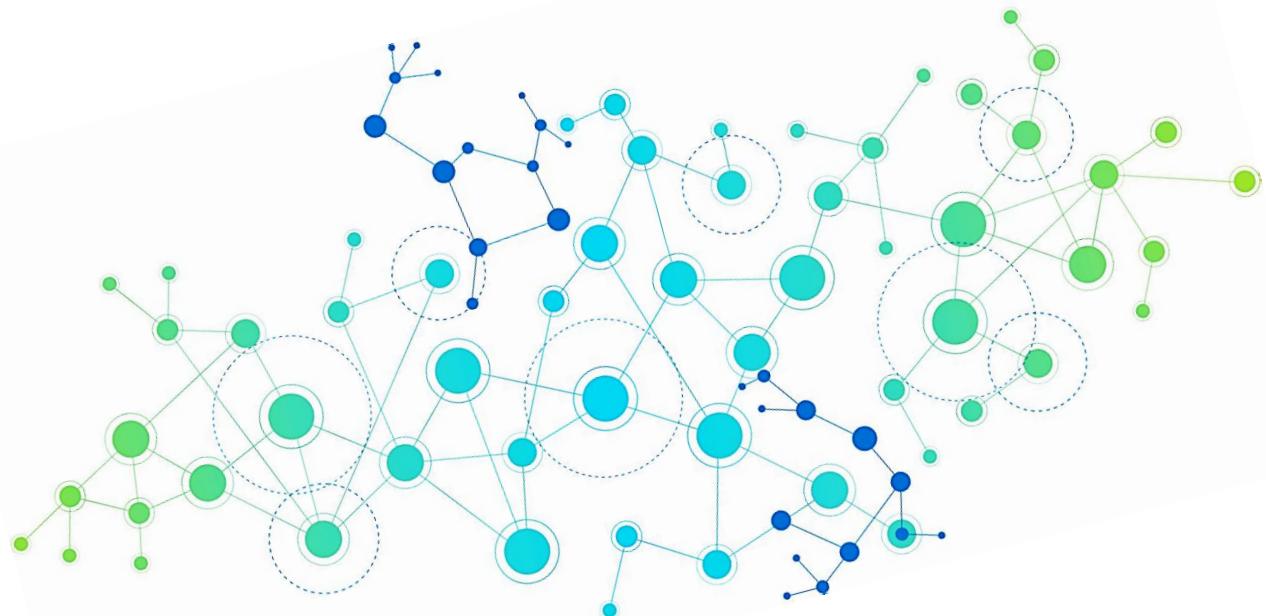


Reti di Calcolatori

Riassunto



Andrea Mansi 2019-2020 – UniUD

Il PDF contiene un riassunto degli argomenti trattati al corso di reti di calcolatori nell'anno accademico 2019-2020

Bibliografia:
Slides Prof. Marino Miculan UniUD
Reti di calcolatori ed.3 Peterson/Davie
(ISBN-13: 978-8838786396)

Indice

Introduzione alle reti	5
Connettività	5
Architettura di rete.....	6
Protocolli di rete	6
Incapsulamento	6
Modello di riferimento ISO OSI (Open System Interconnection).....	7
Modello TCP/IP (architettura Internet).....	9
Switch di livello 1-2-3-4	10
1° livello: Ripetitore	10
2° livello: Bridge	10
3° livello: Router	10
4° livello: Proxy.....	10
Layer fisico	12
Reti a connessione diretta	14
Link (linee di collegamento).....	14
Encoding	14
Framing.....	16
Byte-oriented protocols (BISYNC, DDCMP).....	16
Bit-oriented protocols (HDLC).....	17
Rilevamento degli errori	18
Two-dimensional parity.....	18
Internet checksum algorithm.....	19
CRC (Cyclic Redundancy Check – verifica di ridondanza ciclica)	19
Trasmissione affidabile	21
Stop & wait protocol.....	21
Sliding window protocol.....	22
Ethernet (802.3).....	22
Struttura del frame	23
Indirizzi.....	23
Algoritmo di trasmissione	24
Wireless (802.11).....	27
Frequency hopping spread spectrum.....	27
Direct sequence spread spectrum	27
Tecnologie Wireless	28
Wi-Fi (IEE 802.11).....	28
Collision avoidance	28
Distribution system.....	30
Struttura del frame	31
Bluetooth (802.15.1)	32
Commutazione di pacchetto	34
Switching & forwarding (commutazione e inoltro).....	34
Datagram (connectionless)	35
Virtual Connection (VC).....	36
Source routing (instradamento dalla sorgente)	38

Commutatori per LAN e bridge	38
Internetworking	43
Internetwork.....	43
Semplice interconnessione di reti (Protocollo IP)	43
Modello di servizio IP	43
Consegna di datagrammi	44
Formato del pacchetto (IPv4)	44
Indirizzi globali	47
Inoltro di datagrammi nel protocollo IP	48
Subnetting.....	50
Indirizzamento senza classi (CIDR)	53
Rivisitazione dell'inoltro IP	53
Traduzione degli indirizzi (ARP)	54
Pacchetto ARP	54
Configurazione di host (DHCP – Dynamic Host Configuration Protocol).....	56
Segnalazione di errori (ICMP - Internet Control Message Protocol)	57
Instradamento (Routing).....	58
La rete rappresentata con un grafo.....	58
Vettore di distanza – distance vector routing (RIP).....	59
Stato delle linee – link state routing (OSPF)	62
Differenza tra gli algoritmi a vettore di distanze e a stato delle linee	65
NAT (Network address translation).....	67
IPv6	68
Advanced internetworking	69
Interdomain routing (BGP).....	69
eBGP e iBGP	71
Internet Multicast.....	72
Multicast in IP	72
Multicast a vettore di distanza (DVMRP)	72
Multicast routing (PIM)	74
Protocolli end-to-end	75
UDP (User Datagram Protocol) – semplice demultiplexing	75
TCP (Transport Control Protocol) – flusso affidabile di byte	77
Problemi relativi alle connessioni end-to-end	77
Formato del segmento	78
Instaurazione e terminazione di una connessione	79
Diagramma delle transizioni di stato.....	80
Algoritmo sliding window rivisitato	80
Consegna affidabile e in sequenza	80
Controllo di flusso	81
Controllo della congestione e allocazione delle risorse	85
Problemi nell'allocazione delle risorse	85
Modello della rete	85
Gerarchia.....	86
Allocazione di risorse efficace	86
Allocazione di risorse equa	87

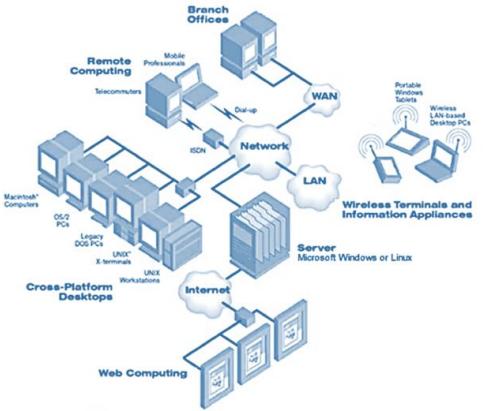
Politiche di gestione delle code	88
FIFO (First-In, First-Out) con taildrop	88
FQ (Fair Queueing).....	88
Controllo di congestione in TCP	91
Aumento additivo/diminuzione moltiplicativa (AIMD).....	91
Partenza lenta	92
Ritrasmissione veloce	92
Recupero veloce	93
Meccanismi di prevenzione della congestione	93
Randomly Early Detection (RED)	93
Impedire la congestione alla sorgente	93
Network Security.....	94
Obiettivi della sicurezza	94
Tipologie di attacco	94
Servizi di sicurezza	96
Modelli per la network security	97
Implementazione dei servizi di sicurezza.....	97
Cifratura	101
Cifratura simmetrica.....	101
Crittoanalisi	102
Sicurezza incondizionata e sicurezza computazionale	104
Cifrario di Vigenere e One-Time Pad.....	104
Cifrari a flusso.....	105
Cifrari a blocco	107
Distribuzione della chiave simmetrica	118
Servizio di integrità e autenticazione di messaggi.....	119
Meccanismi di autenticazione	119
Autenticazione di messaggi tramite crittografia simmetrica	120
Cifratura Asimmetrica	124
Caratteristiche dei sistemi a chiave pubblica	125
RSA.....	125
Firma digitale	127
E-mail Security	132
Autenticazione delle entità e distribuzione delle chiavi	134
Password	134
Protocolli di autenticazione	135
Gestione delle chiavi con codifica a chiave pubblica	140
Procedure di autenticazione in X.509	141
Sicurezza al livello di rete e al livello di trasporto	143
Sicurezza implementata a livello di trasporto: Secure Socket Layer.....	143
Architettura SSL	143
Sicurezza implementata a livello di rete	151
Virtual Private Networks (VPN)	159
Esercizi vari	160

Introduzione alle reti

Una generica rete è composta dall'unione delle seguenti componenti chiave:

- una serie di **nodi indipendenti** tra loro;
- la presenza di **scambio di informazioni**;
- una serie di **canali di comunicazione**.

Con rete si intende quindi: un insieme di nodi indipendenti, interconnessi da canali di comunicazione tramite i quali avviene uno scambio di dati e messaggi. Una rete viene tipicamente instaurata per portare a termine un compito che non può essere eseguito da un singolo nodo. Una rete di calcolatori è una tipologia di rete caratterizzata dal fatto che i singoli nodi sono dispositivi hardware indipendenti che comunicano tra loro mediante opportuni software, tramite dei canali di comunicazione.



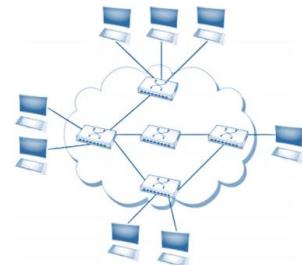
Le reti di calcolatori mirano a soddisfare le seguenti principali caratteristiche:

- essere **general purpose**:
 - non sono ottimizzate e/o specializzate per un'unica applicazione;
 - permettono a più applicazioni di coesistere su di essa.
- avere elevata **flessibilità**:
 - nuove features devono poter essere aggiunte con facilità;
 - devono essere **"vendor-independent"**, ovvero seguire una serie di standardizzazioni (ad esempio i protocolli).

L'Internet è un esempio di rete di calcolatori che soddisfa queste due caratteristiche. Può essere utilizzato per più applicazioni (www, e-mail, streaming, file sharing etc.) e al tempo stesso permette una certa flessibilità (nuove features possono essere aggiunte con il passare del tempo), inoltre, presenta dei protocolli e delle standardizzazioni che ne definiscono il funzionamento.

definizione ricorsiva di rete: una rete di calcolatori può essere definita come un'entità formata da:

- due o più nodi collegati tra di loro (in point-to-point o in accesso multiplo);
- due o più reti collegate tra di loro da un nodo.



Connettività

Segue una serie di nozioni e di termini basilari necessari al fine di comprendere al meglio la materia, tutti legati al concetto di connettività. Essi verranno approfonditi nei successivi capitoli.

Network link: con link, si intende il componente fisico con il quale si collegano due o più nodi di una rete.



Connessione point-to point o ad accesso multiplo: esistono due tipologie di connessioni tra due dispositivi:

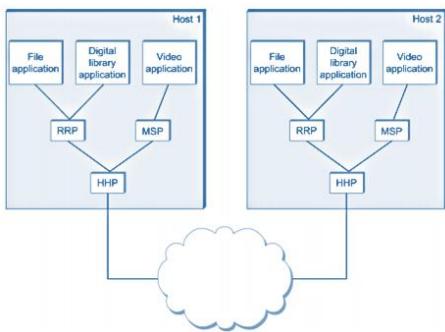
- a) Point to point
- b) Multiple access

Switch: uno switch è un dispositivo che si occupa di collegare tra di loro diversi dispositivi.

Architettura di rete

Possiamo descrivere un'architettura di rete come un insieme di linee guida relative alla progettazione e realizzazione di una rete. Le reti sono molto complesse, per diminuire la difficoltà della loro gestione e progettazione si organizzano a **layer** (strati), costruiti uno sull'altro. Lo scopo di ogni strato è quello di offrire determinati servizi agli strati di livello superiore, schermandoli dai dettagli sull'implementazione dei servizi offerti. Questo concetto familiare è impiegato in tutta l'informatica, dove viene tipicamente definito come **information hiding**, tipi dati astratti, encapsulazione dei dati e programmazione orientata agli oggetti. L'idea di base è che una particolare porzione di software (o hardware) offre un servizio ai propri utenti nascondendogli però i dettagli dello stato interno e gli algoritmi utilizzati.

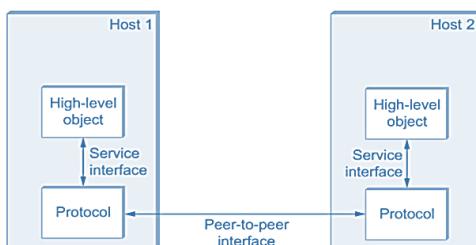
Application programs
Process-to-process channels
Host-to-host connectivity
Hardware



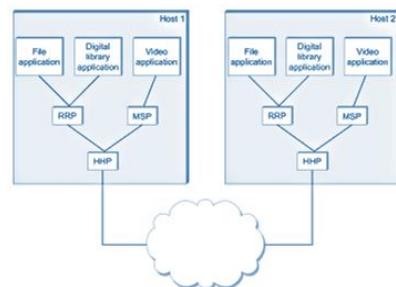
Protocolli di rete

Le regole e le convenzioni usate per far comunicare un layer con un altro, sono globalmente noti come protocolli. Un protocollo è un insieme di regole utilizzate tra le due parti (layer/strati) che comunicano, relative al modo in cui la comunicazione deve procedere. Tra ciascuna coppia di layer contigui si trova un'interfaccia che definisce le operazioni elementari e i servizi che il layer inferiore rende disponibile a quello sovrastante. Più nello specifico, ciascun protocollo di rete prevede due interfacce: la prima, detta **interfaccia di servizio**, che fornisce funzionalità al livello superiore che ne vuole fare uso; la seconda è una **interfaccia peer-to-peer** che definisce la forma e il significato dei messaggi scambiati con un'altra macchina che fa uso dello stesso protocollo. Un elenco di protocolli usati da uno specifico dispositivo è detto **protocol stack**.

