alcuni dati. La nuova finestra può essere notificata dal ricevitore nei segmenti che invia al mittente, se il ricevitore ha dei dati da inviare al mittente (il flusso nell'altra direzione). Tuttavia, se il ricevitore non ha dati da inviare al mittente, esso non potrebbe mai scoprire che la finestra è aperta. Pertanto, se il mittente sa che *AdvertisedWindow*=0, attende un segmento dal ricevitore fino a un determinato timeout; altrimenti, se non riceve alcun segmento, invia un "segmento probe" di 1 byte per "stimolare" una risposta ACK con la finestra corrente. Questo timeout viene aumentato esponenzialmente tra ogni probe successivo (algoritmo di backoff).

Protezione contro il wraparound

Come previsto dal protocollo Sliding Window, bisogna assicurarsi che non ci siano due dati in sospeso (byte) con lo stesso numero di sequenza. La condizione che si ha visto è:

$$\text{larghezza della finestra} < \text{numero di sequenza massimo} / 2$$

equivalente a

$$\text{numero di sequenza massimo} > 2 \times \text{larghezza della finestra}$$

In TCP si ha un *SequenceNum* di 32 bit e una *AdvertisedWindow* di 16 bit. Quindi, il requisito dell'algoritmo della finestra scorrevole è soddisfatto perché $2^{32} \gg 2 \times 2^{16}$. Le recenti implementazioni di TCP hanno più bit per *AdvertisedWindow*, ma la condizione è ancora valida.

Tuttavia il meccanismo di avere uno spazio di numeri di sequenza maggiore della finestra, potrebbe non bastare nel caso di connessioni "riavvolgenti" (*wraparound*). Infatti, se si tramette un flusso di byte fino all'esaurimento dello spazio e perciò si necessita di ricominciare la sequenza, bisogna stare attenti che la ripetizione dei vecchi numeri (0, 1, 2, ecc.) non sia ancora dentro al buffer e quindi venga interpretata come un reinvio dei vecchi pacchetti, che verrebbero scartati. Per risolvere questo inconveniente, si aggiunge un timeout di 120 secondi, chiamato *durata massima del segmento* (MSL), ovvero il tempo massimo che i pacchetti possono sopravvivere in Internet. Il rischio di riavvolgimento del numero di sequenza dipende dalla velocità di trasmissione dei dati su Internet.

Ci sono dei casi in cui i delay sono maggiori di 120 secondi, se la rete dovesse essere molto grande o in base al mezzo trasmissivo utilizzato. Per questo motivo, TCP è fatto per funzionare con delay molto maggiori di questo. Ad esempio, come si può vedere nell'immagine accanto, i diversi mezzi trasmissivi determinano timeout diversi. Esiste un protocollo NON propriamente IP, chiamato *IP over Aviar Carrier* (ideato il 1° di aprile 1999, quindi è un pesce d'aprile) che sfrutta i piccioni viaggiatori per consegnare messaggi IP in hardware tramite un dispositivo legato a una zampina del piccione, il cui *timeo ut* può essere molto più lungo (o uguale, dipende dalla distanza).

Bandwidth	Time until Wraparound
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
Fast Ethernet (100 Mbps)	6 minutes
OC-3 (155 Mbps)	4 minutes
OC-12 (622 Mbps)	55 seconds
OC-48 (2.5 Gbps)	14 seconds

Mantenere la pipe piena

Il campo *AdvertisedWindow* deve essere sufficientemente grande da consentire al mittente di mantenere la pipe piena (chiaramente il destinatario è libero di non aprire la finestra tanto quanto lo consente il campo *AdvertisedWindow*).

Se il destinatario ha spazio sufficiente per il buffer, la finestra deve essere aperta abbastanza per consentire un *ritardo × larghezza di banda* pieno di dati.

Bandwidth	Delay × Bandwidth Product
T1 (1.5 Mbps)	18 KB
Ethernet (10 Mbps)	122 KB
T3 (45 Mbps)	549 KB
Fast Ethernet (100 Mbps)	1.2 MB
OC-3 (155 Mbps)	1.8 MB
OC-12 (622 Mbps)	7.4 MB
OC-48 (2.5 Gbps)	29.6 MB

Required window size for 100-ms RTT

Un campo a 16 bit consente una finestra di 64 Kbyte, che potrebbe non essere sufficiente per le “linee lunghe” (cioè linee con *ritardo × larghezza di banda* $\gg 12500\text{ byte}$). TCP consente di impostare un *fattore di scala* (fino a 2^{14}) durante il protocollo di handshake. Pertanto, la finestra può arrivare a 1 GB ($2^{16} \times 2^{14} = 2^{30}$).

Attivazione della trasmissione

Per decidere di trasmettere un segmento, TCP supporta un’astrazione di flussi di byte. I programmi applicativi scrivono i byte nei flussi e spetta al TCP decidere se ha abbastanza byte per inviare un segmento. In questo caso si ignora il controllo di flusso: si supponga che la finestra sia aperta, come nel caso dell’inizio della connessione.

Il TCP dispone di tre meccanismi per attivare la trasmissione di un segmento:

1. TCP mantiene una dimensione massima del segmento (*Maximum Segment Size*, MSS) variabile e invia un segmento non appena ha raccolto i byte MSS dal processo di invio. La MSS è solitamente impostata sulla dimensione del segmento più grande che il TCP può inviare senza causare la frammentazione dell’IP locale. $MSS = MTU\text{ della rete direttamente connessa} - (\text{intestazione TCP} + \text{intestazione IP})$
2. il processo mittente ha chiesto esplicitamente a TCP di inviarlo (TCP supporta l’operazione push)
3. quando scatta il timer. Il segmento risultante contiene tanti byte quanti ne sono attualmente bufferizzati per la trasmissione (ma sempre fino a MSS e *EffectiveWindow*)

Sindrome della finestra stupida (Silly Window Syndrome)

L’algoritmo TCP a finestre scorrevoli di base non prevede una dimensione minima per i segmenti trasmessi. La sindrome della finestra stupida (SWS) è una situazione in cui vengono inviati molti segmenti piccoli e inefficienti (più overhead), piuttosto che pochi segmenti grandi. Può verificarsi quando un destinatario pubblicizza dimensioni di finestra troppo piccole o un trasmettitore è troppo aggressivo nell’inviare immediatamente quantità di dati molto piccole. Non si tratta di un fallimento dell’algoritmo della finestra scorrevole (che fa il suo lavoro per mantenere pieno il buffer di ricezione), ma di un’inefficienza dovuta all’overhead della rete.

Il caso peggiore è quando: *AdvertisedWindow* è 0 (trasmissione ferma per buffer pieno), poi il processo di ricezione consuma 1 byte, quindi il ricevitore invia al mittente che *AdvertisedWindow* è 1, per cui il mittente vedendo che si è liberato un byte di spazio invia aggressivamente un segmento di 1 byte, chiudendo nuovamente la finestra e costringendo la trasmissione a fermarsi (il buffer del ricevente è saturo) e poi si ripete. Questo porta a una sequenza di segmenti da 1 byte (ciascuno con 40 byte di overhead di intestazione). Sarebbe meglio che il mittente aspettasse un po' prima di inviare un segmento: il processo ricevente consumerà più byte, aprendo così la finestra, e quindi il senso potrebbe inviare un singolo segmento più grande.

Tuttavia questa attesa introduce un rallentamento che potrebbe risultare scomodo per applicazioni interattive come Telnet e SSH, in cui se si preme un pulsante si dovrebbe iniziare a vedere subito una risposta. Per cui serve una via di mezzo per quelle applicazioni che possono inviare grosse quantità di dati, a discapito di un rallentamento (es.: email), e per quelle che devono essere efficienti e veloci.

Algoritmo di Nagle

John Nagle ha introdotto un'elegante soluzione autobloccante, la cui idea chiave è quella in cui finché TCP ha dei dati in volo (transito), il mittente riceverà un ACK, che può essere trattato come un timer che scatta, innescando la trasmissione di altri dati. Per cui, si aspetta un po' prima di inviare un ACK in modo da dare il giusto tempo di svuotamento del buffer e notificando il mittente di quanto si è liberato il ricevente e non appena arriva l'ACK si procede con l'invio effettivo di nuovi dati.

Per cui quando l'applicazione produce nuovi dati da trasmettere, l'algoritmo controlla che:

```
if both the available data and the window are >= MSS
    send a full segment
else
    if there is unACKed data in flight
        buffer the new data until an ACK arrives
    else
        send all the new data now
```

Esercizi:

(Scheda del 2021-06-22):

13) Una sorgente TCP ha inviato tre segmenti da 1200 byte l'uno, con *SequenceNum* pari a 3000, 4200, 5400 rispettivamente, e ha ricevuto i seguenti segmenti in questo ordine: ACK=3000, *AdvertisedWindow*=4200; ACK=5400, *AdvertisedWindow*=1800; ACK=4200, *AdvertisedWindow*=3000. Quanti byte può ancora spedire?

R: in questo caso, dato che i pacchetti non sono arrivati nell'ordine di invio, l'ultimo ha un *AdvertisedWindow* sbagliato (confrontare col primo pacchetto arrivato e che fa testo), per cui bisogna rifare il conto prendendo l'ultimo corretto. In questo modo si ottiene:

$$\text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked}) = 1800 - (6599 - 5399) = 600$$
 dove 6599 si ricava dall'ultimo byte ricevuto decrementato di 1 ($5400 + 1200 - 1$) e 5399 è l'ultimo byte che è stato notificato con un ACK (decrementato di 1).

(Scheda del 2021-07-09):

13) Un'applicazione sta scrivendo su una socket TCP una stringa di 300 byte ogni 10 ms. La connessione TCP ha un MSS di 1460 byte e un RTT di 50ms. Si supponga che *CongestionWindow* e *AdvertisedWindow* siano sufficientemente grandi. (a) Quanto è grande il payload di ogni segmento inviato dall'host, in media? (b) Cosa succede se il RTT scende a 30ms?

R: (a) l'applicazione produce dati a 30 Kbps ($= 300/(10 \cdot 10^{-3})$), quindi quando invia un segmento deve aspettare 50ms di RTT prima di avere un riscontro, per cui nel frattempo accumula 1500 byte, ovvero più grande del

MSS. Il mittente, appena riceve il riscontro dal destinatario (dopo RTT 50 ms) invia 1460 byte, tenendo 40 byte nel buffer. Il mittente continua a accumulare byte e poi li invia al destinatario appena arriva l'ACK e così via, seguendo l'algoritmo di Nagle.

(b) se il RTT scendesse a 30ms, l'applicazione produrrebbe 900 byte nel mentre che un ACK arrivi, quindi si inviano segmenti molto minori del MSS.

Ritrasmissione adattiva e calcolo del timeout

Un segmento deve essere ritrasmesso se non si riceve una conferma per un certo periodo di tempo, per cui bisogna impostare il timeout. Per farlo, bisogna adattarsi ai ritardi. L'algoritmo della finestra scorrevole visto al livello 2 calcola il timeout in base alla tecnologia utilizzata e alla distanza dei due endpoint, risultando un calcolo semplice e deterministico. Al livello 4 è più complesso e tende a cambiare nel tempo, quindi non è sempre prevedibile e dipende dal percorso dei pacchetti (che potrebbero anche andare persi).

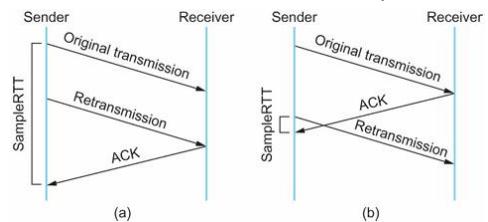
L'algoritmo misura l'RTT campione per ogni coppia segmento/ACK e calcola la media ponderata (pesata) del RTT, per cui ogni volta che un pacchetto viene inviato si tiene conto di quanto tempo serve per avere un riscontro. Ogni volta si ricalcola la media (RTT campione = *SampleRTT*), per cui si ha:

$$EstRTT := \alpha \times EstRTT + (1 - \alpha) \times SampleRTT, \text{ con } \alpha \text{ tra } 0,8 \text{ e } 0,9$$

A questo punto si imposta il timeout in base al valore di *EstRTT*, ovvero *TimeOut* = $2 \times EstRTT$.

Come è possibile notare, la formula è ricorsiva e si necessita di un valore di *EstRTT* iniziale, ovvero quello campionario durante l'handshake.

Questo algoritmo, però, presenta due problemi: l'ACK non conferma realmente una trasmissione, ma solo la ricezione dei dati, e inoltre, quando un segmento viene ritrasmesso e al mittente arriva un ACK, è impossibile decidere se questo ACK debba essere associato alla prima (a) o alla seconda (b) trasmissione per calcolare gli RTT (immagine accanto). Infatti, nel primo caso si tiene conto della ritrasmissione per cui il tempo è molto più lungo, mentre nel secondo caso si prende lo scarto dei due tempi, con risultato che il tempo è molto (troppo) breve.



Algoritmo di Karn-Partridge:

Una soluzione è quella di usare l'*algoritmo di Karn-Partridge* (obbligatorio in TCP, vedere RFC 6298), che dice di non campionare l'RTT durante la ritrasmissione, ma di raddoppiare il timeout dopo ogni ritrasmissione. Inoltre, l'algoritmo ignora i segmenti ritrasmessi quando aggiorna la stima del tempo di andata e ritorno. La stima del tempo di andata e ritorno si basa solo sui riscontri non ambigui, ovvero sui riscontri per i segmenti inviati una sola volta.

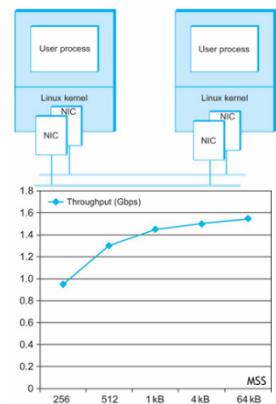
Il raddoppio del tempo serve in quelle situazioni in cui TCP invia un segmento dopo un forte aumento del ritardo, in cui se si utilizzasse la stima del tempo di andata e ritorno precedente, TCP calcolerebbe un timeout che potrebbe ignorare il tempo di andata e ritorno di tutti i pacchetti ritrasmessi, per cui la stima del tempo di andata e ritorno non verrà mai aggiornata e TCP continuerà a ritrasmettere ogni segmento, senza mai adattarsi all'aumento del ritardo.

Per cui la soluzione a questo problema consiste nell'incorporare i timeout di trasmissione con una strategia di backoff a timer, che calcola un timeout iniziale e se il timer scade e causa una ritrasmissione, il TCP aumenta il timeout generalmente di un fattore due. Questo algoritmo è estremamente efficace nel bilanciare le prestazioni e l'efficienza nelle reti con un'elevata perdita di pacchetti, anche se idealmente non sarebbe necessario: le reti che presentano tempi di andata e ritorno elevati e timeout di ritrasmissione dovrebbero essere analizzate con tecniche di analisi delle cause principali.

Prestazioni del TCP

Per quanto riguarda le prestazioni del TCP si tiene conto di due metriche: latenza e throughput.

Ad esempio, se si avesse una configurazione sperimentale con Xeon 2,4 GHz (doppio), una Gigabit Ethernet in aggregazione di link (Linux) e un limite del canale datalink di quasi 2Gbps full duplex, senza perdite, si avrebbe che più grande è l'MSS più alto è il throughput, ma sopra 1KB non aumenta in modo sostanziale. Il MSS tipico è di 1460 byte (adatto quando l'interfaccia ha MTU=1500) che si mantiene al di sotto dei 2Gbps (diversi colli di bottiglia). Nel complesso, il TCP può raggiungere un elevato throughput, in cui a volte il collo di bottiglia è la memoria stessa.



Controllo della congestione e allocazione delle risorse

Il problema che si deve affrontare è come allocare in modo efficace ed equo le risorse tra un insieme di utenti in competizione. Con risorse nelle reti si intendono: larghezza di banda dei collegamenti e i buffer nei router e negli switch.

I pacchetti si contendono presso un router l'uso di un collegamento, e ogni pacchetto viene messo in coda in attesa del suo turno di trasmissione sul collegamento. Quando troppi pacchetti si contendono lo stesso collegamento, la coda si riempie e i pacchetti in più vengono eliminati, col risultato che la rete è congestionata. La rete deve quindi prevedere un meccanismo di controllo/evitamento della congestione.

Il controllo della congestione e l'allocatione delle risorse sono due facce della stessa medaglia. Se la rete assumesse un ruolo attivo nell'allocatione delle risorse, la congestione può essere evitata e non sarebbe necessario il controllo della congestione. Ma allocare in anticipo le risorse con precisione è difficile, in quanto le risorse sono distribuite in tutta la rete e non è facilmente definibile a priori.

D'altra parte, si può sempre lasciare che le sorgenti inviano tutti i dati che vogliono e poi recuperare la congestione quando si verifica (*congestion control*). Questo approccio è più semplice, ma può essere dannoso perché molti pacchetti vengono scartati dalla rete prima che la congestione possa essere controllata.

Un terzo approccio consiste nel lasciare che le sorgenti inviano tutti i dati che vogliono, ma fermandole prima che si verifichi una congestione (*congestion avoidance*). Però è più difficile (deve riconoscere la congestione prima che si verifichi), per cui è meno implementato. Di solito si implementa col controllo della congestione.

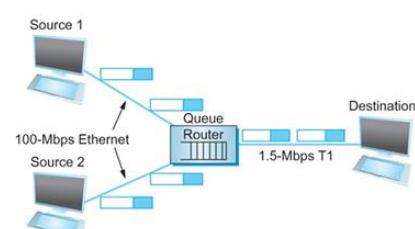
Il controllo della congestione e l'allocatione delle risorse coinvolgono sia gli host che gli elementi di rete, come i router. Per gli *elementi di rete* si possono utilizzare varie discipline di accodamento per controllare l'ordine in cui i pacchetti vengono trasmessi e quali vengono scartati. A *livello di host*, il meccanismo di controllo della congestione stabilisce la velocità con cui le sorgenti possono inviare i pacchetti.

Problemi di allocatione delle risorse

Modello di rete a commutazione di pacchetto:

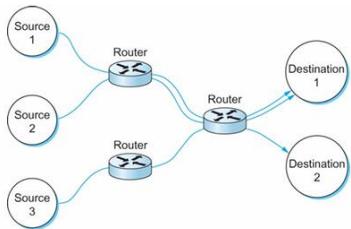
Si consideri, l'allocatione delle risorse in una rete a commutazione di pacchetto (o Internet) costituita da più collegamenti e switch (o router).

In un ambiente di questo tipo, una determinata sorgente può avere una capacità più che sufficiente sul collegamento in uscita immediato per inviare un pacchetto, ma da qualche parte nel mezzo della rete i suoi pacchetti incontrano un collegamento utilizzato da molte sorgenti di traffico diverse, col risultato di *bottle-neck* (= collo di bottiglia, figura accanto).



Modello di rete con flussi senza connessione:

Per gran parte di questa discussione, si assume che la rete sia essenzialmente priva di connessioni, con qualsiasi servizio orientato alla connessione implementato a livello di trasporto (es.: TCP su IP). Si deve qualificare il termine “senza connessioni” perché la classificazione delle reti come senza o orientate alle connessioni è un po’ troppo restrittiva. In particolare, l’ipotesi che tutti i datagrammi siano completamente indipendenti in una rete senza connessione è troppo forte. I datagrammi sono certamente scambiati in modo indipendente, ma di solito un flusso di datagrammi tra una particolare coppia di host passa attraverso un particolare insieme di router.



Multiple flows passing through a set of routers

Uno dei vantaggi dell’astrazione del flusso è che i flussi possono essere definiti a diverse granularità: un flusso può essere *host-to-host* (cioè avere gli stessi indirizzi host di origine/destinazione) o *process-to-process* (cioè avere le stesse coppie host/porta di origine/destinazione). In quest’ultimo caso, un flusso è essenzialmente uguale a un canale, come si è usato finora. Il motivo per cui si è introdotto un nuovo termine è che un flusso è visibile ai router all’interno della rete (cioè a livello di rete), mentre un canale è un’astrazione *end-to-end*.

Poiché attraverso ogni router passano più pacchetti correlati, a volte ha senso mantenere alcune informazioni di stato per ogni flusso (es.: la velocità, la dimensione dei pacchetti, il bitrate, ecc.). Queste informazioni possono essere utilizzate per prendere decisioni sull’allocazione delle risorse per i pacchetti che appartengono al flusso. Questo stato viene talvolta chiamato “*soft state*”.

La differenza principale tra lo stato soft e lo stato “*hard*” è che lo stato soft non deve essere sempre creato e rimosso esplicitamente tramite segnalazione. Spesso viene dedotto “automaticamente” dai router (tecniche recenti adottano il deep learning sui dati di traffico).

Lo stato soft rappresenta una via di mezzo tra una rete puramente priva di connessioni che non mantiene alcuno stato sui router e una rete puramente orientata alle connessioni che mantiene uno stato rigido sui router. Il corretto funzionamento della rete non dipende dalla presenza del soft state (ogni pacchetto viene comunque instradato correttamente senza tener conto di questo stato), ma quando un pacchetto appartiene a un flusso per il quale il router sta mantenendo il soft state, allora il router è in grado di gestire meglio il pacchetto.

Esistono quindi due tassonomie basate sui router, ovvero router-centrico e host-centrico:

- Progetto incentrato sul router (*router-centric*): ogni router si assume la responsabilità di decidere quando i pacchetti devono essere inoltrati e di selezionare quali pacchetti devono essere abbandonati, nonché di informare gli host che generano il traffico di rete sul numero di pacchetti che possono inviare
- Progetto incentrato sull’host (*host-centric*): gli host finali osservano le condizioni della rete (es.: quanti pacchetti riescono a passare attraverso la rete) e regolano il loro comportamento di conseguenza

Si noti che questi due approcci non si escludono a vicenda.

Inoltre, esistono le tassonomie basate su prenotazione e basate su feedback:

- In un sistema basato sulla prenotazione (*reservation-based system*), un’entità (es.: l’host finale) chiede alla rete di allocare una certa quantità di capacità per un flusso. Ogni router alloca quindi risorse sufficienti (buffer e/o percentuale della larghezza di banda del collegamento) per soddisfare questa richiesta. Se la richiesta non può essere soddisfatta da un router, perché ciò comporterebbe un sovraccarico delle sue risorse, il router rifiuta la prenotazione.

- In un approccio basato sul feedback (*feedback-based approach*), gli host finali iniziano a inviare dati senza aver prima prenotato alcuna capacità e poi regolano la loro velocità di invio in base al feedback che ricevono. Questo feedback può essere esplicito (es.: un router congestionato invia all'host un messaggio di "rallentamento") o implicito (es.: l'host finale regola la propria velocità di invio in base al comportamento osservabile dall'esterno della rete, come le perdite di pacchetti).

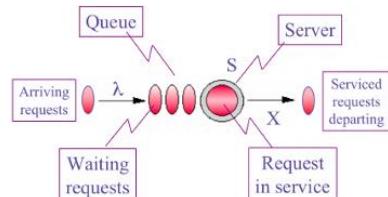
Tra i criteri di valutazione c'è l'efficacia dell'allocazione delle risorse. L'efficacia di uno schema di allocazione delle risorse può essere misurata considerando le due principali metriche del networking: throughput e ritardo. Si vuole il massimo throughput e il minimo ritardo possibile, ma questi obiettivi sono in contrasto tra loro. Si può aumentare il throughput facendo entrare nella rete il maggior numero possibile di pacchetti, in modo da portare l'utilizzo di tutti i collegamenti al 100%, ma l'aumento del numero di pacchetti nella rete aumenta anche la lunghezza delle code in ogni router e quindi comporta ritardi maggiori. Al contrario, i ritardi possono essere ridotti al minimo mantenendo le code il più possibile vuote, ma ciò comporta un minore utilizzo dei collegamenti.

Criterio di valutazione con allocazione efficace delle risorse

Un'analisi dettagliata richiederebbe strumenti e risultati di un'area della probabilità e della statistica chiamata teoria delle code (*Queue Theory*). A titolo di esempio, si considera un flusso completamente casuale di richieste a un server senza memoria (questo tipo di coda è detto M/M/1).

I parametri utilizzati sono:

- Velocità media di arrivo nella coda: λ [richieste/secondo].
- Tempo medio di servizio al server: S [secondi/richiesta].



I risultati che si ottengono sono:

- Tempo medio di permanenza che un cliente impiega per superare il sistema di code (cioè, ritardo medio totale): $R = S/(1 - \lambda S)$.
- Utilizzo del server: $\rho = \lambda S$.
- Lunghezza media della coda del sistema: $Q = \lambda R$.
- Tempo medio di attesa che un cliente può aspettarsi di trascorrere prima di ottenere il servizio: $W = R - S = Sp/(1 - \rho)$.

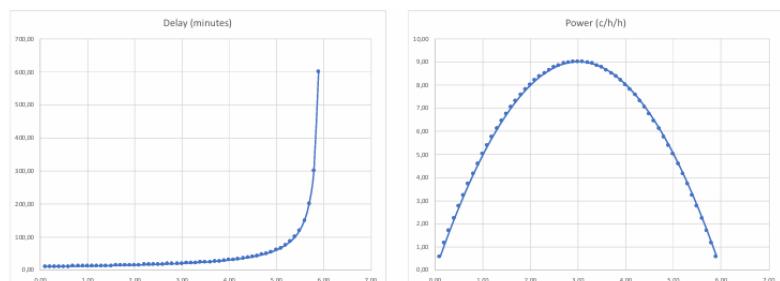
Esistono molti strumenti e librerie per queste simulazioni e analisi.

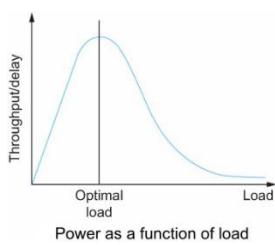
Per descrivere questa relazione, alcuni progettisti hanno proposto di utilizzare il rapporto tra throughput e ritardo come parametro per valutare l'efficacia di uno schema di allocazione delle risorse. Questo rapporto è talvolta indicato come potenza della rete:

$$\text{Potenza} = \text{velocità di trasmissione}/\text{ritardo}$$

Se l'obiettivo è massimizzare il throughput e minimizzare il ritardo, l'obiettivo è quindi massimizzare la potenza (ma in alcuni contesti uno dei due aspetti è più importante e deve essere privilegiato). Nel caso M/M/1 si tratta di: $\text{Potenza} = \lambda/S - \lambda^2$.

Ad esempio, in un semplice ufficio bancario uno sportello potrebbe servire i clienti secondo una tempistica di $S=10$ min/utente. In questo caso λ varia da 0.1 a 5.9 clienti/ora. Il picco di potenza lo si raggiunge al 50% dal carico.





In ambienti reali, la potenza segue una curva come quella riportata qui a fianco. A bassi carichi (bassi λ), le code sono quasi vuote, quindi la potenza aumenta in modo quasi lineare. A un certo punto, la potenza raggiunge il massimo. Il throughput è ben al di sotto della capacità dei canali, ma le code sono ancora corte. Dopo questo punto, il throughput aumenta ma i pacchetti iniziano rapidamente ad accodarsi, aumentando il ritardo (e diminuendo la potenza). L'obiettivo ideale è mantenere il sistema intorno al punto ottimale.

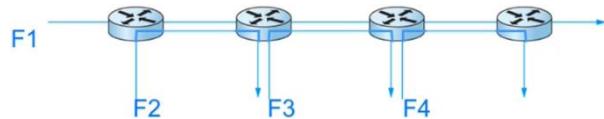
Criteri di valutazione con allocazione equa delle risorse

[Questa sezione è presente solo nelle registrazioni, non nelle slide attuali]

Si consideri un flusso a quattro hop F1 e tre flussi a due hop F2, F3, F4. Sia B la banda complessiva di ciascun router. Complessivamente, le risorse da assegnare sono 4B. Se ogni router assegna parti uguali della sua banda a ogni flusso in arrivo:

- F1 ottiene $B/2 + B/3 + B/3 + B/2 = 5B/3$
- F2 riceve $B/2 + B/3 = 5B/6$
- F3 riceve $B/3 + B/3 = 2B/3$
- F4 ottiene $B/3 + B/2 = 5B/6$

Quindi F1 ottiene molto più degli altri.



In assenza di informazioni esplicite che indichino il contrario, quando più flussi condividono un particolare collegamento, si vorrebbe che ogni flusso ricevesse una quota uguale della larghezza di banda. Questa definizione presuppone che una quota equa di larghezza di banda significhi una quota uguale di larghezza di banda. Ma anche in assenza di prenotazione, una quota uguale potrebbe non corrispondere a una quota equa. Talvolta, si vorrebbe considerare anche la lunghezza dei percorsi confrontati.

Assumendo che equo implica uguale e che tutti i percorsi sono di uguale lunghezza, l'indice di equità di Jain (*Jain's fairness index*) può essere utilizzato per quantificare l'equità di un meccanismo di controllo della congestione. Dato un insieme di throughput di flusso x_1, x_2, \dots, x_n (misurati ad esempio in bps), l'indice di equità f di Jain è definito come segue:

$$f(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n x_i^2}$$

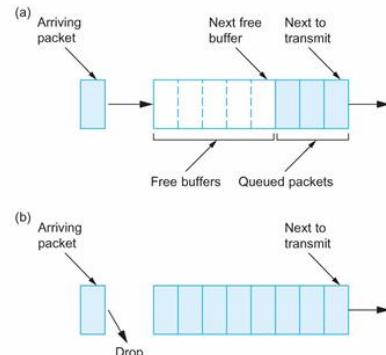
Può essere applicato anche a altre risorse (es.: memoria, energia, ecc.) e anche in altri settori (es.: economia). L'indice di equità risulta essere sempre un numero compreso tra $1/n$ e 1 (più è grande, meglio è). Il caso $1/n$ si ottiene quando tutti gli x_i tranne uno sono 0, cioè quando tutte le risorse vanno a un unico flusso (minore equità), mentre il caso a 1 quando $x_i = x_j$ per tutti gli i, j (massima equità).

Nell'esempio dei quattro flussi sopra: $f\left(\frac{5B}{3}, \frac{5B}{6}, \frac{2B}{3}, \frac{5B}{6}\right) = 0,867$, il che significa che c'è una leggera equità, ma uno dei flussi tende a prevalere un po' sugli altri.

In realtà, esistono anche altre politiche di allocazione delle risorse, ma in questo documento vengono analizzate solo queste due appena viste. Di seguito, invece, vi sono delle applicazioni un po' più concrete di quanto appena visto.

Disciplina di accodamento FIFO con caduta della coda (tail drop)

L'idea della coda FIFO, detta anche coda *first-come-first-served* (FCFS), è semplice: il primo pacchetto che arriva a un router è il primo ad essere trasmesso. Dato che lo spazio del buffer di ogni router è finito, se un pacchetto arriva e la coda (spazio del buffer) è piena, il router lo scarta. Questo avviene senza tener conto del flusso a cui il pacchetto appartiene o della sua importanza. Questa procedura viene talvolta chiamata *tail drop*, poiché i pacchetti che arrivano alla fine della FIFO vengono scartati. Si noti che tail drop e FIFO sono due concetti separati: la FIFO è una disciplina di programmazione che determina l'ordine di trasmissione dei pacchetti, il tail drop è una politica di drop che determina quali pacchetti vengono scartati.



(a) FIFO queuing; (b) tail drop at a FIFO queue

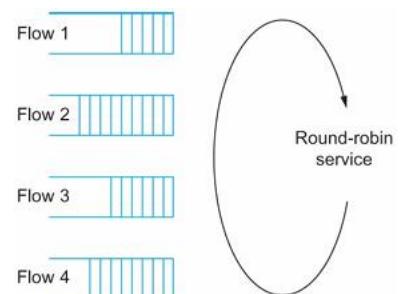
Una semplice variante dell'accodamento FIFO di base è l'accodamento prioritario (*priority queuing*). L'idea è quella di contrassegnare ogni pacchetto con una priorità. Il contrassegno può essere riportato, ad esempio, nell'intestazione IP (TOS/DSCP) o determinato in altri modi.