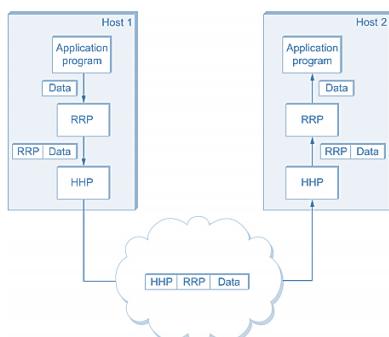
funzionalità al livello superiore che ne vuole fare uso; la seconda è una **interfaccia peer-to-peer** che definisce la forma e il significato dei messaggi scambiati con un'altra macchina che fa uso dello stesso protocollo. Un elenco di protocolli usati da uno specifico dispositivo è detto **protocol stack**.



Interfacce di servizio e peer to peer



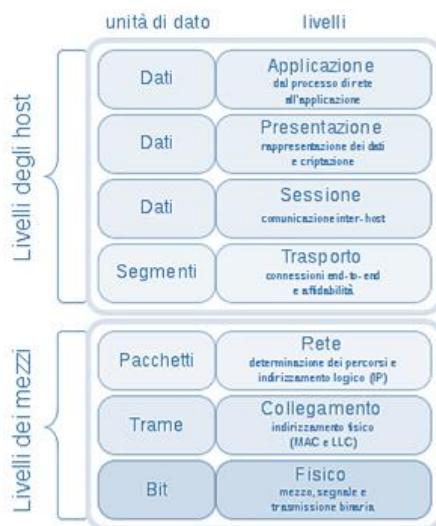
Grafo dei protocolli



Incapsulamento

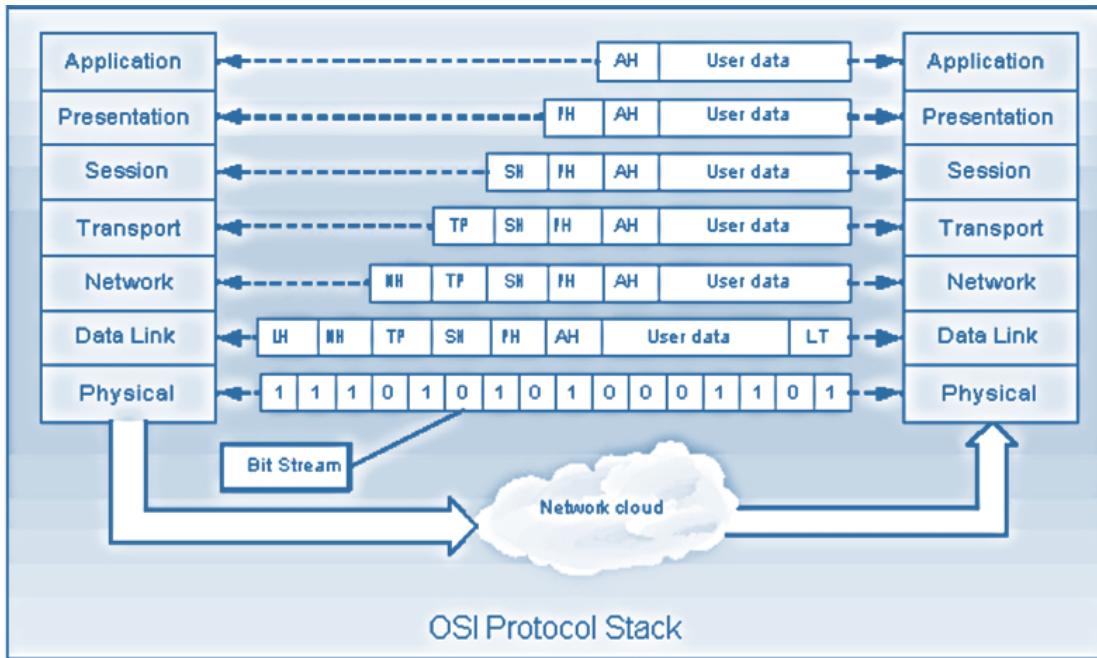
L'incapsulamento (o imbustamento), nelle reti di calcolatori, è un termine che indica l'operazione, spesso ripetuta più volte, di inserimento di un payload di un certo livello n dello strato architettonico tra dati di controllo (intestazioni o header) di livello n-1. Il risultato è un pacchetto di livello n-1, che diventa quindi il carico utile per il protocollo di rete di livello n-2, e così via.

Modello di riferimento ISO OSI (Open System Interconnection)



Il modello OSI è uno standard per le funzioni di comunicazione nelle telecomunicazioni o di sistemi informatici, stabilito nel 1984 dall'International Organization for Standardization (ISO). Segue la stratificazione di riferimento proposta dallo standard con relativa descrizione delle funzionalità dei vari layer:

- 1. Layer fisico:** lo strato fisico si occupa della trasmissione di bit grezzi sul canale di comunicazione.
- 2. Layer di collegamento (data link):** permette il trasferimento affidabile di dati attraverso il livello fisico. Invia frame di dati con la necessaria sincronizzazione ed effettua un controllo degli errori e delle perdite di segnale. Tutto ciò consente di far apparire, al livello superiore, il mezzo fisico come una linea di trasmissione esente da errori di trasmissione. Questo livello si occupa in primis di formare i dati da inviare attraverso il livello fisico, incapsulando il pacchetto proveniente dallo strato superiore in un nuovo pacchetto provvisto di un nuovo header (intestazione) e tail (coda); usati anche per sequenze di controllo. Questa frammentazione dei dati in specifici pacchetti è detta framing e i singoli pacchetti sono detti frame. Come controllo degli errori, per ogni pacchetto ricevuto, il destinatario invia al mittente un pacchetto ACK (acknowledgement) contenente lo stato della trasmissione: il mittente deve ripetere l'invio dei pacchetti mal trasmessi e di quelli che non hanno ricevuto riscontro/risposta.
- 3. Layer di rete:** rende i livelli superiori indipendenti dai meccanismi e dalle tecnologie di trasmissione usate per la connessione; inoltre, si prende carico della consegna a destinazione dei pacchetti. Questo layer è responsabile di effettuare il routing (scelta ottimale del percorso di rete da utilizzare per garantire la consegna delle informazioni dal mittente al destinatario, scelta svolta dal router attraverso dei particolari algoritmi di routing e tabelle di routing.). Traduce gli indirizzi. La sua unità di dati è il pacchetto.
- 4. Layer di trasporto:** la funzione essenziale dello strato trasporto è quella di accettare dati dallo strato superiore, dividerli in unità più piccole quando necessario, passarle allo strato network e assicurarsi che tutti i pezzi arrivino correttamente all'altra estremità. L'unità di dato atomico è il messaggio.
- 5. Layer di sessione:** si occupa di controllare la comunicazione tra applicazioni. Instaurare, mantenere ed abbattere connessioni (sessioni) tra applicazioni cooperanti. Si occupa anche della sincronia di invio/ricezione messaggi. Si occupa anche di inserire dei punti di controllo nel flusso dati: in caso di errori nell'invio dei pacchetti, la comunicazione riprende dall'ultimo punto di controllo andato a buon fine.
- 6. Layer di presentazione:** l'obiettivo di questo layer è quello di trasformare i dati forniti dalle applicazioni in un formato standardizzato e offrire servizi di comunicazione comuni, come la crittografia, la compressione del testo e la riformattazione.
- 7. Layer dell'applicazione:** lo scopo di base è interfacciare utente e macchina. Fornisce un insieme di protocolli che operano a stretto contatto con le applicazioni.



Riassunto OSI Protocol Stack:

I tre livelli più bassi dello stack (layer 1, 2 e 3) sono implementati nei nodi della rete.

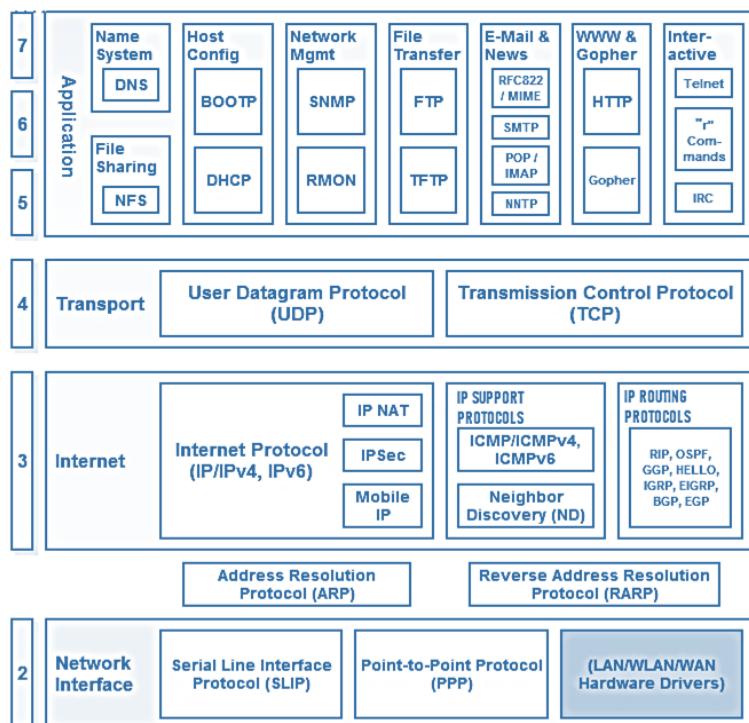
1. Layer **fisico**: gestisce la trasmissione grezza di bit attraverso un canale di comunicazione. Qui vengono affrontati i problemi di encoding e decoding.
2. Layer **datalink (collegamento)**: colleziona un flusso di bit in un blocco aggregato detto frame. Viene eseguito un controllo degli errori.
3. Layer di **rete (network)**: gestisce il routing attraverso i vari nodi. L'unità di dati scambiata in questo livello sono i pacchetti.
4. Layer di **trasporto**: crea dei collegamenti host to host. Si occupa di stabilire, mantenere e terminare una connessione tra host. I dati ricevuti dai livelli superiori vengono suddivisi in segmenti/frammenti (datagram).
5. Layer di **sessione**: controlla la comunicazione tra applicazioni. Consente di inserire servizi più avanzati a quelli già forniti dal livello di trasporto come la sincronizzazione.
6. Layer di **presentazione**: trasforma i dati forniti dalle applicazioni in un formato standardizzato e offre servizi di comunicazione comuni.
7. Layer dell'**applicazione**: fornisce un insieme di protocolli che operano a stretto contatto con le applicazioni.

Modello TCP/IP (architettura Internet)

La cosiddetta Suite di Protocolli Internet (Internet Protocol Suite), meglio nota come architettura TCP/IP, è un insieme di protocolli di rete su cui si basa il funzionamento della rete Internet. Così come il modello ISO/OSI, anche l'architettura TCP/IP si basa su una suddivisione in livelli dei protocolli di rete. In questo caso i livelli sono solo quattro e sono mostrati in figura.

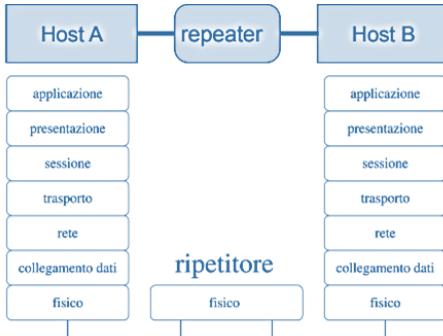
- Livello di accesso alla rete (Network Access Layer – Link Layer):** sotto il livello Internet, il modello di riferimento TCP/IP specifica solo che ci deve essere un livello di accesso alla rete in grado di spedire i pacchetti IP. Nel modello ISO/OSI questo strato corrisponde ai primi due livelli: il livello di collegamento e il livello fisico. In realtà i protocolli di questo livello non sono specificati nell'architettura TCP/IP: cioè essa non si occupa di fornire esplicite indicazioni sui protocolli del livello più basso, ma usa a tale scopo i protocolli sviluppati per altre architetture di rete.
- Livello di Internet (di rete - Internet layer):** offre un servizio al livello delle applicazioni avvalendosi dei servizi del sottostante livello di rete. Si occupa dell'instradamento dei pacchetti nella rete e dell'interconnessione fra reti diverse, senza effettuare controlli su eventuali errori di trasmissione. Può suddividere le dimensioni del messaggio in pacchetti più piccoli, per adattarli alle specifiche della rete. A ciascun nodo, durante questa fase, viene assegnato un indirizzo IP che lo identificherà in modo non ambiguo nella rete. Le funzionalità di instradamento consentono di selezionare il percorso migliore per veicolare un messaggio verso un dato nodo destinatario, noto che sia il suo indirizzo IP.
- Livello di trasporto (Transport Layer):** gestisce la comunicazione fra un host e un altro, fornendo un canale logico di comunicazione per l'applicazione. Può presentare funzioni di verifica e di correzione errori. Può anche suddividere il messaggio in parti più piccole (segmentazione).
- Livello di applicazione (Application Layer):** il primo livello è quello dell'applicazione: esso rappresenta l'interfaccia con l'utente ed abilita, ad esempio, la consultazione di pagine web, stabilendo e gestendo le sessioni di lavoro dei processi client tra il nostro browser ed il server web. Realizza i servizi applicativi per Internet, quali la posta elettronica, la navigazione sul web, il trasferimento di file etc. etc.

tabella rappresentante i principali protocolli della suite TCP/IP, la loro distribuzione sui diversi livelli e le loro reciproche relazioni.



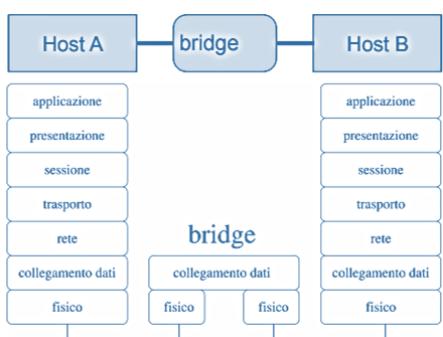
Switch di livello 1-2-3-4

Segue una breve introduzione sugli switch di 1°, 2°, 3° e 4° livello



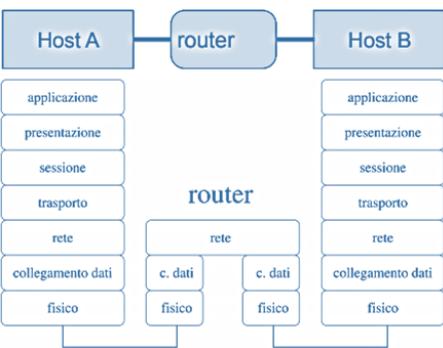
1° livello: Ripetitore

Uno switch di primo livello viene chiamato ripetitore: è un semplice dispositivo di rete che non gestisce il traffico che lo attraversa. Qualsiasi pacchetto che entra in una porta viene allagato o "ripetuto" su ogni altra porta, ad eccezione di quella di entrata. Uno switch di rete crea virtualmente la connessione end-to-end del livello 1. I bit in input vengono decodificati e ricodificati dall'altro lato



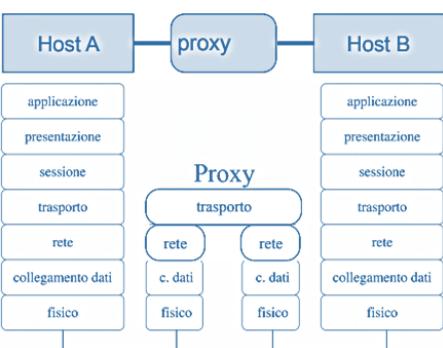
2° livello: Bridge

Il bridge (letteralmente ponte) è un dispositivo di rete che si colloca al livello datalink del modello ISO/OSI e che traduce da un mezzo fisico a un altro all'interno di una stessa rete locale. Esso è quindi in grado di riconoscere, nei segnali elettrici che riceve dal mezzo trasmissivo, dei dati organizzati in strutture a pacchetto detti frame, di individuare all'interno di esse l'indirizzo del nodo mittente e quello del nodo destinatario e in base a questi operare un indirizzamento dei frame tra più segmenti di rete a esso interconnessi.



3° livello: Router

È un dispositivo di rete che, in una rete informatica a commutazione di pacchetto, si occupa di instradare i dati, suddivisi in pacchetti, fra sottoreti diverse. È quindi, a livello logico, un nodo interno di rete deputato alla commutazione di livello tre del modello OSI. L'instradamento può avvenire verso sottoreti direttamente connesse, su interfacce fisiche distinte, oppure verso altre sottoreti non limitrofe che, grazie alle informazioni contenute nelle tabelle di instradamento (di routing), siano raggiungibili attraverso altri nodi della rete.



4° livello: Proxy

Uno switch di quarto livello traduce messaggi tra due diversi host. Lavora nel livello di trasporto, ovvero il quarto livello del modello ISO OSI.

ES: Per ciascuna delle seguenti applicazioni di rete, si dica se è più sensibile all'**ampiezza di banda**, alla **latenza** (delay di trasmissione), o alla variazione della latenza (**jitter**):

- (a) accesso terminale remoto;
- (b) invio mail;
- (c) streaming audio;
- (d) videoconferenza.

- Risposta:

- (a): latenza;
- (b): banda;
- (c): jitter;
- (d): latenza e banda;

Layer fisico

Segue un riassunto delle principali nozioni relative alla parte di programma di fisica:

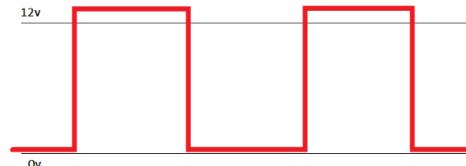
Baud-rate e Bit-rate:

La velocità con cui i simboli di una codifica di base n vengono inviati viene indicata con l'unità **baud** (simboli al secondo). Non corrisponde alla velocità con cui viene inviata l'informazione (in termini di bit/s); questo perché ad ogni simbolo corrisponde potenzialmente a una quantità di bit diversa da 1.

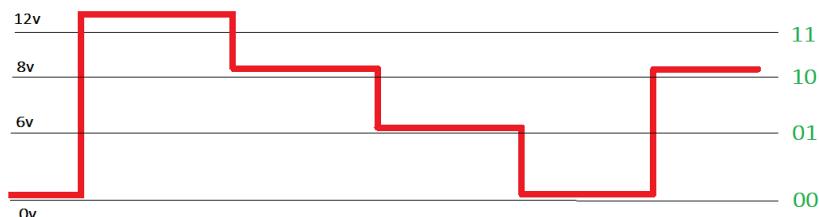
In un segnale digitale di base n (ovvero n stati/simboli diversi distinguibili possibili – anche detti intervalli di ampiezza, tipicamente 2^M) possiamo codificare più bit per ciascun simbolo:

$\log_2(n)$ = bit codificabili da 1 simbolo, oppure “ n ° bit necessari a distinguere un valore tra n possibili”

Ad esempio, in un segnale digitale che può assumere solo lo stato on/off, avremmo due simboli distinguibili, di conseguenza 1 baud (simbolo) codifica 1 bit. Si veda l'immagine a destra: se con il segnale >12v codifichiamo l'1 e con il segnale prossimo a 0v codifichiamo lo 0, abbiamo che con 1 simbolo (che può assumere 2 valori distinti; quindi $n=2$) possiamo codificare 1 bit; infatti $\log_2(2) = 1$.



Un altro esempio: si consideri invece il caso nell'immagine sotto; se possiamo distinguere 4 stati distinti ($n=4$) abbiamo che con 1 stato possiamo codificare 2 bit; ad esempio come nell'immagine assegnamo ai 12v il valore 11, agli 8v il valore 10, etc. Infatti, $\log_2(4) = 2$. Se ipotizziamo che i due esempi abbiano lo stesso baud-rate; ovvero lo stesso numero di simboli per secondo; nel secondo caso, codificando 2 bit invece che 1 con uno stato; avremmo un bit rate doppio. Vediamo ora la formula per calcolare il bitrate (grezzo) nel caso generico.



$$\text{bitrate (grezzo)} = \text{baudrate} * \log_2(n)$$

analisi unità di misura: [bit/s] = [simbolo/s] * [bit/simbolo]
nella moltiplicazione simbolo e simbolo si semplificano, il risultato è quindi bit/sec

Bitrate netto e grezzo e concetto di code-rate:

Il coderate indica il rapporto tra dati utili e dati totali (ovvero utili + overhead di ulteriori codifiche, ad esempio per la correzione degli errori).

coderate: bit con informazione/bit totali

Ad esempio, la codifica Manchester ha un code-rate di 0.5; quella 4B/5B di 4/5.

Il bitrate netto è quindi quello grezzo (che passa nel canale fisico) per il code-rate; ovvero il bitrate dei dati utili.

$$\text{bitrate (netto)} = \text{coderate} * \text{bitrate (grezzo)} = \text{coderate} * \text{baudrate} * \log_2(n)$$

Rumore sul canale: una misura della rumorosità di un canale è il rapporto segnale rumore SNR, definito come il rapporto tra la potenza associata al segnale e quella al rumore.

$$\text{SNR} = P_{\text{sign}} / P_{\text{rumore}}$$

$$\text{SNR (in dB)} = 10 \cdot \log_{10}(\text{SNR})$$

deciBel: l'entità (significatività) della potenza trasmessa da un'onda può essere valutata confrontandola in ordine di grandezza con quella di un fenomeno di riferimento. Detta P_0 la potenza del fenomeno di riferimento, si dirà che un segnale avrà una potenza espressa in deciBel:

$$\text{dB} = 10 * \log_{10}(P/P_0)$$

Teorema di Nyquist [sulla max freq.]: a Nyquist è legato non solo il nome del teorema di campionamento ma anche quello di un teorema sulla massima frequenza f_{pulse} con cui possono essere inviati dei segnali digitali su un canale privo di rumore, con la larghezza di banda B :

$$f_{\text{pulse}} < 2B \quad (\text{quindi } f_{\text{pulse}} = \text{baudrate})$$

Teorema di Shannon Hartley: Quindi, la massima velocità di trasmissione R si ottiene nel seguente modo, dove M = numero di simboli distinguibili (ovvero lo scorso n).

$$R = f_{\text{pulse}} * \log_2(M) \leq 2B * \log_2(M)$$

ovvero:

$$R = \text{baudrate} * \log_2(n) \leq 2B * \log_2(n)$$

I seguenti risultati sono in riferimento ad assenza di rumore sul canale fisico.
La capacità C di un canale, indica la velocità limite di trasmissione senza errori.
Dove B è la banda (baudrate/2 perché per Nyquist $f_{\text{pulse}} < 2B$).

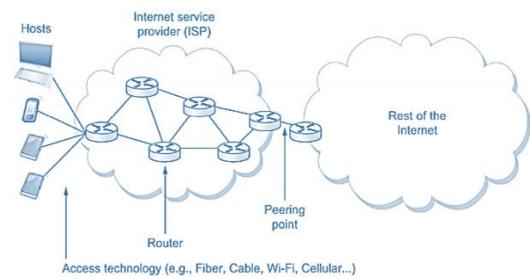
$$R < C = B * \log_2(1 + \text{SNR})$$

Reti a connessione diretta

Connettere fisicamente dei calcolatori (host) mediante un mezzo fisico (cavo, fibra ottica etc.) non è complesso, tuttavia, prima che possano scambiarsi dei messaggi bisogna risolvere cinque problemi fondamentali:

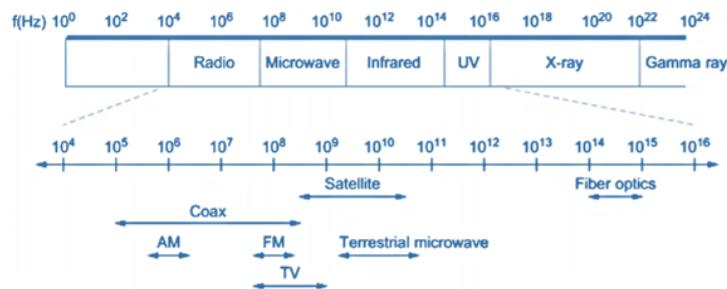
- **Encoding:** codifica dei bit sul mezzo di comunicazione in modo che possano essere compresi dall'host ricevente.
- **Framing:** delineare la sequenza di bit trasmessi lungo il collegamento fisico, in modo tale che possano rappresentare un messaggio completo.
- **Error detection:** rilevare errori e intraprendere azioni appropriate.
- **Consegna affidabile:** far apparire affidabile un collegamento nonostante il fatto che di tanto in tanto alcuni frame risultino persi o corrotti.
- **Media access control:** mediare l'accesso al collegamento fisico nel caso sia condiviso da più host.

È importante riprendere il concetto di nodo e di link. Un nodo è un qualsiasi dispositivo hardware del sistema in grado di comunicare con gli altri dispositivi che fanno parte della rete; può quindi essere un computer, una stampante, un fax, un modem, uno switch etc. In ogni caso il nodo deve essere dotato di una scheda di rete. Un link è una linea di collegamento, utilizzata appunto, per collegare più host tra di loro.



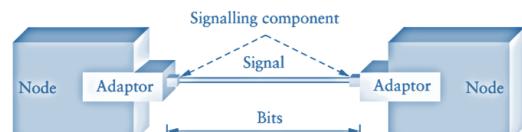
Link (linee di collegamento)

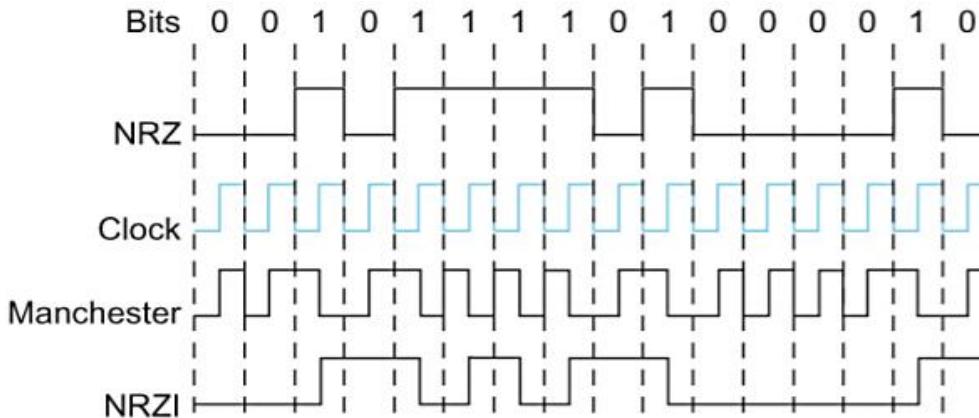
I collegamenti di rete (link) sono implementati su una varietà di supporti fisici diversi, tra cui il doppino, cavo, fibra ottica e lo spazio. Qualunque sia il mezzo fisico, viene utilizzato per propagare segnali. Questi segnali sono in realtà onde elettromagnetiche che viaggiano alla velocità della luce. Una proprietà importante di un'onda elettromagnetica è la frequenza, misurata in hertz [Hz]; velocità con cui oscilla l'onda. La distanza tra una coppia di massimi adiacenti o minimi di un'onda, in genere misurati in metri, è chiamata lunghezza d'onda dell'onda. Tali collegamenti forniscono le basi per la trasmissione di tutti i tipi di informazioni, incluso il tipo di dati che siamo interessati a trasmettere. Diciamo che i dati binari sono codificati nel segnale.



Encoding

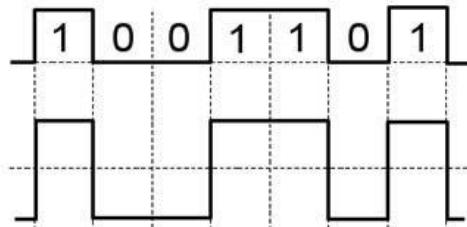
Il primo passo per trasformare nodi e collegamenti in blocchi utilizzabili è capire come collegarli in modo tale che i bit possano essere trasmessi da un nodo all'altro è la codifica. Come menzionato nella sezione precedente, i segnali si propagano su collegamenti fisici. L'attività da svolgere, pertanto, è quella di codificare i dati binari che il nodo di origine desidera inviare nei segnali che i collegamenti sono in grado di trasportare, e quindi, poi decodificare il segnale nei corrispondenti dati binari nel nodo ricevente. Ignoriamo i dettagli di modulazione e supponiamo che stiamo lavorando con due segnali discreti: alto e basso. Segue una lista delle principali tecniche di codifica.





NRZ (Non Return to Zero):

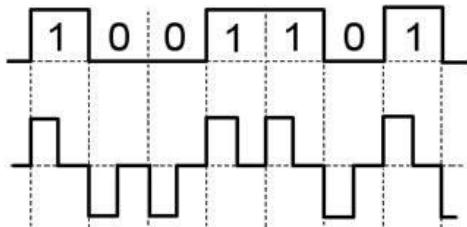
- Lo stato digitale “1” è rappresentato con un segnale alto.
- Lo stato digitale “0” è rappresentato con un segnale basso.



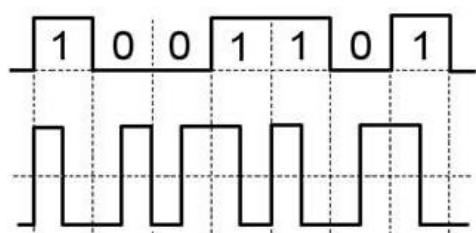
Questo metodo è facilmente ottenibile e non richiede circuiti complicati anche perché non si tratta di una vera e propria codifica, visto che i dati vengono passati direttamente come tali in uscita. Si ha inoltre una alta robustezza agli errori, anche se lunghe stringhe di “0” o di “1” potrebbero causare la perdita del sincronismo.