I router implementano quindi code FIFO multiple, una per ogni classe di priorità. Il router trasmette sempre i pacchetti dalla coda di priorità più alta, se questa non è vuota, prima di passare alla coda di priorità successiva. All'interno di ogni priorità, i pacchetti sono ancora gestiti in modo FIFO.

Disciplina di accodamento con accodamento equo

Il problema principale dell'accodamento FIFO è che non discrimina tra le diverse sorgenti di traffico o non separa i pacchetti in base al flusso a cui appartengono. L'accodamento equo (*fair queuing*, FQ) è un algoritmo proposto per risolvere questo problema. L'idea di FQ è di mantenere una coda separata per ogni flusso attualmente gestito dal router. Il router serve poi queste code in una sorta di round robin.

La complicazione principale dell'accodamento equo è che i pacchetti elaborati da un router non sono necessariamente della stessa lunghezza. Per allocare veramente la larghezza di banda del collegamento in uscita in modo equo, è necessario prendere in considerazione la lunghezza dei pacchetti.



Infatti, ad esempio: un router gestisce quattro flussi, uno con pacchetti da 1000 byte e gli altri con pacchetti da 100 byte. Un semplice servizio di round-robin dalla coda di ogni flusso darà al primo flusso $1000/1300 = 77\%$ della larghezza di banda del collegamento e solo $100/1300 = 7,7\%$ della sua larghezza di banda a ciascun altro flusso. In questo modo non è equo, poiché non si ha tenuto conto della lunghezza dei pacchetti.

Quello che si vorrebbe è un round-robin bit per bit, cioè, il router trasmette un bit dal flusso 1, poi un bit dal flusso 2 e così via. Chiaramente, non è possibile interlacciare i bit dei diversi pacchetti. Ogni pacchetto deve essere trasmesso interamente.

Il meccanismo FQ approssima quindi questo comportamento determinando prima il momento in cui un dato pacchetto finirebbe di essere trasmesso se venisse inviato utilizzando il round-robin bit per bit, e poi utilizzando questo tempo di completamento per ordinare i pacchetti per la trasmissione.

Per comprendere l'algoritmo di approssimazione del round robin bit per bit, si consideri il comportamento di un singolo flusso. Si ha che:

- P_i : indica la lunghezza del pacchetto i (misurata in tempo)

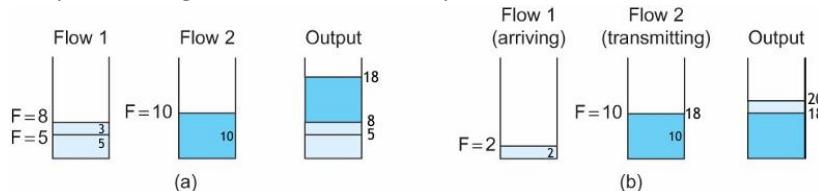
- S_i : tempo in cui il router inizia a trasmettere il pacchetto i
- F_i : tempo in cui il router termina la trasmissione del pacchetto i
- La formula che si userà è: $F_i = S_i + P_i$

Bisogna determinare quando si può iniziare a trasmettere il pacchetto i . Dipende se il pacchetto i è arrivato prima o dopo che il router ha finito di trasmettere il pacchetto $i - 1$ per il flusso. Sia A_i il tempo in cui il pacchetto i arriva al router. Allora il tempo di inizio è $S_i = \max(F_{i-1}, A_i)$, quindi $F_i = \max(F_{i-1}, A_i) + P_i$, dunque si considera il tempo maggiore tra la trasmissione del pacchetto precedente (tenendo conto della sua lunghezza) e il tempo di arrivo del pacchetto.

L'algoritmo, quindi, per ogni flusso, calcola un timestamp F_i per ogni pacchetto in arrivo come segue. Si ha F_{i-1} come il timestamp dell'ultimo pacchetto, il tempo di arrivo A_i e lunghezza P_i , per cui come appena visto $F_i = \max(F_{i-1}, A_i) + P_i$. Quindi, il prossimo pacchetto da trasmettere è sempre quello che ha il timestamp più basso tra tutti i flussi, cioè, il pacchetto che dovrebbe terminare la trasmissione prima di tutti gli altri. Questo garantisce differenze minime tra i flussi e quindi la massima equità.

Esempio di accodamento equo in azione (immagine sotto):

- (a) i pacchetti che terminano prima vengono inviati per primi;
- (b) l'invio di un pacchetto già in corso viene completato



Nel caso (a) nel Flow 1 arriva un pacchetto di lunghezza 5 e essendo l'unico pacchetto presente il suo timestamp sarà $F_i = S_i + P_i = 5$, poiché S_i è 0. Successivamente arriva un secondo pacchetto con lunghezza 3, il cui timestamp vale $F_i = S_i + P_i = 8$ in quanto il pacchetto precedente termina la sua trasmissione nel momento 5 (era partita da 0) e $P_i = 3$. Nel flow 2 nel frattempo arriva un pacchetto da 10, il quale avrà un timestamp F_i di 10, poiché è il primo della sua lista. L'algoritmo dunque invierà i pacchetti nell'ordine: pacchetto 1 ($F_i = 5$), pacchetto 2 ($F_i = 8$) e pacchetto 3 ($F_i = 18$, poiché inizia la trasmissione nel momento 8 e, essendo di lunghezza 10, termina la trasmissione nel tempo 18).

Nel caso (b), invece, si presuppone che la situazione sia la stessa, solo che nel mentre che il pacchetto 3 viene inviato, nel Flow 1 arrivi un nuovo pacchetto il cui timestamp iniziale è $F_i = 2$ poiché è il primo della sua lista. Esso partirà in realtà dopo del pacchetto 3, in quanto non si ferma la trasmissione già in atto e quindi il suo timestamp di partenza sarebbe $\max(F_{i-1}, A_i) = \max(18, 0) = 18$. Nel caso in cui, invece il pacchetto 4 fosse arrivato in contemporanea alla decisione di partenza, si sarebbe dovuto prendere il timestamp minore dei due (nel caso del pacchetto 4, visto che il suo timestamp $F_i = 2$ è un tempo già passato si deve ricalcolarlo).

Si consideri l'esempio visto prima di un router che gestisce quattro flussi: F1 con pacchetti da 1000 byte e F2, F3, F4 con pacchetti da 100 byte. Secondo l'accodamento equo si trasmette prima un pacchetto da F2, poi uno da F3, uno da F4, uno da F2, ecc. fino a quando non vengono inviati 10 pacchetti da ciascun flusso; infine, è possibile trasmettere un pacchetto da F1. Poi tutta la procedura si ripete. La banda assegnata a ciascun flusso è la stessa, per cui si ha *equità = 1*.

Esercizio:

Un router applica l'accodamento *FairQueing* a tre flussi A, B, C, con attualmente pacchetti della seguente durata (in qualche unità di tempo): A: 100, 200, 100, 1000; B: 300, 200, 500; C: 400. (a) A che istante termina la trasmissione del pacchetto di C? (b) Se all'istante 1200 arriva un altro pacchetto del flusso C di durata 100, quando inizia la sua trasmissione?

Packet	Duration	Finish time
A1	100	100
A2	200	300
A3	100	400
A4	1000	1400

Packet	Duration	Finish time
B1	300	300
B2	200	500
B3	500	1000

Packet	Duration	Finish time
C1	400	400
C2	100	1300

(rappresentazione grafica tempi)

R: (a) per questa risposta va considerato che C ha solo C1 come pacchetto (dell'immagine sopra) per cui costruendo una tabella di output si ottiene che C1 venga inviato nel tempo 1000. (Nel caso di parità di timestamp essendoci una sorta di round robin si dovrebbe dare priorità a una coda usata meno, anche se nell'immagine della soluzione affianco non sempre viene seguita questa regola.)

(b) Seguendo le stesse regole di decisione di invio dei pacchetti, C2 viene inviato nel momento 1800.

Packet	Output order	
	Start	Finish time
A1	0	100
B1	100	400
A2	400	600
C1	600	1000
A3	1000	1100
B2	1100	1300
B3	1300	1800
C2	1800	1900
A4	1900	2900

Controllo della congestione di TCP

Il controllo della congestione TCP è stato introdotto in Internet alla fine degli anni '80 da Van Jacobson, circa otto anni dopo che lo stack di protocollo TCP/IP era diventato operativo. Prima di allora, Internet soffriva di un collasso della congestione (*congestion collapse*). Infatti, gli host inviavano i loro pacchetti in Internet alla velocità massima consentita dall'*AdvertisedWindow*, tuttavia i router intermedi potevano avere capienza più limitata di essa, per cui si verificava una congestione in qualche router (che causava la caduta dei pacchetti), gli host andavano in time out e ritrasmettevano i loro pacchetti, causando una congestione ancora maggiore.

Da quel momento si sono introdotti degli algoritmi di controllo della congestione, quindi tutte le architetture TCP/IP devono obbligatoriamente introdurre questi algoritmi (oltre a quelli di controllo del flusso). L'idea del controllo della congestione TCP è che ogni sorgente deve determinare quanta capacità è disponibile nella rete, in modo da sapere quanti pacchetti può far transitare con sicurezza.

Tra di essi c'è l'algoritmo di Nagle, che, una volta che una determinata sorgente ha un numero di pacchetti in transito, utilizza l'arrivo di un ACK come segnale che uno dei suoi pacchetti ha lasciato la rete e che quindi è sicuro inserire un nuovo pacchetto nella rete senza aumentare il livello di congestione. Utilizzando gli ACK per ritmare la trasmissione dei pacchetti, si dice che il TCP è autococcante (*self-clocking*).

Come già detto, esistono diversi algoritmi di controllo del flusso (la pagina web di Wikipedia elenca più di 30 algoritmi). Essi si differenziano per: feedback osservato per il controllo, nodi coinvolti nell'algoritmo (mittente / ricevitore / router), parametri che mira a ottimizzare (latenza / larghezza di banda / ritrasmissioni / ecc.), equità, ecc. La maggior parte di questi algoritmi può coesistere sulla stessa rete, per cui si possono avere diversi algoritmi di controllo della congestione in diversi nodi.

Infatti, oggi ogni sistema operativo ha un algoritmo "general purpose" di default:

- CUBIC: predefinito nei kernel Linux dalla versione 2.6.19. (novembre 2006)
- TCP composto: predefinito in Windows da Vista e Server 2008.

L'amministratore può cambiare l'algoritmo predefinito con altri, se necessario (con molta attenzione!). Ad esempio, gli algoritmi più recenti includono:

- Proportional Rate Reduction (PRR): disponibile nei kernel Linux per migliorare il recupero delle perdite dalla versione 3.2 (gennaio 2012). Dovrebbe essere più efficace quando si ha a che fare con reti wireless con perdita di pacchetti
- Bottleneck Bandwidth and Round trip-time (BBR): sviluppato da Google per QUIC (2016, protocollo di trasporto all'interno del browser e poi portato su kernel, per cui è sostituibile a quello di CUBIC) e

disponibile nei kernel Linux per consentire un controllo della congestione basato su modelli dalla versione 4.9. Non si basa sulla perdita di pacchetti.

Verranno approfonditi in seguito solo alcuni aspetti adottati negli algoritmi classici (*Tahoe* 1988 e *Reno* 1990), ma utilizzati ancora oggi.

Finestra di congestione (congestion window)

Il protocollo TCP mantiene una nuova variabile di stato per ogni connessione, chiamata *CongestionWindow* o *cwnd*, che viene utilizzata dalla sorgente per limitare la quantità di dati in transito in un determinato momento (è un singolo numero che dice quanti dati sono ancora inviabili sulla rete senza mandare in congestione i router intermedi). La finestra di congestione è la controparte del controllo di congestione della finestra pubblicizzata (*Advertized Window*) del controllo di flusso. Il TCP è stato modificato in modo tale che il numero massimo di byte di dati non riconosciuti consentito sia ora il minimo della finestra di congestione e della finestra pubblicizzata.

A questo punto, la finestra effettiva di TCP viene modificata come segue:

- $\text{MaxWindow} = \text{MIN}(\text{CongestionWindow}, \text{AdvertisedWindow})$
- $\text{EffectiveWindow} = \text{MaxWindow} - (\text{LastByteSent} - \text{LastByteAcked})$

In altre parole, *MaxWindow* sostituisce *AdvertisedWindow* nel calcolo di *EffectiveWindow*. In questo modo, una sorgente TCP non può inviare più velocemente di quanto possa fare il componente più lento, la rete o l'host di destinazione.

Il problema è come TCP viene a sapere un valore appropriato per la *CongestionWindow*. A differenza della finestra pubblicizzata, che viene inviata dal lato ricevente della connessione, non c'è nessuno che invii una finestra di congestione adeguata al lato mittente di TCP. Una soluzione potrebbe essere quella che sia il router intermedio a avvisare la sorgente di riallentare l'invio dei pacchetti, tuttavia non è sicuro fidarsi dei pacchetti che girano in rete da parte di dispositivi sconosciuti. Per cui la risposta è che la sorgente TCP debba dedursi da sé il livello di congestione (lo percepisce dallo stato/signali impliciti della rete e poi si calcola un valore approssimato) e impostare di conseguenza la finestra di congestione in base al livello di congestione che percepisce nella rete. In particolare, un mittente testa la capacità della rete provando a inviare su di essa il numero di pacchetti in base alla finestra pubblicizzata e poi "resta in ascolto" di una eventuale congestione (controllo della congestione), da cui poi capisce/deduca quanto grande è (approssimativamente) la finestra di congestione.

Questo comporta una diminuzione della finestra di congestione quando il livello di congestione aumenta e un aumento della finestra di congestione quando il livello di congestione diminuisce. Questo meccanismo è comunemente chiamato *aumento additivo/diminuzione moltiplicativa* (*additive increase / multiplicative decrease*, AIMD).

Algoritmo AIMD:

L'algoritmo AIMD, quindi, segue una formula matematica in cui si ha:

- $w(t)$: finestra di congestione nella fascia temporale t
- $a > 0$: parametro di aumento additivo
- $0 < b < 1$: fattore di diminuzione moltiplicativo

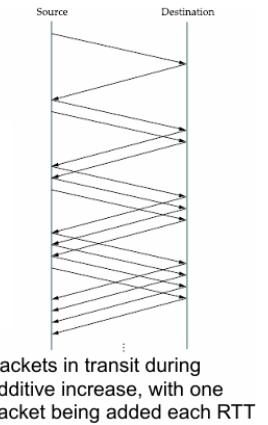
Il mittente invia uno slot di pacchetti e aspetta un riscontro/segnale dalla rete. Se vede che non si è verificato nulla, allora prova a "tirare un po' di più la corda" inviando qualche pacchetto in più, altrimenti se si era verificata una congestione riduce i pacchetti. Questo è riassumibile in una formula, ottenendo che la finestra nella fascia temporale $t + 1$ è definita da:

$$w(t + 1) = \begin{cases} w(t) + a, & \text{se la congestione non è stata rilevata} \\ w(t) \times b, & \text{se la congestione è rilevata} \end{cases}$$

In TCP, si ha che $a = 1$ MSS e $b = 0.5$. Sebbene la *CongestionWindow* sia definita in termini di byte, questi valori sono da intendersi come numero di pacchetti (interi). La *CongestionWindow* non può scendere al di sotto della dimensione di un singolo pacchetto, cioè della dimensione massima del segmento (MSS).

A questo punto resta da chiarire come fa la sorgente a determinare che la rete è congestionata e che deve diminuire la finestra di congestione. La risposta sta sul motivo principale per cui i pacchetti non vengono consegnati (e si verifica un timeout): il pacchetto è stato abbandonato a causa della congestione di un router intermedio. Questo perché è raro che un pacchetto venga abbandonato a causa di un errore durante la trasmissione (eccezione per le connessioni wireless), per cui l'unico motivo per cui si verifica è che stato buttato via volontariamente (o perso). Pertanto, TCP interpreta i timeout come un segnale di congestione e riduce la velocità di trasmissione. In particolare, ogni volta che si verifica un timeout, la sorgente imposta la *CongestionWindow* alla metà del suo valore precedente ($b = 0.5$).

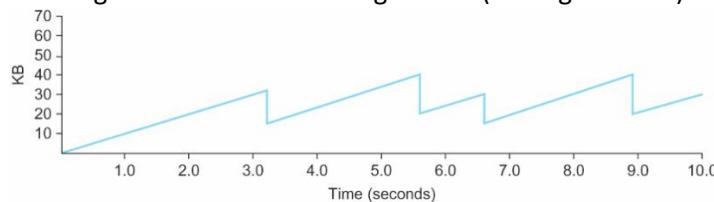
Invece, per quanto riguarda l’“aumento additivo” della finestra di congestione per sfruttare la nuova capacità disponibile nella rete, ogni volta che la sorgente invia con successo (senza errori, perdite, rinvii, ecc.) un pacchetto di *CongestionWindow*, cioè ogni pacchetto inviato durante l’ultimo RTT ha ricevuto un ACK, aggiunge l’equivalente di un pacchetto alla *CongestionWindow* ($a = 1$).



In realtà, TCP non aspetta un’intera finestra di ACK per aggiungere un pacchetto alla finestra di congestione, ma incrementa *CongestionWindow* di una frazione di MSS ogni volta che viene ricevuto un ACK. Supponendo che l’ACK ricevuto confermi la ricezione di (nuovi) k byte, la frazione è $k/CongestionWindow$. In particolare, quando si riceve un ACK per nuovi k byte (cioè si incrementa *LastByteAck* di k byte) la finestra di congestione viene incrementata come segue:

- $Incremento = MSS \times (k/CongestionWindow)$
- $CongestionWindow = CongestionWindow + Incremento$

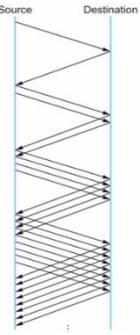
In questo modo, poiché durante un round (che dura circa RTT) viene inviata un’intera *CongestionWindow*, se tutti questi dati vengono ACKati l’incremento complessivo è esattamente di 1 MSS. Se vengono inviati meno dati, l’incremento complessivo è proporzionalmente inferiore a 1 MSS. Questo meccanismo prende anche il nome di “modello a dente di sega” della finestra di congestione (immagine sotto).



Partenza lenta (slow start):

Il meccanismo di aumento additivo appena descritto è l’approccio giusto da utilizzare quando la sorgente sta operando vicino alla capacità disponibile della rete, ma richiede troppo tempo per aumentare una connessione quando questa parte da zero. Il TCP prevede quindi un secondo meccanismo, chiamato (ironicamente) *slow start* (partenza lenta), che viene utilizzato per aumentare rapidamente la finestra di congestione a partire da un avvio a freddo (es.: subito dopo l’handshake). L’avvio lento aumenta effettivamente la finestra di congestione in modo esponenziale, anziché lineare, che è comunque più lento rispetto al lancio di una quantità di dati di una finestra pubblicizzata in una volta sola (per questo è “lenta”, perché rispetto all’aumento “a cannone” in cui si usa subito tutta la finestra, questa invia più piano).

In particolare, la sorgente inizia impostando la *CongestionWindow* a un unico pacchetto. Quando arriva l'ACK per questo pacchetto, TCP aggiunge 1 a *CongestionWindow* e invia due pacchetti. Dopo aver ricevuto i due ACK corrispondenti, TCP incrementa *CongestionWindow* di 2 per ogni ACK e poi invia quattro pacchetti. E così via. Il risultato finale è che TCP raddoppia effettivamente il numero di pacchetti in transito a ogni RTT. Quando poi si raggiunge il limite della finestra si applica la stessa regola del dimezzo (diminuzione moltiplicativa).



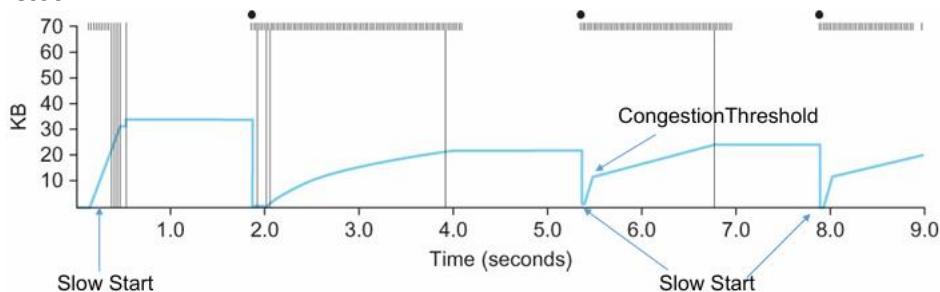
Oltre all'inizio di connessione (handshake), la seconda situazione in cui viene utilizzato l'avvio lento si verifica quando la connessione si interrompe in attesa di un timeout. Questo ricalca l'algoritmo della finestra scorrevole di TCP: quando un pacchetto viene perso, la sorgente raggiunge un punto in cui ha inviato tutti i dati consentiti dalla finestra pubblicizzata e quindi si blocca in attesa di un ACK che non arriverà. Alla fine si verifica un timeout, ma a questo punto non ci sono più pacchetti in transito, il che significa che la sorgente non riceverà alcun ACK per "cronometrare" la trasmissione di nuovi pacchetti. Dopo il timeout, la sorgente reinvia il pacchetto perso e riceve un singolo ACK cumulativo che riapre l'intera finestra pubblicizzata. Ora, si può scaricare sulla rete un'intera finestra di dati tutta in una volta sarebbe troppo aggressivo per la rete. Invece, la sorgente utilizza l'avvio lento per riavviare il flusso di dati.

Dopo il timeout, anche se la sorgente utilizza di nuovo l'avvio lento, ora conosce più informazioni rispetto all'inizio della connessione. La sorgente ha un valore attuale (e utile) di *CongestionWindow*: il valore di *CongestionWindow* che esisteva prima dell'ultima perdita di pacchetti, diviso per 2 come risultato della perdita. Questo valore può essere considerato come la finestra di congestione "target". L'avvio lento viene utilizzato per aumentare rapidamente la velocità di invio fino a questo valore, e poi viene utilizzato un aumento additivo oltre questo punto.

È necessario ricordare la finestra di congestione "target" risultante dalla diminuzione moltiplicativa e la finestra di congestione "effettiva" utilizzata dall'avvio lento. Il TCP introduce una variabile temporanea per memorizzare la finestra target, tipicamente chiamata *CongestionThreshold* o *ssthreshold*, che viene impostata uguale al valore di *CongestionWindow* risultante dalla diminuzione moltiplicativa. La variabile di *CongestionWindow* viene quindi reimpostata a un pacchetto e viene incrementata di un pacchetto per ogni ACK ricevuto finché non raggiunge il *CongestionThreshold*, a quel punto viene incrementata di un pacchetto per RTT (come per il normale AIMD).

Di seguito un'immagine esplicativa del fenomeno del controllo della congestione TCP in AIMD con avvio lento (e nessun'altra tecnica) e la legenda dei segni/linee utilizzati:

- linee colorate: rappresentano il valore di *CongestionWindow* nel tempo;
- punti pieni nella parte superiore del grafico: sono i timeout;
- segni di hash nella parte superiore del grafico: indicano il tempo di trasmissione di ogni pacchetto;
- barre verticali: il tempo in cui è stato trasmesso per la prima volta un pacchetto che alla fine è stato ritrasmesso



Lo slow start sarebbe da considerarsi esponenziale, anche se nell'immagine non sembra. La prima parte del grafico rappresenta i pacchetti che sono stati inviati e di cui non si ha ricevuto il riscontro (barre verticali

lunghe), per cui ci si ferma per un periodo di tempo, fino allo scadere dei timeout, dopo dei quali si interviene. A questo punto si riparte da un segmento solo (con partenza lenta) e si reinviando i dati come fatto prima. Si verifica poi un'altra perdita di pacchetti, per cui scadrà un altro timeout. Si riparte nuovamente con una slow start fino a metà della capacità di prima, prima della perdita del pacchetto (*CongestionThreshold*), da lì in poi si procede lineari, fino a quando non si perde un altro pacchetto e si ricomincia con questa seconda strategia.

Ritrasmissione veloce:

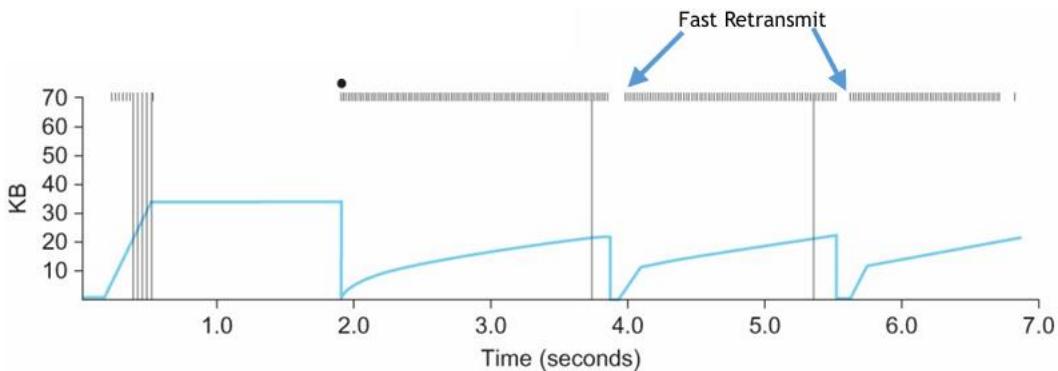
I meccanismi descritti finora facevano parte della proposta originale (di Van Jacobson) di aggiungere il controllo della congestione al TCP. Ben presto si scoprì che l'implementazione a grana grossa dei timeout TCP portava a lunghi periodi di tempo in cui la connessione si spegneva in attesa della scadenza del timer. Per questo motivo, fu aggiunto al TCP un nuovo meccanismo chiamato *fast retransmit* (ritrasmissione veloce, in 4.3BSD Unix "Tahoe", 1988). La ritrasmissione veloce è un'euristica che a volte attiva la ritrasmissione di un pacchetto caduto prima del normale meccanismo di timeout.

Si ricorda che ogni volta che un pacchetto di dati arriva al lato ricevente, quest'ultimo risponde con una conferma, anche se questo numero di sequenza è già stato riconosciuto. Pertanto, quando un pacchetto arriva fuori ordine, cioè TCP non può ancora riconoscere i dati contenuti nel pacchetto perché i dati precedenti non sono ancora arrivati, TCP reinvia lo stesso riconoscimento inviato l'ultima volta. Questa seconda trasmissione dello stesso riconoscimento è chiamata ACK duplicato. Un ACK duplicato è un messaggio di riscontro dello stesso pacchetto, che non si sovrappone ai dati e non modifica la finestra pubblicizzata del destinatario.

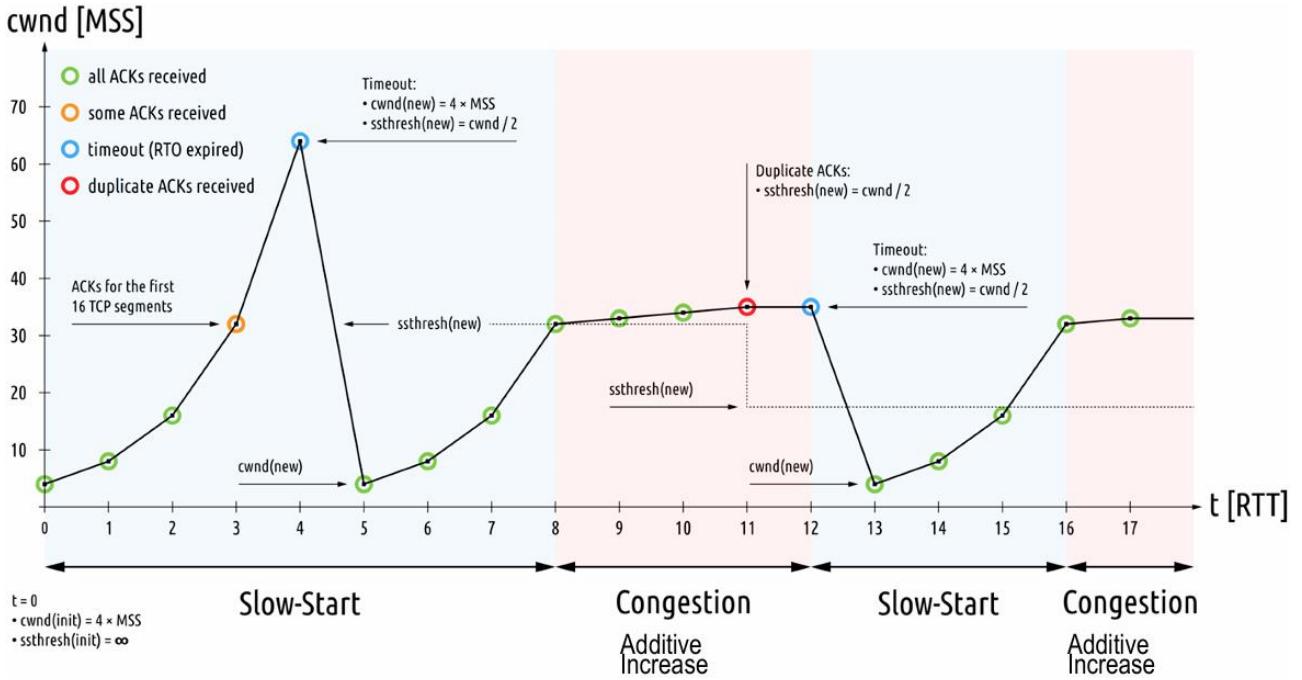
Quando il lato mittente vede un ACK duplicato, sa che l'altro lato deve aver ricevuto un pacchetto fuori ordine, il che suggerisce che un pacchetto precedente potrebbe essere stato perso. Poiché è anche possibile che il pacchetto precedente sia stato solo ritardato e non perso, il mittente aspetta di vedere un certo numero di ACK duplicati e poi ritrasmette il pacchetto mancante. In pratica (in Tahoe e Reno), prima di ritrasmettere il pacchetto, TCP aspetta di vedere tre ACK duplicati (cioè quattro ACK che riconoscono lo stesso pacchetto). Quindi, Tahoe imposta la soglia di avvio lento alla metà della finestra di congestione corrente, riduce la finestra di congestione al valore di base (es.: 1 MSS) e ripristina lo stato di avvio lento.

Di seguito un'immagine esplicativa del fast retransmission di Tahoe e la legenda:

- linea colorata: finestra di congestione;
- pallino solido: timeout;
- segni di hash: tempo in cui ogni pacchetto viene trasmesso;
- barre verticali: tempo in cui è stato trasmesso per la prima volta un pacchetto che alla fine è stato ritrasmesso



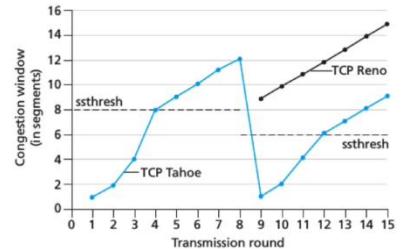
Di seguito, un'altra immagine più esplicativa del fenomeno analizzato (in particolare nel passaggio da esponenziale a lineare della trasmissione, post timeout):



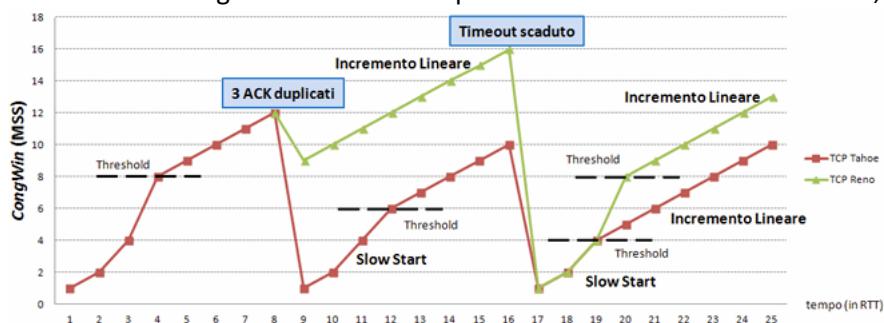
In questo caso si parte con SS (slow start) esponenziale fino al timeout. Nel mentre si ricevono solo parte degli ACK di risposta, che verranno considerati come *threshold*, da cui poi si procederà linearmente (alla ritrasmissione successiva dopo del timeout). E poi i passaggi successivi ricalcano questo appena descritto.