RZ (Return to Zero):

- Lo stato digitale “1” è rappresentato con un segnale alto.
- Lo stato digitale “0” è rappresentato con un segnale basso.



Ad ogni semiperiodo il segnale torna sempre a zero. Come nel metodo precedente, non si ha una vera e propria codifica dei dati. Il ricevitore deve però distinguere tra 3 livelli, anziché tra 2; quindi la probabilità di errore è più grande rispetto a quella che si ha nell’NRZ. Il vantaggio è che lunghe stringhe di “0” o di “1” non causano la perdita del sincronismo.



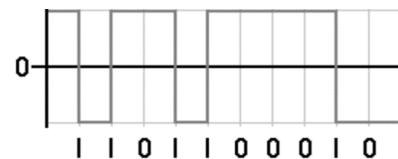
Manchester:

- Lo stato digitale “1” è rappresentato con una transizione del semiperiodo dal segnale alto al segnale basso.
- Lo stato digitale “0” è rappresentato con una transizione del semiperiodo dal segnale basso al segnale alto.

Come nell’RZ, in questo metodo lunghe stringhe di “0” o “1”

non causano la perdita del sincronismo. Inoltre, lavorando con solo due livelli, viene garantita un’alta robustezza agli errori. La codifica Manchester richiede un circuito più complicato rispetto a quelli per l’RZ e l’NRZ.

NRZI: (Non Return to Zero Inverted): risolve il problema di 1 consecutivi, ma non quello degli 0 consecutivi. L’uno viene rappresentato dalla transizione dalla fase di risalita/discesa del clock.



4B/5B: la codifica finale che consideriamo, chiamata 4B / 5B, tenta di affrontare l'inefficienza della codifica Manchester senza soffrire del problema di avere durate estese di segnali alti o bassi. L'idea della codifica 4B/5B è quella di inserire bit extra nel flusso di bit in modo da suddividere lunghe sequenze di 0 o 1. In particolare, ogni sequenza di 4 bit di dati effettivi viene codificata in un codice a 5 bit che viene quindi trasmesso al ricevitore; da qui il nome 4B / 5B. I codici a 5 bit sono selezionati in modo tale che ciascuno non abbia mai più di uno 0 iniziale e non più di due 0 finali. In questo modo, tre 0 consecutivi non possono mai apparire, anche tra due diversi codici a 5 bit. In figura un esempio di mappatura di 5 valori codificati in 4B/5B.

Nome	4B	5B
0	0000	11110
1	0001	01001
2	0010	10100
3	0011	10101
4	0100	01010

Framing

Ora che abbiamo visto come trasmettere una sequenza di bit su un collegamento punto-punto, da un adattatore all'altro, concentriamoci sulle reti a commutazione di pacchetto, il che significa che blocchi di dati (chiamati frame), non flussi di bit, vengono scambiati tra i nodi. È la scheda di rete che consente ai nodi di scambiare i frame. Quando il nodo A desidera trasmettere un frame al nodo B, comunica alla sua scheda di rete di trasmettere un frame dalla propria memoria. Ciò comporta l'invio di una sequenza di bit sul collegamento fisico che collega A e B. L'adattatore di rete del nodo B raccoglie quindi la sequenza di bit in arrivo sul collegamento e deposita il frame corrispondente nella memoria di B. Riconoscendo esattamente quale set di bit costituisce l'unità logica detta frame, ovvero determinare dove inizia e finisce il frame; questa è la sfida centrale affrontata dalla scheda di rete.

Esistono diversi metodi per affrontare il problema del framing. Segue una lista dei principali protocolli.

- Protocolli **Byte-oriented**;
- Protocolli **Bit-oriented**.

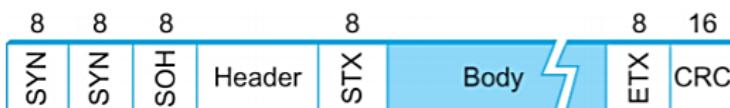
Byte-oriented protocols (BISYNC, DDCMP)

Uno dei più datati approcci al framing consiste nel visualizzare ogni frame come una raccolta di byte (caratteri) anziché una raccolta di bit. Segue una lista dei principali protocolli byte-oriented:

BISYNC (Binary Synchronous Communications) – sentinel approach:

I frame vengono trasmessi a partire dal campo più a sinistra.

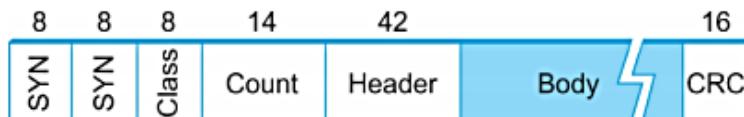
- L'inizio di un frame è indicato dall'invio di due **SYN** (carattere speciale di sincronizzazione);
- La parte dei dati del frame è contenuta tra le sentinelle speciali, denotate dal carattere **STX** (inizio del testo) ed dal carattere **ETX** (fine del testo); inoltre:
 - **SOH**: carattere indicante l'inizio dell'intestazione (header);
 - **DLE**: Data Link Escape ("escape"): Body vs. ETX;
 - **CRC**: controllo di ridondanza ciclico.



Un problema legato all'approccio con uso di sentinella è che se il carattere ETX è contenuto nei dati (porzione body) potrebbe venir inteso (erroneamente) come ETX indicante la fine dei dati stessi. Per ovviare a questo problema si utilizza il carattere DLE di "escaping" che può a sua volta esser preceduto da un altro DLE. Questa tecnica di "escaping" viene a volte chiamata "**character stuffing**". Il frame include anche un campo con etichetta CRC (cycle redundancy check - controllo di ridondanza ciclico) che viene utilizzato per rilevare errori di trasmissione; sono diversi gli algoritmi per il rilevamento degli errori che verranno presentati nella sezione successiva. Infine, il frame contiene ulteriori campi di intestazione che sono utilizzati, tra l'altro, per l'algoritmo di consegna a livello datalink.

DDCMP (Digital Data Communications Message Protocol) – byte-counting approach:

L'approccio byte-counting usato dal protocollo DDCMP non utilizza delle sentinelle per individuare l'inizio e la fine del body ma contiene una variabile **COUNT** che indica quanti byte sono inclusi nel body, così da sapere quanti dei prossimi byte che verranno letti sono dati e quali no. Se COUNT è corrotto, il frame è irrecuperabile.



Bit-oriented protocols (HDLC)

Un protocollo orientato ai bit semplicemente non visualizza il flusso di dati come una sequenza di byte ma come una sequenza di bit.

HDLC (High level Data Link Control)

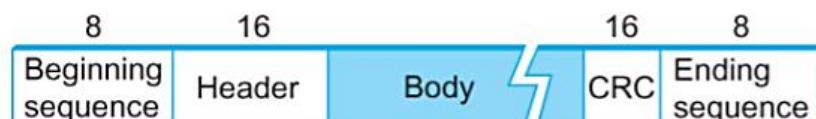
Il protocollo HDLC indica sia l'inizio che la fine di un frame con la distinta sequenza di bit **01111110**. Inoltre, questa sequenza viene trasmessa in qualsiasi momento in cui il collegamento sia inattivo in modo che il mittente e il ricevitore possano rimanere sincronizzati. Siccome la sequenza 01111110 può apparire ovunque nella sezione dei dati, il protocollo HDLC utilizza (similmente al concetto di DLE nel BISYNC) una tecnica chiamata **bit-stuffing**.

Il bit stuffing nell'HDLC funziona nel seguente modo:

- Dal lato del trasmettitore, ogni volta che vengono trasmessi cinque 1 consecutivi nel body del messaggio (escludendo quando il mittente invia la sequenza 01111110 di sincronizzazione) il trasmettitore inserisce uno 0 prima di trasmettere il bit successivo.
- Dal lato del ricevente, quando arrivano cinque 1 consecutivi, in base al bit successivo si possono verificare più casi:
 - Se il bit successivo è uno 0, allora è un bit "stuffed" e va scartato;
 - Se il bit successivo è un 1, allora le possibilità sono due:
 - È la sequenza "01111110" indicante la fine del body;
 - Oppure è stato introdotto un errore nel flusso di dati.

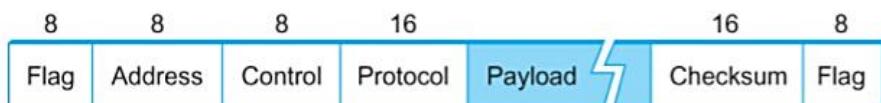
Questi due casi sono distinguibili guardando il bit ancora successivo:

- Se questo è un 1 (01111111) ci dev'esser stato un errore e questo frame va scartato;
- Se questo è uno 0, il ricevitore deve aspettare il secondo "01111110" per potere ricevere nuovamente dei dati.



PPP (Point-to-Point Protocol)

Il protocollo PPP deriva dall'HDLC; usa il "sentinel approach" e il "bit-stuffing".



Rilevamento degli errori

Nelle reti, a livello datalink, alcuni errori possono venir introdotti nei frame durante la loro trasmissione. Questo accade, ad esempio, a causa di interferenze elettriche o a causa di rumore termico. Sebbene gli errori siano rari, specialmente sui collegamenti ottici, è necessario un meccanismo per rilevarli in modo da poter intraprendere azioni correttive. Il rilevamento degli errori è solo una parte del problema. L'altra parte sta nel correggere gli errori rilevati precedentemente. Esistono due approcci di base che possono essere adottati quando il destinatario di un messaggio rileva un errore. Il primo consiste nell'informare il mittente che il messaggio è stato danneggiato in modo che il mittente possa ritrasmetterne una copia. In alternativa, ci sono alcune tipologie di algoritmi di rilevamento degli errori che consentono al destinatario di ricostruire il messaggio corretto anche se danneggiato; tali algoritmi si basano su codici di correzione degli errori, discussi di seguito. Infine, si può ignorare il pacchetto errato come se non fosse mai arrivato.

L'idea alla base di qualsiasi schema di rilevamento degli errori è quella di aggiungere informazioni ridondanti per ciascun frame che possano essere utilizzate per determinare se sono stati introdotti degli errori. All'estremo, potremmo immaginare di trasmettere due copie complete dei dati. Se le due copie sono identiche al ricevitore, probabilmente entrambi i frame sono corretti. Se essi differiscono, è stato probabilmente introdotto un errore in uno (o entrambi) di essi, e devono quindi essere scartati. Questo è uno schema di rilevamento degli errori piuttosto scarso per due motivi: Innanzitutto, invia n bit ridondanti per un messaggio n-bit (scarsa efficienza). In secondo luogo, molti errori non verranno rilevati, ad esempio un errore presente nelle stesse posizioni di entrambi i frame non verrà rilevato (scarsa efficacia).

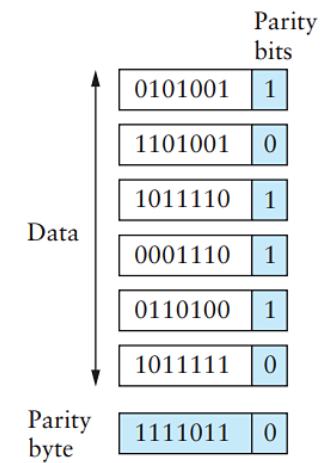
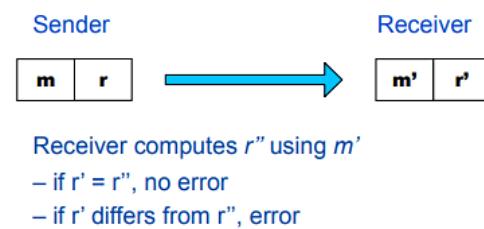
Diciamo che i bit extra che inviamo sono ridondanti perché non aggiungono nuove informazioni al messaggio. Infatti, sono derivati direttamente dal messaggio originale usando un algoritmo ben definito che sia il mittente che il destinatario conoscano con precisione. All'invio di un frame, il mittente applica l'algoritmo al messaggio per generare i bit ridondanti, quindi trasmette sia il messaggio che quei pochi bit extra appena calcolati. Quando il destinatario applicherà lo stesso algoritmo al messaggio ricevuto, dovrebbe (in assenza di errori) ottenere lo stesso risultato inviato dal mittente. Se i due messaggi corrispondono, il ricevente può concludere (con alta probabilità) che non sono stati introdotti errori nel messaggio durante la trasmissione. Se non corrispondono, può assicurarsi che il messaggio (o i bit ridondanti) sia stato corrotto, e successivamente attuare l'azione appropriata, che può essere eliminare il messaggio o se possibile correggerlo. Segue una lista dei principali codici di rilevamento e/o correzione degli errori:

Two-dimensional parity

Tale sistema prevede l'aggiunta di un bit ridondante ai dati, calcolato a seconda che il numero di bit che valgono 1 sia pari o dispari. I bit di parità sono uno dei codici di rilevazione d'errore più semplici.

Esempio: $1101001 \rightarrow 01101001$ oppure $1001111 \rightarrow 11001111$

Se un numero dispari di bit (incluso il bit di parità) è cambiato durante la trasmissione di un insieme di bit, allora il bit di parità non sarà corretto e indicherà che è avvenuto un errore durante la trasmissione. Quindi, il bit di parità è un codice di controllo, ma non è un codice di correzione d'errore, poiché non c'è modo di determinare quale particolare bit è sbagliato. Il "Two-dimensional parity" esegue un calcolo simile per ogni posizione di bit attraverso ciascuno dei byte contenuti nel frame. Ciò comporta l'aggiunta di un byte di parità per l'intero frame, oltre a una parità bit per ogni byte.



Internet checksum algorithm

L'idea alla base dell'Internet checksum è molto semplice: vengono sommati i bit del messaggio e il risultato (checksum) viene trasmesso insieme al frame al ricevente. All'arrivo del frame il ricevitore esegue lo stesso calcolo sui dati ricevuti e confronta il risultato con il checksum ricevuto. Nell'eventualità che essi non siano uguali il frame verrà considerato errato. Questo schema di controllo errori non viene effettuato a livello datalink ma a layer più alti.

CRC (Cyclic Redundancy Check – verifica di ridondanza ciclica)

Il cyclic redundancy check (ovvero controllo di ridondanza ciclico, CRC) è un metodo per il calcolo di somme di controllo (checksum). Il nome deriva dal fatto che i dati d'uscita sono ottenuti elaborando i dati di ingresso i quali vengono fatti scorrere ciclicamente in una rete logica. Il controllo CRC è molto diffuso perché la sua implementazione binaria è semplice da realizzare e si presta bene a rilevare errori di trasmissione.

L'idea basilare che sta dietro ogni algoritmo per il calcolo del CRC è abbastanza semplice: trattare il flusso di dati come un enorme numero binario, dividerlo per un altro numero binario fisso (costante), ed utilizzare il resto di questa divisione come valore di CRC. In realtà, il divisore, il dividendo (il flusso dati), il quoziente, ed il resto non vanno considerati come interi positivi, ma piuttosto come polinomi con coefficienti binari. Questo avviene considerando ciascuno di questi numeri come una sequenza di bit che non sono altro che i coefficienti di un polinomio. Vediamo come ciò avviene con un esempio: il numero decimale 35 viene rappresentato come 23H in esadecimale, e 100011 in formato binario. Se ai bit venissero fatti corrispondere i coefficienti di un polinomio, il numero rappresenterebbe anche il seguente polinomio: $1*x^5+0*x^4+0*x^3+0*x^2+1*x^1+1*x^0 = x^5 + x^1 + 1$.

Al fine di eseguire il calcolo del CRC, la prima cosa da fare è quella di scegliere un divisore, che tecnicamente viene definito come il "polinomio generatore". Un'importante proprietà del polinomio generatore è l'ampiezza (width), definite come la posizione del suo bit più significativo con valore 1. Valori tipici per l'ampiezza sono 16 e 32, perché essi semplificano l'implementazione dell'algoritmo sui normali computer.

Alcuni polinomi generatori comunemente usati sono i seguenti:

- CRC-8-ATM: viene utilizzato per calcolare il valore del campo HEC della cella ATM (Asynchronous Transfer Mode). Il polinomio è $x^8 + x^2 + x + 1$, e la corrispondente rappresentazione è 0x07
- CRC-8-CCITT: viene utilizzato per i dispositivi che operano sul bus 1-Wire (Maxim/Dallas ha anche introdotto un altro tipo di CRC leggermente diverso da questo). Il polinomio è $x^8 + x^7 + x^3 + x^2 + 1$, e la corrispondente rappresentazione è 0x8D
- CRC-16 CCITT: viene impiegato in svariate applicazioni tra cui: Bluetooth, X.25, XMODEM, PPP, ecc. Il polinomio è $x^{16} + x^{12} + x^5 + 1$, e la corrispondente rappresentazione è 0x1021

L'algoritmo: il calcolo del CRC fa riferimento a un insieme di dati (per esempio un messaggio oppure un array), di conseguenza, avremo a che fare con un flusso di dati composto da una sequenza di bit. Occorre quindi scegliere un polinomio divisore. Successivamente, il calcolo consiste in una divisione tra il flusso di bit ed il polinomio. Un numero W di bit uguali a zero deve essere appeso al flusso di bit prima di eseguire la divisione, essendo W l'ampiezza del polinomio. Ciò significa che nella matematica del CRC (o meglio, nella matematica dei polinomi) l'addizione e la sottrazione possono essere eseguite tramite la funzione XOR. A livello implementativo in hardware, tutte le operazioni sono eseguite tramite un registro a scorrimento a sinistra R con un numero di bit uguale all'ampiezza del polinomio, cioè W.

L'algoritmo consiste dei seguenti passi:

1. aumento del messaggio aggiungendo W bit (ampiezza del polinomio divisore) 0 alla sua fine;
2. caricare il registro R con bit tutti a 0;
3. se esistono ancora bit da processare eseguire lo step 4, altrimenti saltare allo step 7;
4. far scorrere verso sinistra di 1 bit il registro R, caricando il prossimo bit del messaggio nel bit di posizione 0 del registro;
5. se dal registro esce un bit a 1 (cioè se il bit più significativo del registro prima di eseguire lo shift era un 1) porre: $R = R \text{ XOR polinomio}$;
6. saltare allo step 3;
7. il registro R contiene ora il resto, cioè il CRC.

Quindi, nelle applicazioni relative ai protocolli di comunicazione (come l'Ethernet), il CRC viene aggiunto alla fine del messaggio prima di eseguire la trasmissione. Lato ricevente, è possibile sia ricalcolare il CRC e confrontarlo con quello ricevuto, o più semplicemente calcolare il CRC su tutto il messaggio ricevuto: se non si sono verificati errori, il risultato sarà uguale a 0.

$$\begin{array}{r}
 1011011101 \ 0000 \quad | \ 10101 \\
 10101 \\
 \hline
 00011111 \\
 10101 \\
 \hline
 010100 \\
 10101 \\
 \hline
 000011 \ 000 \\
 10 \ 101 \\
 \hline
 01 \ 1010 \\
 1 \ 0101 \\
 \hline
 0 \ 1111
 \end{array}$$

ES: Si vuole trasmettere la sequenza di bit 1011011101 aggiungendo un codice CRC usando il polinomio generatore $x^4 + x^2 + 1$. Come è fatto il messaggio complessivo?

- Il messaggio complessivo è 1011011101111. Qui a lato il calcolo del CRC.
Il resto è 1111.

NB risoluzione: aggiunto 4 zeri perché il polinomio è di grado quattro.

Trasmissione affidabile

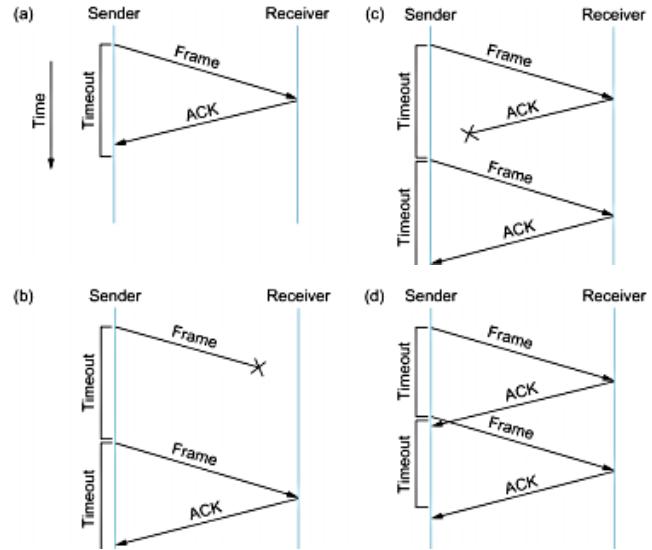
Come abbiamo visto nella sezione precedente, i frame a volte vengono danneggiati durante il trasporto; con un codice di errore come il CRC possiamo rilevare tali errori. Esistono alcuni codici di errore abbastanza sofisticati da permettere anche la correzione di alcuni errori, ma tipicamente non vengono utilizzati perché l'overhead è troppo elevato. Anche quando vengono utilizzati codici di correzione degli errori (ad es. su collegamenti wireless), alcuni errori saranno troppo gravi per essere corretti. Di conseguenza, alcuni frame danneggiati devono essere eliminati. Un protocollo a livello datalink (di collegamento) che desidera fornire frame in modo affidabile deve in qualche modo recuperare questi frame scartati (persi). Questo si ottiene usando una combinazione di due fondamentali meccanismi: **acknowledgments** (riconoscimenti) & **timeouts**. Un riconoscimento (**ACK** in breve) è un piccolo frame di controllo che un protocollo rimanda al suo mittente dicendo che ha ricevuto un frame precedente. Se il mittente non riceve un ACK (riconoscimento) dopo un ragionevole lasso di tempo, ritrasmette il frame originale. Questa azione di attesa per un ragionevole lasso di tempo si chiama timeout. La strategia generale per implementare la consegna affidabile con ack e timeouts è talvolta chiamata richiesta di ripetizione automatica (normalmente abbreviata **ARQ – Automatic Repeat Request**). Seguono le principali strategie ARQ:

Stop & wait protocol

Lo schema ARQ più semplice è l'algoritmo stop-and-wait. L'idea che sta alla base di questo algoritmo è semplice: dopo aver trasmesso un frame, il mittente attende un riconoscimento prima di trasmettere il frame successivo. Se il riconoscimento non arriva dopo a certo periodo di tempo, il mittente scade (timeout) e ritrasmette il frame originale. Cronologia che mostra quattro diversi scenari per l'algoritmo stop-and-wait.

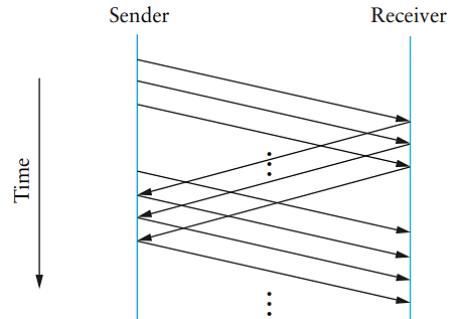
- a) L'ACK è ricevuto prima della scadenza del timer;
- b) Il frame originale è andato perso;
- c) l'ACK è perso;
- d) Il timeout scatta presto.

I casi b e c sono problematici!

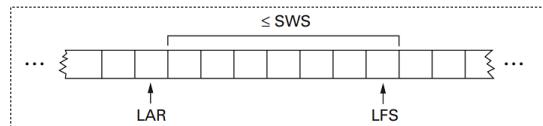


Sliding window protocol

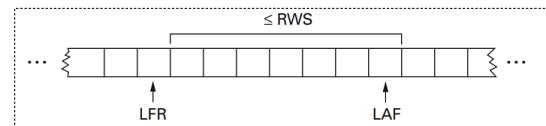
Il protocollo “sliding window” invece funziona nel seguente modo: il mittente assegna un numero di sequenza (SeqNum) a ciascun frame, inoltre, mantiene tre variabili: **“SendWindowSize”** (SWS) che identifica il limite superiore dei frame che può spedire; **“LastAckReceived”** (LAR) che identifica il numero dell’ultimo ACK ricevuto e **“LastFrameSequencenum”** (LFS) che denota il numero (il SeqNum) dell’ultimo frame inviato. Inoltre, il mittente mantiene in memoria la seguente invariante: **LFS-LAR <= SWS**.



Sliding window del mittente:



Sliding window del ricevente:



Quando arriva un ACK, il mittente muove LAR a destra ($LAR++$) permettendogli così di trasmettere un altro frame. Il mittente associa a ciascun frame un timer, che ritrasmette nel caso il relativo ACK non sia ricevuto entro lo scadere del tempo. Il ricevente mantiene tre variabili: **“ReceiveWindowSize”** (RWS) che delimita il limite superiore di frame che può ricevere, **“LargestAcceptableFrame”** (LAF) che delimita il SeqNumber più elevato accettabile e **“LastFrameReceived”** (LFR) che denota la sequenza dell’ultimo frame ricevuto. Inoltre il ricevente mantiene in memoria la seguente invariante: **LAF-LFR <= RWS**.

All’arrivo di un frame di numero SeqNumber, il ricevitore si può trovare in più casi:

- SeqNum \leq LFR oppure SeqNum $>$ LAF: Il frame viene scartato;
- LFR $<$ SeqNum \leq LAF: Il frame viene accettato e viene spedito un ACK.

Problemi del protocollo Sliding Window: quando avviene un timeout, la quantità di dati trasmessa diminuisce. Quando un pacchetto viene perso, la banda non viene sfruttata interamente. Possibili soluzioni:

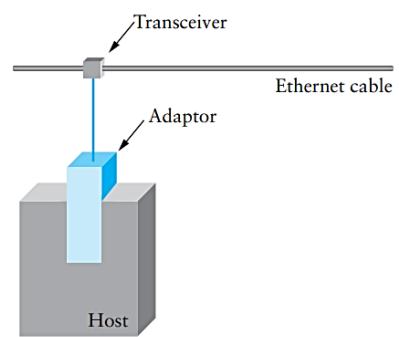
- Negative Acknowledgement (NAK);
- Additional Acknowledgement;
- Selective Acknowledgement.

Ethernet (802.3)

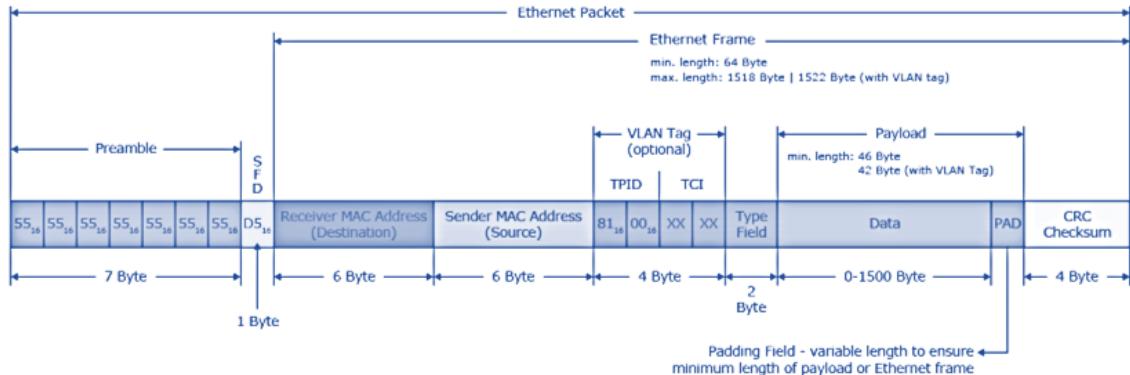
<https://www.youtube.com/watch?v=iKn0GzF5-IU>

L’Ethernet è una tecnologia per reti locali (LAN – Local Area Network). Ethernet (su cavo half-duplex – una sola direzione di trasmissione) utilizza il protocollo **CSMA/CD (Carrier Sense Multiple Access with Collision Detection)**. Il protocollo viene implementato nell’adattatore di rete e possiede le seguenti caratteristiche:

- **Carrier Sense**: sta a indicare che ciascun nodo può rilevare se il mezzo di comunicazione (il cavo ethernet in questo caso) è in stato idle o è busy (in attesa o occupato).
- **Multiple Access**: sta a indicare che ogni nodo può accedere al mezzo di comunicazione in maniera concorrente.
- **Collision Detection**: sta a indicare che un nodo “ascolta” mentre trasmette per verificare che non ci siano collisioni di frame trasmessi da un altro nodo.



Struttura del frame



- Il preambolo consente al ricevitore di sincronizzarsi con il segnale. Consiste in una sequenza di zeri e uni alternati.
- SFD:** Start of Frame Delimiter (8bit 10101011) è una sentinella indicante l'inizio del frame.
- Sia il trasmettitore che il ricevente sono identificati da un indirizzo MAC a 48bit (6 byte) ciascuno.
- Type Field:** 2 byte indicanti la tipologia di pacchetto. Informazione utilizzata dai protocolli di livello più alto.
- Data:** I dati del pacchetto, da 0 a 1500Byte.
- PAD:** nel caso in cui i dati del pacchetto siano troppo piccoli, serve a riempire il frame per garantire la dimensione minima.
- CRC:** Checksum dell'algoritmo CRC-32
- L'**Header** è quindi grande 14+4 opzionali (byte).