Recupero veloce:

Quando il meccanismo di ritrasmissione veloce segnala una congestione, invece di far scendere la finestra di congestione fino a un solo pacchetto ed eseguire l'avvio lento, è possibile utilizzare gli ACK ancora presenti nella pipe per cronometrare l'invio dei pacchetti. Questo meccanismo, chiamato *fast recovery*, elimina di fatto la fase di avvio lento che si verifica tra il momento in cui fast retransmit rileva un pacchetto perso e l'inizio dell'incremento additivo.



È implementato ad esempio in Reno, in cui se vengono ricevuti tre ACK duplicati si esegue la ritrasmissione veloce, si impone $ssthresh$ alla metà della finestra di congestione e si setta la nuova finestra di congestione a $ssthresh + 3$ MSS, invece di impostarla a 1 MSS come in Tahoe (recupero veloce). In ogni caso, se un ACK va in time out, riduce la finestra di congestione a 1 MSS e ripristina lo stato di avvio lento. Sotto, i due a confronto:



In questo caso l'algoritmo può riconoscere in automatico i pacchetti duplicati come congestione, senza aspettare il timeout, e quindi ricominciare da capo a dimezzare la finestra, ripartire da 0, ecc. (restando con l'incremento esponenziale).

Come appena visto, con questi algoritmi si osservano due aspetti della rete per capire la presenza di una congestione: la perdita di pacchetti e i timeout scaduti. Infatti, talvolta si mal interpreta questi “segnali” come una congestione quando magari è solo un problema del canale e non di un qualche router congestionato, con la conseguenza di dimezzo della finestra e rallentamento della comunicazione.

Esercizi:

Un certo host TCP, che usa un algoritmo di congestion control con partenza lenta e ritrasmissione veloce, apre una nuova connessione, con $MSS=1400$. Esegue 3 round completi in partenza lenta, partendo da $CongestionWindow = 2 \text{ MSS} = 2800$. Se non vengono ricevuti gli ACK per due segmenti inviati nel terzo round, quanto vale $CongestionWindow$ all'inizio del quarto round?

R: Partendo con $CongestionWindow = 2 \text{ MSS}$ si inviano 2 pacchetti nel primo round; dopo si raddoppia la $CongestionWindow = 4 \text{ MSS}$, quindi 4 pacchetti inviati; al terzo round $CongestionWindow = 8 \text{ MSS}$ con 8 pacchetti inviati; dopo il terzo round la $CongestionWindow = 16 \text{ MSS}$ a cui vanno sottratti 2 segmenti ACK non ricevuti per cui $CongestionWindow = 14 \text{ MSS} \rightarrow 19600$.

Una connessione TCP, con $MSS=1400$ e recupero veloce, si trova nella fase additiva, e inizia il round con $CongestionWindow = 10000$. Dopo aver inviato un po' di segmenti lunghi MSS e ricevuto 5 ACK, scade il timeout di un segmento inviato in precedenza. Quanto diventa la $CongestionWindow$?

R: l'incremento ricevuto per ogni ACK è di $MSS * MSS / CW = 196$ byte. Quindi, dopo 5 segmenti riscontrati, CW è diventato $10000 + 196 * 5 = 10980$. A questo punto scade il timeout, e quindi CW viene dimezzata diventando 5490.

La differenza sostanziale tra Tahoe e Reno è che il primo aspetta di ricevere quattro ACK uguali per accettare l'ipotesi di una congestione (in realtà al terzo duplicato e poi al quarto interviene) e poi riparte da 1 ritrasmettendo il segmento richiesto dall'ACK, invece Reno oltre a questi passaggi fa anche altre cose e parte dal *threshold* + un certo valore. Tahoe mette il *threshold* a metà del corrente valore della *CongestionWindow* e riparte da 1 MSS in slow start esponenziale fino al raggiungimento del valore impostato e poi procede linearmente, invece Reno non fa la partenza lenta, ma invia il segmento mancante al destinatario e poi parte direttamente dalla metà della *CongestionWindow* attuale + 3 MSS (*threshold*) e procede linearmente. Solo in caso di timeout ripartono entrambi da 1 (vedi immagine precedente sul confronto dei due meccanismi).

Meccanismi di evitamento della congestione

È importante sottolineare che la strategia del TCP consiste nel controllare la congestione una volta che si verifica, invece di cercare di evitarla in primo luogo. Infatti, il TCP aumenta ripetutamente il carico che impone alla rete nel tentativo di trovare il punto in cui si verifica la congestione (pacchetti persi), per poi fare marcia indietro da questo punto.

Un'alternativa interessante, ma non ancora ampiamente adottata, è quella di prevedere quando sta per verificarsi una congestione e quindi ridurre la velocità di invio dei dati da parte degli host appena prima che i pacchetti inizino a essere scartati. Questa strategia si chiama “evitamento della congestione” (*congestion avoidance*), per distinguerla dal controllo della congestione.

Rilevamento precoce casuale (RED)

Un primo meccanismo interessante è il Rilevamento precoce casuale (*Random Early Detection*, RED) inventato da Sally Floyd e Van Jacobson nei primi anni Novanta. Secondo questo meccanismo, ogni router è programmato per monitorare la lunghezza della propria coda e, quando rileva che la congestione è imminente, notifica la sorgente di regolare la propria finestra di congestione. Inoltre, piuttosto che inviare

esplicitamente un messaggio di notifica di congestione alla sorgente, RED è più comunemente implementato in modo da notificare implicitamente la sorgente della presenza di una congestione facendo cadere volontariamente uno dei suoi pacchetti. La sorgente viene quindi effettivamente notificata dal successivo timeout o dall'ACK duplicato.

Questo funziona perché RED è stato progettato per essere utilizzato insieme a TCP, che attualmente rileva la congestione tramite timeout (o altri mezzi per rilevare la perdita di pacchetti, come gli ACK duplicati). Inoltre, come già ribadito, è meglio non fidarsi dei pacchetti che circolano in rete, per cui questo meccanismo di segnalazione indiretta va più che bene, in quanto i mittenti non vengono in realtà notificati direttamente.

Come suggerisce la parte "early" dell'acronimo RED, il gateway lascia cadere il pacchetto prima di quanto dovrebbe, in modo da notificare la sorgente che dovrebbe diminuire la sua finestra di congestione prima di quanto avrebbe fatto normalmente. In altre parole, il router lascia cadere alcuni pacchetti prima di aver esaurito completamente lo spazio del buffer, in modo da indurre la sorgente a rallentare, con la speranza di non doverne lasciare molti in seguito.

Resta quindi da decidere quando far cadere un pacchetto e quale. Per capire l'idea di base di RED, si consideri una semplice coda FIFO. Piuttosto che aspettare che la coda sia completamente piena e quindi essere costretti a lasciare cadere ogni pacchetto in arrivo, si potrebbe decidere di lasciare cadere ogni pacchetto in arrivo con una certa probabilità di caduta ogni volta che la lunghezza della coda supera un certo livello (quindi con probabilità crescente all'aumentare del riempimento della coda FIFO). Questa idea è chiamata *early random drop*. L'algoritmo RED definisce i dettagli di come monitorare la lunghezza della coda e quando far cadere un pacchetto.

In primo luogo, RED calcola la lunghezza media della coda utilizzando una media mobile ponderata simile a quella utilizzata nel calcolo del timeout TCP originale. Cioè, *AvgLen* calcolato come:

$$AvgLen = (1 - w) \times AvgLen + w \times SampleLen$$

dove $0 < w < 1$ e *SampleLen* è la lunghezza della coda quando viene effettuata una misurazione a campione.

Nella maggior parte delle implementazioni software, la lunghezza della coda *SampleLen* viene misurata ogni volta che un nuovo pacchetto arriva al gateway. Nell'hardware, potrebbe essere calcolata a un intervallo di campionamento fisso.

Come è possibile vedere dal grafico, i problemi sorgono quando il router si sovraccarica velocemente o resta per tanto tempo con un alto carico di pacchetti da smaltire. In realtà, il carico transitorio è un problema relativo, in quanto si riesce a gestire abbastanza bene, ma quello più grave è quello più duraturo (es.: FTP, HTTP pesante, download di file grossi, situazioni peer-to-peer, ecc.).

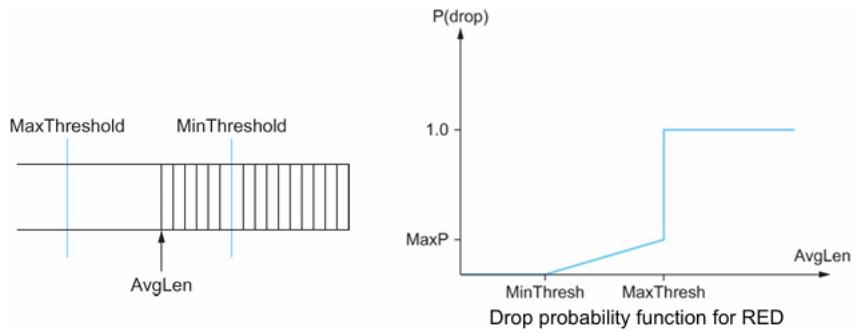
In secondo luogo, RED ha due soglie di lunghezza della coda che attivano determinate attività: *MinThreshold* e *MaxThreshold*. Quando un pacchetto arriva al gateway, RED confronta l'*AvgLen* corrente con queste due soglie, secondo le regole seguenti:

- se $AvgLen \leq MinThreshold$: mette in coda il pacchetto
- se $MinThreshold < AvgLen < MaxThreshold$: calcola la probabilità $P(drop)$ e lascia cadere il pacchetto in arrivo con la probabilità $P(drop)$
- se $MaxThreshold \leq AvgLen$: lascia cadere il pacchetto in arrivo

$P(drop)$ è calcolato come segue (dove *MaxP* è un parametro, descritto sotto):

$$P(drop) = MaxP \times \frac{AvgLen - MinThresh}{MaxThresh - MinThresh}$$

Per decidere se scartare/tenere un pacchetto, si prende un numero random, che se è inferiore alla probabilità $P(drop)$ allora viene scartato, altrimenti si tiene.



MaxP serve a regolare l’“aggressività/conservatività”, ovvero l’angolo della linea obliqua della probabilità nel range $[\text{minTresh}, \text{maxTresh}]$, quindi il livello di arrivo di maxTresh e dopo del quale i pacchetti vengono scartati. In realtà per rendere la probabilità più o meno aggressiva/conservativa si possono cambiare anche i parametri di minTresh e maxTresh , in modo da cambiare l’inclinazione della retta obliqua.

Questo meccanismo di *Congestion Avoidance* è uno di quei servizi che non sono implementabili in un solo livello dello stack, ma necessitano della cooperazione di due o più livelli (livello 3 e 4). Tuttavia, non è nemmeno l’unico implementabile. Esistono alcuni algoritmi lato Sender o lato Receiver o lato Router (e varianti combinate) che consentono di gestire l’evitamento di congestione da punti diversi della rete (vedere pagina Wikipedia che ne elenca e descrive qualcuno), sotto ne verrà illustrato qualcuno.

Esercizi:

Un utente si accorge che un certo router, che implementa la RED con $\text{MinThreshold} = 20$ KByte e $\text{MaxThreshold} = 100$ KByte, perde circa il 10% dei pacchetti. Quanto è piena la coda sul router, in kByte?

R: Dato che MaxP della probabilità non viene fornito, si considera $\text{MaxP}=1$. Dunque, per fare una probabilità di 0.1, bisogna che sia $0.1 = \frac{x-20}{100-20}$, quindi $8 = x - 20$ da cui $x = 28$ KByte.

Un router applica la politica RED ad una coda, con $\text{Minthreshold} = 10$ KByte e $\text{Maxthreshold} = 20$ KB. La coda attualmente è piena a 9 KByte. Arrivano in rapida successione tre pacchetti, di rispettivamente 2 KByte, 3 KByte e 2 KByte. Qual è la probabilità che tutti e tre vengano accodati?

R: In questo caso si fa finta che HeaderLen sia uguale alla SampleLen (?). Il primo pacchetto viene accodato sicuramente (quindi $P_1 = 1$), e la coda si allunga a 11 KByte. La probabilità che il secondo pacchetto sia scartato è $\frac{11-10}{20-10} = \frac{1}{10}$, quindi $P_2 = 1 - \frac{1}{10} = 0.9$. Se il secondo pacchetto è stato accodato, la coda diventa 14 KByte. La probabilità che il terzo pacchetto sia scartato è $\frac{14-10}{20-10} = \frac{4}{10}$, quindi $P_3 = 1 - \frac{4}{10} = 0.6$. La probabilità che tutti e tre i pacchetti vengano accodati è quindi $P = P_1 P_2 P_3 = 1 \times 0.9 \times 0.6 = 0.54 = 54\%$.

Evitare le congestioni basate sulle sorgenti

L’idea generale di queste tecniche consiste nell’osservare qualche segnale proveniente dalla rete che indica che la coda di un router si sta accumulando e che presto si verificherà una congestione se non si interviene. Ad esempio, la sorgente potrebbe notare che, man mano che le code dei pacchetti si accumulano nei router della rete, si verifica un aumento misurabile dell’RTT per ogni pacchetto successivo inviato. In questo caso si usa una tecnica di evitamento di congestione basata sulla sorgente (*source-based congestion avoidance*), di solito utilizzato (con varianti) in Vegas, BBR, ecc.

Un algoritmo di questo genere sfrutta questa osservazione come segue: la finestra di congestione aumenta normalmente come in TCP, ma ogni due ritardi di andata e ritorno l’algoritmo controlla se l’RTT corrente è

maggiori della media degli RTT minimi e massimi visti finora. In caso affermativo, l'algoritmo diminuisce la finestra di congestione di un ottavo, ovvero:

$$\text{CongestionWindow} = \text{CongestionWindow} * 0,875$$

Altrimenti un'altra idea potrebbe essere quella di correggere la grandezza della finestra in base al cambio di RTT e della grandezza della finestra corrente, ovvero una volta ogni due round-trip delay si esegue:

$$(\text{CurrentWindow} - \text{OldWindow}) \times (\text{CurrentRTT} - \text{OldRTT})$$

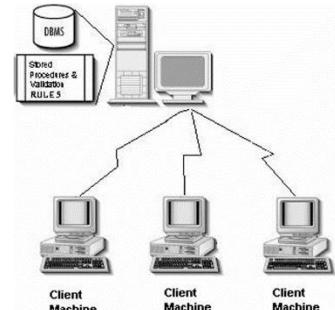
Se il risultato è positivo, significa che sia la finestra si è ingrandita, ma l'RTT è cresciuto, oppure che l'RTT è diminuito e che la finestra si sia rimpicciolita. Invece, se il risultato è 0 o negativo, vuol dire che la rete è stabile e veloce di prima. Da notare che la finestra cambia a ogni aggiustamento e è solita oscillare attorno al punto ottimale.

Sicurezza della rete

Le persone autorizzate possono inviare, recuperare e modificare informazioni a distanza, per cui la restrizione fisica non è più possibile. Minacce alla sicurezza non solo sul computer in cui sono memorizzate le informazioni, ma anche sui canali utilizzati per le trasmissioni.

In generale, la strategia da adottare per la sicurezza è:

1. Identificare gli asset da proteggere
2. Definire gli obiettivi di sicurezza
3. Stabilire la politica di sicurezza
4. Identificare le minacce
5. Sviluppare servizi di sicurezza per implementare controlli/contromisure e piani di emergenza/recupero



Come si suol dire, la sicurezza è sempre un costo, ma l'insicurezza costa di più. Infatti, la sicurezza si ottiene implementando servizi che hanno un costo: economico, di risorse computazionali, di risorse umane per l'implementazione e la gestione, di progettazione e implementazione di programmi più complessi, di problemi pratici per gli utenti, ecc. Ma la mancanza di sicurezza può costare molto di più, ad esempio la perdita di beni vitali (es.: informazioni, dati, ecc.) e/o conseguenze sociali, legali e penali.

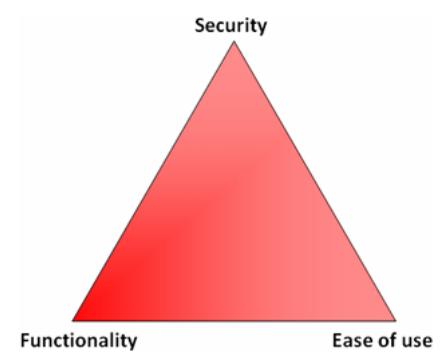
Per cui la sicurezza è sempre un compromesso rispetto al valore degli asset:

$$\text{Costo dei servizi di sicurezza} \leq \text{Costo della perdita di asset} * \text{Probabilità di perdita}$$

Per diminuire il costo dei servizi di sicurezza, si deve diminuire la probabilità di perdita, che a sua volta dipende da: probabilità di presenza di vulnerabilità e servizi di sicurezza implementati per evitare gli attacchi. Ovvero:

$$\text{Probabilità di perdita} = \text{Probabilità di vulnerabilità} * \text{Probabilità di attacco}$$

Ma è anche un compromesso rispetto a funzionalità e usabilità. Non è possibile massimizzare tutti e tre gli aspetti contemporaneamente e il miglioramento di un aspetto diminuisce gli altri. Per cui è importante stabilire quali sono più rilevanti e quali possono essere sacrificati dipendentemente dalla situazione. È da ricordare che è possibile implementare discretamente solo al massimo due delle funzionalità e che una tenderà a essere meno fattibile in presenza delle altre (es.: più funzionalità si aggiungono, più il sistema diventa vulnerabile agli attacchi, in quanto "si allarga" la superficie di attacco, oppure più fattori di sicurezza si implementano, più scomodo da usare diventa il sistema).



Obiettivi e terminologie

Per quanto riguarda gli obiettivi della sicurezza informatica, esistono molti e diversi modelli, ma uno abbastanza comune è la triade della CIA (immagine sotto).



- Riservatezza: si deve proteggere le proprie informazioni riservate. Un'organizzazione deve difendersi da quelle azioni dannose che mettono in pericolo la riservatezza delle sue informazioni. Infatti, anche solo usando Wireshark si riesce a "sniffare" tutto il traffico di pacchetti che arriva su un PC. Allo stesso modo qualsiasi router o dispositivo intermedio a una comunicazione può essere sotto attacco informatico, per cui anche le informazioni riservate potrebbero essere rivelate
- Integrità: le informazioni devono essere modificate costantemente. Le modifiche devono essere effettuate solo da soggetti autorizzati e attraverso meccanismi autorizzati. Ad esempio in un DBMS, i privilegi di inserimento e modifica dei dati deve essere fatto solo da chi è autorizzato. In realtà è lo stesso anche per quanto riguarda i dispositivi intermedi alla comunicazione, infatti anche solo nel meccanismo store-and-forward, nulla vieta a uno switch/router di leggere in contenuto e di inoltrarlo modificato (volontariamente) con un checksum differente
- Disponibilità: le informazioni create e conservate da un'organizzazione devono essere disponibili alle entità autorizzate, quando esse ne hanno necessità. Questa regola va di pari passo con la precedente descritta

Questi tre fattori sono indipendenti tra di loro per cui è possibile combinarli abbastanza liberamente.

Terminologie sulla sicurezza

Di seguito un po' di terminologie comuni sulla sicurezza:

- Minaccia alla sicurezza (*Security Threat*): una potenziale violazione della sicurezza, che esiste quando c'è una circostanza, una capacità, un'azione o un evento che potrebbe violare gli obiettivi di sicurezza e causare danni (= c'è una possibilità/rischio di attacco, ma non è detto che si verifichi)
- Attacco alla sicurezza (*Security Attack*): un attacco alla sicurezza del sistema che deriva da una minaccia intelligente (essere umano), cioè qualsiasi azione che comprometta gli obiettivi di sicurezza delle informazioni di un'organizzazione (es.: dalla minaccia teorica si è passati ai fatti/azione)
- Detezione di sicurezza (*Security Detection*): riconoscere/rilevare che un dispositivo sconosciuto (o con indirizzo IP diverso dal solito) ha effettuato un nuovo accesso con credenziali di qualcun altro, per cui si invia una richiesta di verifica al proprietario dell'account
- Servizio di sicurezza (*Security Service*): un servizio per migliorare la sicurezza del sistema e dei trasferimenti di informazioni. È destinato a contrastare gli attacchi alla sicurezza, utilizzando i meccanismi di sicurezza. È da tenere in considerazione che i dispositivi col passare del tempo diventano vulnerabili (anche per via di malfunzionamenti, es.: condensatori che si bruciano, ecc.) e/o i loro software diventano obsoleti al progredire della tecnologia, per cui diventano meno sicuri
- Meccanismo di sicurezza (*Security Mechanism*): meccanismo progettato per rilevare, prevenire o recuperare un attacco di sicurezza, dunque sono i "mattoni" (singole funzionalità semplici) su cui poi viene costruita tutta la parte di sicurezza

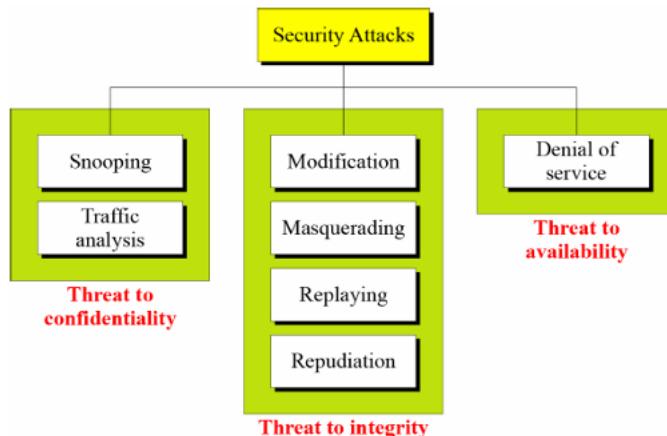
Ad esempio, nel computer di una persona ci sono dei documenti importanti (asset), a cui alcune persone sono interessate ad accedervi (minaccia); a volte il proprietario del computer si allontana lasciando il computer da

solo (vulnerabilità) e mentre è via, una persona non autorizzata può sedersi e usare il suo computer (attacco). Una contromisura può essere quella di implementare un servizio di autenticazione che chieda e garantisca l'identità degli utenti del computer (meccanismi, es.: nome utente/password, impronta digitale (fingerprint), smartcard, tono vocale, scansione retinica, guardia di sicurezza, mail di riscontro di accesso, ecc.).

Molto spesso una contromisura che viene utilizzata è quella di chiedere via mail a un utente se è stato lui ad effettuare un accesso. La mail di verifica può essere inviata a ogni singolo accesso al sistema con tali credenziali, oppure a seguito di un accesso da un dispositivo sconosciuto (es.: tramite riconoscimento di un indirizzo IP diverso dal solito o da un dispositivo nuovo/sconosciuto).

Altro termine importante è quello del recovery, ovvero il ripristino del sistema dopo di un attacco, anche con un *rollback*. Ad esempio, se ci si accorge che un utente non autorizzato ha effettuato modifiche non autorizzate al sistema, si prevede un ripristino con un rollback e si riporta la situazione come era prima dei cambiamenti. Per fare questo serve, quindi, un meccanismo/algoritmo di *backup*, in modo da poter salvare lo stato del sistema.

|| Tipi di attacchi



Attacchi alla riservatezza:

- **Snooping**: accesso non autorizzato ai dati (o ai metadati, che è peggio in quanto si mette in luce gli endpoint, la quantità di dati trasferita, la frequenza delle comunicazioni, luoghi, ecc.) e al computer da parte di hacker, virus, abusi di password, social engineering, ecc. Comprende anche l'intercettazione dei dati durante le trasmissioni (*sniffing*) e i tabulati (es.: delle chiamate telefoniche), per cui è meglio utilizzare l'*encipherment* (= cifratura) dove possibile
- **Analisi del traffico**: ottenere un altro tipo di informazioni monitorando il traffico online. Ci sono molti modi per divulgare informazioni parziali, anche se i dati sono criptati (es.: con chi si parla)

Attacchi all'integrità:

- **Modifica**: significa che l'aggressore intercetta il messaggio e lo modifica. A volte l'aggressore si limita a cancellare o ritardare il messaggio, ma di solito segue un attacco di snooping
- **Mascheramento o spoofing**: avviene quando l'aggressore si spaccia per qualcun altro. L'attaccante può fingere di essere il mittente, il destinatario o entrambi ("man-in-the-middle")
- **Replaying**: l'attaccante ottiene una copia di un messaggio inviato da un utente e successivamente cerca di riprodurlo (senza modifiche)
- **Ripudio**: il mittente del messaggio potrebbe in seguito negare di averlo inviato, oppure il destinatario del messaggio potrebbe in seguito negare di averlo ricevuto. Si noti che questo attacco viene eseguito da una delle due parti legittime e non da una terza entità

Attacchi alla disponibilità:

- Denial of Service (DoS): è un attacco molto comune. Può rallentare o interrompere completamente il servizio di un sistema, generando richieste fasulle per ottenere un carico pesante sulla rete, oppure cancellando le risposte del server per costringere il client a non rispondere, o cancellando le richieste del client, costringendolo a ripetere le richieste. Può essere attuato da una rete coordinata di attaccanti (*distributed DoS, DDOS*). Una contromisura può essere quella di fare in modo che il carico dell'attaccante non sia inferiore a quello dell'obiettivo oppure deviare le richieste illecite, se possibile

Gli attacchi all'integrità sono molto critici e pericolosi: si pensi al caso delle banche, in cui se una comunicazione venisse intercettata e modificata si potrebbe incorrere in grossi rischi (es.: dirottamento dei dati a altri utenti o dei bonifici su altri conti), oppure un utente malintenzionato si finge qualcun altro e accede a conti correnti di un'altra persona. Anche l'invio duplicato di messaggi o di replaying costituisce un serio problema per gli utenti che potrebbero confermare e/o fornire dei dati a terzi. Per questi motivi, è necessario che ci sia un meccanismo di autenticazione e di scambio di chiavi di sicurezza tra i soli endpoint della comunicazione.

Lo scoglio più grande da affrontare è proprio quello dello scambio delle chiavi di sicurezza, infatti, una volta aver implementato quello in maniera funzionante, la sicurezza vien da sé. Se ben progettato, questo meccanismo rende difficile per un attaccante esterno comprendere i dati e modificarli affinché risultino validi.

Il ripudio, invece, è un problema più complesso. Si pensi ad esempio se due host si mettessero d'accordo su un certo scambio di dati e al termine di una transazione andata a buon fine e confermata dal sistema, il mittente improvvisamente ripudiasse la comunicazione e dicesse che non è stato lui a mandare i dati, a quel punto il destinatario penserebbe di aver ricevuto dati sbagliati, quando in realtà magari sono giusti, ma potrebbe non avere modo di verificarli e non sa se scartarli o meno (intanto il mittente ha ricevuto i dati in cambio da parte dell'altro endpoint).

Oppure potrebbe verificarsi il caso in cui è il destinatario a ripudiare i dati per cui il mittente penserebbe di aver mandato dei propri dati al ricevente sbagliato e non avrebbe modo di verificarne l'autenticità. In questo caso la situazione diventa difficile perché il mittente non sa a chi sono arrivati i dati e il destinatario ha i dati, ma dice di non averli ricevuti, quindi si aspetta un reinvio.

Anche in questo caso un sistema di autenticazione renderebbe le cose più sicure, ma non è completamente sufficiente. Ad esempio per garantire un non-ripudio del mittente si potrebbe utilizzare un meccanismo di firma digitale o un altro host intermedio che faccia da "notaio" tra i due endpoint della comunicazione (*notarization*). Invece, per un servizio di non-ripudio del destinatario si può usare la PEC oppure una sorta di raccomandata con ricevuta di ritorno, in modo da garantire che il destinatario abbia ricevuto.

Tuttavia anche attacchi DOS (e DDOS) sono rognosi da gestire. Infatti, essi vengono creati da un qualsiasi host che tenta di connettersi a un server inviando un pacchetto SYN, il server accetta, crea una socket con un timeout di 60 secondi e attende un pacchetto ACK dal mittente. Quando al server, invece che arrivare una sola richiesta SYN, ne riceve tante, inizia a creare tante socket fino a riempire la propria tabella interna e esse restano occupate in attesa di un pacchetto ACK.

Per un host è semplicissimo da implementare, in quanto basta inviare dei pacchetti (il client non alloca nessuna risorsa: è solo il server che lo fa in questo caso). Per questo motivo si cerca sempre di implementare dei meccanismi che fanno in modo che il server non allochi subito le risorse e che i client "facciano un po' più fatica a richiedere il servizio" (es.: SCTP Cookie). Inoltre, tecniche più moderne controllano che non vi siano troppe richieste dallo stesso indirizzo e soprattutto se non vanno a buon fine vengono scartate.

Esistono altre categorie di attacchi, ad esempio le classificazioni tra attacchi attivi e quelli passivi, che si differenziano nel modo di approcciare, ovvero:

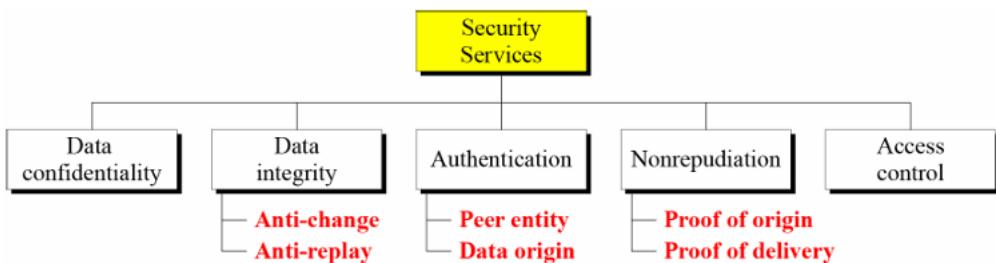
Attacks	Passive/Active	Threatening
Snooping Traffic analysis	Passive	Confidentiality
Modification Masquerading Replaying Repudiation	Active	Integrity
Denial of service	Active	Availability

- Attacchi passivi: apprendono o utilizzano informazioni dai sistemi senza influire sulle risorse del sistema (es.: controllano i metadati dei pacchetti)
- Attacchi attivi: alterano le risorse del sistema o le loro operazioni

Gli attacchi passivi non modificano i dati né influiscono sul sistema, il che li rende difficili da rilevare, per cui l'enfasi è sulla prevenzione, piuttosto che sul rilevamento.

Gli attacchi attivi, invece, modificano i dati o influenzano il sistema, quindi sono più visibili, ma sono difficili da prevenire completamente, però si possono applicare contromisure adeguate. Una contromisura può essere quella di controllare che il pacchetto ricevuto sia corretto dal punto di vista del contenuto e in caso sia una cosa completamente diversa da quella che il destinatario si aspettava, può richiedere l'invio da parte del mittente, anche se talvolta può essere utopico, perché potrebbe non essere semplice da rilevare.