Ethernet è quindi un protocollo bit-oriented (in riferimento alle tecniche di framing appena viste).

ES: Un ricevitore ethernet, come capisce quando inizia e finisce il frame?

- per il preambolo di 7 byte di sincronizzazione più il SFD (tecnica sentinel-approach).

- capisce che il frame è finito per mancanza di segnale.

Indirizzi

Ogni host nel protocollo Ethernet possiede un indirizzo univoco. L'indirizzo in realtà appartiene all'adattatore, non all'host. Esso è tipicamente scritto in ROM del dispositivo. Gli indirizzi sono tipicamente scritti in formato leggibile dall'essere umano: una sequenza di 6 numeri separati da colonne, ogni numero corrisponde a un byte dei 6, ed è fornito da una coppia di valori esadecimale. Ad esempio: 8:0:2:b:e4:b1:2 corrisponde a 00001000 00000000 00101011 11100100 10110001 00000010.

Ogni frame trasmesso in una rete ethernet è ricevuto da ciascun adattatore che riconosce quei frame ad egli indirizzati, ne controlla il CRC e in caso di esito positivo spedisce il frame al suo host.

Per essere certi che ogni adattatore abbia un indirizzo univoco, a ciascun produttore di dispositivi Ethernet viene assegnato un diverso prefisso che deve costituire la parte iniziale dell'indirizzo di tutti gli adattatori che costruisce e vende.

In Ethernet esiste un indirizzo speciale avente tutti i bit al valore 1 (255:255:255. etc.) che ha funzione di indirizzo broadcast. I frame con destinazione broadcast vengono passati al proprio host da tutti gli adattatori di rete. Un indirizzo che ha il primo bit a 1, ma che non sia broadcast, è detto multicast.



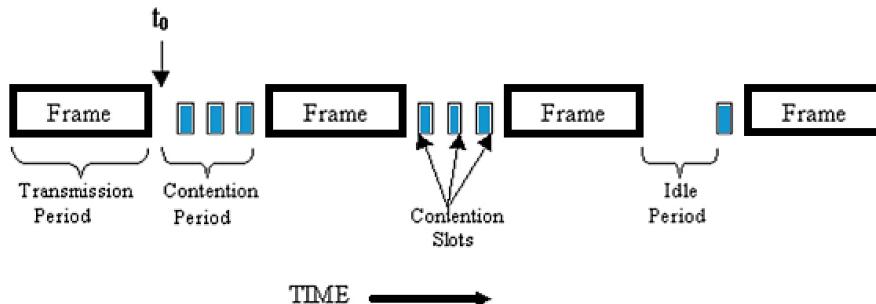
Un adattatore riceve tutti i frame in arrivo dalla propria rete ethernet mentre accetta soltanto:

- i frame destinati al proprio indirizzo;
- i frame destinati all'indirizzo broadcast;
- i frame destinati a un indirizzo multicast, se ha ricevuto istruzioni per ascoltare il traffico destinato a tale indirizzo;
- tutti i frame, se è stato configurato in modalità promiscua (accetta tutto il traffico sulla rete).

Algoritmo di trasmissione

L'algoritmo di trasmissione del protocollo Ethernet è chiamato **Ethernet's Media Access Control (MAC)** e viene implementato via hardware (negli adattatori). Segue il funzionamento:

- Quando un adattatore ha un frame da mandare e la linea è in idle, quest'ultimo viene inviato immediatamente. L'upper bound di 1500Bytes per il messaggio fa sì che un adattatore occupi una linea per un lasso di tempo di lunghezza finita.
- Quando un adattatore ha un frame da mandare e la linea risulta occupata (busy) aspetta che la linea vada in idle (per un tempo minimo detto **Inter Packet Gap (IPG)**) e poi trasmette immediatamente. I tempi di attesa sono determinati da un algoritmo di backoff esponenziale spiegato successivamente.



Poiché non esiste controllo centralizzato, è possibile che due o più adattatori inizino a trasmettere nello stesso momento; quando ciò accade, si dice che i frame **collidono** sulla rete; dato che Ethernet è in grado di rilevare le collisioni, ciascuna sorgente può stabilire che si sta verificando una collisione. Nel momento in cui un adattatore si accorge che il frame che ha trasmesso sta entrando in collisione con un altro, per prima cosa trasmette una sequenza di disturbo di 32 bit (**jамming sequence**), poi interrompe la trasmissione. In questo modo, in caso di collisione, un trasmettitore invierà almeno 96 bit: 64 bit di preamble e 32 bit di sequenza di disturbo (96 bit solo quando i due host che hanno generato la collisione sono vicini). Per avere la certezza che il frame appena inviato non abbia colpito con un altro frame, può darsi che il trasmettitore debba inviare fino a 512 bit (non a caso un frame Ethernet deve essere lungo almeno 512 bit/64 byte). Perché proprio 512 bit? Perché una lunghezza limitata a 2500 metri? La risposta a entrambe le domande ha a che fare con il fatto che più due nodi sono distanti, più tempo serve al frame inviato da un nodo per raggiungere l'altro nodo, e in tale intervallo di tempo la rete è vulnerabile al fenomeno della collisione.

Riassumendo:

In un protocollo CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) ci sono tre possibili stati:

- **Idle**: nessuna stazione ha frame da trasmettere;
- **Contention**: qualche stazione vuole trasmettere, e deve aspettare che la linea diventi libera, ovvero aspettare che non ci siano segnali sulla linea da almeno un tempo ITP (9.6 µs – Inter Packet gap);
- **Transmission**: una o più stazioni trasmettono:
 - siccome non c'è un controllo centralizzato, può capitare che due o più stazioni trasmettano allo stesso tempo: perché entrambi trovano la linea libera allo stesso istante; quando questo accade si dice che i due frame collidono nella rete.

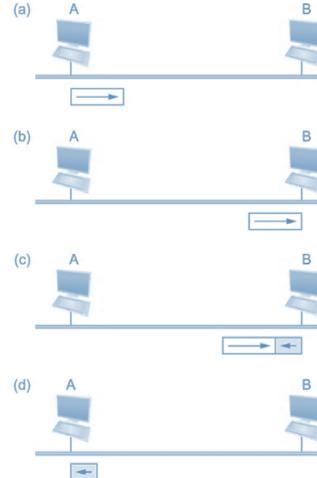
Come viene rilevata una collisione? l'elettronica confronta ciò che trasmette con ciò che riceve, se la differenza non è zero, allora c'è una collisione.

Worst-case-scenario: denotiamo con t_{prop} la latenza di un link (tempo di propagazione).

- A inizia a trasmettere un frame all'istante t
- Il primo bit del frame di A arriva a B all'istante $t+t_{prop}$
- Un istante prima che arrivi il frame di A, l'host B inizia a trasmettere il suo frame: questo causa un'immmediate collisione tra i due frame, che B rileverà e inizierà a inviare la jamming-sequence di 32-bit.
- L'host A non saprà della collisione finché non verrà raggiunto dal frame di B, che arriverà all'istante $t+2t_{prop}$

Di conseguenza, A deve continuare a trasmettere fino a questo momento per far sì che sia in grado di rilevare la collisione. A deve trasmettere per almeno $2+t_{prop}$ per essere sicura di rilevare tutte le possibili collisioni.

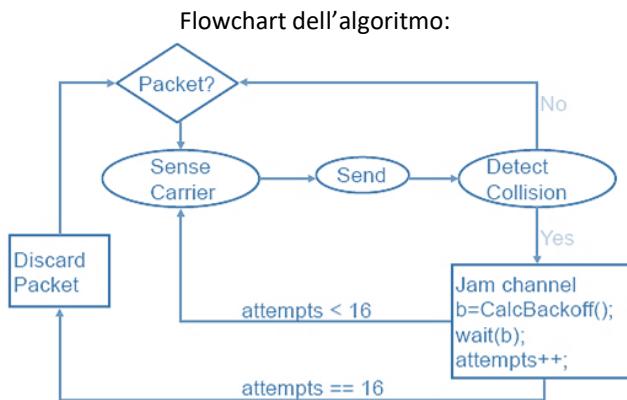
In generale, il massimo round-trip delay (latenza massima) è stata determinata essere $51.2\mu\text{s}$ ($t_{prop} \leq 25.6\mu\text{s}$) con il limite di 2500m.



Ogni volta che un adattatore rileva una collisione, ferma la trasmissione e aspetta un certo ammontare di tempo e poi riprova. Ogni volta che l'adattatore riprova e fallisce, duplica il tempo di attesa. Questa strategia viene chiamata **backoff esponenziale**.

Algoritmo di backoff esponenziale: chiamiamo SlotTime = $51.2\mu\text{s}$

Al primo tentativo, l'adattatore invia subito il frame dopo aver atteso ITP. Se non ci sono collisioni nel lasso di tempo SlotTime, tutto è ok. Se invece ci sono, l'adattatore invia un runt-frame (jamming-sequence), aspetta ITP, e riprova dopo aver aspettato **0 o 1 SlotTime, selezionato a random**. Se il tentativo fallisce nuovamente, aspetta una quantità random tra 0,1 o 3 SlotTime (sempre a random), al terzo fallimento aspetta una quantità random tra 0,1,2,3... 2^3-1 SlotTime e così via. In generale l'algoritmo aspetta una quantità pari a **k*SlotTime con k = 0... 2^c-1** , con c limitato a 15 e c=n° tentativo (1,2,3...15). Quindi cresce alla 2.

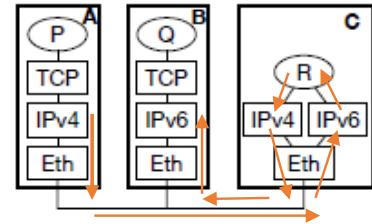


ES: Due stazioni Ethernet A e B hanno già tentato di trasmettere una volta, rilevando collisione. Al secondo tentativo, anche una terza stazione C prova a trasmettere (per la prima volta). Qual è la probabilità che la trasmissione avvenga senza collisioni?

- A causa del backoff esponenziale, A e B aspettano un tempo random compreso di 0 o 1 SlotTime (= 51.2μs), mentre C non aspetta niente (perchè è il suo primo tentativo). Quindi affinchè non ci sia collisione bisogna che sia A sia B scelgano di aspettare 1 SlotTime (altrimenti vanno in collisione con C). La probabilità che questo avvenga è $0.5 * 0.5 = 0.25$

ES: Nello schema a lato, i tre calcolatori sono collegati alla stessa rete Ethernet. I processi P e Q possono comunicare, eventualmente passando per C? A talfine, cosa deve fare il processo R?

- Non possono comunicare direttamente, ma possono farlo passando per C. Il processo R deve tradurre pacchetti IPv4 in IPv6 e viceversa: toglie l'intestazione di un tipo e mette quell'altro, mantenendo il payload di livello 4. (Un po' come un NAT).



ES: Un ricevitore Ethernet...

- (a) ...come capisce quando inizia un frame?
- (b) ...e come capisce quando finisce?

- Risposta:

- (a) Per il preambolo di 7 byte + il SFD: start frame delimiter (tecnica sentinel)
- (b) Per la mancanza di segnale.

Wireless (802.11)

I link wireless trasmettono segnali elettromagnetici: onde radio, micro-onde, infrarossi etc., inoltre, condividono tutti lo stesso mezzo di comunicazione, l'aria; quindi l'obiettivo è quello di ridurre le interferenze e permettere che la comunicazione sia più efficiente possibile.

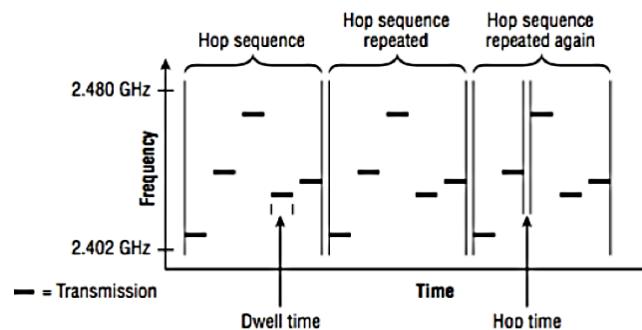
Frequenze specifiche sono assegnate a particolari utilizzi, come ad esempio:

- uso governativo, militare;
- AM/FM radio, televisione, comunicazioni satellitari e telefonia mobile.

Poiché tutti i collegamenti senza fili condividono lo stesso mezzo trasmissivo, è fondamentale rendere efficiente la sua condivisione. La maggior parte di queste condivisioni viene realizzata suddividendo il mezzo lungo la dimensione della frequenza dello spazio; si può assegnare a un'unica entità l'utilizzo esclusivo di una determinata frequenza all'interno di una determinata area geografica. Queste assegnazioni esclusive sono regolate da agenzie governative. Esistono alcune bande di frequenza riservate a un utilizzo che non richiede licenze particolari. Anche i dispositivi che usano queste frequenze "libere" sono, però, soggetti ad alcune limitazioni. La prima riguarda la potenza di trasmissione, la quale limita il raggio d'azione del segnale, riducendo così la probabilità di interferenza con altri segnali. Quando lo spettro elettromagnetico è condiviso da più dispositivi e applicazioni, viene spesso utilizzata una tecnica denominata spread spectrum, basata sull'idea di distribuire il segnale su una banda di frequenza più ampia del normale, allo scopo di minimizzare l'impatto derivante dall'interferenza di altri dispositivi.

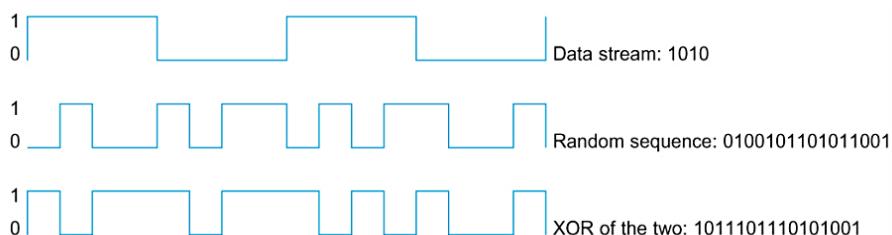
Frequency hopping spread spectrum

Con "frequency hopping" si indica la modalità di trasmissione del segnale utilizzando una sequenza randomica di frequenze. La sequenza di frequenze è calcolata da un generatore di numeri casuali (pseudorandom number generator). Sia il ricevente che il mittente conoscono il seme e quindi possono rimanere sincronizzati sulle frequenze corrette.