Servizi di sicurezza



- Riservatezza dei dati: protezione dei dati e dei metadati (come il flusso di traffico) dalla divulgazione non autorizzata (es.: ACL)
- Integrità dei dati: garanzia che i dati ricevuti corrispondano a quelli inviati da un'entità autorizzata
- Autenticazione: garanzia che l'entità comunicante sia quella dichiarata. Si suddivide in autenticazione dell'origine dei dati (*data origin*), che è più semplice da realizzare, e autenticazione dell'entità peer (*peer entity*), effettuata all'avvio della connessione e durante la connessione. In realtà in italiano è molto ambigua come parola, perché in inglese si hanno authenticity (= autenticità) e authentication (= autenticazione), che sono un po' diverse, infatti la prima riguarda i dati (autentici, quindi non modificati), mentre la seconda le persone (peer) che stanno parlando (essere sicuri di chi ci parla)
- Non-reputazione: protezione contro la negazione da parte di una delle parti di una comunicazione. Si suddivide in origine, ovvero la prova che il messaggio è stato inviato dalla parte specificata, e in destinazione, cioè la prova che il messaggio è stato ricevuto dalla parte specificata
- Controllo degli accessi: prevenzione dell'uso non autorizzato (lettura, scrittura, modifica, esecuzione, ecc.) di una risorsa. In realtà ha un significato molto ampio, ma il Denial of Service non rientra in questa categoria

In realtà entra in gioco anche la disponibilità, ovvero la garanzia che una risorsa sia disponibile per gli usi previsti. Essa è definita come una proprietà di altri servizi, non come un servizio a sé stante.

Meccanismi di sicurezza

Come già detto, essi sono funzioni di base (“mattoni”) progettate per rilevare, prevenire o recuperare il sistema da un attacco di sicurezza. Non esiste un unico meccanismo in grado di supportare tutti i servizi richiesti. Spesso si replica le funzioni normalmente associate ai documenti fisici (es.: firme e date, protezione dalla divulgazione, dalla manomissione o dalla distruzione, autenticazione notarile o testimoniale, registrazione o licenza, ecc.).

I meccanismi di sicurezza possono essere incorporati nel livello di protocollo appropriato per fornire dei servizi, anche trasversali, come:

- Encipherment: serve per nascondere o coprire i dati e può fornire riservatezza, ma può fornire anche altri servizi come la crittografia e steganografia.
- Integrità dei dati: si cerca di rilevare le modifiche dei dati (es.: valori di checksum, hash, ecc.)
- Firma digitale: garantisce al destinatario l’origine del messaggio
- Scambio di autenticazione (exchange): le parti dimostrano reciprocamente la propria identità
- Padding del traffico: inserimento di dati fasulli nel traffico dati reale, per evitare l’analisi del traffico
- Controllo dell’instradamento: selezione e modifica dei percorsi del traffico per evitare intercettazioni, o per evitare zone “scomode” (come già detto potrebbero esserci questioni politiche, economiche, ecc. per evitare di passare su alcuni router o reti, oppure per usarne delle altre)
- Notarizzazione (notarization): una terza parte fidata controlla le comunicazioni (es.: per evitare il ripudio)
- Controllo degli accessi: metodi per dimostrare che un’entità ha i diritti di accesso alle risorse

Service	Mechanism							
	Enciph-erment	Digital signature	Access control	Data integrity	Authenti-fication exchange	Traffic padding	Routing control	Notari-za-tion
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Non-repudiation		Y		Y				Y
Availability				Y	Y			

Tra servizi e meccanismi, c’è in realtà una relazione (immagine accanto), anche se non tutti i meccanismi possono essere utilizzati per implementare un determinato servizio. I meccanismi discussi in precedenza sono solo “ricette” teoriche per implementare la sicurezza: l’implementazione effettiva dei servizi di sicurezza richiede alcune tecniche.

Modelli per la sicurezza di rete

Un modello è un’astrazione per considerare le minacce e progettare servizi di sicurezza implementando meccanismi di sicurezza e decidendo le strategie. Un modello solitamente nasconde dei dettagli, ma in maniera positiva: consente di potersi concentrare maggiormente sugli aspetti importanti. In questo modo ci si può focalizzare direttamente su un preciso aspetto di sicurezza che si vuole garantire.

Esistono due modelli principali:

- Modello del canale insicuro, per ragionare sulla sicurezza dei dati trasmessi su una rete insicura
- Modello dell’accesso alla rete, per ragionare sulle minacce al nostro sistema informativo

Come è facilmente intuibile, riguardano due aspetti differenti della rete. In questo documento verrà approfondito solo il primo, in quanto è focalizzato sulla sicurezza della rete, mentre il secondo riguarda la sicurezza del sistema.

Modello del canale insicuro (Dolev-Yao)

Quando si parla di modello del canale insicuro si intende quello di Dolev-Yao. Questo modello è applicabile a tutti i livelli di comunicazione, non solo ai canali di rete, indipendentemente dai tipi di host che ne usufruiscono. La situazione con cui si ha a che fare è quella tra un sender e un receiver che sfruttano un canale informativo per comunicare tra loro (figura accanto) e a cui può accedere anche un opponent e eventualmente una terza parte (onesta). In questo caso non è importante sapere come sia stato implementato tale canale. L'opponent può cercare di hackerare liberamente il sistema/canale, leggere e intercettare i pacchetti, modificare o duplicare i messaggi, generare attacchi DOS/DDOS, ecc. a tal punto che si dice (ironicamente, ma non troppo) che il canale sia l'opponent stesso (gli endpoint consegnano a lui i messaggi). In questa situazione si devono implementare dei meccanismi di sicurezza con dei servizi di sicurezza come quelli visti poco fa. Si può dire che il meccanismo diventi una trasformazione di sicurezza (i cerchietti nell'immagine sopra), cioè il messaggio originario non deve essere mandato così com'è sul canale, ma viene prima trasformato in qualche maniera (la trasformazione dipende dal servizio richiesto, es.: un messaggio di confidenzialità dovrà essere cifrato, mentre uno di non-riprudio del mittente con firma digitale, avrà tutto in chiaro ma con la firma affianco che ne garantisca l'autenticità e il destinatario avrà un algoritmo per verificarne la validità).

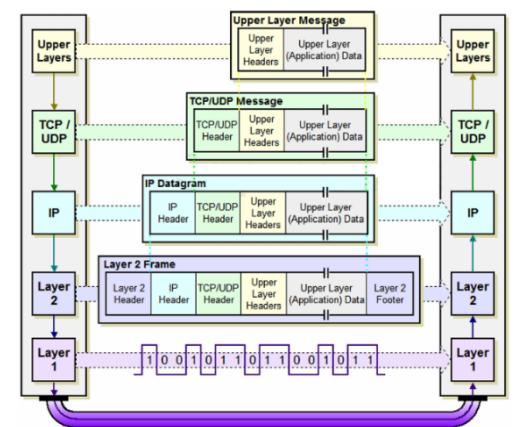
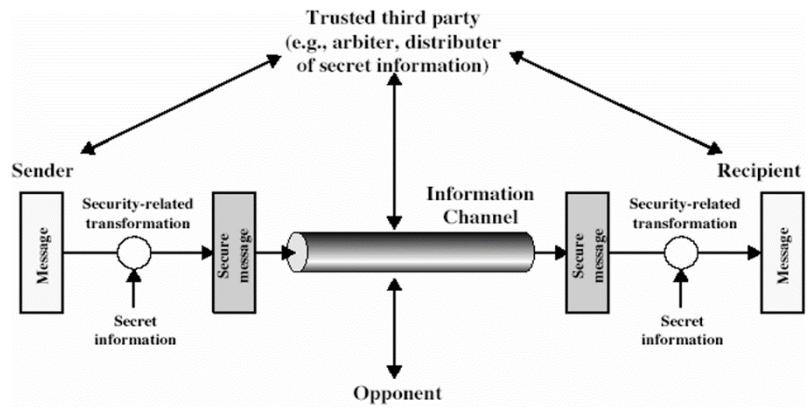
Per poter implementare un servizio che sia fattibile dai due endpoint senza che un opponent combini guai, bisogna poter garantire il passaggio di informazioni segrete (e questa è l'unica cosa veramente segreta di tutto lo schema). Il modello di Dolev-Yao assume che queste informazioni segrete siano state scambiate in precedenza, ma non fornisce quale metodo sia stato utilizzato per fare ciò.

L'eventuale terza parte, invece, può fare da tramite (utilizzando sempre il canale per la comunicazione) e può svolgere diversi ruoli, onesti (es.: coagulare la comunicazione, può scambiare le chiavi di autenticazione o generarne, ecc.).

L'utilizzo di questo modello richiede di:

- progettare un algoritmo adeguato per la trasformazione della sicurezza
- generare le informazioni segrete (chiavi) usate dall'algoritmo
- sviluppare metodi per distribuire e condividere le informazioni segrete
- specificare un protocollo che consenta ai committenti di utilizzare la trasformazione e le informazioni segrete per un servizio di sicurezza.

Questo modello viene ampiamente utilizzato quando si deve analizzare, progettare e implementare delle trasformazioni di sicurezza (o in generale per affrontare la tematica della sicurezza) su un canale insicuro. In realtà non è il solo modello esistente, ma ve ne sono anche altri. Quando si deve, poi, passare alla realizzazione vera e propria, bisogna prendere delle decisioni su come implementare realmente il canale di comunicazione e non è unico. Infatti, durante la comunicazione tra processi, ve ne possono essere diversi (es.: il mezzo fisico su cui vengono trasmessi i segnali, ma anche i canali virtuali/astratti che vengono implementati al di sopra di esso). In realtà, ogni livello del

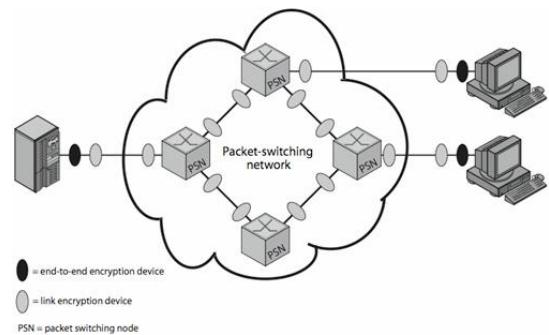


modello OSI o TCP/IP implementa un canale (virtuale) e i servizi di sicurezza possono essere implementati su (quasi) tutti questi canali, con risultati diversi a seconda di quello scelto. E poi resta da chiarire quali di essi è l'effettivo canale insicuro del modello e dove è posizionato.

Posizionamento della sicurezza

Quando si tratta di scegliere dove posizionare il canale, si hanno due alternative principali di posizionamento (*placement of security*):

- **Sicurezza del collegamento** (*link security*): la trasformazione della sicurezza avviene in modo indipendente su ogni collegamento, implica la decodifica del traffico tra i collegamenti e richiede molti dispositivi, ma con chiavi accoppiate. Questo riguarda i primi due livelli dello stack
- **Sicurezza end-to-end**: la trasformazione della sicurezza (es.: la crittografia) avviene tra la sorgente originale e la destinazione finale e necessita di dispositivi ad ogni estremità con chiavi condivise. Questa sicurezza riguarda i livelli dal layer 3 in su dello stack



Queste due alternative sono implementabili entrambe (anzi consigliato), quindi NON è una scelta tra le due. Come è possibile vedere dai pallini in grigio dell'immagine appena sopra (che equivalgono agli stessi cerchietti dell'immagine vista prima di essa), i dati quando devono essere inviati vengono prima cifrati/trasformati e poi decodificati al momento di arrivo al destinatario, mentre quando passano tra un router e un altro del canale sono in chiaro, in quanto la trasformazione coinvolge soltanto il collegamento. Quindi per proteggersi dai possibili sniffing dei router (o qualsiasi altra minaccia), è meglio utilizzare la crittografia end-to-end.

In realtà, quando si utilizza la crittografia end-to-end si deve lasciare le intestazioni in chiaro, in modo che la rete possa instradare correttamente le informazioni. Quindi, sebbene i contenuti siano protetti, i flussi di traffico non lo sono. Per cui l'ideale sarebbe avere entrambe le cose insieme: l'end-to-end protegge i contenuti dei dati sull'intero percorso e fornisce l'autenticazione, mentre il collegamento protegge i flussi di traffico dal monitoraggio. Ogni protocollo di comunicazione o applicazione potrebbe avere le sue regole di trasformazione dei dati (es.: WPA2, SSH, PCP, SMAIL, ecc.).

Sicurezza nei vari livelli

Come già detto, la funzione di sicurezza può essere collocata a diversi livelli del modello di riferimento OSI. Con il passaggio a livelli più alti vengono protette meno informazioni, ma è più sicuro, nonostante sia più complesso e con più entità e chiavi. Il numero di chiavi necessarie in una rete è $O(n^2)$, dove n è il numero di parti. A livello di rete, si ha una chiave per ogni coppia di host, mentre a livello di applicazione, una chiave per ogni coppia di applicazioni.

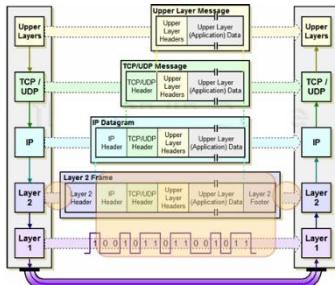
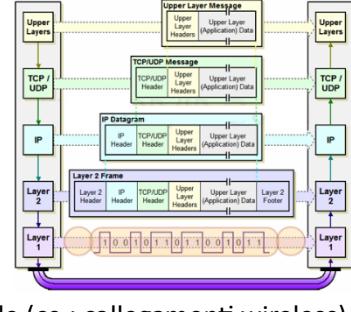
Infatti, a livelli inferiori si hanno più informazioni che sono protette dal servizio, ma sono anche più trasparenti per l'utente/programmatore, tuttavia è più complesso e costoso (a volte impossibile) da implementare efficacemente (e vulnerabile negli switch).

A livelli più alti, invece, si hanno meno informazioni protette, ma meglio, però il meccanismo è più invasivo per l'utente/programmatore e necessita di più chiavi.

Di seguito sono riportate le diverse implementazioni di ogni livello dello stack.

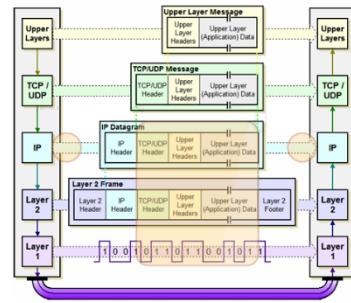
Sicurezza a livello fisico:

Le comunicazioni sono sicure a livello di collegamento e sono implementate da sensori per rilevare le intrusioni fisiche sul mezzo (es.: sigilli, barriere a infrarossi, sensori di pressione, ecc.). Sono gestite dalle telecomunicazioni e coprono *tutto*. Per poter hackerare il mezzo bisogna esserci fisicamente collegati. A livello militare i cavi fisici vengono fatti passare in un tubo sotto pressione e se qualcuno cerca di tagliarlo per accedere ai cavi, i sensori di pressione chiudono automaticamente la connessione. Tuttavia è un servizio costoso: ogni collegamento richiede la propria sicurezza e non sempre possibile (es.: collegamenti wireless).



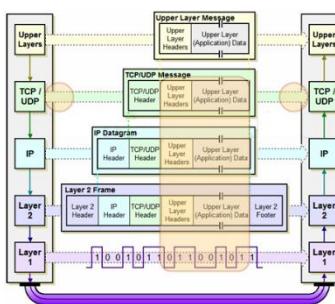
Sicurezza a livello di collegamento dati:

Le comunicazioni sono sicure a livello di collegamento e sono implementate nelle schede di rete (es.: WEP, WPA). Sono gestite da un amministratore di sistema e coprono i dati dell'applicazione, il trasporto e l'intestazione della rete, ma non l'intestazione del frame (bisogna poter leggere a chi è indirizzato). Tuttavia la sicurezza dei datalink è vulnerabile se il traffico deve attraversare i router: questi ultimi devono decriptare i messaggi per ottenere informazioni di routing, che possono quindi essere intercettate da intrusi. In realtà essendoci i MAC address in chiaro, anche solo "sniffando" pacchetti (senza essere connessi in rete) è possibile intercettare il traffico di dati e che device riguardano (es.: Alexa che invia ciò che sente a Amazon).



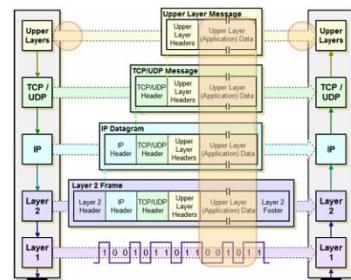
Sicurezza a livello di rete:

Le comunicazioni sono sicure a livello di sistema (non ad-hoc per le applicazioni) e sono implementate nel kernel. Sono gestite dagli amministratori di sistema, sono trasparenti per gli sviluppatori e coprono i dati dell'applicazione e l'intestazione del trasporto (no intestazioni IP). Ad esempio: IPsec è un'integrazione IP che può implementare sia l'autenticazione che la confidenzialità. A questo livello sono implementate anche le VPN.



Sicurezza a livello di presentazione/trasporto:

Le comunicazioni sono sicure lato end-to-end e sono implementate come librerie o API di sistema (libere e scaricabili). I programmati devono utilizzare esplicitamente il servizio, ma non programmarlo. Qui si coprono i dati dell'applicazione e l'intestazione (se presente), ma non gli indirizzi e le porte utilizzate (perché bisogna poterli leggere). Ad esempio: TLS/SSL, viene utilizzato per "mettere in sicurezza" molti protocolli esistenti orientati alla connessione (http → HTTPS, POP → POP3, IMAP → IMAPS, ecc.).



Sicurezza a livello di applicazione:

È implementata direttamente nelle applicazioni (es.: Telegram usa un suo algoritmo per la cifratura dei messaggi) e copre solo i dati dell'applicazione, infatti si implementano specifici servizi di sicurezza per risolvere specifici problemi (fino a prima erano problematiche generiche relative alla rete, qui è per le app). Infatti, qui si forniscono molti protocolli di sicurezza "ad hoc" (es.: SSH per le connessioni remote; PGP, S/MIME, PEC per le email; SET/EMV per le transazioni con carta di credito; Kerberos, OAuth2 per l'autenticazione; Signal, MTProto2 per le chat, ecc.). In questo livello viene cifrato solamente il payload (header in chiaro).

Di seguito una tabella riassuntiva:

Level	Services	Protocols/applications
7 Application	End-to-end services	PGP, S/MIME, SAML, SSH, EMV, SET, ...
6 Presentation		XML security
5 Session	Authentication encryption	SSL/TLS, Secure UDP
4 Transport	traffic analysis...	
3 Network	Authentication, encryption	IPSec
2 Datalink	Authentication for link access, confidentiality	WEP, WPA, 802.1X
1 Physical		

Queste cifrature vengono implementate a strati, stile cipolla, in cui ogni livello aggiunge il proprio pezzettino di cifratura. Per cui se si vuole inviare ad esempio una mail, si ha che ogni strato cifra il pacchetto del livello superiore, quindi scendendo fino al livello fisico si hanno quattro cifrature differenti.

Crittografia

Adesso che sono stati illustrati i meccanismi di sicurezza, è possibile implementare i servizi di sicurezza, primo tra tutti la crittografia. È un meccanismo di sicurezza fondamentale, utilizzato in molti servizi, e esiste in due tipi principali: la crittografia simmetrica (“a chiave privata”) e la crittografia asimmetrica, o “a chiave pubblica”. Non si tratta di un’alternativa, ma di due strumenti diversi con applicazioni diverse.

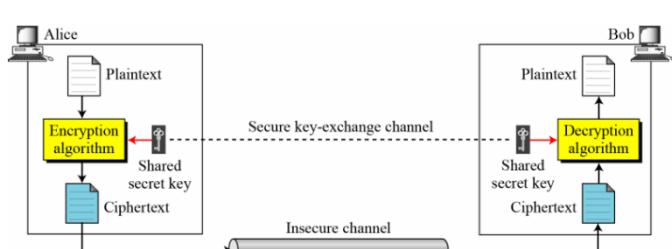
Sotto la terminologia che verrà utilizzata in seguito:

- testo in chiaro (plaintext): messaggio originale
- testo cifrato (ciphertext): messaggio codificato
- cifrario (cipher): algoritmo per trasformare il testo in chiaro in testo cifrato
- chiave (key): informazione utilizzata nel cifrario nota solo al mittente/ricevitore
- cifrare (encipher o encrypt): conversione del testo in chiaro in testo cifrato
- decifrare (decipher o decrypt): recuperare il testo cifrato dal testo in chiaro
- crittografia (cryptography): studio dei principi/metodi di crittografia
- criptoanalisi (cryptanalysis o codebreaking): studio dei principi/metodi di decifrazione di un testo cifrato senza conoscere la chiave
- crittologia (cryptology): campo che comprende sia la crittografia che la crittoanalisi

Crittografia simmetrica

Esistono diversi tipi, ad esempio quella convenzionale, quella a chiave privata, quella a chiave singola, ecc. in questa tecnica, il mittente e il destinatario condividono una chiave comune (condividono le stesse conoscenze sulla comunicazione), segreta all’intruso.

Tutti gli algoritmi di crittografia classici sono a chiave privata, in quanto era l’unico tipo prima dell’invenzione della chiave pubblica nel 1970, per cui è di gran lunga il più utilizzato. È da tenere presente che non è poi stata sostituita dalla crittografia asimmetrica (cioè a chiave pubblica).



Il modello è esattamente quello di Dolev-Yao visto precedentemente, in cui è presente un canale insicuro, su cui passa il testo, che da testo in chiaro viene trasformato in un testo cifrato da un algoritmo di cifratura, tramite chiave segreta. Qui nell’immagine, i riquadri gialli corrispondono le palline/cerchietti del modello di Dolev-Yao.

Requisiti:

Servono due requisiti per un uso sicuro della crittografia simmetrica, ovvero:

- un algoritmo di crittografia forte/robusto, cioè senza conoscere la chiave, deve essere impossibile ottenere il testo in chiaro dal testo cifrato. Questa chiave è segreta e nota solo al mittente/ricevitore
- matematicamente si devono avere due funzioni, tali che (a livello di bit):

$$\begin{aligned} Y &= E_K(X), && \text{cifratura} \\ X &= D_K(Y), && \text{decifrazione} \\ D_K(E_K(X)) &= X, && \text{testo originale} \\ D_K(E_H(X)) &= \text{errore}, && \text{se } K \neq H \end{aligned}$$

Il compito di un crittografo è quello di trovare le due funzioni D e E affinché l'algoritmo e tecnica di cifratura/decifratura possa funzionare e in maniera tale che sia difficile o impossibile scoprire X senza avere K (key)

Principio di Kerckhoff

Quanto appena visto corrisponde al principio di Kerckhoff. In questo caso si deve supporre che l'algoritmo di cifratura sia noto all'avversario. Infatti, il principio di Kerckhoff (*La cryptographie Militaire*, 1883) afferma che la sicurezza del messaggio deve dipendere solo dalla segretezza della chiave, non dalla segretezza dell'algoritmo di cifratura. Quindi la chiave è segreta/privata, ma l'algoritmo è pubblico.

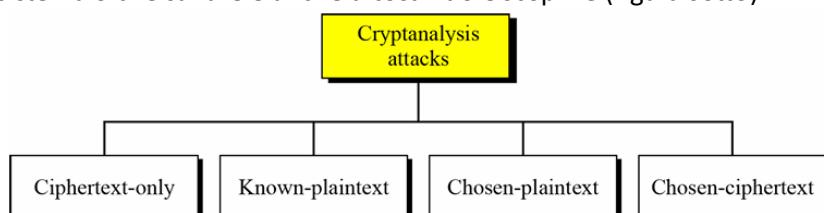
Questo principio deriva dalla massima di Shannon: “*Il nemico conosce il sistema*”. Per cui l'unica informazione che manca all'avversario è la chiave; tutto il resto è noto (il che implica un canale sicuro per distribuire la chiave, che può essere soggetta alla scoperta dell'avversario).

Tuttavia, nulla impedisce di tenere segreto anche il codice sorgente e il cifrario (anche se discutibile...), ma questo NON può essere un requisito per la sicurezza. Infatti, qualsiasi progetto di software di sicurezza che non presupponga che il nemico possieda il codice sorgente è già inaffidabile. Anche se può sembrare un controsenso, basarsi sulla “sicurezza” di qualcosa che dice di essere sicuro perché non è conosciuto agli attaccanti della rete (quindi è un software privato, o a sorgente chiusa), è in realtà una vulnerabilità poiché prima o poi qualcuno potrebbe scoprire il funzionamento dell'algoritmo (e anche perché con buona probabilità, circa 9/10, è stato progettato con una backdoor di sicurezza al suo interno e che può essere attivata a insaputa di chi usufruisce del sistema e cambiare tutto il sistema/algoritmo una volta che è stato scoperto è un “bagno di sangue”, mentre cambiare una key è molto più facile). Per cui è meglio non fidarsi mai degli algoritmi e dei protocolli di crittografia a sorgente chiusa.

Infatti, tutti i software di cifratura che si usano oggigiorno (es.: RC4, AES, ecc.) sono open-source e si conosce il loro funzionamento, non solo a livello di codice, ma anche di principio del progetto.

Crittoanalisi

Come già detto, la crittoanalisi è il processo di tentativo di scoprire il testo in chiaro corrispondente a un testo cifrato, o meglio la chiave. Esistono vari modi per tentare un attacco di crittoanalisi, distinti in base a ciò che l'attaccante sa del sistema o che sa fare e anche a cosa vuole scoprire (figura sotto).



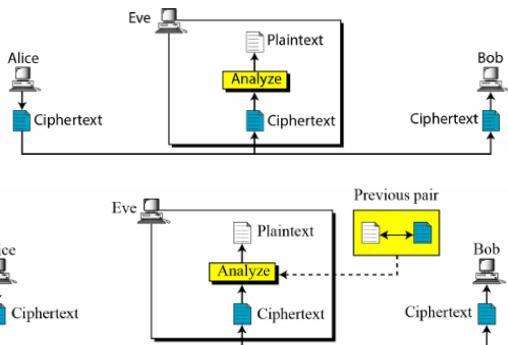
Tra i vari attacchi crittoanalitici si possono distinguere (ordinati in modo crescente per conoscenza delle informazioni da parte dell'attaccante):

- solo testo cifrato (ciphertext only): il crittoanalista conosce solo l'algoritmo e il testo cifrato. È il più semplice in quanto si tratta di ascoltare le comunicazioni tra i router e prendere i pacchetti come sono
- conosce il testo in chiaro (known plaintext): conosce/sospetta uno o più testi in chiaro e il corrispondente testo cifrato, per cui cerca di risalire alla chiave utilizzata. È molto frequente perché molto spesso si sa che cosa potrebbe essere stato scambiato durante la comunicazione (l'intestazione dei pacchetti è sempre in chiaro, come già detto) e si cerca di risalire alla chiave calcolando la cifratura del testo in chiaro e confrontandola col risultato. La parte di recupero della chiave, tuttavia, è molto difficile, in quanto un buon cifrario deve essere complesso in modo da non essere scoperto. Questa situazione è simile alla storia di Enigma, in cui si sapeva che i testi cifrati dei tedeschi finivano sempre con un saluto al leader, per cui bastava partire da quelle due parole e risalire alla chiave
- testo in chiaro scelto (chosen plaintext): selezionare il testo in chiaro e ottenerne il testo cifrato. Questo attacco un po' più forte dei precedenti, e si dice anche di "avere un *oracolo cifratore*", ovvero quando l'attaccante riesce a far cifrare quello che vuole lui, quindi sceglie un testo e lo fa cifrare al cifratore e osserva quello che ottiene. Esistono delle tecniche per manipolare il cifratore a cifrare quello che si vuole. Sempre riprendendo l'esempio di Enigma, gli inglesi sapendo a che ora e luogo sarebbe avvenuto un attacco, componevano un messaggio, lo crittavano e lo inviavano, aspettando una risposta dagli ufficiali tedeschi che pensavano fosse un messaggio dei loro soldati
- testo cifrato scelto (chosen ciphertext): selezionare il testo cifrato e ottenerne il testo in chiaro
- testo scelto (chosen text): selezionare il testo in chiaro o il testo cifrato da decifrare

I buoni algoritmi sono progettati per resistere almeno a un attacco di tipo *known-plaintext*.

Gli attacchi più comuni sono quelli di solo testo cifrato (*Ciphertext-Only*, nella prima immagine a destra), in cui il crittoanalista conosce solo l'algoritmo e il testo in chiaro, di conseguenza il suo obiettivo è scoprire il testo in chiaro e/o la chiave; e gli attacchi a testo noto (*known-plaintext*, seconda immagine a destra), in cui il crittoanalista conosce/sospetta uno o più testi in chiaro e il corrispondente testo cifrato, quindi il suo obiettivo è scoprire nuovi testi in chiaro e la chiave.

Nel primo caso il crittoanalista sa che le chiavi appartengono a un insieme finito per cui anche il numero di chiavi generabili è finito. La prima idea che salta in mente a questo punto è quella di generare tutte le chiavi e a ogni nuova comunicazione le si prova una alla volta fino a scoprire il testo (*attacco di forza bruta*). Tuttavia, è sempre possibile solamente in linea teorica, poiché richiede risorse computazionali molto elevate e il tempo a disposizione è poco (la comunicazione di solito ha tempi brevi, per cui non ha senso impiegare sei mesi a rompere il cifrario se serviva avere la risposta subito).



Per cui la sicurezza che si deve ottenere dalla crittografia è:

- sicurezza incondizionata: indipendentemente dalla potenza del computer o dal tempo a disposizione, il cifrario non può essere violato poiché il testo cifrato non fornisce informazioni sufficienti per determinare in modo univoco il testo in chiaro corrispondente
- sicurezza computazionale: il costo della violazione del cifrario supera il valore delle informazioni criptate; oppure con risorse di calcolo limitate (es.: il tempo necessario per i calcoli è troppo grande), il cifrario non può essere decifrato

Tuttavia, nonostante queste premesse, la ricerca a forza bruta è sempre possibile: basta provare tutte le chiavi possibili, finché non si trova quella corretta, il che richiede di riconoscere quando la chiave è sbagliata o corretta, cioè:

$$D_{K'}(E_{K(M)}) = \text{errore}, \quad \text{se e solo se } K' \neq K$$

in cui si presuppone che il testo in chiaro sia noto o riconoscibile. Questo infatti, è l'attacco più elementare e è proporzionale alla dimensione della chiave, quindi diventa rapidamente irrealizzabile. Di seguito una tabella che riporta le stime sui tempi in base alla complessità (lunghezza) della chiave (presupponendo di avere un calcolatore con prestazioni ragionevoli, circa un tentativo ogni microsecondo, e uno con prestazioni più elevate, es.: i computer della CIA), in cui il tempo stimato viene conteggiato tenendo conto della metà delle combinazioni, ovvero 2^{n-1} con n numero di bit della key, essendo una media (la combinazione corretta potrebbe trovarsi all'inizio del cifrario come anche alla fine):

Key Size (bits)	Number of Alternative Keys	Time required at 1 decryption/μs	Time required at 10 ⁶ decryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31} \mu\text{s} = 35.8 \text{ minutes}$	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55} \mu\text{s} = 1142 \text{ years}$	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127} \mu\text{s} = 5.4 \times 10^{24} \text{ years}$	$5.4 \times 10^{18} \text{ years}$
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167} \mu\text{s} = 5.9 \times 10^{36} \text{ years}$	$5.9 \times 10^{30} \text{ years}$
26 diff. chars (perms)	$26! = 4 \times 10^{26}$	$2 \times 10^{26} \mu\text{s} = 6.4 \times 10^{12} \text{ years}$	$6.4 \times 10^6 \text{ years}$

Pertanto, la forza bruta è sempre possibile in teoria, ma non in pratica: il suo costo computazionale è troppo elevato. Un'altra strategia è l'attacco crittoanalitico, ovvero la ricerca di debolezze matematiche nel cifrario, eventualmente per chiavi e testi specifici. Come è facile immaginare, questo si basa molto di più sulle competenze del crittoanalista rispetto alla tecnica di forza bruta.

Questa seconda strategia, può essere utile per ridurre lo spazio delle chiavi in cui applicare gli attacchi *brute force* (= forza bruta), infatti, alcuni cifrari noti e utilizzati hanno subito attacchi crittoanalitici. Tuttavia, anche ridurre lo spazio di ricerca può non essere pratico: ad esempio, ridurre lo spazio delle chiavi da 2^{128} a 2^{126} è considerato un attacco crittoanalitico, ma in pratica non indebolisce il cifrario.

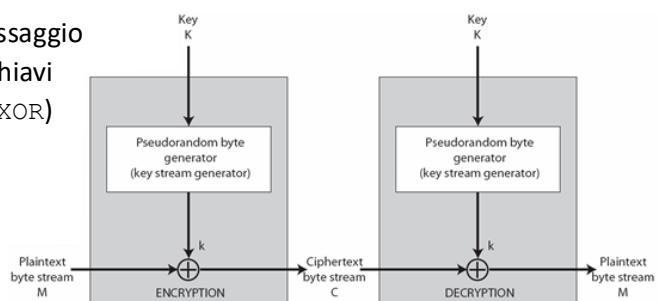
Cifrari di flusso

Ci sono due diversi cifrari utilizzabili per la crittografia simmetrica, ovvero quelli a flusso e quelli a blocchi.

Nel cifrario a flusso (*stream cipher*) si elabora il messaggio byte per byte (come un flusso). Si ha un flusso di chiavi (*keystream*) pseudo-casuali, che vanno combinate (*XOR*) con il testo in chiaro bit per bit:

$$C_i = M_i \oplus K_i$$

La stessa operazione va eseguita per decifrare il messaggio.



Questo algoritmo di cifratura è molto comodo in quanto E e D visti prima sono dentro alla stessa funzione (quindi non serve generare due funzioni distinte). Inoltre, è molto efficace perché la casualità del flusso di chiavi distrugge completamente le proprietà statistiche del messaggio.

I due keystream per le chiavi devono essere generati in contemporanea e allo stesso modo, infatti i due generatori di chiavi necessitano dello stesso *seed* (chiave). Però, cosa molto importante, una volta aver generato un flusso di chiavi, non lo si deve MAI riutilizzare (némeno per due porzioni dello stesso messaggio se non di comunicazioni differenti), altrimenti:

$$C \oplus C' = (M \oplus K) \oplus (M' \oplus K) = M \oplus M'$$

ovvero mescolando i due keystream si riottiene il messaggio in chiaro, poiché la casualità dei due flussi si annulla, per cui la cifratura risulta vana.

Tuttavia i generatori hanno memoria limitata, per cui accade deterministicamente che prima o poi le combinazioni (stati) di bit si esauriscono e serve ripetere lo stesso valore (e in realtà deve essere così per funzionare). Quindi a questo punto il problema è poter costruire un meccanismo che, data una chiave (seed) robusta e grande, riesca a generare un flusso di byte così ampio da non dover riscontrare la duplicazione dello stesso valore, se non in un tempo così lungo da finire su un'altra comunicazione.

Per cui, alcune considerazioni di progettazione sono:

- si deve avere un periodo lungo senza ripetizioni
- il flusso di chiavi deve essere statisticamente casuale
- si dipende da una chiave (seed) sufficientemente grande
- si genera una grande complessità lineare

RC4

Quindi, se progettato correttamente, può essere sicuro quanto un cifrario a blocchi con chiave della stessa dimensione, ma di solito è più semplice e veloce.

Un'implementazione molto comune è RC4, ampiamente utilizzato (es.: web SSL/TLS, wireless WEP, WPA, ecc.) e che sfrutta un periodo di 2^{100} combinazioni (stati), che dovrebbe essere sufficiente alla totale trasmissione dei dati di una determinata comunicazione.

Questo è un cifrario proprietario della RSA DSI, ovvero un altro progetto di Ron Rivest, semplice ma efficace. È un cifrario a flusso orientato ai byte, con dimensione variabile della chiave (fino a 2048 bit).

Il periodo è molto probabile che sia superiore a 10^{100} , ma lo stesso è molto veloce (8-16 operazioni macchina per byte), il che lo rende ampiamente utilizzato (come già detto).

La chiave forma una permutazione casuale di tutti i valori a 8 bit e utilizza tale permutazione per “strapazzare” (*scramble*), ovvero rimescolare, le informazioni in ingresso elaborate un byte alla volta.

L'algoritmo inizia con un array S di numeri da 0 a 255 e usa la chiave per rimescolare, fino a quando S forma lo stato interno del cifrario.

```
for i = 0 to 255 do
    S[i] = i
    T[i] = K[i mod keylen]
j = 0
for i = 0 to 255 do
    j = (j + S[i] + T[i]) (mod 256)
    swap (S[i], S[j])
```

Quindi, la crittografia continua a mescolare i valori dell'array. La somma delle coppie rimescolate seleziona il valore “chiave del flusso” dalla permutazione. Si effettua uno XOR su $S[t]$ con il byte successivo del messaggio da cifrare/decifrare.

```
i = j = 0
for each message byte Mi
    i = (i + 1) (mod 256)
    j = (j + S[i]) (mod 256)
    swap(S[i], S[j])
    t = (S[i] + S[j]) (mod 256)
Ci = Mi XOR S[t]
```

In realtà ci sono tanti altri algoritmi e metodi per calcolare le chiavi. Ad esempio anche i polinomi generatori visti per la creazione del CRC, sono in grado di generare un flusso di byte pseudo-casuale. Infatti, utilizzava un sistema a 32 bit a scorrimento, che in questo caso può essere riciclato per questo scopo inserendo il seed nel sistema e lasciandolo girare per un periodo di tempo.

Però a questo punto sorge un'altra idea: se al posto di un flusso pseudo-casuale si avesse un keystream realmente casuale, quindi non generato da un seed, ma un flusso di byte estratti tutti casualmente, si avrebbe quindi un cifrario “perfetto”. L'*One-Time Pad* è esattamente questo.

|| One-Time Pad

Si utilizza un flusso di chiavi veramente casuale per tutta la durata del messaggio, col risultato che il cifrario sarà incondizionatamente sicuro. Questo metodo si chiama *One-Time Pad* (o *cifrario di Vernam*).

In questo caso, la chiave è il flusso di chiavi e è infrangibile poiché il testo cifrato non ha alcuna relazione statistica con il testo in chiaro (dimostrato da Shannon), poiché per ogni testo in chiaro (*any plaintext*) e per ogni testo cifrato (*any ciphertext*) esiste una chiave che mappa l'uno con l'altro, quindi non c'è modo di trovare la vera chiave: molte chiavi hanno senso.

Detto in altre parole si ha un flusso completamente casuale che viene “iniettato” nel testo in chiaro generando un rumore, statistico, che distrugge tutte le informazioni in esso contenute. Questo funziona, poiché eseguendo lo `XOR` su dei byte random, si ottiene a sua volta un risultato random.

Esempio: il testo cifrato è 'A' = 01000001. Se la chiave è 00000011, il testo in chiaro è 01000010 = 'B'; se la chiave è 00000010, il testo in chiaro è 01000011 = 'C'; e così via per tutte le possibili chiavi. Tutte sono ugualmente possibili, poiché il testo originale non è conosciuto, quindi non si può dire quale sia testo in chiaro che è stato realmente utilizzato.

Anche in questo caso, si può usare la chiave solo una volta, altrimenti la chiave non è più casuale, e si hanno gli stessi problemi nella generazione e distribuzione sicura della chiave. Infatti, la chiave deve essere lunga quanto il messaggio originale e deve essere nota sia al mittente che al destinatario e non è chiaro come trasmetterla in maniera sicura (deve essere pre-condivisa prima della comunicazione effettiva). Bisognerebbe che venga trasmessa in un canale sicuro, in un tempo diverso da quello della comunicazione effettiva, infatti l'*One-Time Pad* prende questo nome in quanto si usa una volta sola lo scambio di chiavi e poi si resetta la comunicazione per il flusso effettivo di byte.

|| Cifrari a blocchi

I cifrari a blocchi elaborano i messaggi in blocchi, ognuno dei quali viene poi crittato/decrittato. Funziona come una sostituzione (funzione biiettiva) su caratteri molto grandi (es.: 64 bit o più). I cifrari moderni funzionano con 128 bit, in quanto il vantaggio sta nel fatto che più grande è il blocco, meno efficaci sono gli attacchi statistici. Essi sono ben analizzati, per più ampia gamma di applicazioni.

I cifrari a blocchi sembrano una sostituzione estremamente grande, in quanto si avrebbe bisogno di una tabella di 2^{64} voci per un blocco a 64 bit: l'utilizzo della memoria risulterebbe di $2^{64} \times 2^6 = 2^{70}$ byte, il che non è fattibile. Invece, è più facile creare la sostituzione da blocchi più piccoli.

Molti cifrari a blocchi simmetrici classici sono basati su una *struttura di cifratura di Feistel* (DES). Mentre, i cifrari più moderni sono diversi (es.: AES).

AES

Il *Advanced Encryption Standard* (AES) è stato pubblicato dal NIST degli Stati Uniti in un bando per i cifrari nel 1997, che però è stato selezionato come standard nel 2001. Nacque per la semplice ragione che il DES era un algoritmo vecchio e ormai obsoleto (usava 64 bit, con 56 bit di chiave), per cui si è deciso di creare un bando per un nuovo algoritmo che lo rimpiazzasse, i cui requisiti necessari fossero:

- un cifrario a blocchi simmetrico a chiave privata
- dati a 128 bit, con chiavi a 128/192/256 bit
- più forte e più veloce del Triple-DES
- vita attiva di 20-30 anni (+ uso in archivio)
- fornire specifiche complete e dettagli di progettazione
- implementazioni in C e Java

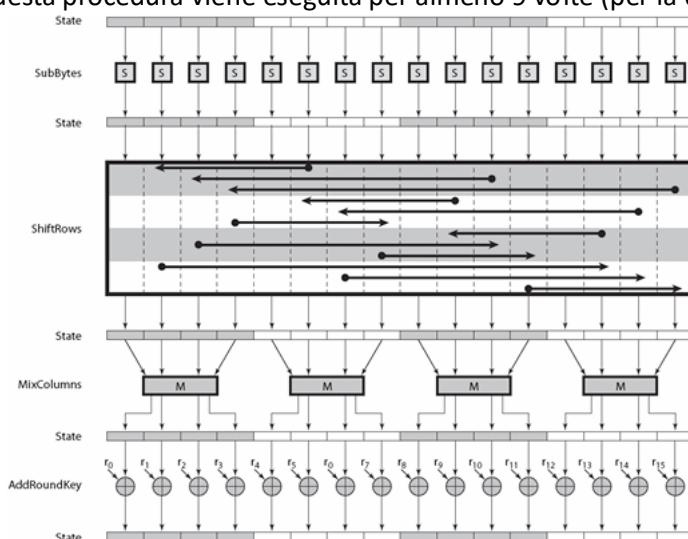
A questo punto vennero accettati 15 candidati (giugno del '98) e solo 5 vennero selezionati nell'agosto del 1999. Alla fine è stato scelto Rijndael (dal nome degli inventori Rijmen e Daemen) come AES nell'ottobre 2000, ma venne rilasciato come standard FIPS PUB 197 nel novembre 2001. È ormai utilizzato in molti contesti (es.: stream dati dei blueray o del digitale terrestre o dal satellitare) e ha sostituito il 3DES. Tuttavia, anche gli altri algoritmi Serpent e Twofish di quel bando sono stati implementati e qualche volta si trovano nelle librerie (invece, gli altri del bando sono MARS, che era un algoritmo dell'IBM, e RC6, che sta per Ron's Ciphers, dal nome dell'ideatore).

	Rijndael	Serpent	Twofish	MARS	RC6
General Security	2	3	3	3	2
Implementation Difficulty	3	3	2	1	1
Software Performance	3	1	1	2	2
Smart Card Performance	3	3	2	1	1
Hardware Performance	3	3	2	1	2
Design Features	2	1	3	2	1
Total	5	16	14	13	10

Per quanto riguarda la sua implementazione e realizzazione, si necessita di 3 o 4 porte logiche, che in tecnologie attuali corrisponde a un quadratino di silicio delle dimensioni di 0.1 x 0.1 mm, quindi si può anche integrare in un qualsiasi sistema, anche un RFID.

Utilizza chiavi a 128/192/256 bit, con dati a 128 bit. È un cifrario iterativo piuttosto che Feistel (DES) e elabora i dati come blocchi di 4 colonne di 4 byte, operando sull'intero blocco di dati in ogni round. È progettato per essere resistente agli attacchi noti, veloce e compatto nel codice su molte CPU e è semplice da progettare.

Di seguito un'immagine di un round di AES. Esso itera per 9, 11 o 13 volte i 16 byte di stato iniziali. Per prima cosa una parte di sostituzione, ovvero una tabella fissa che sostituisce ogni byte, e è nota a tutti in quanto specifica dell'algoritmo. A seguire c'è una sequenza di permutazioni (*shift*), succedute da un'operazione di mescola (*mix*) delle colonne con somme e prodotti invertibili e poi infine viene aggiunto bit a bit in `XOR` la chiave di round. Tutta questa procedura viene eseguita per almeno 9 volte (per la decifratura si fa l'inverso).



Per quanto riguarda la sicurezza di AES, finora, gli unici attacchi riusciti sono quelli a canale laterale. Infatti, per la sua robustezza, è consentito dal governo degli Stati Uniti per documenti SECRET e TOP SECRET e è stato studiato a fondo da molti crittografi. La maggior parte degli attacchi recenti non è ancora fattibile (es.: l'attacco a chiave correlata di Biryukov & Khovratovich ha una complessità di $2^{99.5}$, infatti sono state usate chiavi molto simili a quelle generate).

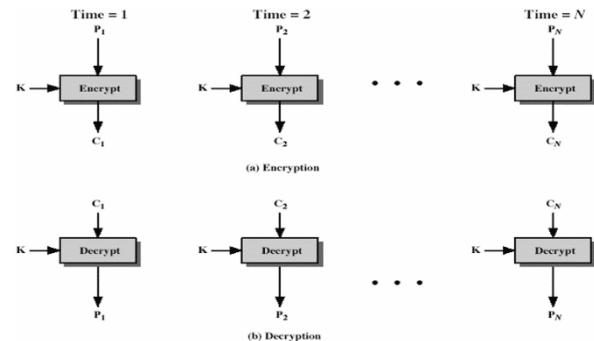
Col comando `man ssh_config` (file di configurazione di SSH) della console è possibile vedere i cifrari supportati da utilizzare per crittare i dati. Mentre con `man openssl`, nella sezione `/cipher`, è possibile vedere qualche altra informazione relativa ai cifrari utilizzabili, invece con `openssl list-cipher-algorithms` viene fuori una lista più completa di tutti gli algoritmi di cifratura supportati da openssl.

Modalità di funzionamento dei cifrari a blocchi

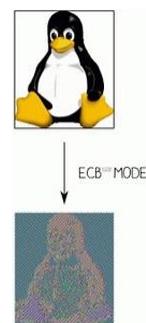
Si supponga di avere a disposizione un cifrario a blocchi con delle chiavi (es.: AES a 128 bit, Blowfish, RC4, ecc.), essi processano i dati a blocchi di dimensione fissa (es.: DES critta blocchi di 64 bit con chiave di 56 bit, AES critta blocchi di 128 bit con chiavi di 128 / 192 / 256 bit, ecc.), tuttavia ci sono casi in cui i blocchi sono più piccoli del messaggio. Infatti, nella pratica è necessario un modo per criptare/decriptare quantità arbitrarie di dati. Il *ANSI X3.106-1983 Modes of Use* (ora FIPS 81) definisce 4 possibili modalità, che poi successivamente sono diventate 5 per definire AES e DES, ovvero ECB, CBC, CFB, OFB, CTR (in realtà ne sono state proposte molte altre).

CodeBook elettronico (ECB):

Nel caso in cui i blocchi fossero più corti del messaggio da crittare, la prima idea che viene in mente è quella di replicare il cifrario, ovvero suddividere il messaggio in blocchi grandi quanto il cifrario e crittarli in maniere indipendente tra loro. Ogni blocco è un valore che viene sostituito, codificato indipendentemente dagli altri blocchi: $C_i = DES_{K_1}(P_i)$. Questo sistema è applicabile a ogni cifrario.



Quest'idea sembra tanto facile e buona da implementare, ma in realtà... è brutta... Infatti, tende a non funzionare, soprattutto perché a blocco uguale corrisponde il valore uguale, e nonostante non si riesca comunque a risalire al plaintext, ma la struttura si riesce a capire.



Quindi, le ripetizioni del messaggio possono comparire nel testo cifrato, se allineato al blocco di messaggi, in particolare con dati come i grafici, o con messaggi che cambiano molto poco, il che diventa un problema di analisi dei CodeBook. La debolezza è dovuta all'indipendenza dei blocchi di messaggi crittati, quindi l'uso principale di questo algoritmo è l'invio di pochi blocchi di dati (di solito di un singolo valore).

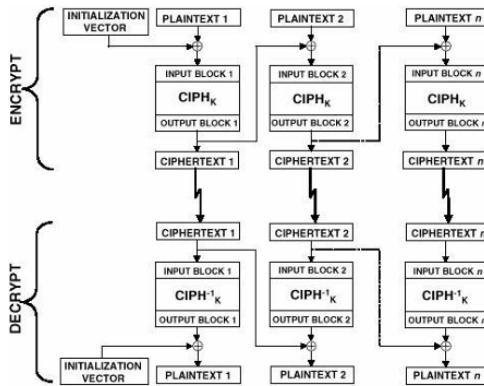
Cifratura a blocchi (CBC):

In questo caso, il messaggio viene suddiviso in blocchi che vengono poi collegati tra loro nell'operazione di crittografia. Ogni blocco crittato precedente viene concatenato con il blocco di testo in chiaro corrente, da cui il nome *Cipher Block Chaining* (CBC).

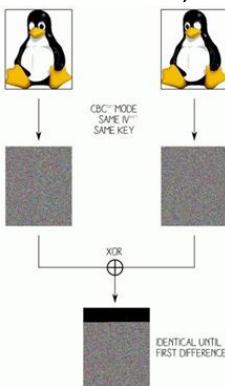
Si utilizza il vettore iniziale (*Initial Vector*, IV), che NON è una chiave, per avviare il processo:

$$C_i = E_K(P_i \text{ XOR } C_{i-1})$$

$$C_0 = IV$$



Quindi con questa tecnica, ogni blocco viene creato per “disturbare” il precedente, dunque per romperlo nella cifratura. Infatti, viene utilizzato per la crittografia di grandi quantità di dati (es.: e-mail, file, web, ecc.) o per l'autenticazione.



Di conseguenza, un blocco di testo in chiaro dipende da tutti i blocchi che lo precedono, per cui qualsiasi modifica a un blocco di testo in chiaro si ripercuote su tutti i blocchi di testo cifrato successivi.

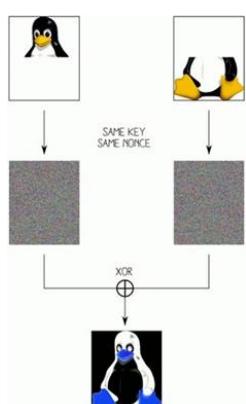
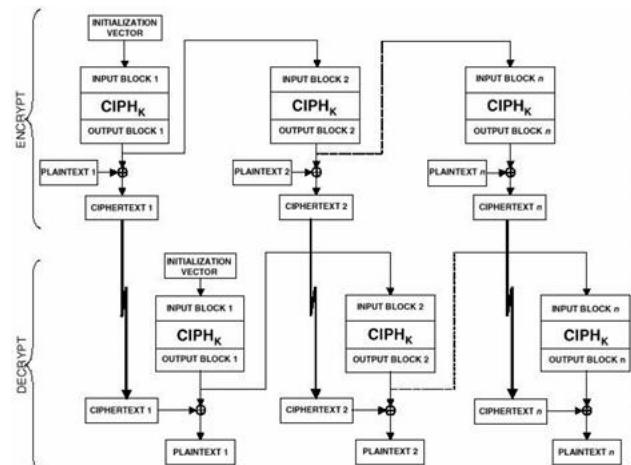
Inoltre, si necessita di un vettore di inizializzazione (*/V*) che deve essere noto al mittente e al destinatario. Il problema sta nel fatto che se inviato in chiaro, l'attaccante può cambiare i bit del primo blocco e modificare l'*/V* per compensarlo. Pertanto, l'*/V* deve essere un valore fisso (come in EFTPOS) oppure deve essere inviato criptato in modalità ECB prima del resto del messaggio.

FeedBack in uscita (OFB):

Qui il messaggio viene trattato come un flusso di bit e non più come un blocco. L'uscita del cifrario viene aggiunta al messaggio e l'output viene quindi restituito (da qui il nome *Output FeedBack*, OFB). Il feedback è indipendente dal messaggio e può essere calcolato in anticipo:

$$\begin{aligned} C_i &= P_i \text{ } XOR \text{ } O_i \\ O_i &= E_K(O_{i-1}) \\ O_0 &= IV \end{aligned}$$

Questo algoritmo si usa per la crittografia di flussi su canali rumorosi. Infatti, è comodo perché gli errori di bit non si propagano. Tuttavia è più vulnerabile alla modifica del flusso di messaggi.

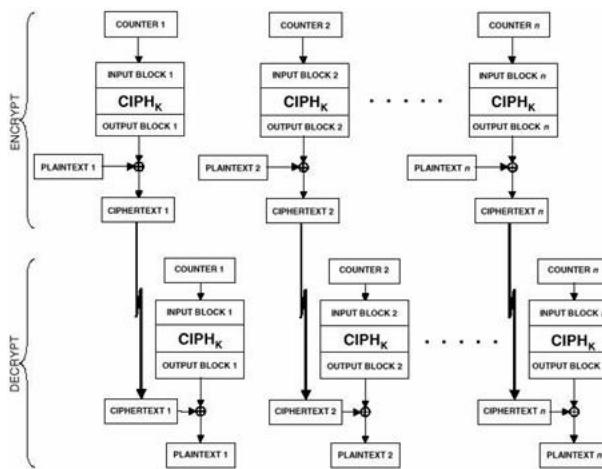


Si tratta di una variante del cifrario di Vernam, quindi non deve mai riutilizzare la stessa sequenza “pseudo-random” (*chiave+IV*). Dunque fissato il valore di *k* e di */V*, l'algoritmo darà sempre la stessa sequenza di byte (è deterministico), per cui crittografare sequenze uguali comporta a risultati uguali (problema analogo a ECB e come è possibile vedere in immagine accanto). Inoltre, il mittente e il destinatario devono rimanere sincronizzati.

Originariamente era specificato con feedback a *m*-bit, ma ricerche successive hanno dimostrato che si dovrebbe usare solo un feedback a blocco completo (cioè CFB-64 o CFB-128).

Contatore (CTR):

È una modalità “nuova”, sebbene sia stata proposta fin dall'inizio. È una via di mezzo tra ECB e OFB, ma critta il valore del contatore piuttosto che qualsiasi valore di feedback. Quindi, deve avere una chiave e un valore del contatore diversi per ogni blocco di testo in chiaro (mai riutilizzato):



$$C_i = P_i \text{ XOR } O_i$$

$$O_i = E_K(i)$$

Si usa per la crittografia di reti ad alta velocità (es.: ATM, IPsec, ecc.). Infatti, questo algoritmo è efficiente, può eseguire crittografie in parallelo in HW o SW e può essere preprocessato in anticipo rispetto alle necessità. Quindi, è ottimo per collegamenti ad alta velocità e a raffica, con accesso casuale ai blocchi di dati crittografati (es.: file system crittografati) e con sicurezza dimostrabile (buona come altre modalità).

Tuttavia bisogna assicurarsi che non vengano mai riutilizzati i valori delle chiavi e dei contatori, altrimenti potrebbero rompersi (cfr. OFB).

Integrità e autenticazione dei messaggi

Ci sono altri tipi di protezione dei dati, oltre a quelli di crittografia. Infatti, l'integrità e l'autenticazione dei messaggi si occupano di proteggere l'integrità di un messaggio (ovvero la protezione del messaggio dalla modifica dei dati) e di convalidare l'identità del mittente. Per la considerazione dei requisiti di sicurezza, si hanno tre funzioni alternative: la crittografia del messaggio, il codice di autenticazione del messaggio (MAC) e le funzioni hash.

Per quanto riguarda i requisiti di sicurezza, esistono diversi tipi di attacchi alla sicurezza:

- divulgazione
 - analisi del traffico
 - mascheramento
 - modifica del contenuto
 - modifica della sequenza
 - modifica della tempistica
 - ripudio della fonte
 - ripudio della destinazione
- attacchi (passivi) alla riservatezza
- attacchi (attivi) all'autenticità dei dati, del mittente (es.: firme digitali) e del destinatario

Invece, per quanto riguarda i meccanismi di autenticazione, in generale, si utilizza un modello basato su quello di Dalle-Yao, in cui c'è un'applicazione di trasformazione applicata al



sender, ovvero l'autenticatore, che serve a garantire l'autenticità dei dati, e una trasformazione all'indietro (riportare in plaintext), chiamata verificatore. Ogni meccanismo è composto da un autenticatore e da un verificatore, eseguiti separatamente.

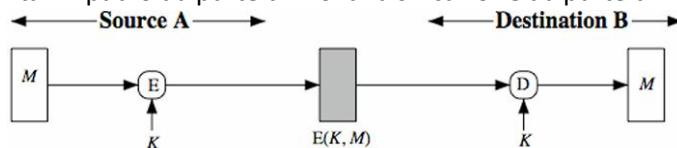
Quindi, il testo che viaggia sulla rete non è solo messaggio puro e cifrato, ma viene aggiunta qualche altra informazione (da parte dell'autenticatore) in modo da garantire l'attendibilità dei dati trasmessi. I dati prodotti dagli autenticatori possono essere verificati in modo indipendente (nei protocolli).

I tipi di autenticatori esistenti sono: crittografia dei messaggi, codici di autenticazione dei messaggi e funzioni di hash.

Autenticazione dei messaggi tramite crittografia simmetrica

Se si utilizza la crittografia simmetrica allora:

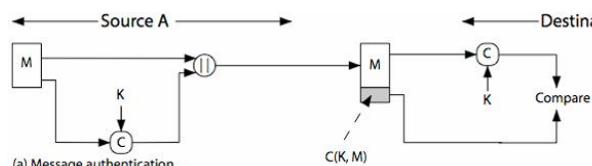
- il destinatario sa che il mittente deve averla creata poiché solo il mittente e il destinatario conoscono la chiave utilizzata
- il destinatario sa che il contenuto non può essere stato alterato, poiché se il messaggio ha una struttura adeguata, si possono rilevare eventuali modifiche grazie alla ridondanza o a un checksum
- tuttavia, non si evita il ripudio da parte di A o la falsificazione da parte di B



(a) Symmetric encryption: confidentiality and authentication

Codice di autenticazione del messaggio (MAC)

Il MAC (*Message Authentication Code*) è generato da un algoritmo che crea un piccolo blocco di dimensioni fisse e che dipende dal messaggio e da una chiave. È come la crittografia, però non è necessario che sia



reversibile. Il MAC viene aggiunto dal mittente al messaggio come firma e il destinatario esegue lo stesso calcolo sul messaggio per verificare la corrispondenza con il MAC. Questo meccanismo fornisce al destinatario la certezza che il messaggio è inalterato e proviene dal mittente, in quanto sia il mittente che il destinatario condividono la chiave e possono crearla. Si noti che il MAC non è una firma digitale.

Questo meccanismo inoltre, è utile perché non serve che venga cifrato assieme al codice, per cui non serve che altri dati vengano fatti passare nell'algoritmo di cifratura, rendendo più facile la scoperta della chiave. Infatti, una vulnerabilità già vista è quelle che più dati vengono spediti, più si aumenta la probabilità che l'attaccante trovi il codice della chiave, poiché ha più dati su cui fare confronti e corrispondenze. Invece, qui, non serve crittografare anche il MAC, lasciando quindi questa informazione in chiaro e crittografando solo i dati strettamente necessari alla comunicazione.

Il MAC è quindi un checksum crittografico dato dalla formula $MAC = C_K(M)$, che condensa un messaggio di lunghezza variabile M utilizzando una chiave segreta K a un autenticatore di dimensioni fisse. È una funzione multi-a-uno, in quanto potenzialmente molti messaggi hanno lo stesso MAC (in quanto la dimensione è limitata/fissa, al contrario dei messaggi), ma trovarli deve essere molto difficile.

Funzioni di hash

Le funzioni di hash riducono un messaggio arbitrario a una dimensione fissa $h = H(M)$ e di solito si assume che la funzione di hash sia pubblica e non codificata (cfr. MAC che è codificato). L'hash viene utilizzato per rilevare le modifiche al messaggio e può essere utilizzato in vari modi con il messaggio, ma più spesso lo si usa per creare MAC e firma digitale.

In questo caso le funzioni di hash che si usano sono funzioni non invertibili, tuttavia sono molto robuste in termini di sicurezza, il che le rende "un muro" per un attaccante. Infatti, una funzione di hash buona ha come caratteristiche:

1. può essere applicato a qualsiasi messaggio di dimensioni M
2. produce un output di lunghezza fissa h
3. è facile calcolare $h = H(M)$ per qualsiasi messaggio M (l'inversione invece non è semplice)

4. deve essere resistente in tre modi:

- Resistenza alla pre-immagine: dato h è impossibile trovare x data $H(x) = h$
- Seconda resistenza alla pre-immagine: dato x è impossibile trovare y se $H(y) = H(x)$
- Resistenza alla collisione (clash): è impossibile trovare qualsiasi x, y se $H(y) = H(x)$

Algoritmo di Sicure Hash (SHA)

Lo SHA (*Secure Hash Algorithm*) era originariamente progettato dal NIST e dalla NSA nel 1993, ma poi è stato rivisto nel 1995 come SHA-1. È uno standard statunitense per l'uso con lo schema di firma DSA (lo standard è FIPS 180-1 1995, anche Internet RFC3174, per cui se l'algoritmo è SHA, lo standard è SHS) e è basato sul design di MD4 con differenze di chiave. Esso produce valori hash a 160 bit. Nel 2005, alcuni risultati sulla sicurezza di SHA-1 hanno sollevato preoccupazioni sul suo utilizzo in applicazioni future, infatti il clash può essere riscontrato in 2^{69} operazioni anziché 2^{80} .

Successivamente è stata effettuata una revisione dello standard Secure Hash. Il NIST ha pubblicato la revisione FIPS 180-2 nel 2002, che aggiunge altre 3 versioni di SHA: SHA-256, SHA-384 e SHA-512. Questa revisione è stata progettata per la compatibilità con la maggiore sicurezza fornita dal cifrario AES. In realtà la struttura e i dettagli sono simili a quelli di SHA-1, quindi l'analisi dovrebbe essere simile, ma i livelli di sicurezza sono più elevati.

Da terminale è possibile crittografare un messaggio, anche su più righe e di lunghezza variabile, digitando il comando `shasum`, scrivendo il testo e poi al termine di esso premere `ctrl+D`. La riga successiva che viene mostrata a riga di comando è il codice hash a 160 caratteri che corrisponde al testo digitato. Di default viene eseguito lo SHA-1, ma è possibile specificare un preciso codice di crittazione digitando ad esempio il comando `shasum -a 512` per crittare con lo SHA-512 (per gli altri vedere le impostazioni con `shasum -h`). Altrimenti per utilizzare altri algoritmi di cifratura, si specifica come comando il medesimo algoritmo, ad esempio per il md5 si usa il comando `md5`, ottenendo un testo cifrato a 128 caratteri.

Uso degli Hash Digest nell'autenticazione dei messaggi

Supponendo a questo punto di avere un'ottima funzione di hash, la si può utilizzare per costruire un MAC, quindi anziché utilizzare un cifratore si usa una funzione di hash congiunta con una chiave. Un digest di messaggio non autentica il mittente del messaggio, dunque per fornire l'autenticazione del messaggio, un utente A deve fornire la prova che è stato lui a inviare il messaggio e non un impostore. Il digest creato dalla funzione hash è chiamato codice di rilevamento delle modifiche (*modification detection code*, MDC). Per l'autenticazione del messaggio è necessario un codice di autenticazione del messaggio (MAC).