Direct sequence spread spectrum

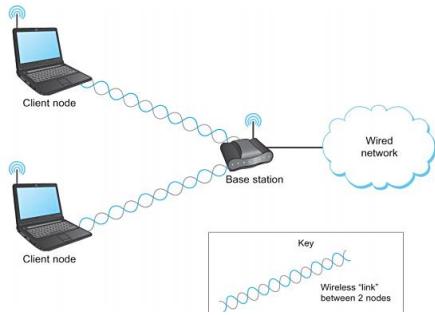
Ogni bit di un frame è rappresentato da una molteplicità di bit nel segnale trasmesso. Per ciascun bit il trasmettitore spedisce uno XOR del bit e di n bit randomici detti "chipping sequence". La sequenza randomica di bit è calcolata da un generatore di numeri casuale il cui seme è conosciuto da ricevitore e mittente. Il ricevitore per recuperare il dato esegue uno XOR del segnale ricevuto con la sequenza randomica.



Tecnologie Wireless

Le tecnologie wireless più comuni differiscono per banda offerta, distanza di collegamento tra dispositivi.

- Bluetooth
- Wi-Fi (802.11)
- WiMAX (802.16)
- 3G/4G/5G cellular wireless



	Bluetooth (802.15.1)	Wi-Fi (802.11)	3G Cellular
Typical link length	10 m	100 m	Tens of kilometers
Typical data rate	2 Mbps (shared)	54 Mbps (shared)	Hundreds of kbps (per connection)
Typical use	Link a peripheral to a computer	Link a computer to a wired base	Link a mobile phone to a wired tower
Wired technology analogy	USB	Ethernet	DSL

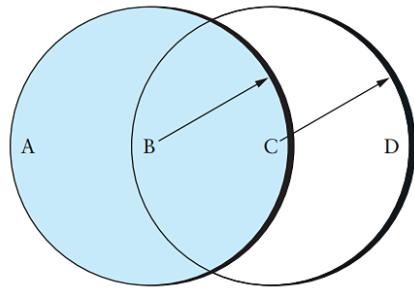
Le comunicazioni wireless supportano le connessioni point-to-multipoint. Le comunicazioni tra i nodi “non-base” ovvero i client possono essere diretti (come nel WiFi) o instradati tramite una stazione base (line telefoniche mobile).

Wi-Fi (IEE 802.11)

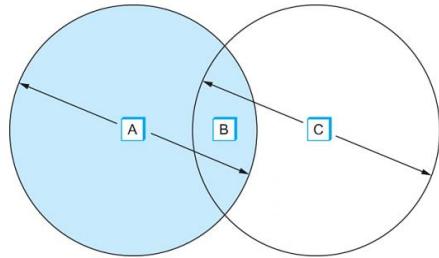
Come Ethernet, lo standard 802.11 è stato progettato per l'utilizzo in un'area geograficamente limitata e il suo obiettivo principale consiste nel mediare l'accesso a un mezzo di comunicazione condiviso: in questo caso, segnali che si propagano nello spazio.

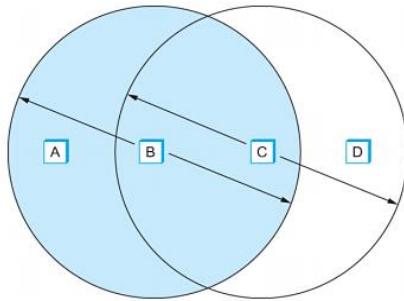
Collision avoidance

A primo avviso, si può pensare che il protocollo wireless segua lo stesso approccio del protocollo ethernet (aspetta che il link diventi libero prima di trasmettere e fermati nel caso che avvenga una collisione). In prima approssimazione questo è ciò che accade, ma con qualche differenza. In una rete wireless il problema è più complesso perché non tutti gli host sono tra di loro raggiungibili.



Esempio: si consideri lo schema a destra dove ciascun nodo può trasmettere al nodo immediatamente alla sua sinistra o destra. Ad esempio, B può scambiare dati con A e C ma non con D; C può raggiungere B e D ma non A. Supponiamo che A e C vogliano comunicare con B. A e C non sono ignari della presenza reciproca. I loro due frame spediti a B collidono, ma a differenza dell'ethernet nessuno dei due tra A e C sono consapevoli della collisione. A e C sono detti **nodi nascosti** (reciprocamente).





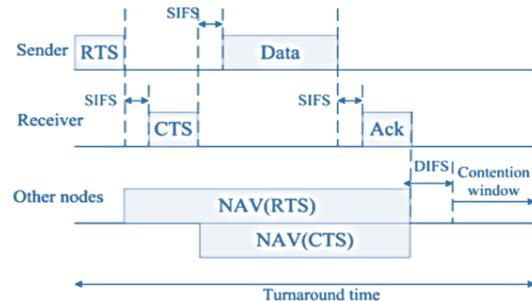
Esiste un altro problema, quello dei **nodi esposti**. Supponiamo che B stia trasmettendo ad A. C è consapevole di questa comunicazione perché “sente” la trasmissione di B. Sarebbe un’errore però, se C concludesse che non può trasmettere a nessuno perché sente B trasmettere solo perché non vede A. Supponiamo che C voglia trasmettere al nodo D: questo non risulta un problema siccome la trasmissione tra C e D non interferirebbe con quella tra A e B.

Il protocollo 802.11 risolve questi problemi con un’algoritmo chiamato **CSMA/CA** o **MACA** (Carrier Sense Multiple Access with Collision Avoidance). L’idea alla base dell’algoritmo è la seguente:

- trasmettitore e ricevitore si scambiano frame di controllo prima che il trasmettitore scambi dei dati.
- questa trasmissione di frame informa tutti i nodi nelle vicinanze che una trasmissione sta per iniziare.
- il trasmettitore trasmette un frame: “**request to send (RTS)**” al ricevitore.
 - il frame RTS include un campo che indica per quanto tempo il trasmettitore vuole trattenere il mezzo (lunghezza del data-frame da trasmettere).
- il ricevitore risponde con un frame “**clear to send (CTS)**” al trasmettitore se non è impegnato.
 - questo frame restituisce (eco) le informazioni presenti nell’RTS.

Funzionamento:

Ogni nodo che riceve il frame CTS sa di essere vicino al ricevitore, e quindi non può trasmettere per un determinato lasso di tempo (quello indicato nel frame CTS, chiamato **Network Allocation Vector (NAV)**). Ogni nodo che riceve il frame RTS ma non il CTS conclude di non essere vicino al ricevitore, (ma solo al trasmettitore) e che quindi è libero di trasmettere.



Esempio: riferendosi al caso precedentemente illustrato, se C volesse comunicare con D, trasmetterebbe un RTS che verrebbe ricevuto da D e da B. D risponderebbe con un CTS che verrebbe raggiunto da C e NON da B. C che riceve l’ok, trasmette a D per un determinato lasso di tempo indicato nel CTS. B di conseguenza, non ricevendo il CTS sa di non essere vicino al trasmettitore e di conseguenza può trasmettere (ad esempio ad A).

Se due o più nodi rilevano un nodo in stato di idle e trasmettono il loro frame RTS allo stesso tempo, i due frame collideranno. Questo avviene perché lo standard 802.11 non supporta il rilevamento di collisioni. Il trasmettitore non può controllare la linea durante la trasmissione perché il mezzo può funzionare solo in trasmissione o ricezione (unidirezionale). Il trasmettitore quindi conclude che è avvenuta una collisione quando non riceve il frame CTS dopo un determinato lasso di tempo. In questo caso aspetta una quantità randomica di tempo (è importante che sia randomica per evitare che i due ritrasmettano l’RTS nello stesso istante) e rispedisce il CTS. La quantità di tempo da aspettare è definita come nell’algoritmo di backoff esponenziale di ethernet.

Intervalli temporali dell’algoritmo:

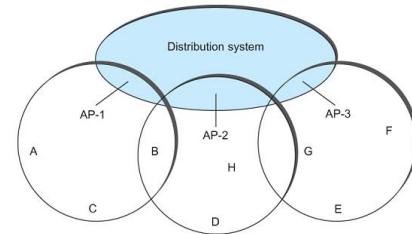
- **Slot time:** il minimo intervallo utilizzato nell’algoritmo di backoff esponenziale.
- **SIFS: Shortest Inter Frame Space:** il tempo utilizzato per processare e rispondere ad un frame.
- **PIFS: Point Control Function Interframe Space:** tempo che ciascun access point deve aspettare prima di inviare un nuovo RTS.
- **DIFS: Distributed Control Function Interframe Space:** tempo che ciascuna stazione deve aspettare prima di inviare un nuovo RTS.

Esempio riassuntivo comunicazione tra A e B:

- A invia un RTS a B;
- B riceve l'RTS e passa un tempo pari a SIFS, poi invia un CTS ad A;
- A riceve il CTS e passa un tempo pari a SIFS, poi invia i dati per il tempo prestabilito;
- B riceve i dati e passa un tempo pari a SIFS, poi invia un ACK di ricezione dei dati;
- Poi in contemporanea:
 - A riceve l'ACK e sa di aver inviato i dati correttamente.
 - Ciascuna stazione deve aspettare DIFS prima di inviare un nuovo RTS.

Distribution system

Oggi quasi tutte le reti 802.11 usano una topologia orientata all'uso di stazioni base: invece di avere una rete con nodi tutti uguali, alcuni nodi possono spostarsi all'interno della rete, mentre altri sono connessi a un'infrastruttura di rete cablata. Questi ultimi, che sono stazioni base, vengono chiamati access point (punti di accesso) e sono connessi l'uno all'altro mediante ciò che viene chiamato sistema di distribuzione. Ogni access point opera su un certo canale, caratterizzato da un'opportuna banda di frequenza e, tipicamente, ciascuno di essi opererà su un canale diverso da quello usato dagli access point adiacenti. L'aspetto importante è che la rete di distribuzione operi all'interno dello strato di linea di collegamento, lo stesso strato di protocolli in cui si trovano i collegamenti wireless: in altre parole, che non dipenda da alcun protocollo appartenente a uno strato più elevato. L'idea che sta alla base di questa configurazione è che ciascun nodo si associa a un access point. Perché il nodo A possa comunicare con il nodo E, per esempio, A invia per prima cosa un frame al proprio access point (AP-1), il quale inoltra il frame attraverso il sistema di distribuzione ad AP-3, il quale finalmente trasmette il frame a E. Ciò che viene specificato da 802.11 è il modo in cui i nodi selezionano il proprio access point e come funziona tale algoritmo in presenza di nodi che si spostano da una cella a un'altra.

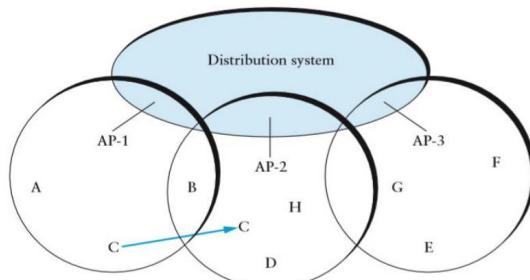


La tecnica per la selezione di un access point viene chiamata **scanning** (scansione) e richiede i seguenti passi:

1. il nodo invia un frame di tipo **Probe**;
2. tutti gli access point raggiungibili rispondono con un frame di tipo **Probe Response**;
3. il nodo seleziona uno degli access point e gli invia un frame di tipo **Association Request**;
4. l'access point risponde con un frame di tipo **Association Response**.

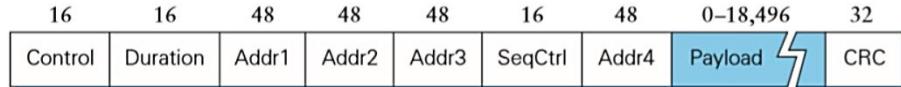
Ogni volta che un nodo si associa a un nuovo access point, tale AP segnala il cambiamento all'access point precedente tramite il sistema di distribuzione.

Nella situazione proposta nella figura soprastante, il nodo C si sposta dalla cella servita da AP-1 verso la cella servita da AP-2. Mentre si sposta invia il frame Probe, che sollecita il frame Probe Response da parte di AP-2. A un certo punto C preferirà AP-2 rispetto ad AP-1 e chiederà l'associazione a tale access point. Il meccanismo appena descritto prende il nome di **scansione attiva**, perché il nodo ricerca attivamente un access point. Oltre a ciò, gli access point inviano periodicamente un frame di tipo **Beacon** che pubblicizza le caratteristiche dell'access point stesso, tra le quali figurano le velocità di trasmissione consentite. Questa modalità viene chiamata **scansione passiva** e un nodo può cambiare il proprio access point in base al frame Beacon ricevuto, rispondendo semplicemente con un frame Association Request all'access point.



Struttura del frame

Il frame wireless contiene gli indirizzi dei nodi sorgente e destinazione (ciascuno lungo 48bit – 6 byte), fino ad un massimo di 2312 byte di dati e un codice CRC a 32bit. I campi precedenti al payload sono il MAC header.



Descrizioni degli altri campi del frame (del MAC header):

- **Control:** contiene tre sottocampi:
 - **Type:** 6 bit che indicano se il frame trasporta dati, se è RTS o CTS etc.
 - **toDS e fromDS**

Il frame del protocollo 802.11 come appena descritto contiene quattro indirizzi (Addr1-2-3-4). Come essi vengano interpretati dipende dai valori toDS e fromDS del campo Control. Essi servono per gestire la possibilità che il frame debba essere inoltrato attraverso il sistema di distribuzione e di conseguenza, il mittente originario non è necessariamente lo stesso nodo che ha trasmesso il frame più recente.

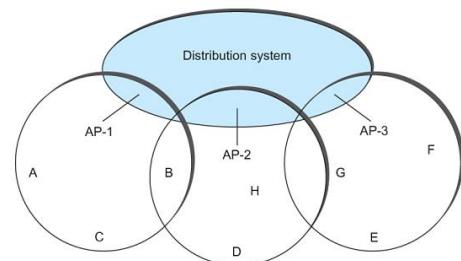
Ad esempio, nel caso più semplice, quando un nodo sta trasmettendo ad un altro direttamente, entrambi i bit toDS e fromDS valgono 0, Addr1 quindi identifica il nodo destinazione e Addr2 il nodo sorgente.