Le funzioni di hash per la sicurezza dei MAC garantiscono sicurezza da:

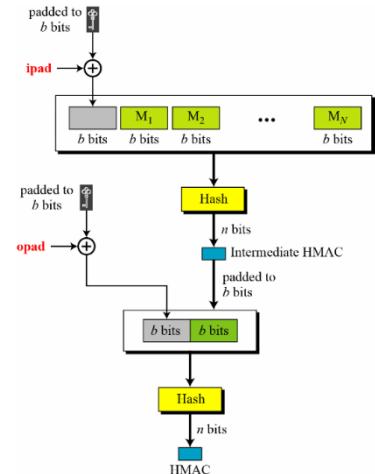
- attacchi di forza bruta che sfruttano degli hash con forte resistenza alle collisioni, che hanno un costo di $2^{m/2}$, proposta di cracking HW MD5 (64 bit) e un hash a 160 bit (perché l'hash a 128 bit sembrava vulnerabile); inoltre il MAC con coppie messaggio-MAC note può attaccare sia lo spazio delle chiavi (cfr. ricerca delle chiavi) sia il MAC (è necessario un MAC di almeno 128 bit)
- attacchi crittoanalitici che sfruttano la struttura; come i cifrari a blocchi, vogliono che gli attacchi di forza bruta siano l'alternativa migliore

Per poter utilizzare una funzione di hash per il MAC, c'è uno standard chiamato HMAC, ovvero un MAC basato su funzioni di hash con una sorta di hash con chiave. Esso è specificato come standard Internet RFC2104 e utilizza una funzione di hash sul messaggio:

$$HMAC_K = \text{Hash}[(K^+ \text{ XOR } opad) \parallel \text{Hash}[(K^+ \text{ XOR } ipad) \parallel M]]$$

dove K^+ è la chiave imbottita (*padded*) e $opad = 0x5C = '\backslash'$, $ipad = 0x36 = '6'$ sono le costanti di imbottitura (*padding*) specificate.

L'overhead è di solo tre calcoli di hash in più rispetto a quelli necessari per il solo messaggio, inoltre è possibile utilizzare qualsiasi funzione hash (es.: MD5, SHA-1, RIPEMD-160, Whirlpool, ecc.).

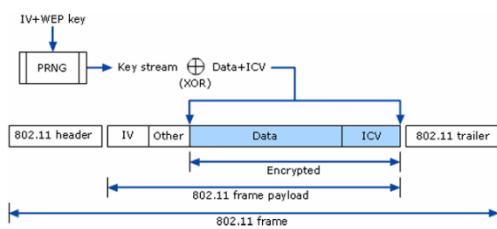


La sicurezza dimostrata di HMAC è legata a quella dell'algoritmo di hash sottostante (se è buona quella è buono anche il HMAC). Per cui un attacco a HMAC richiede un attacco a forza bruta sulla chiave utilizzata o un "attacco di compleanno" (*birthday attack*), dal momento che la chiave deve osservare $2^{n/2}$ messaggi generati con la stessa chiave alcuni di essi campitano nella stessa ripetizione ("compleanno"). Per cui bisogna scegliere la funzione hash in base alla velocità rispetto ai vincoli di sicurezza (es.: MD5 è sufficiente e più veloce di SHA-1).

Wired Equivalent Privacy (WEP)

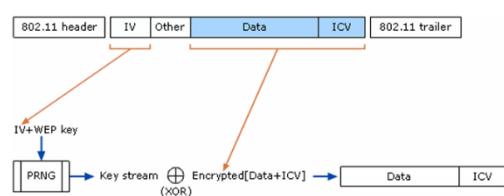
Di seguito un esempio di applicazione dell'autenticazione di messaggi a chiave simmetrica: *Wired Equivalent Privacy* (1999). Lo standard 802.11 definisce WEP, come uno schema di crittografia a chiave simmetrica tra host e access point. Esso intende offrire la stessa riservatezza delle connessioni cablate e limitare l'accesso all'infrastruttura solo agli host autorizzati, in cui solo i pacchetti autenticati possono accedere alla rete. L'idea era quindi quella di rendere il WI-FI come le reti Ethernet, dunque la comunicazione può avvenire solo se si è connessi direttamente al mezzo e nel caso del WI-FI questa implementazione corrisponde a conoscere la password di accesso, che in questo secondo caso svolge anche il ruolo di controllo di accesso (quindi solo i pacchetti degli host autenticati, e riconoscibili all'interno della rete, possono transitare, mentre gli altri messaggi vengono rigettati).

La chiave è condivisa tra host e AP e può essere (di solito lo è) la stessa per tutti gli host. Non è specificato come questa chiave venga scambiata tra host e AP (di solito, tramite canali esterni), né per quanto tempo è valida (spesso rimane invariata per anni).



Per quanto riguarda la parte di crittazione, il *payload + checksum* (ICV) di 802.11 viene crittografato utilizzando (una variante di) RC4, con un IV a 24 bit (3 byte). L'IV viene inviato insieme al testo cifrato. Nel caso in cui non venga impostata la cifratura (bit WEP a 0), il frame sarebbe composto da intestazione, payload e trailer (CRC) normali dell'802.11, invece se il bit WEP è a 1, il frame va letto in maniera leggermente differente, ovvero come mostrato nell'immagine accanto a sinistra, in cui tra intestazione e data si hanno il IV e altre informazioni supplementari e tra data e trailer c'è il ICV (per validare).

Invece, per quanto riguarda la decodifica, la stazione ricevente inizializza il PRNG con l'IV appena ricevuto, per ottenere il flusso di chiavi, poi processa con uno XOR il flusso di dati con il payload crittato e ottiene un nuovo flusso di dati. Per capire se i dati sono corretti, il destinatario prende i dati e li passa sotto funzione di hash, se ottiene lo stesso ICV li accetta. Immagine accanto a destra.



La cifratura/decifratura può essere effettuata dall'hardware WiFi (quindi l'IV viene inviato prima del testo cifrato). Lo standard 802.11 non specifica come l'IV debba essere generato e cambiato (ma si raccomanda di cambiarlo spesso, per evitare attacchi di riutilizzo, anche se nella realtà può non accadere). In effetti, molte schede WEP, ad ogni pacchetto: o autoincrementano l'IV, partendo da 0, oppure generano un IV pseudo-casuale, partendo da un seme fisso o casuale.

Il grande errore:

Fin qui può sembrare funzionante, infatti venne analizzato, implementato e venduto in milioni di copie in schede WI-FI, ma solo in un secondo momento ci si rese conto di un grossissimo problema (tra i tanti).

Il flusso di chiavi è identificato dalla coppia (*key, IV*) e lo scopo della crittoanalisi è di trovare almeno due pacchetti criptati con la stessa coppia per poter riconoscere la chiave. Di solito la chiave è invariata per molto tempo (mesi o anni), per cui se l'IV è autoincrementato, si ripete al massimo dopo $2^{24} \approx 16 \times 10^6$ pacchetti (poiché si hanno al massimo 3 byte di IV), ma con IV più bassi sono più comuni (a causa dei riavvii della scheda wireless). Invece, se l'IV è pseudorandom, per il "paradosso del compleanno" si ha che:

$$n(0,5) \approx 1.18 * (2^{24/2}) = 1.18 * 2^{12} = 4833 \\ p(10000) \approx 0,95$$

Per cui basta sniffare un certo numero di pacchetti fino a quando non si riconosce lo stesso IV, il che può accadere in tempi relativamente brevi (a causa del random potrebbero esserci gli stessi valori ricorrenti). A quel punto basta mettere in XOR i dati cifrati, si annulla la parte variabile dei dati e si ottiene lo XOR dei bit in chiaro. Da qui in poi è possibile effettuare delle analisi statistiche e/o inserire altri contenuti nei pacchetti.

Infatti, se un frame Ethernet trasporta fino a 1500 byte, 1 MB di dati corrisponde a almeno 700 frame, quindi per arrivare a 10000 frame della probabilità sopra basta ascoltare fino a 15 MB. Basta tenere conto che lo standard 802.11g può trasmettere fino a 14000 pacchetti/s per capire che questi dati arrivano in tempi molto brevi. A quel punto, dopo un numero sufficiente di sniffing, si osservano due (o più) pacchetti i cui payload C1 e C2 sono codificati con lo stesso keystream KS. Allora si può eseguire:

$$C1 \oplus C2 = (P1 \oplus KS) \oplus (P2 \oplus KS) = P1 \oplus P2$$

Con qualche analisi (es.: statistica, testo noto, frequenza, ecc.) si può recuperare i testi in chiaro P1 e P2, ovvero si ha un attacco passivo alla riservatezza dei dati!

Inoltre, $C1 \oplus P1 = P1 \oplus KS \oplus P1 = KS$ può essere eseguito per recuperare un pezzo di flusso di chiavi che può essere usato per crittografare e trasmettere pacchetti non più lunghi di P1. Con un numero sufficiente di pacchetti, si può recuperare pezzi di keystream sufficienti per trasmettere ciò che si vuole, ad esempio effettuare un attacco attivo all'accesso della risorsa (l'AP), e poiché il tempo di vita dell'IV non è specificato, il punto di accesso deve accettare qualsiasi IV, anche sempre lo stesso.

Altro grande errore:

Altro problema che fa acqua da tutte le parti è che per alcuni IV, il RC4 è debole. Nel WEP, gli IV deboli sono della forma $(a + 3, 255, x)$, per cui un aggressore che conosce il byte m -esimo del flusso di chiave può ricavare il byte $m+1$ -esimo.

Inoltre, in WEP, il primo byte di dati proviene spesso dal *Subnetwork Access Protocol* (SAP), per cui è possibile (statisticamente) ricavare in sequenza ogni byte della chiave dai *keystream chunks* e dall'IV. Ripetendo a sufficienza l'attacco di compleanno si possono recuperare abbastanza chunks e IV da recuperare l'intera chiave.

Correzione del WEP:

Per cui a seguito di queste fallo che sono state scoperte, è stata effettuata una correzione. Nel 2004 l'IEEE ha "prontamente" proposto una soluzione più robusta nell'emendamento 802.11i, che definisce un protocollo

di handshake a 4 vie per la mutua autenticazione, basato su un segreto condiviso chiamato *Pairwise Master Key* (PMK), e per la condivisione di una nuova chiave di sessione chiamata *Pairwise Transient Key* (PTK).

Inoltre, sono stati definiti due protocolli per la riservatezza e l'integrità dei dati:

- *Temporal Key Integrity Protocol* (TKIP): disciplina l'uso degli IV. Utilizza il cifrario RC4, quindi può essere utilizzato su hardware legacy. Obbligatorio in WPA, opzionale in WPA2, ora deprecato.
- *Counter-Mode/CBC-Mac Protocol* (CCMP): più robusto, utilizza AES in modalità CTR, quindi richiede nuovo hardware. Opzionale in WPA, obbligatorio in WPA2

Quindi, con questa nuova correzione si sono utilizzate delle chiavi temporanee (TKIP) che venivano sostituite regolarmente (dopo pochi secondi) durante le comunicazioni. Quindi, invece che usare sempre la vera WEP key per cifrare si usa una chiave temporanea, ovvero la chiave master WEP resta sempre la stessa, ma la chiave utilizzata in RC4 è la temporanea. Tuttavia si è visto che anche questa tecnica non è troppo robusta.

Altra falla riscontrata è quella avvenuta in un attacco di reinstallazione della chiave (*Key Reinstallation Attack*, KRACK, 2017), ovvero di recente, Vanhoef e Piessens hanno scoperto un attacco all'handshake a 4 vie (il bug è nel protocollo!), che sfrutta la ritrasmissione dei messaggi, consentita in caso di perdita di messaggi. Questa ritrasmissione consente a un aggressore di reinstallare una nuova chiave di sessione tra un client e il punto di accesso.

Inoltre, nel caso di Android 6.0, la chiave può essere forzata a 0. In CCMP, un avversario può riprodurre e decifrare (ma non falsificare) i pacchetti crittografati. In TKIP (e GCMP), un avversario può riprodurre, decifrare e falsificare i pacchetti crittografati. Sono in corso patch e soluzioni (ma molti sistemi sono ancora vulnerabili).

Crittografia a chiave pubblica

La crittografia tradizionale a chiave privata/singola utilizza una chiave, condivisa da mittente e destinatario. Se questa chiave viene rivelata, le comunicazioni sono compromesse. Per cui è ottima per garantire la riservatezza (attacchi passivi) e in quanto simmetrica, le parti sono uguali, quindi non protegge il mittente dalla falsificazione di un messaggio da parte del destinatario e dalla pretesa che sia stato inviato dal mittente (attacco attivo).

Probabilmente il progresso più significativo nei 3000 anni di storia della crittografia è stata l'invenzione della chiave pubblica dovuta a Whitfield Diffie e Martin Hellman all'Università di Stanford nel 1976. In realtà era già conosciuta in precedenza dalla comunità riservata, ad esempio dal CESG del Regno Unito (James Ellis, 1970) e dalla NSA (metà degli anni '60).

Questa tecnica utilizza due chiavi, una pubblica e una privata, per cui è asimmetrica, e proprio perché le parti non sono uguali, la riservatezza viene garantita maggiormente. Le implementazioni attuali utilizzano un'applicazione intelligente dei concetti della teoria dei numeri per funzionare (facile in un senso, difficile nell'altro) e integra piuttosto che sostituire la crittografia a chiave privata.

La crittografia a chiave pubblica/doppia chiave/asimmetrica prevede quindi l'uso di due chiavi:

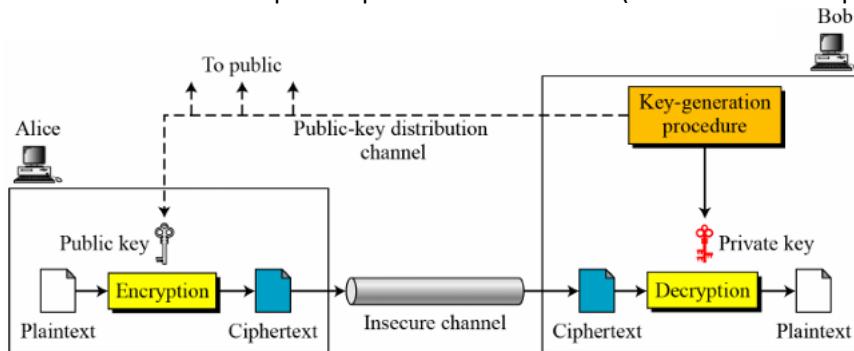
- una chiave pubblica (PU_A), che può essere conosciuta da chiunque e può essere utilizzata per crittografare i messaggi e verificare le firme
- una chiave privata (PR_A), nota solo al creatore della chiave, utilizzata per decifrare i messaggi e firmare (creare) le firme

È asimmetrica perché chi critta i messaggi o verifica le firme non può decriptare i messaggi o creare le firme.

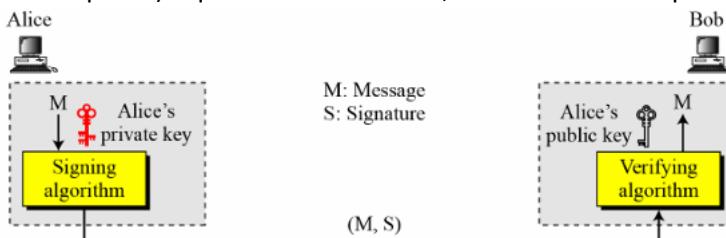
Le applicazioni delle chiavi pubbliche possono essere classificate in tre categorie (immagini sotto):

- crittografia/decrittografia (fornire segretezza): ovvero il mittente critta un messaggio con la chiave pubblica del destinatario. Quando un mittente A vuole mandare un messaggio a un destinatario B,

esso utilizza lo stesso schema di trasformazione delle chiavi visto prima, solo che con una chiave pubblica, mentre per la decrittazione il destinatario utilizza la chiave privata. Quindi, per comunicare, il mittente A critta un messaggio che vuole inviare a un destinatario B utilizzando la chiave pubblica di B, esso poi utilizzerà la sua chiave privata per decrittare il testo (e viceversa farà B per rispondere)



- firme digitali (forniscono autenticazione e non ripudio): il mittente “firma” un messaggio con la sua chiave privata. In questo caso se il mittente A vuole mandare un messaggio al destinatario B, crea il messaggio e prima di inviarlo genera una firma del messaggio (*signatures*) utilizzando il messaggio e la chiave privata, dunque ciò che viaggia è la coppia (M, S) . Da notare che in questo caso il messaggio non è cifrato. Il destinatario, invece, deve verificare con la chiave pubblica la validità del pacchetto (in modo da garantire il non ripudio) e quando esso l'accetta, nessuna delle due parti può ripudiare l'altra



- scambio di chiavi (chiavi di sessione): utilizzare la PKC per implementare un canale sicuro per lo scambio delle chiavi di sessione

Alcuni algoritmi sono adatti a tutti gli usi, altri sono specifici per uno di essi.

Il punto di forza di questa tecnica sta, quindi, nel fatto che gli algoritmi a chiave pubblica si basano su due chiavi meglio definibili come due parti della stessa chiave, dove:

- è computazionalmente impossibile trovare la chiave privata conoscendo solo l'algoritmo e la chiave pubblica
- è computazionalmente facile cifrare/decifrare i messaggi quando è nota la relativa chiave di cifratura. Infatti, gli schemi a chiave pubblica utilizzano funzioni unidirezionali: problemi facili (di tipo P) in un senso, ma difficili (di tipo NP) nell'altro.
- una delle due chiavi può essere usata per la crittografia, mentre l'altra è usata per la decrittografia (per alcuni algoritmi)

RSA

L'algoritmo RSA è stato inventato da Rivest, Shamir e Adleman del MIT nel 1977. Esso è lo schema a chiave pubblica più noto e diffuso e che si basa sull'esponenziazione in un campo finito (*Galois*) su numeri interi modulo un primo. L'esponenziazione richiede operazioni $O((\log n)^3)$, che sono facili, e utilizza numeri interi di grandi dimensioni (es.: 1024-2048 bit). La sicurezza è dovuta al costo della fattorizzazione di grandi numeri, che richiede operazioni $O(e^{\log n \log \log n})$, che sono difficili.

Funzione toziente di Eulero $\phi(n)$:

Quando si esegue l'aritmetica modulo n , l'insieme completo dei residui è: $0 \dots n - 1$, invece l'insieme ridotto dei residui è costituito da quei numeri (residui) che sono relativamente primi a n (ovvero che non hanno fattori in comune con n , es.: per $n=10$, l'insieme completo dei residui è $\{0,1,2,3,4,5,6,7,8,9\}$, mentre quello ridotto dei residui è $\{1,3,7,9\}$). I residui ridotti hanno inversioni moltiplicative (inoltre moltiplicando tra loro i numeri dell'insieme e effettuando il modulo, si ottengono sempre quei fattori come risultato) e il numero di elementi nell'insieme ridotto dei residui è chiamato *funzione di toziente di Eulero $\phi(n)$* (o *Euler Totient Function $\phi(n)$* , in cui ϕ si pronuncia 'phi').

Per calcolare il toziente può essere facile o difficile, perché c'è una proprietà interessante che viene utilizzata. Se n è un numero primo, $\phi(n) = n - 1$, poiché vanno presi tutti i numeri primi relativi con n , però essendo n un numero primo si ottengono tutti i numeri inferiori a esso (tranne sé stesso).

Inoltre, se n è dato da due numeri primi p e q , il suo toziente è dato da $\phi(n) = (p - 1)(q - 1)$.

Configurazione della chiave RSA e utilizzo:

Ogni utente genera una coppia di chiavi pubbliche/private selezionando due grandi primi a caso p e q . Calcolando il loro modulo di sistema $n = p * q$, si ottiene $\phi(n) = (p - 1)(q - 1)$, come visto prima.

A questo punto si può selezionare un numero a caso che sia co-primo con ϕ e che rappresenti la chiave di crittografia e , dove $1 < e < \phi(n)$, $\gcd(e, \phi(n)) = 1$. Quindi, ora per risolvere l'equazione per trovare la chiave di decrittazione d (inverso del modulo di $\phi(n)$) si deve calcolare $e * d = 1 \bmod \phi(n)$ e $0 \leq d \leq n$.

Dunque, la chiave pubblica di cifratura è data dalla coppia $PU = (e, n)$, mentre quella privata è $PR = (d, n)$.

Per criptare un messaggio M del mittente si prende la chiave pubblica del destinatario $PU = (e, n)$ e si calcola $C = M^e \bmod n$, dove $0 \leq M < n$. Invece, per decifrare il testo cifrato C il proprietario utilizza la propria chiave privata $PR = (d, n)$ e calcola $M = C^d \bmod n$.

Si noti che il messaggio M deve essere più piccolo del modulo n (blocco se necessario). Gli esponenti grandi possono essere calcolati in modo efficiente con l'algoritmo del quadrato e della moltiplicazione.

RSA funziona grazie al Teorema di Eulero: $a^{\phi(n)} \bmod n = 1$ dove $\gcd(a, n) = 1$. Infatti, in RSA si hanno le due equazioni $n = p * q$ e $\phi(n) = (p - 1)(q - 1)$ e la scelta accurata di e e d come inversi mod $\phi(n)$, per cui $ed = 1 + k * \phi(n)$ per qualche k .

Quindi:

$$C^d = M^{e*d} = M^{1+k*\phi(n)} = M^1(M^{\phi(n)})^k = M^1(1)^k = M^1 = M \bmod n$$

Tuttavia, questo ragionamento non è corretto se M non è co-primo con n , ma anche in questo caso il risultato vale per il *piccolo teorema di Fermat (Fermat's little theorem)*. Per una prova più completa, si veda https://www.dimgt.com.au/rsa_theory.pdf

Di seguito un esempio, in cui per l'impostazione delle chiavi si eseguono i passi:

1. Selezionare i numeri primi: $p = 17$ e $q = 11$
2. Calcolare $n = p * q = 17 * 11 = 187$
3. Computare $\phi(n) = (p - 1)(q - 1) = 16 * 10 = 160$
4. Selezionare e : $\gcd(e, 160) = 1$; scegliere $e = 7$
5. Determinare d : $d * e = 1 \bmod 160$ e $d < 160$, quindi $d = 23$ poiché $23 * 7 = 161 = 1 * 160 + 1$
6. Pubblicare la chiave pubblica $PU = (7, 187)$
7. Mantenere segreta la chiave privata $PR = (23, 187)$

Poi per la cifratura/decifrazione:

- dato il messaggio $M = 88$ (da notare che $88 < 187$)
- per crittografare si calcola $C = 88^7 \text{ mod } 187 = 11$
- per la decifrazione si calcola $M = 11^{23} \text{ mod } 187 = 88$

In riassunto per la generazione delle chiavi, gli utenti di RSA devono determinare due primi a caso (p e q), scegliere e o d e calcolare l'altro (di solito $e = 65537$). I primi p e q non devono essere facilmente ricavabili dal modulo $n = p * q$, il che significa che n deve essere sufficientemente grande e di solito si prova con un numero di Mersenne ($2^p - 1$) o di Fermat ($2^{2^n} + 1$) e si usa il test di primalità AKS o Miller-Rabin. Gli esponenti e e d sono inversi, quindi si usa l'algoritmo dell'inverso per calcolare l'altro.

Altri crittosistemi a chiave pubblica

Altri crittosistemi a chiave pubblica che non vengono approfonditi in questo documento sono:

- ElGamal: basato sul *logaritmo discreto*, in cui per p grande numero primo, e_1 una radice primitiva (cioè un generatore dei residui ridotti) si ha che:
dato d , è facile calcolare $e_2 = e_1^d \text{ mod } p$
dato e_2 , calcolare $d = \log_{e_1} e_2 \text{ mod } p$ è difficile
- Curve ellittiche: basate sulle soluzioni in un campo finito di un'equazione della forma
$$y^2 = x^3 + ax^2 + b$$

Questo insieme forma un gruppo in cui la moltiplicazione è facile, ma il logaritmo discreto è molto più difficile della fattorizzazione, quindi sono sufficienti chiavi più corte (una chiave pubblica a 256 bit di curva ellittica è equivalente a una chiave pubblica RSA a 3072 bit). È implementato in Algoritmo di firma digitale (DSA), Algoritmo di firma digitale a curva ellittica (ECDSA), ecc.

Sicurezza degli schemi a chiave pubblica

Come per gli schemi a chiave privata, l'attacco di ricerca esaustiva a forza bruta (*brute force exhaustive search*) è sempre teoricamente possibile, ma in questo caso le chiavi utilizzate sono troppo grandi (> 1024 bit). Infatti, uno schema a chiave privata a 64 bit ha una sicurezza approssimativamente simile a quella di un RSA a 512 bit.

La sicurezza si basa su una differenza di difficoltà abbastanza grande tra i problemi facili (en/decrypt) e quelli difficili (crittanalisi). Più in generale, il problema difficile è noto, ma viene reso sufficientemente difficile da non poter essere violato, per questo richiede l'uso di numeri molto grandi e è quindi lento rispetto agli schemi a chiave privata.

Come già detto, in RSA i possibili approcci all'attacco sono: la ricerca a forza bruta delle chiavi, che è resa inaccessibile dalla dimensione dei numeri, gli attacchi matematici basati sulla difficoltà di calcolare $\phi(n)$ fattorizzando il modulo n (anche questi resi difficili), gli attacchi temporali (sull'esecuzione della decifrazione) e attacchi a scelta del testo cifrato (date le proprietà di RSA).

Firme digitali

Le firme digitali consentono di verificare autore, data e ora della firma, di autenticare il contenuto del messaggio e sono verificabili da terze parti fidate per risolvere controversie (es.: arbitro, notaio, ecc., ma non l'attaccante). In realtà, le firme digitali includono la funzione di autenticazione con funzionalità aggiuntive.

La firma digitale deve dipendere dal messaggio firmato e deve utilizzare informazioni uniche sul mittente per prevenire sia la falsificazione che il rifiuto; essa deve essere relativamente facile da produrre e relativamente

facile da riconoscere e verificare, però essere computazionalmente impossibile da falsificare con un nuovo messaggio per una firma digitale esistente o con una firma digitale fraudolenta per un dato messaggio. Infine, deve essere pratica da salvare in memoria.

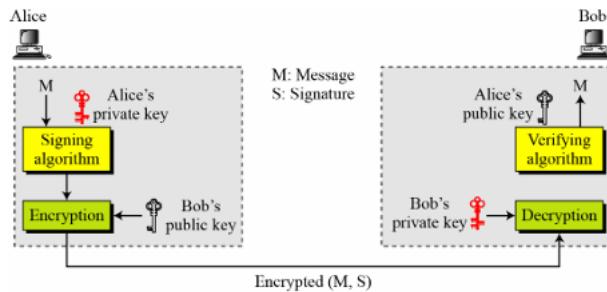
Firma digitale a chiave asimmetrica necessita di un sistema a chiave pubblica. Il firmatario firma con la sua chiave privata, mentre il verificatore verifica con la chiave pubblica del firmatario.



Come per la crittografia asimmetrica che usa e per crittare o alla d per decrittare, in questo caso si usano d per firmare e e per verificare la firma.

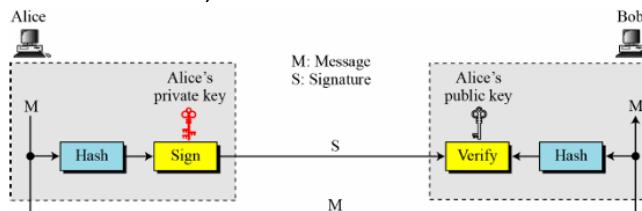
In realtà sulla stessa base si può creare un modello simmetrico aggiungendo una terza parte fidata (es.: notaio, ecc.), che consenta di fare da tramite e in modo tale per cui le chiavi utilizzate risultino “speculari”.

Tuttavia, la firma digitale non garantisce la riservatezza. Se necessario, si utilizza un secondo livello (es.: tramite chiavi pubbliche). Quindi, se si necessita di avere dati crittografati, prima li si firma e poi li si critta prima di spedirli al destinatario.

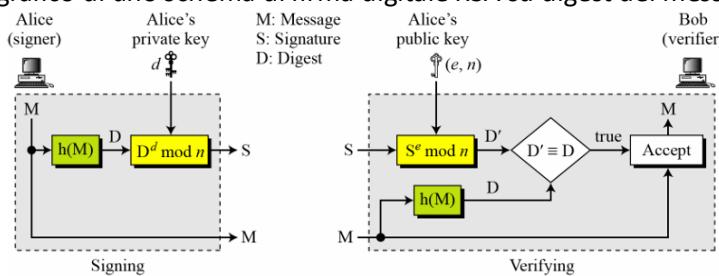


Questo procedimento funziona, ma a livello pratico non lo si utilizza quasi mai, soprattutto se il messaggio è molto lungo, perché si appesantiscono i calcoli computazionali di crittazione. Per questo motivo, di solito si utilizza uno schema più semplice e compatto, ovvero la firma del digest.

La firma del digest consiste nell'applicare la funzione di hash solo al messaggio e è una procedura necessaria perché i crittosistemi asimmetrici sono lenti, inoltre è sicura se le funzioni hash sono sufficientemente sicure.



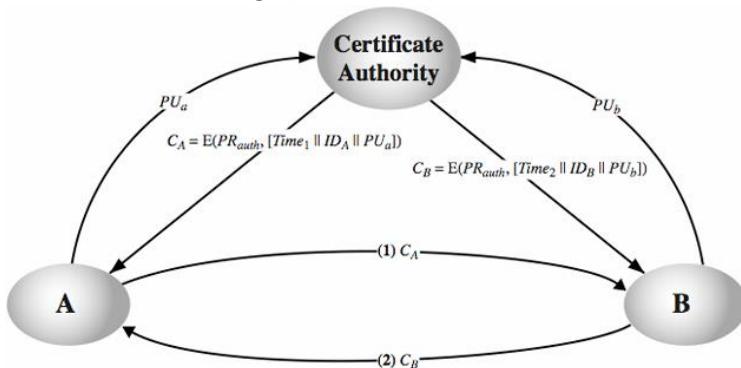
Di seguito un esempio grafico di uno schema di firma digitale RSA su digest del messaggio:



Certificati a chiave pubblica

Un momento critico della comunicazione asimmetrica, è quello della distribuzione delle chiavi pubbliche. Infatti, nel momento in cui un mittente A prende la chiave pubblica di un destinatario B per mandargli un messaggio, potrebbe esserci un attaccante E che prende anch'esso la chiave pubblica di B e gli invia un messaggio fingendosi A.

Si necessita allora di un meccanismo che consenta di legare l'identità di un endpoint alla chiave pubblica (di solito con altre informazioni come il periodo di validità, i diritti d'uso, ecc.). per questo motivo si usano i certificati a chiave pubblica, in cui tutti i contenuti sono firmati da una chiave pubblica di fiducia o da un'autorità di certificazione (CA). Essi possono essere verificati da chiunque conosca la chiave pubblica dell'autorità di certificazione. In realtà, i certificati sono generati solo dalla CA, ma non sono conservati nella CA (per evitare manomissioni e colli di bottiglia).



Ogni committente richiede una volta alla CA di generare un certificato, che viene conservato localmente, per cui la CA non ha bisogno di conservare copie dei certificati. I timestamp sono necessari per implementare l'obsolescenza dei vecchi certificati.

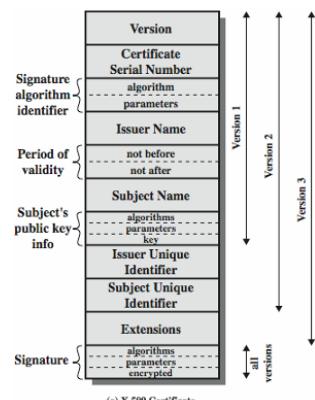
Quindi, A chiede il suo certificato alla CA e quest'ultima glielo da. Poi in qualsiasi momento, A invia il proprio certificato a B; B può “decriptare” il certificato utilizzando la chiave pubblica della CA. Questo assicura che il certificato proviene dalla CA. Se il certificato non è obsoleto, il PU_A è valido. Se il PR_A viene preso da un intruso, A genera una nuova coppia e chiede alla CA un nuovo certificato. Anche se in realtà, fino al timeout, l'intruso può essere autenticato come A. Ma, il problema si può ridurre con le liste di revoca (conservate nella CA).

Servizio di autenticazione X.509:

Fa parte degli standard del servizio di directory ITU-T X.500, ovvero server distribuiti che mantengono il database delle informazioni sugli utenti. Prima nel 1988, vi furono state numerose revisioni, ma nel 2002 divenne X.509v3 (RFC 3280).

Definisce il framework per i servizi di autenticazione, in cui la directory può memorizzare certificati a chiave pubblica dell'utente e firmata dall'autorità di certificazione. Definisce inoltre i protocolli di autenticazione, utilizza la crittografia a chiave pubblica, firme digitali e algoritmi non standardizzati, ma raccomandati da RSA. I certificati X.509 sono ampiamente utilizzati.

Viene rilasciato da una Autorità di Certificazione (CA), contenente: versione (1, 2 o 3), numero di serie (univoco all'interno della CA) che identifica il certificato, identificatore dell'algoritmo di firma, nome X.500 dell'emittente (CA), periodo di validità (da – a date), nome X.500 del soggetto (nome del proprietario), informazioni sulla chiave pubblica del soggetto (algoritmo, parametri, chiave), identificatore univoco dell'emittente (v2+), identificatore univoco del soggetto (v2+), campi estensione (v3), firma (dell'hash di tutti i campi del certificato). La notazione CA << A >> denota “certificato per A firmato da CA”.



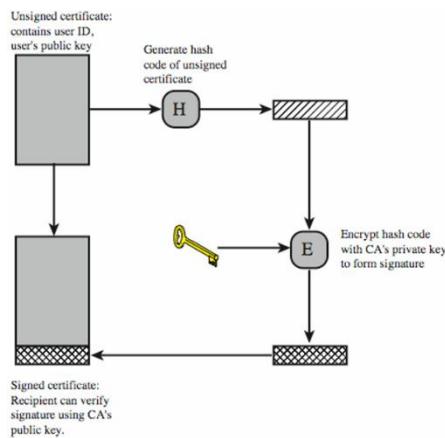
Estensioni del certificato:

Si possono aggiungere delle estensioni al certificato, come le informazioni sulla chiave e sulla politica. Esse trasmettono informazioni sulle chiavi del soggetto e dell'emittente, oltre a indicatori della politica del certificato, nonché ai livelli crescenti di controlli e quindi di fiducia (vedere sotto).

Classe	Controlli di identità	Uso
1	verifica nome/email	navigazione web/email
2	+ iscrizione/addr controllo	e-mail, sottoscrizioni, convalida SW
3	+ documenti d'identità	accesso a e-banking/servizi

Altre informazioni extra possono essere attribuiti del soggetto e dell'emittente del certificato, come il supporto di nomi alternativi (es.: e-mail), in formati alternativi per il soggetto del certificato e/o l'emittente. Oppure vincoli sul percorso del certificato, in modo da consentire vincoli sull'uso dei certificati da parte di altre CA.

Ottenere un certificato e come revocarlo:



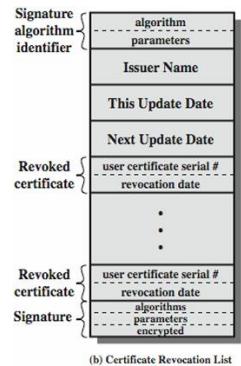
Qualsiasi utente che abbia accesso alla CA può ottenere qualsiasi certificato da essa. L'utente genera una chiave privata/pubblica, si autentica alla CA, invia la chiave pubblica alla CA in una "richiesta di certificato" e quest'ultima firma la richiesta (con la propria chiave privata e crittato) e restituisce il certificato.

Solo la CA può generare un certificato (che è un file locale), perché non può essere falsificato, poi i certificati possono essere inseriti in una directory pubblica e possono essere rinnovati prima della scadenza.

In realtà ci sono anche altri tipi di certificati e di metodi di generazione di essi, ad esempio, la tessera sanitaria regionale che ha un chip al suo interno, è in grado di generare dei certificati a chiavi RSA se attivata con un lettore apposito. Infatti essa genera delle chiavi pubbliche/private temporanee per poter accedere ai servizi pubblici (online o fisici).

Come già ribadito, i certificati hanno un periodo di validità e talvolta può essere necessario revocare il certificato prima della scadenza (es.: la chiave privata dell'utente è compromessa, l'utente non è più certificato da quella CA, il certificato della CA è compromesso, ecc.).

Le CA mantengono un elenco di certificati revocati e può essere pubblicato nella *Certificate Revocation List* (CRL) nel caso quei certificati non fossero ancora scaduti nel momento della revoca, e dopo del termine dei timestamp vengono rimossi da essa. Gli utenti dovrebbero controllare i certificati con la CRL della CA o verificare la validità di un certificato chiedendo alla CA tramite il OCSP (*Online Certificate Status Protocol*).

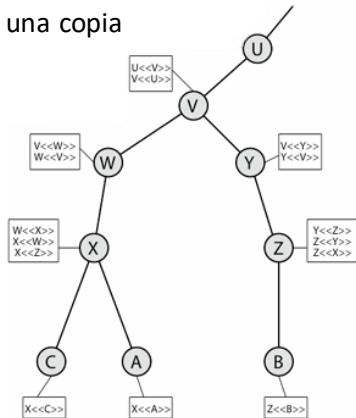


Gerarchia delle CA:

Se entrambi gli utenti condividono una CA comune, si presume che ne conoscano la chiave pubblica, altrimenti le CA devono formare una gerarchia, per questioni di scalabilità. Si utilizzano i certificati che collegano i membri della gerarchia per convalidare le altre CA. Ogni CA ha certificati per i clienti (in avanti) e per i genitori (indietro) e ogni cliente si fida dei certificati dei genitori. Questo meccanismo consente la verifica di qualsiasi certificato di una CA da parte degli utenti di tutte le altre CA della gerarchia.

Ad esempio, un utente C, sapendo solo la chiave pubblica di V, può ottenere una copia verificata della chiave pubblica di B:

1. B invia a C una catena di certificati, U<<V>>, V<<Y>>, Y<<Z>>, Z<>.
2. C convalida V<<Y>> utilizzando la chiave pubblica di V (U<<V>> non è necessario)
3. C estrae la chiave pubblica di Y da V<<Y>>.
4. C valida Y<<Z>> usando la chiave pubblica di Y.
5. C estrae la chiave pubblica di Z da Y<<Z>>.
6. C valida Z<> usando la chiave pubblica di Z.
7. C estrae la chiave pubblica di B da Z<>



È importante tenere presente che quando una CA genitore scade, anche tutti i suoi figli scadranno. Per cui andranno sostituiti tutti affinché essi continuino a valere (a livello gerachico).

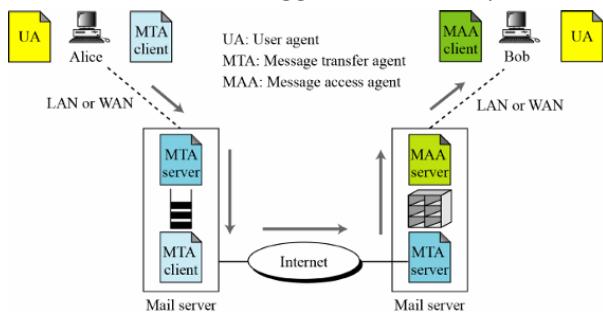
Inoltre, se un certificato venisse rubato, è possibile che venga utilizzato come se fosse il proprietario, ad esempio se rubassero il certificato del sito dell'università, si potrebbe mettere in piedi un altro dominio uguale a esso e farlo valere come tale. Il problema cresce se il certificato che viene rubato è quello di una qualche CA genitore, poiché potrà rilasciare certificati a chiunque (anche con chiavi di altri utenti, ma figurandoli come di altri) e/o ritirare quelli già rilasciati. Questo può essere uno svantaggio della gerarchia, anche se a mano a mano che si sale, il livello di sicurezza diventa estremo e difficile da attaccare (inteso fisicamente blindato come bunker con guardie armate e con accesso solo con autorizzazione firmata). Questa sicurezza è dovuta al fatto che in passato alcune CA sono state attaccate (es.: lo scandalo della DigiNotar che rilasciava certificati per alcuni dominii Gmail, che venne attaccata da un'operazione chiamata Black Tulip) e sono state sfruttate per creare certificati fraudolenti.

Autorità radice di certificazione:

Le CA radice (*root*) sono autocertificate (*self-certified*), ovvero si firmano il proprio certificato. Questi certificati devono essere distribuiti in modo sicuro, infatti di solito vengono forniti con il sistema operativo o con le applicazioni. È necessario fidarsi di loro se ci si fida del proprio sistema operativo o della propria applicazione.

Sicurezza delle e-mail

La posta elettronica è uno dei servizi di rete più diffusi e apprezzati. In realtà, nella normale posta elettronica, il contenuto dei messaggi non è sicuro: possono essere ispezionati e modificati sia in transito, sia da utenti



opportunamente privilegiati sul sistema intermedio o di destinazione, è facile impersonare qualsiasi utente, ecc. Per farla breve è molto simile alle cartoline delle vacanze per cui si necessita di una protezione, specialmente in termini di riservatezza (protezione dalla divulgazione), autenticazione del mittente del messaggio, integrità del messaggio (protezione dalla modifica), non ripudio dell'origine (protezione dalla negazione da parte del mittente), non ripudio della destinazione (protezione dalla negazione del destinatario).

La rete per il sistema di invio delle e-mail è una cosiddetta *rete overlay*, ovvero una rete costituita da tanti nodi/switch che lavorano sopra Internet (a livello applicativo). Questi nodi/switch vengono chiamati server di posta elettronica o server MTA (*message transfer agent*). Lo scopo di questi server è di guardare le mail che arrivano e di mandarle al prossimo server. Quindi quando un utente crea una e-mail con un MTA client e la invia, questa passa per qualche server prima di arrivare in quello del destinatario (stile rete a commutazione di pacchetto). Il destinatario a sua volta, per prendere le e-mail usa un client MAA (*message access agent*).

Per fare in modo che la comunicazione avvenga, si usa il protocollo SMTP (*Simple Mail Transfer Protocol*) per inviare la e-mail, mentre per poterla leggere, il destinatario utilizza i protocolli POP o IMAP. Questi protocolli sono implementati entrambi nei servizi di posta elettronica (es.: Thunderbird, Outlook, ecc.). In realtà queste applicazioni e i browser nascondono questi meccanismi.

Quindi, l'invio di un'e-mail è un'attività una-tantum (*one-time activity*), in cui non esiste una "sessione", né uno scambio reciproco. In termini di sicurezza, il mittente del messaggio deve includere gli identificatori degli algoritmi utilizzati nel messaggio. Devono essere utilizzati alcuni algoritmi a chiave pubblica. La cifratura/decifratura può essere effettuata con un algoritmo a chiave simmetrica, ma la chiave segreta per decifrare il messaggio è cifrata con la chiave pubblica del destinatario ed è inviata insieme al messaggio.

Modello PGP

Questo modello appena descritto si chiama *Pretty Good Privacy* (PGP), sviluppato da Phil Zimmermann negli anni '90, e è uno standard di fatto per la sicurezza delle e-mail. Può fornire riservatezza, firma digitale (non ripudio), o entrambe, nel corpo dell'email. Per rendere disponibile questo servizio di sicurezza sono stati selezionati i migliori algoritmi di crittografia a disposizione e integrati in un unico programma (su Unix, PC, Macintosh e altri sistemi) e originariamente era gratuito, ma ora ha anche versioni commerciali.

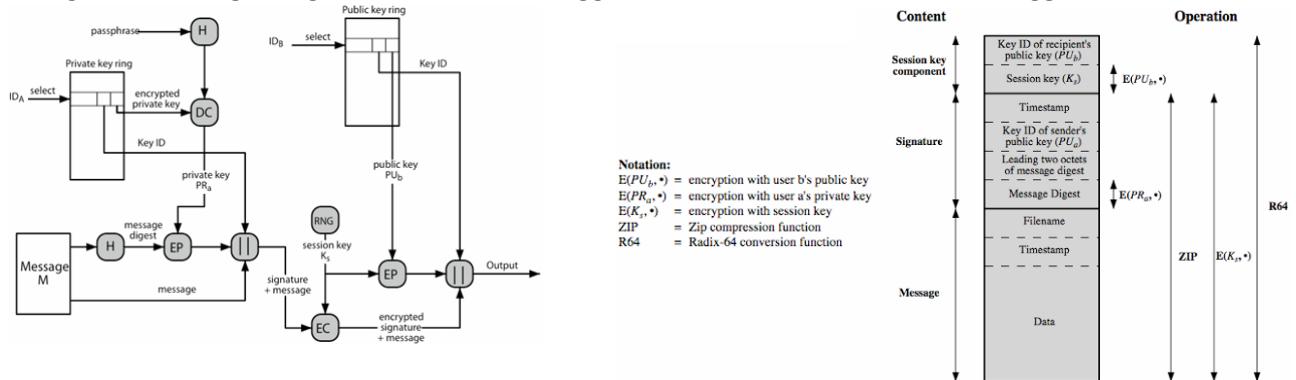
Ogni utente PGP ha una coppia di portachiavi:

- l'anello delle chiavi pubbliche, che contiene tutte le chiavi pubbliche di altri utenti PGP noti a questo utente, indicizzate per ID di chiave
- l'anello delle chiavi private, che contiene la coppia di chiavi pubbliche/private di questo utente, indicizzate per ID di chiave e crittografate da una *hashed passphrase* (frase hash di accesso)

La sicurezza delle chiavi private dipende quindi dalla sicurezza della *passphrase* (frase di accesso).

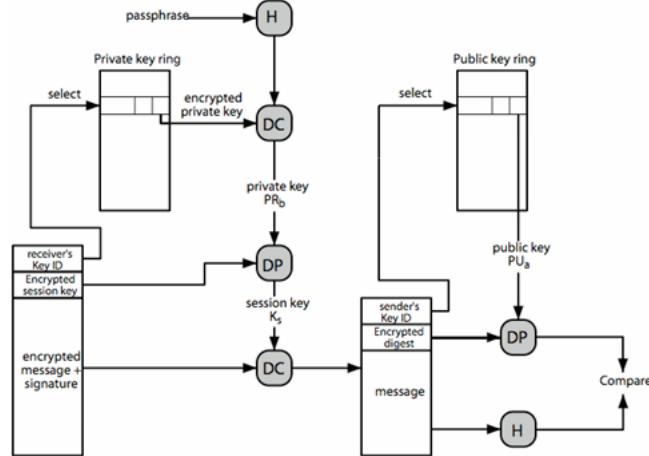
Per quanto riguarda l'autenticazione e l'invio del messaggio cifrato con hash e chiave RSA (già vista in realtà), dunque ogni utente crea un messaggio, calcola la hash, lo firma con la propria chiave privata e lo invia. Il ricevitore, invece, utilizza la chiave pubblica del mittente per decrittare e recuperare l'hash del messaggio, verifica la correttezza del testo e lo compara con quello decrittato.

Di seguito le immagini di generazione del messaggio (sinistra) e del formato del messaggio PGP (destra):



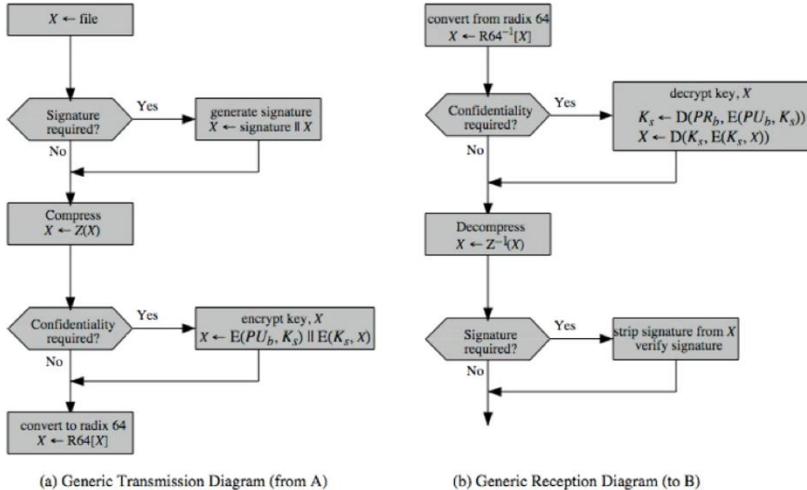
Per quanto riguarda la confidenzialità, invece, la si implementa col meccanismo della cifratura simmetrica. Quindi, il mittente crea il messaggio e genera una chiave random di sessione (runtime, quindi ogni e-mail ha una chiave diversa e può essere dimensioni variabili, a seconda dell'algoritmo di cifratura utilizzato, di solito a 128 bit, e generata in modalità ANSI X12.17), cifra il messaggio usando un algoritmo di cifratura apposito (es.: CAST-128, IDEA, 3 DES in modalità CBC, ecc.) con la chiave di sessione, inserisce nel pacchetto la chiave di sessione crittografata in RSA con la chiave pubblica del destinatario (perché altrimenti il receiver non saprebbe come decrittare le informazioni) e invia il messaggio così ottenuto. Da notare che viene definita "chiave di sessione" in quanto l'invio di una e-mail corrisponde a una sessione di comunicazione (non si

attende subito la risposta, per cui nasce e termina lì, questa è la durata). A questo punto, il destinatario prende il pacchetto, estrae la chiave simmetrica usando la sua chiave privata, estrae dal contenuto ottenuto la chiave di sessione e decifra il messaggio. Di seguito un'immagine che mostra i passaggi della ricezione del messaggio:



In realtà non sempre i messaggi in PGP vengono firmati, né non è d'obbligo che siano crittografati per la confidenzialità. Inoltre, di default PGP comprime i messaggi (usando l'algoritmo ZIP) dopo che sono stati firmati, ma prima di essere crittografati. Inoltre, per compatibilità con i formati e-mail (di solo testo), i dati grezzi vengono codificati in caratteri ASCII stampabili tramite algoritmo radix-64 (ogni byte contiene solo 6 bit significativi, ovvero 64 caratteri stampabili, e vengono mappati 3 byte in 4 caratteri stampabili), questo perché la posta è un servizio vecchio che in passato funzionava solo con ASCII. PGP segmenta i messaggi se sono troppo lunghi.

Di seguito un'immagine che riassume tutti i passaggi che vengono eseguiti in PGP:



Gestione delle chiavi PGP:

In PGP ogni utente è la propria “autorità di certificazione” e può firmare le chiavi di utenti che conosce direttamente. In questo modo si forma una “rete di fiducia” (“web of trust”) in cui gli utenti si fidano delle chiavi che hanno firmato. Inoltre, un utente può fidarsi delle chiavi firmate da altri se ha una catena di firme che le collega a loro (quindi chiede ai suoi “amici” se conoscono quel mittente e sanno che può fidarsi).

Questa rappresenta a tutti gli effetti una rete distribuita, invece che gerarchica come X.509, in cui l’anello portachiavi delle chiavi include indicatori di fiducia e gli utenti possono anche revocare le loro chiavi (es.: quando sono compromesse o semplicemente vecchie).

S/MIME (Secure/Multipurpose Internet Mail Extensions)

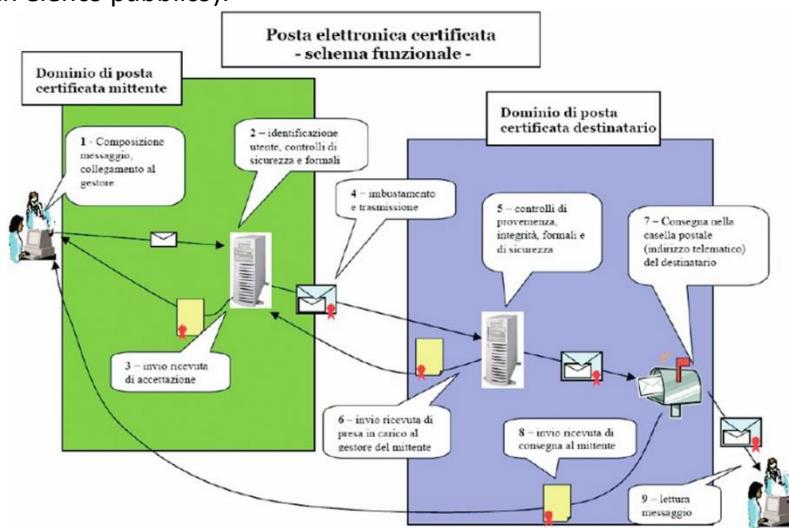
La posta elettronica originale di Internet RFC822 era solo testuale. Invece, MIME ha fornito il supporto per diversi tipi di contenuto e per i messaggi in più parti, con la codifica dei dati binari in forma testuale. Poi S/MIME ha aggiunto dei miglioramenti alla sicurezza.

I principi di funzionamento sono simili a quelli di PGP, ma con implementazioni diverse originariamente sviluppate da RSA e poi standardizzate da IETF. È diventato lo standard de jure per la posta elettronica sicura (RFC 3851 e 5751) e al giorno d'oggi, la maggior parte degli agenti di posta supporta S/MIME (es.: MS Outlook, Mozilla, Thunderbird, Mac Mail, ecc.). Quindi se arriva una e-mail codificata con S/MIME, automaticamente il client la decodifica e va a vedere la firma dell'attachment (file *smime.p7s*) se è certificata e in caso lo specifica graficamente all'utente.

Posta Elettronica Certificata (PEC)

C'è un ulteriore miglioramento della sicurezza della posta elettronica S/MIME, ovvero la *Posta Elettronica Certificata* (PEC). Essa aggiunge la non ripudiabilità del destinatario (che S/MIME non garantisce) e il timestamp di invio certificato (da parte del server), per cui è simile alla posta raccomandata (*registered mail*).

Quando il messaggio viene consegnato nella casella di posta del destinatario, il mittente riceve una ricevuta firmata. Tuttavia questo non garantisce che il destinatario abbia effettivamente letto l'email, ma non può affermare di non averla ricevuta. Inoltre, questa ricevuta ha valore legale, se i server sono ufficialmente riconosciuti (esiste un elenco pubblico).



Gestione delle chiavi e autenticazione delle entità

L'autenticazione delle entità è una tecnica progettata per consentire a una parte di provare l'identità di un'altra parte. Un'entità può essere una persona, un processo, un client, un server, ecc.

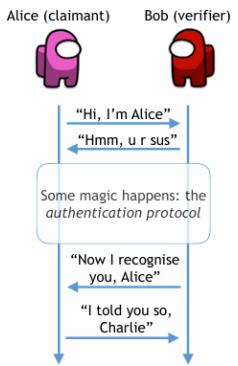
L'entità la cui identità deve essere provata è il richiedente (*claimant*). La parte che cerca di provare l'identità del richiedente è chiamata verificatore (*verifier*). Essi hanno una serie di approcci basati sull'utilizzo della crittografia a chiave pubblica, soprattutto per la distribuzione delle chiavi di sessione.

È, inoltre, necessario assicurarsi che le chiavi pubbliche delle altre parti siano corrette, anche se questo potrebbe non essere il caso (es.: chiavi obsolete/revocate). Esistono vari protocolli che utilizzano timestamp o nonces (e anche in questo caso, molti protocolli pubblicati sono risultati difettosi).

Autenticazione del messaggio e dell'entità

È importante non confondere l'autenticazione dei messaggi con l'autenticazione delle entità. L'autenticazione dei messaggi autentica un messaggio alla volta e potrebbe non avvenire in tempo reale. L'autenticazione di entità autentica il richiedente per l'intera durata di una sessione (livello 5 dello stack).

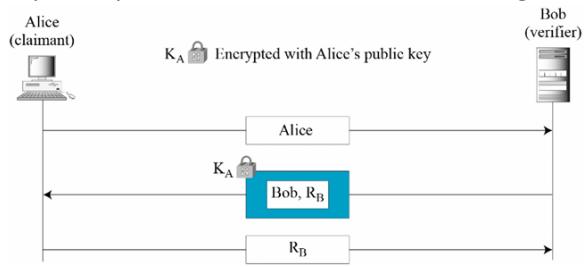
In questo caso si dice che “il richiedente (A) è autenticato per il verificatore (B)”. Questo di solito corrisponde alla creazione di un segreto condiviso (es.: una nuova chiave di sessione). L'autenticazione delle entità modifica la conoscenza interna dei partecipanti: c'è un *prima* e un *dopo*.



Autenticazione con crittografia a chiave asimmetrica:

Se si ha una crittografia asimmetrica, si può identificare un partecipante attraverso la sua chiave segreta, tramite protocollo di autenticazione unidirezionale:

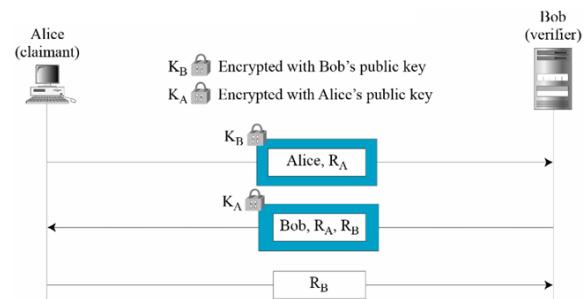
1. $A \rightarrow B: A$
2. $B \rightarrow A: E_{K_A}(B, R_B)$
3. $A \rightarrow B: R_B$



R_B è un numero casuale generato sul posto, chiamato *nonce*. I passi 2-3 sono una sfida-risposta (*challenge-response*): B lancia una sfida che solo il vero A può risolvere. Sono possibili molte varianti (domande diverse), a seconda di ciò che A conosce. Si noti che la sfida viene lanciata dal verificatore e a cui risponde il richiedente. Questo non autentica B per A.

Invece, col protocollo di autenticazione bidirezionale si ha:

1. $A \rightarrow B: E_{K_B}(A, R_A)$
2. $B \rightarrow A: E_{K_A}(B, R_A, R_B)$
3. $A \rightarrow B: R_B$



R_A e R_B sono entrambi nonces. I passi 1-2 sono una sfida-risposta per B. Questo autentica B per A. Invece, i passi 2-3 sono una sfida-risposta per A, che autentica A per B. In questo caso entrambe le parti devono disporre di una chiave pubblica e privata.

Procedure di autenticazione in X.509

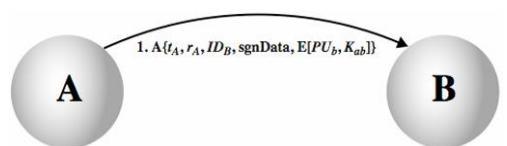
Lo standard X.509 definisce 3 procedure di autenticazione:

- A una via / unidirezionale (One-Way): per messaggi unidirezionali
- a due vie (Two-Way): sessione interattiva, basata su timestamp (richiede una certa sincronizzazione)
- a tre vie (Three-Way): sessioni interattive, basate su nonces, senza timestamp

Tutti utilizzano firme a chiave pubblica e un formato del messaggio (in cui non tutti i campi sono sempre utili) del tipo: $A\{timestamp, nonce, dest, dati firmati, dati segreti\}$. Il fatto che i campi vengono racchiusi tra parentesi graffe, significa che i dati sono stati tutti firmati da A.

Autenticazione a una via (unidirezionale):

Un messaggio ($A \rightarrow B$) viene utilizzato per stabilire l'identità di A e l'integrità e l'originalità del messaggio. Il messaggio proviene da A e è destinato a B, e deve includere timestamp, nonce, identità di B e firma di A ($A\{\dots\}$). Può includere informazioni aggiuntive per B (es.: la chiave di sessione).

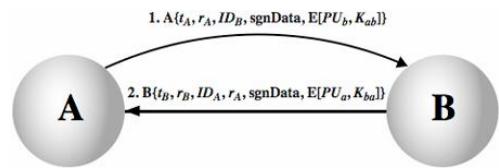


In questo caso, A effettua un claim a B e gli invia un messaggio firmandolo. All'interno di questo messaggio vi sarà l'identità di A, il fatto di essere destinato a B, la chiave condivisa K_{ab} (segreto condiviso che viene utilizzato in seguito) che viene cifrata con la chiave pubblica di B (e che viene allegata) e altri dati firmati.

Quando B riceve questo messaggio vede che è firmato, per cui procede con la verifica con la chiave pubblica di A che il messaggio sia autentico. Successivamente prende la sua chiave privata e decodifica la parte finale del messaggio, ottenendo la chiave K_{ab} generata da A.

Autenticazione a due vie:

In questo caso si condividono due messaggi ($A \rightarrow B$, $B \rightarrow A$) che stabiliscono inoltre: l'identità di B e che la risposta proviene da B, e che la risposta è destinata ad A, nonché l'integrità e l'originalità della risposta. La risposta include il nonce originale di A, nonché il timestamp e il nonce di B. Inoltre, può includere informazioni aggiuntive per A (es.: la chiave di sessione).



La prima parte è come nell'autenticazione unidirezionale, solo che in più si aggiunge la risposta di B, ovvero lo stesso meccanismo solo che rivolto all'utente A. Inoltre, nell'invio di messaggio di B, viene allegata anche la risposta al claim di A.

Sia questa che l'autenticazione precedente, talvolta hanno uno svantaggio dovuto ai timestamp. Nel caso in cui un attaccante intercetti un messaggio, lo prenda, decodifichi le informazioni (K_{ab}) e lo rimandi uguale in un secondo momento (anche a distanza di anni), se l'utente, ad esempio B, che si vede arrivare questo messaggio non guardasse i timestamp, inizierebbe una nuova sessione (lui non si ricorderebbe che quella stessa conversazione era già iniziata tanto tempo indietro). A quel punto B potrebbe interpretare l'attaccante (a sua insaputa) come A. Se, invece B guardasse i timestamp potrebbe accorgersi della truffa e rifiutare il messaggio. L'attaccante quindi modifica i timestamp in modo che siano validi: ora B pensa che quel mittente sia valido (non si accorge della truffa, per cui essi risultano inutili).

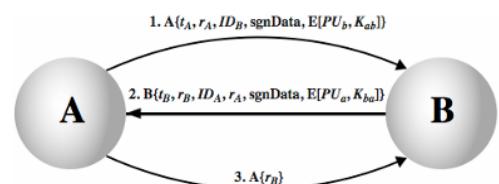
Questo tipo di attacco appena descritto si chiama attacco di replica (*replay attack*: un attaccante può riutilizzare un vecchio messaggio di A per indurre B a utilizzare una vecchia chiave di sessione K_{ab}).

Come già detto, per evitarlo, si controlla il timestamp t_A su B: il messaggio è accettato se $|t_B - t_A| < \Delta$, dove t_B è l'ora locale di B quando riceve il messaggio da A.

A questo punto sorge il problema di come scegliere Δ : se troppo grande, si è più a rischio di attacchi replay; se troppo piccolo, si può rifiutare molti messaggi validi a causa dei ritardi della rete. Inoltre, come mantenere sincronizzati gli orologi di A e B? questo può essere un grosso problema in quanto gli orologi hardware sono spesso molto imprecisi, tuttavia esistono protocolli di sincronizzazione (NTP), ma anch'essi non sono perfetti (fino a ~5-10ms) e possono quindi essere attaccati.

Autenticazione a tre vie:

In questo terzo caso si inviano tre messaggi ($A \rightarrow B$, $B \rightarrow A$, $A \rightarrow B$) che consentono l'autenticazione come già visto, ma senza l'utilizzo di timestamp. Questo perché si ha come terzo messaggio, la risposta da A a B contenente una copia firmata del nonce di B, quindi significa che non è necessario controllare o fare affidamento sui timestamp (quindi può essere ignorato). La magia sta sul fatto che si effettuano controlli sulle chiavi utilizzate, per cui è difficile che un attaccante abbia anche le chiavi, o che comunque le abbia subito al momento della risposta. In questo modo solo i veri endpoint possono firmare le chiavi correttamente.



Diffie-Hellman Key Agreement

Il primo schema di tipo “chiave pubblica” proposto da Diffie & Hellman nel 1976 è insieme all’esposizione dei concetti di chiave pubblica, anche se si è scoperto che Williamson (UK CESG) aveva proposto segretamente il concetto nel 1970.

È un metodo pratico per creare una chiave di sessione simmetrica senza la necessità di una terza parte (“centro di distribuzione delle chiavi”). Non può essere utilizzato per scambiare un messaggio arbitrario, piuttosto può stabilire una chiave comune nota solo ai due partecipanti. Il valore della chiave dipende dai partecipanti (e dalle loro informazioni sulle chiavi private e pubbliche). Si basa sull’esponenziazione in un campo finito (*Galois*, modulo di un primo o di un polinomio), che è facile da computare, e la sicurezza si basa sulla difficoltà di calcolo dei logaritmi discreti (simile alla fattorizzazione), che è difficile.

Per quanto riguarda lo scambio delle chiavi, tutti gli utenti concordano sui parametri globali:

- intero primo grande o polinomio p
- g è una radice primitiva $\text{mod } p$ per cui vale:

$$\{g^i \text{ mod } p \mid 0 \leq i \leq p-1\} = \{0 \dots p-1\}$$

Ogni utente (es.: A) genera la propria chiave scegliendo una chiave segreta (numero) $x < p$ e calcola la propria chiave pubblica (o metà di essa) con la formula $R_1 = g^x \text{ mod } p$. Ogni utente rende pubblica la chiave R_1 .

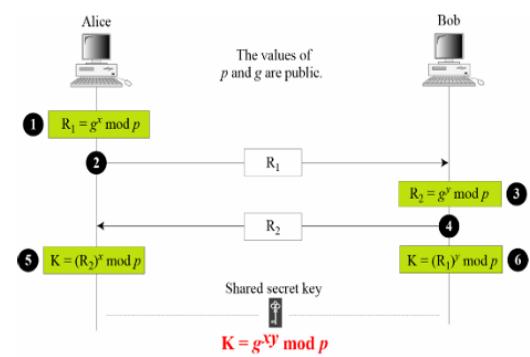
Quindi, la chiave di sessione condivisa dagli utenti A e B è K , data da:

$$\begin{aligned} K &= g^{x+y} \text{ mod } p \\ &= R_1^y \text{ mod } p \quad (\text{che B può calcolare}) \\ &= R_2^x \text{ mod } p \quad (\text{che A può calcolare}) \end{aligned}$$

Ora K può essere usata come chiave di sessione in uno schema di crittografia a chiave privata o in altri schemi tra A e B. Se A e B comunicano successivamente, avranno la stessa chiave di prima, a meno che non scelgano nuove chiavi pubbliche. L’attaccante ha bisogno di x o y , e deve risolvere il log discreto (difficile).

Ad esempio, gli utenti A e B che desiderano scambiarsi le chiavi:

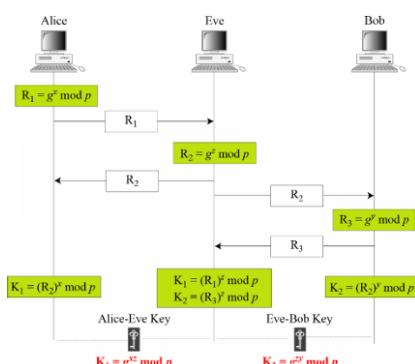
1. si accordano sul primo $p = 353$ e $a = 3$
2. scelgono chiavi segrete casuali: A sceglie $x_A = 97$, B sceglie $x_B = 233$
3. calcolano le rispettive chiavi pubbliche:
 - $R_A = 3^{97} \text{ mod } 353 = 40$ (A)
 - $R_B = 3^{233} \text{ mod } 353 = 248$ (B)
4. calcolano la chiave di sessione condivisa come:
 - $K_{AB} = R_B^{x_A} \text{ mod } 353 = 248^{97} = 160$ (A)
 - $K_{AB} = R_A^{x_B} \text{ mod } 353 = 40^{233} = 160$ (B)



Attacco Man-in-the-middle

La DH semplice è vulnerabile a un attacco *man-in-the-middle*, perché i messaggi non sono autenticati.

Un attaccante E agisce come intermediario tra A e B, eseguendo due scambi DH in parallelo: uno con A, l’altro con B. Quindi E può intercettare, decifrare, leggere/modificare e ricifrare tutti i messaggi. Si noti che si tratta di un attacco attivo: se l’aggressore effettuasse solo attacchi passivi (sniffing), non sarebbe un problema.



In questo caso A invia il messaggio verso B, ma viene intercettato da E, il quale computa R_2 , che poi invia a entrambi gli endpoint e essi rispondono di conseguenza pensando di essere l'altro utente. Alla fine della configurazione A e B sono ignari che in realtà ci sia un tramite E che li ascolta e interviene a suo piacimento.

In realtà questa tecnica può avvenire anche in altri sistemi, ad esempio la crittografia end-to-end di WhatsApp e Telegram, o anche altre (in realtà è poco probabile che avvengano perché sono molto robusti, quindi accade solo in presenza di applicazioni contraffatte in cui è stato inserito un codice che scopre le chiavi oppure infiltrandosi nei server di scambio delle chiavi/messaggi).

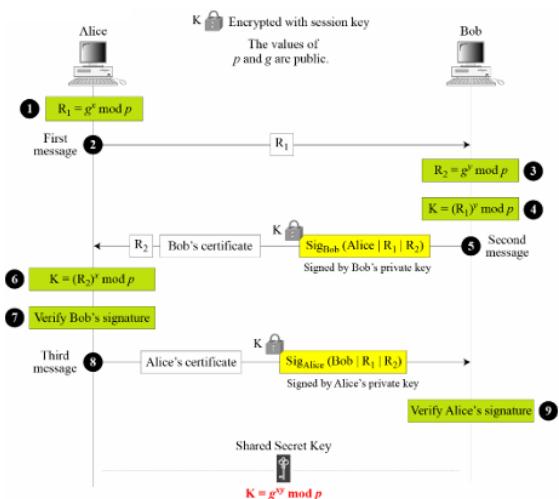
In realtà il problema non sta nell'attacco alla confidenzialità dei messaggi (R_1 , R_2 , ecc.), che sono in chiaro (infatti DH funziona anche senza la crittografia), ma nello schema stesso che non garantisce l'autenticità dei messaggi, cioè quando a B arriva R_1 crede che arrivi da A quando potrebbe non essere così.

Accordo di chiave sicura da stazione a stazione

La soluzione è che le chiavi pubbliche DH devono essere autenticate (*Safe station-to-station key agreement*). Di solito le chiavi sono firmate utilizzando certificati a chiave pubblica. In questo caso la crittografia asimmetrica e le firme digitali risultano utili. Soluzioni simili sono utilizzate in SSL e IPSec.

In questo caso, quando viene condivisa la chiave e poi computata la sua rispettiva, viene allegata la firma digitale, in modo da poter garantire all'altro utente che quella chiave è valida. Lo stesso farà poi l'altro utente.

In questa maniera un utente E non riesce a falsificare la chiave di crittazione o la firma digitale, per cui non può impersonare un altro utente.



Sicurezza Web

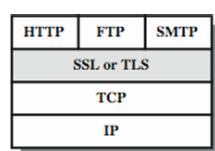
Il Web è ampiamente utilizzato da aziende, enti pubblici e privati, ma Internet e il Web sono vulnerabili, infatti hanno una varietà di minacce (es.: integrità, riservatezza, denial of service, autenticazione, ecc.). Per cui necessitano di meccanismi di sicurezza aggiuntivi.

Sicurezza a livello di trasporto (protocollo SSL/TLS)

Si effettua tramite TLS (*Transport Layer Security*), ovvero il vecchio SSL (*Secure Socket Layer*), un servizio di sicurezza a livello di trasporto e di sessione che offre integrità e riservatezza, anche sul flusso di traffico (in qualche modo).

Si utilizza il protocollo TCP per fornire un servizio end-to-end affidabile. SSL era originariamente sviluppato da Netscape negli anni '90, mentre la versione 3 è stata progettata con il contributo del pubblico e solo successivamente è diventato uno standard Internet noto come TLS (*Transport Layer Security*).

Tutto quanto è sviluppato in librerie, che sono perfettamente integrabili anche in vecchie macchine.



Protocol	Published	Status
SSL 1.0	Unpublished	Unpublished
SSL 2.0	1995	Deprecated in 2011
SSL 3.0	1996	Deprecated in 2015
TLS 1.0	1999	Deprecated in 2020
TLS 1.1	2006	Deprecated in 2020
TLS 1.2	2008	
TLS 1.3	2018	

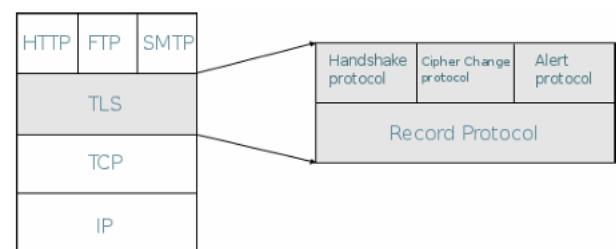
Porting delle applicazioni a SSL/TLS:

SSL/TLS è pensato per essere facile da usare anche per i programmati. Infatti, non è difficile, ma richiede l'accesso e la modifica del codice sorgente. Alcune chiamate sono specifiche, altre sono molto simili a quelle dei socket:

- `SSL_library_init(3), SSL_CTX_new(3)`: impostano varie opzioni relative a certificati, algoritmi ecc.
- `SSL_new(3)`: crea la sessione (oggetto SSL)
- `SSL_set_fd(3)`: associa la rete all'oggetto
- `SSL_accept(3), SSL_connect(3)`: Handshake TLS/SSL (dal server, dal client)
- `SSL_read(3), SSL_write(3)`: lettura e scrittura di dati sulla connessione TLS/SSL
- `SSL_shutdown(3)`: chiude la connessione TLS/SSL

Architettura TLS

TLS ha due livelli di protocolli. Il primo è il *Protocollo Record* che implementa l'effettiva trasformazione di sicurezza dei dati dell'applicazione (payload) nel canale TCP insicuro e viceversa. Esso utilizza i socket TCP per la trasmissione dei dati e viene utilizzato direttamente dai protocolli applicativi (es.: HTTP, IMAP, POP, ecc.). Può essere visto come un protocollo di presentazione (livello 6) e inoltre utilizza informazioni mantenute nello stato di connessione e gestite da altri protocolli (in quanto per funzionare necessita di condivisione di chiavi e per fare ciò necessita di una sessione).

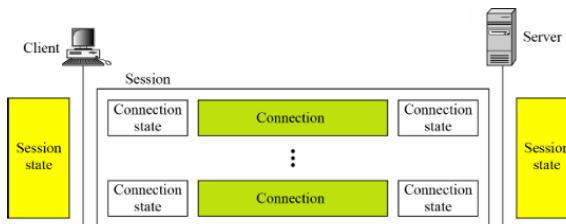


Il secondo livello è quello dei protocolli Handshake, Change Cyper Spec e Alert, che sono utilizzati per l'autenticazione delle entità, per impostare e gestire le informazioni necessarie per i servizi di sicurezza (chiavi segrete). Dovrebbero essere visti come protocolli di sessione (livello 5) e servono a far funzionare il protocollo TLS Record appena descritto.

Stati di sessione e di connessione TLS:

Ci sono inoltre delle informazioni da mantenere, ovvero quelle di sessione e di connessione:

- Sessione TLS: un'associazione tra applicazioni client e server che viene creata dal protocollo Handshake e definisce un insieme di parametri crittografici. Può essere condivisa da più connessioni TLS
- Connessione TLS: è un collegamento transitorio di comunicazione peer-to-peer che viene associato a una sessione SSL



Uno stato di sessione contiene i seguenti dati: identificatore di sessione, certificato del peer (opzionale) cioè il certificato X.509v3, metodo di compressione, specifiche di cifratura (algoritmi di cifratura e di hash e relativi dati) e *Master Secret* ovvero una chiave condivisa di 48 byte (384 bit).

Uno stato di connessione ha: server e client casuali (SR, CR), sei segreti derivati dal *Master Secret* e da SR e CR in cui sia server che client scrivono il MAC secret con delle chiavi di scrittura (e il IV per i cifrari CBC), e il numero di sequenza ($0 - 2^{64}-1$).

Stato della sessione TLS:

La combinazione di algoritmi di scambio di chiavi, hash e crittografia definisce una suite di cifratura (*cipher suite*) per ogni sessione TLS. Quindi all'interno della sessione vengono decisi questi parametri che poi valgono per tutte le connessioni.

Ad esempio: `SSL_DHE_RSA_WITH_DES_CBC_SHA` significa “stabiliamo il segreto principale utilizzando Diffie-Hellman effimero autenticato con RSA; quindi i dati vengono crittografati utilizzando DES in modalità CBC e autenticati con SHA-1”.

Parameter	Description
Session ID	A server-chosen 8-bit number defining a session.
Peer Certificate	A certificate of type X509.v3. This parameter may be empty (null).
Compression Method	The compression method.
Cipher Suite	The agreed-upon cipher suite.
Master Secret	The 48-byte secret.
Is resumable	A yes-no flag that allows new connections in an old session.

Oppure: `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256` significa “stabiliamo il segreto principale usando la curva ellittica Diffie-Hellman autenticata con RSA; quindi i dati vengono crittografati usando la modalità AES CBC con chiavi di 128 bit e autenticati con SHA-256”.

In TLS 1.3, molti algoritmi tradizionali (quelli più deboli) sono stati abbandonati nel tentativo di rendere il protocollo più sicuro.

TLS 1.0-1.2:

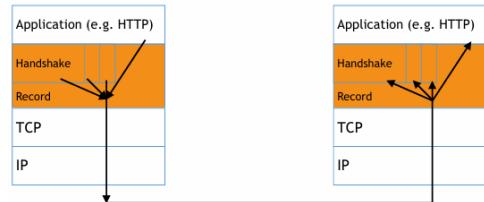
Key exchange/agreement	Authentication	Data ciphers	Message authentication
RSA	RSA	RC4	Hash-based MD5
Diffie-Hellman	DSA	Triple DES	SHA hash function
ECDH	ECDSA	AES	
SRP		IDEA	
PSK		DES	
		Camellia	

Protocollo Record

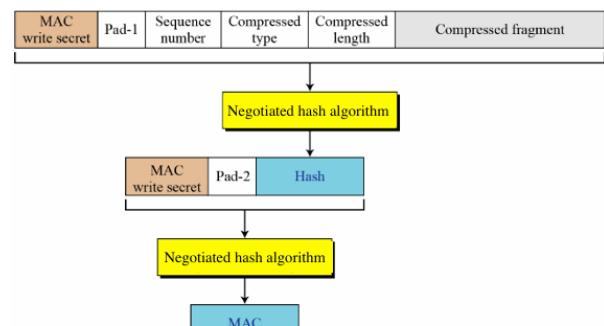
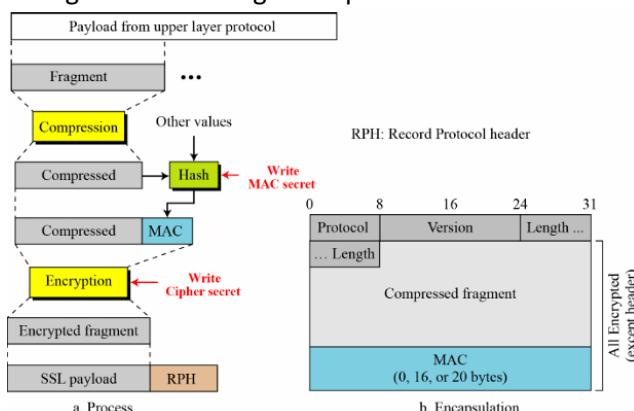
TLS Record implementa le vere trasformazioni di sicurezza, utilizzando le informazioni specifiche della connessione stabilita a partire dalle chiavi di sessione.

Per quanto riguarda l'integrità del messaggio, si utilizza un MAC con chiave segreta condivisa, simile a HMAC, ma con un padding differente e con meccanismi anti-replay.

Invece, per quanto riguarda la riservatezza, si utilizza una crittografia simmetrica con una chiave segreta condivisa definita dal protocollo Handshake (es.: NULL, AES, RC4, DES, 3DES, ecc.), con cui il messaggio viene frammentato (frammenti fino a $2^{14} = 16384$ byte) e compresso prima della crittografia.



Di seguito due immagini del processo di trasformazione (mittente + destinatario) e del calcolo del MAC:



Si noti che il numero di sequenza viene utilizzato nel calcolo del MAC, per evitare attacchi di replay.

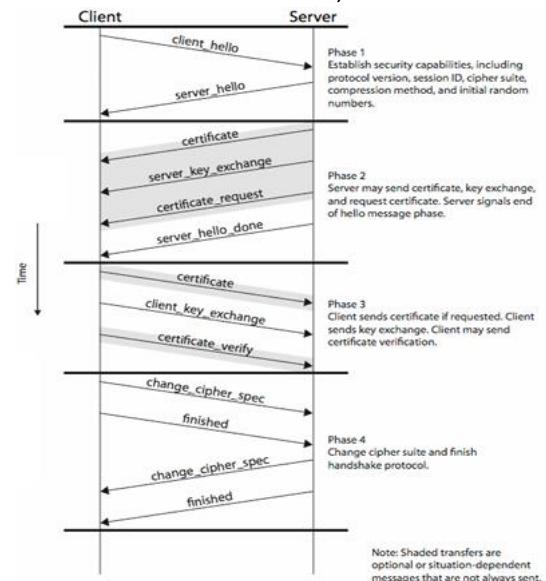
Protocollo Handshake

Questa è la fase in cui si determina il Master Secret e si stabiliscono i dati di sessione. Infatti, è la fase cruciale di tutto il meccanismo di sicurezza e sta al protocollo di Handshake generare un Master Secret per tutta la durata della comunicazione.

Si prefissa anche di essere versatile e di adattarsi a molte situazioni diverse, a seconda di quali sono le informazioni a disposizione del client e del server. Inoltre, consente al server e al client di autenticarsi a vicenda, negoziare algoritmi di crittografia, MAC e le chiavi crittografiche da utilizzare.

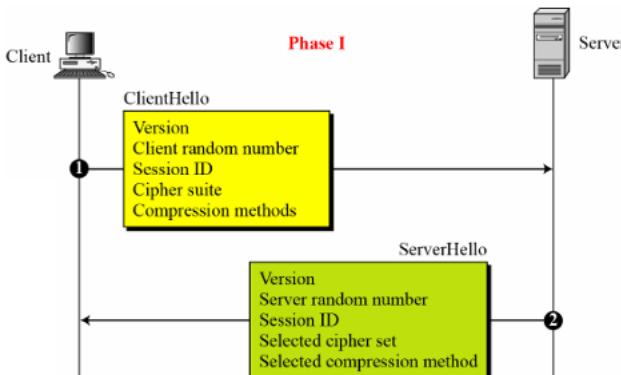
Comprende una serie di messaggi in fasi:

1. Stabilire le capacità di sicurezza
2. Autenticazione del server e scambio di chiavi (es.: per Diffie-Hellman)
3. Autenticazione e scambio di chiavi del client
4. Terminare



Fase 1

Questa fase è proprio l'inizio della connessione, ovvero i primi messaggi di scambio. Il client manda un pacchetto al server dicendogli chi è, che versione di SSL/TLS utilizza, il suo numero random, l'ID di sessione e poi specifica tutte le cipher suite e metodi di compressione che supporta (in ordine di preferenza).



Il server risponde restituendo la sua versione di SSL/TLS, il suo numero random, il numero ID di sessione e poi specifica al client quale algoritmo della cipher suite del client preferisce utilizzare e che supporta (di solito il più sicuro di quelli forniti).

Quindi al termine di questa fase, il client e il server si

conoscono e hanno deciso: la versione di SSL/TLS, gli algoritmi per lo scambio di chiavi, l'autenticazione dei messaggi e la crittografia (in questo punto si decide come si svolgono le fasi 2 e 3), il metodo di compressione e i due numeri casuali per la generazione delle chiavi.

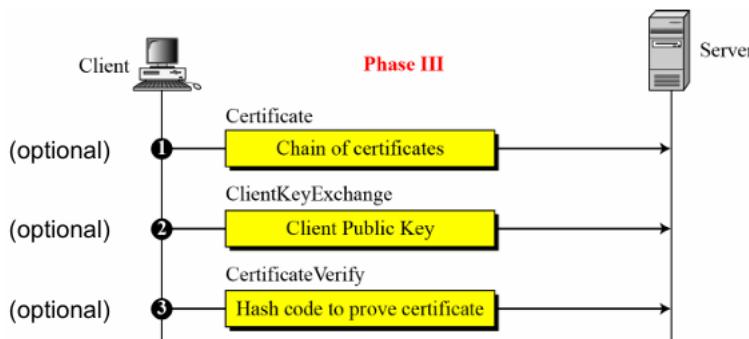
Fase 2

La fase 2 inizia dal server, che in via diversi messaggi a seconda di quale autenticazione è stata eseguita al passaggio precedente (dunque alcuni di questi parametri sono opzionali per la comunicazione).

Al termine di questa fase, il server risulta autenticato dal client e il client conosce la chiave pubblica del server, se necessario.



Fase 3

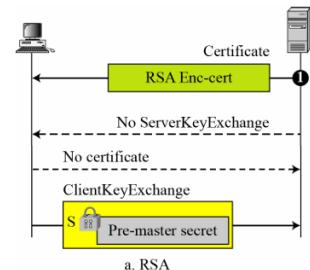


La fase 3 è simile alla precedente, solo che viene svolta dal client. Infatti, è lui che invia alcuni parametri opzionali, che dipendono sempre da quale tipo di autenticazione è stata eseguita al passaggio 1.

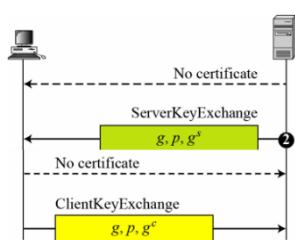
Dopo di questa fase, il client risulta autenticato per il server e sia il client che il server conoscono il segreto pre-master.

Le fasi 2 e 3 si compongono in realtà di scambi di messaggi differenti a seconda di cosa è stato selezionato nella fase 1.

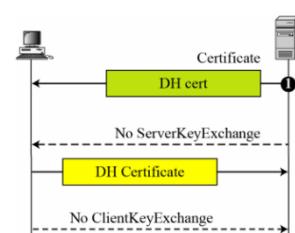
Ad esempio, nel caso della RSA (situazione più diffusa/comune), la chiave viene generata dal client e inviata al server crittografata con la sua chiave pubblica RSA. Il server deve avere un certificato (X.509), ma il client non lo ha. Il server quindi, manda al client il suo certificato, il client lo valida e se lo accetta, estrae la chiave pubblica dal certificato, genera una chiave casuale e la cifra con la chiave pubblica appena estratta e invia tutto quanto al server (pre-master secret). Da notare che solo il server può decifrarlo, visto che è la sua chiave pubblica.



Nel caso di Diffie-Hellman anonimo, invece, si esegue un Diffie-Hellman senza autenticazione. In questo caso il server sceglie g , p e s casuale (per determinare g^s) e invia questi valori al client, il quale sceglie un valore casuale per c (per determinare g^c) che sia compreso in $0 < c < p - 1$ e lo restituisce al server. In questo modo si crea una chiave condivisa con Diffie-Hellman (pre-master secret), senza l'utilizzo di certificati. Tuttavia, per il fatto di essere anonimo e non autenticato, è vulnerabile agli attacchi Man-In-The-Middle. Per questo è ammesso fino a TLS 1.2 e non da TLS 1.3 in poi. Quindi, questa strategia va utilizzata solo in casi di canali sicuri protetti da attacchi attivi.



Nel caso di Diffie-Hellman fisso, si ha che le parti pubbliche fisse sono contenute nei certificati (e firmate dalla CA). quindi, si genera una chiave fissa per ogni coppia di peer, anche tra sessioni diverse. In realtà è come se le chiavi fossero mezze, perché i valori di g , p e g^x sono contenuti nei certificati X.509 per cui non serve fare molto per la connessione, se non scambiarci i certificati e validarli.



Tuttavia, non è tutto rosa e fiori: a ogni comunicazione, dato che i certificati sono quelli, si generano sempre le stesse chiavi (pre-master secret).

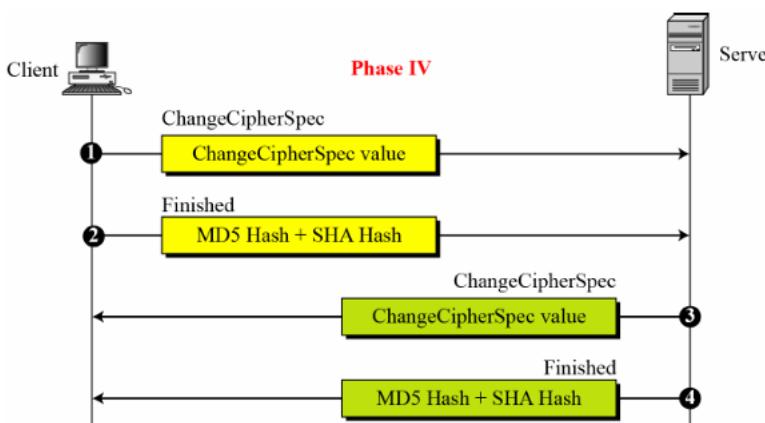
Infine, nel caso di Diffie-Hellman effimero, ovvero è un meccanismo Diffie-Hellman in cui si utilizzano numeri casuali, inviati e firmati con RSA o DSS. Questa è la scelta più robusta e migliore (quando possibile), però richiede che sia il client che il server abbiano un certificato X.509, cosa che non è sempre possibile. Quindi, in questo caso, il server invia il suo certificato al client e poi genera e invia i valori g , p e g^s . Il client valida il certificato e risponde col suo, poi computa g , p e g^c e invia questi valori al server. Questi passaggi sono tutti in chiaro e visibili con Wireshark e hanno una durata effimera (da cui il nome del processo).



Fase 4

Al termine della fase 3, qualunque sia la scelta di autenticazione, si ha costruito una sorta di pre-master secret

più o meno sicuro, si ha autenticato solo il server (caso RSA) o sia server che client (casi D-H fisso e D-H effimero) oppure nessuno dei due (caso D-H anonimo).

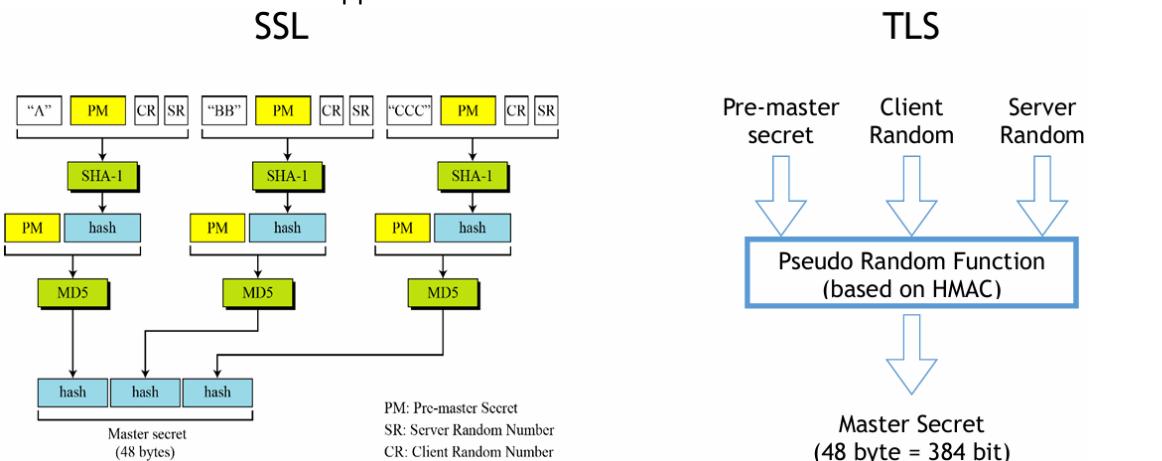


Quindi quando avviene la fase di scelta del cifrario (*Change Cypher Spec*), viene inviato un singolo messaggio, che fa sì che lo stato in sospeso diventi attuale, per cui si aggiorna la suite di cifratura in uso e il numero di sequenza viene riavviato.

Inizialmente tutti i parametri di write e read sono a NULL e poi da quando viene inviato il *Change Cypher Spec* i dati che il client invia saranno cifrati, per cui verranno presi i dati dalla tabella *pending* e inseriti in quella *active* per l'invio/lettura. Il server quando vedrà comparire i dati, inizierà a spostare i dati anche lui da *pending* a *active*, nella colonna read/write opposta a quella dell'azione eseguita dal client.

Per questi passaggi si deve inoltre generare il master secret effettivo, la cipher key e la MAC key, i cui passaggi vengono illustrati in seguito.

Per quanto riguarda il master secret, si utilizza una funzione di generazione che preleva il pre-master secret della fase 3 (chiamato PM) e i numeri random di client e server della fase 1. Questi vengono computati e trasformati in modo da ottenere un hash da 48 byte. Di seguito un'immagine che mostra come SSL codificava questo valore e come viene invece rappresentato tutt'ora con TLS:

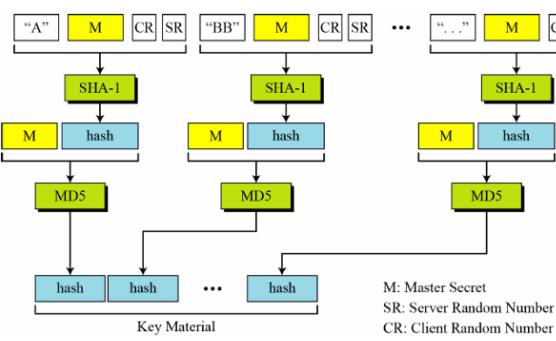


Dopo aver creato una sessione, è possibile stabilire una o più connessioni.

Per ognuna di esse, i segreti e gli IV sono derivati dal Master Secret (dallo stato della sessione) e dai nuovi valori random del Server e del Client, durante l'handshake della connessione.

Quindi per ognuna delle connessioni si creano delle chiavi necessarie per la comunicazione e questo si chiama *key material*. In realtà anche il key material viene generato allo stesso modo del master secret, solo che invece di prendere il pre-master secret e i numeri random iniziali si prende il master secret effettivo e i numeri random iniziali (immagine sotto a sinistra).

Allo stesso modo si può creare i cosiddetti segreti della connessione TLS (immagine sotto a destra)



Parameter	Description
Server and client random numbers	A sequence of bytes chosen by the server and client for each connection.
Server write MAC secret	The outbound server MAC key for message integrity. The server uses it to sign; the client uses it to verify.
Client write MAC secret	The outbound client MAC key for message integrity. The client uses it to sign; the server uses it to verify.
Server write secret	The outbound server encryption key for message integrity.
Client write secret	The outbound client encryption key for message integrity.
Initialization vectors	The block ciphers in CBC mode use initialization vectors (IVs). One initialization vector is defined for each cipher key during the negotiation, which is used for the first block exchange. The final cipher text from a block is used as the IV for the next block.
Sequence numbers	Each party has a sequence number. The sequence number starts from 0 and increments. It must not exceed $2^{64} - 1$.