Nel caso più complesso entrambi valgono 1, indicando cioè che il messaggio è stato trasmesso da un nodo wireless al sistema di distribuzione e poi dal sistema di distribuzione ad un altro nodo wireless. In questo caso:

- **Addr1:** identifica la destinazione finale;
- **Addr2:** è l'indirizzo del trasmettitore attuale (cioè quello che ha inoltrato il frame dal sistema di distribuzione alla destinazione finale);
- **Addr3:** identifica la destinazione intermedia (quella che accetta il frame da un nodo wireless e lo inoltra attraverso il sistema di distribuzione);
- **Addr4:** è l'indirizzo della sorgente originaria.

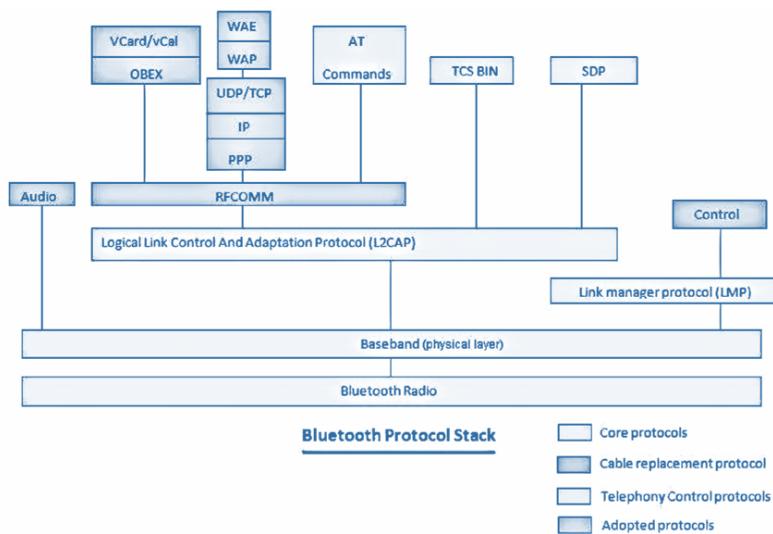
Esempio: Riferendosi sempre allo stesso schema: A comunica ad E

- **Addr1:** E
- **Addr2:** AP-3
- **Addr3:** AP-1
- **Addr4:** A



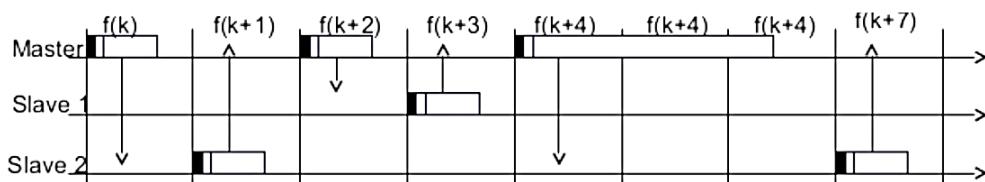
Bluetooth (802.15.1)

Bluetooth si colloca nell'ambito delle comunicazioni a corto raggio. È un'alternativa più conveniente al collegamento via cavo tra due dispositivi. In applicazioni di questo tipo non è necessario avere una grande banda disponibile. Ciò significa che i trasmettitori radio Bluetooth possono usare una potenza di trasmissione abbastanza modesta. Bluetooth opera nella banda a 2.45 Ghz, che non ha bisogno di concessioni o licenze. I collegamenti Bluetooth consentono velocità tipiche che vanno da 1 a 3 Mbps e hanno un raggio operativo di circa 10 metri. Lo standard Bluetooth è specificato da un consorzio industriale, Bluetooth Special Interest Group, tramite un intero sistema di protocolli, ben oltre lo strato di collegamento, giungendo fino ai protocolli dello strato applicativo, detti profili. Alcuni esempi: esiste un profilo per sincronizzare un palmare con un PC; un altro che fornisce l'accesso a una rete a un computer portatile tramite un modem simulato.



Piconet

Una rete bluetooth è detta piconet. Consiste nella presenza di un dispositivo **master** e fino a sette dispositivi **slave**. Qualsiasi comunicazione avviene tra il master e uno slave: gli slave non comunicano direttamente tra loro. Dal momento che gli slave hanno un ruolo semplice, per tali dispositivi l'hardware e il software può essere altrettanto semplice ed economico. Per convivere con le possibili interferenze è d'obbligo utilizzare una tecnica basata sulla strategia spread spectrum, usando frequency hopping con 79 canali (cioè frequenze diverse) ciascuno usato per $625\mu s$. Questa modalità identifica in modo naturale finestre temporali in cui Bluetooth può operare con multiplexing sincrono a divisione di tempo: un frame può occupare 1, 3 o 5 intervalli di tempo consecutivi.



Negli intervalli di tempo etichettati con un numero dispari soltanto il master può iniziare la trasmissione di un frame; uno slave può trasmettere in un intervallo con numero pari, ma soltanto in risposta a una richiesta inviata dal master nell'intervallo precedente. Un dispositivo slave può essere "parcheggiato" (parked), cioè reso inattivo in uno stato di basso consumo di potenza. Un dispositivo parcheggiato non può comunicare all'interno della piconet: può solamente essere riattivato dal master. Una piconet può avere fino a 255 dispositivi parcheggiati.

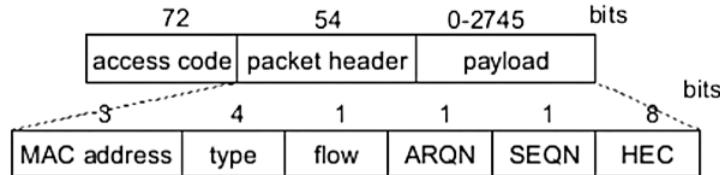
Ogni nodo, ha un bluetooth device address: BD_ADDR.

Scatternet

Più piconet possono essere unite per formare una scatternet. Una scatternet è l'insieme di due o più piconet che condividono almeno un nodo. Questo può agire come un bridge per le reti: riceve un frame da un master e lo invia a un altro. I master di ogni piconet devono tenere traccia della posizione dei nodi per instradare i frame correttamente.

Frame format

Un frame Bluetooth viene chiamato pacchetto, e ha il seguente formato:



Descrizione dei campi:

- **access code**: serve per la sincronizzazione derivata dal master quando attiva la piconet.
- **packet header**: contiene altri ulteriori sottocampi, tra cui quello relativo all'indirizzo MAC, al tipo del link e all'algoritmo di checksum.

ES: In una certa cella Bluetooth (versione 2), l'accesso al mezzo è a divisione di tempo, con slot di $625\mu s$ e un bitrate grezzo di 3Mbit/s; il master trasmette negli slot pari, mentre gli slave negli slot dispari. Supponendo che un frame occupi un solo slot, e ricordando che ogni frame ha un'intestazione di 54 bit e un preambolo ("access code") di 72 bit, a quale bitrate massimo netto può trasmettere il nodo master?

- In uno slot ci stanno $625 * 10^{-6} * 3 * 10^6 = 1875$ bit, a cui bisogna togliere 54+72 bit di overhead, per cui i bit netti trasmessi sono 1749 bit, in ogni slot. Dato che il master può trasmettere solo negli slot pari, il bitrate complessivo con cui può trasmettere è $1749/(2 * 625 * 10^{-6}) = 1,4$ Mbps.

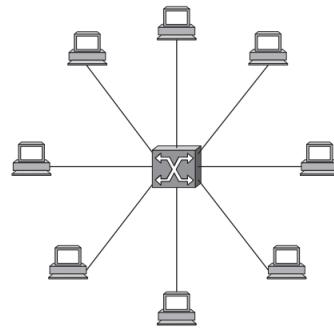
Commutazione di pacchetto

https://www.youtube.com/watch?v=1z0ULvg_pW8

Si è visto come connettere un nodo ad un altro oppure a una rete esistente, ma come è possibile costruire reti di scala mondiale? Una singola rete Ethernet non può collegare più di 1024 host, un collegamento punto-a-punto ne collega soltanto due e le reti wireless sono limitate dal raggio d'azione dei propri apparati radio. Dato che l'obiettivo è quello di costruire reti che possano assumere dimensioni mondiali, il problema successivo da affrontare è, quindi, quello di consentire la comunicazione fra host che non siano direttamente connessi. Sono i commutatori (switch) a dare la sensazione di essere collegati tra host. Le reti di calcolatori usano commutatori di pacchetti per consentire ai pacchetti di viaggiare da un host ad un altro, anche senza che esista una connessione diretta fra tali host. Seguirà l'approccio della commutazione in reti locali (LAN switching) che è frutto di un'evoluzione dell'interconnessione (bridging) di reti ethernet, divenuta una delle tecnologie dominanti nelle reti locali odierne estese (extended LAN).

Switching & forwarding (commutazione e inoltro)

Uno switch è un meccanismo che ci permette di interconnettere link per creare reti più grandi. Uno switch è un dispositivo con molteplici input e output che trasferisce pacchetti da un input ad uno o più output. Il suo compito quindi è quello di ricevere pacchetti in arrivo da una linea e inoltrarli verso l'uscita corretta, in modo che possano raggiungere la propria destinazione. Esistono più modalità mediante le quali uno switch può determinare l'uscita corretta per un pacchetto, modalità che possono essere catalogate in approcci privi di connessione (**connectionless**) e approcci orientati alla connessione (**connection-oriented**). Lo switch permette di implementare la topologia a stella, essa ha alcuni vantaggi:



- Anche se uno switch ha un numero prefissato di ingressi e di uscite, che limita il numero di host che possono essere connessi ad un singolo commutatore, si possono costruire grandi reti interconnettendo un certo numero di switch tra di loro.
- Usando linee di collegamento punto-punto, possiamo connettere uno switch ad un altro switch, oppure ad un host, in modo da realizzare reti ad ampia estensione geografica.
- L'aggiunta di un nuovo host alla rete mediante la sua connessione ad uno switch non implica necessariamente che gli host precedentemente connessi osserveranno una diminuzione di prestazioni della rete. Questo non è vero per le reti a mezzo fisico condiviso, ad esempio l'ethernet. In una rete commutata, ciascun host ha la propria linea di collegamento verso lo switch, per cui è possibile che molti host trasmettano alla piena velocità; purché lo switch sia in grado di gestire una banda in entrata complessiva così elevata.

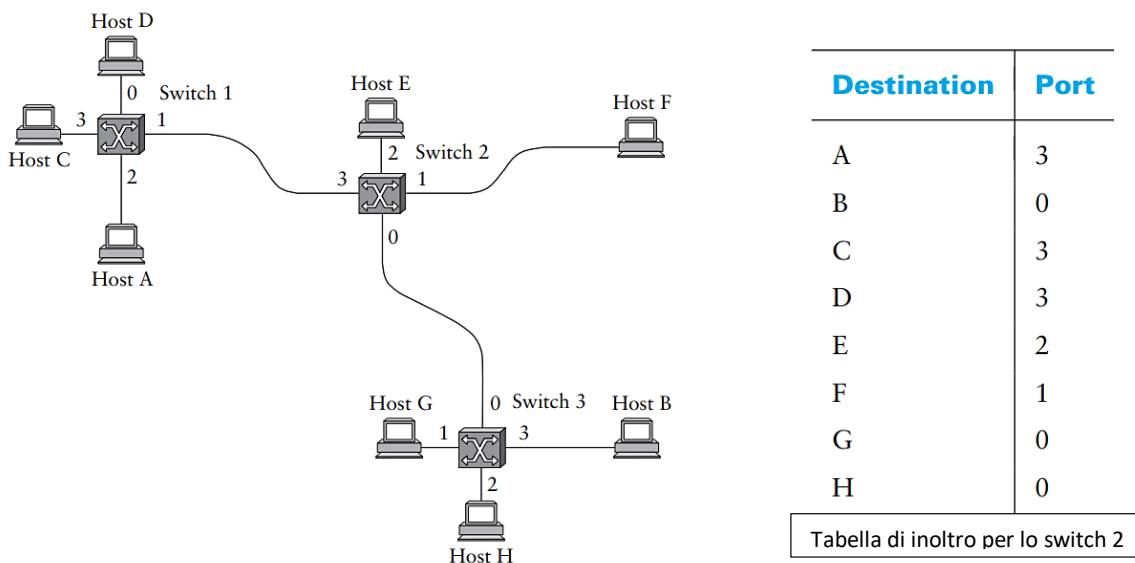
Uno switch è connesso ad un insieme di linee di connessione e, per ciascuna di esse esegue un appropriato protocollo di livello data link (livello 2) per comunicare con il nodo che si trova all'altro capo della linea. Il compito principale di uno switch è quello di ricevere i pacchetti in arrivo da una delle sue linee e di trasmetterli verso qualche altra linea. Questa funzione viene detta **switching o forwarding** (inoltro) ed è la funzione principale dello strato di rete dell'architettura OSI (livello 3).

Come fa uno switch a decidere in quale porta d'uscita inoltrare un pacchetto in entrata? Analizzando l'intestazione del frame può giungere ad una conclusione. Come interpretare l'intestazione dipende dalla modalità utilizzata. Ne esistono principalmente due: **datagram (connectionless)** oppure l'approccio a **circuito virtuale (connection-oriented)**. Esiste anche un terzo approccio, meno comune: **intradamento dalla sorgente (source-routing)**. Una caratteristica necessaria è la presenza di una modalità di identificazione dei nodi terminali. Tale identificazione viene fatta assegnando un indirizzo univoco a ciascun host.

Datagram (connectionless)

L'idea che sta alla base dei datagrammi è incredibilmente semplice: ciascun pacchetto deve contenere informazioni sufficienti per consentire a qualsiasi switch di decidere come fargli raggiungere la propria destinazione, cioè ogni pacchetto contiene l'indirizzo di destinazione completo. Si consideri l'esempio: per decidere come inoltrare un pacchetto, uno switch deve consultare una tabella di inoltro (forwarding table). Costruire la tabella di inoltro è un problema noto come **"intradamento"**, realizzabile a mano per reti molto piccole; in modo automatico mediante complessi algoritmi in caso di reti di grandi dimensioni. Le reti datagram (connectionless) hanno le seguenti caratteristiche:

- Un host può inviare un pacchetto ovunque in qualsiasi momento, perché ogni pacchetto che si presenta in ingresso ad uno switch può venire inoltrato immediatamente.
- Quando un host invia un pacchetto, non ha alcun modo di sapere se la rete sia in grado di consegnarlo, né addirittura se l'host destinatario sia operativo.
- Ciascun pacchetto viene inoltrato in modo indipendente dai pacchetti precedenti che possono esser stati inviati alla medesima destinazione, per cui due pacchetti consecutivi inviati da A verso B possono seguire due percorsi completamente diversi.
- Il malfunzionamento di uno switch o di una linea potrebbe non avere effetti sulle comunicazioni nella rete se è possibile trovare un percorso alternativo.



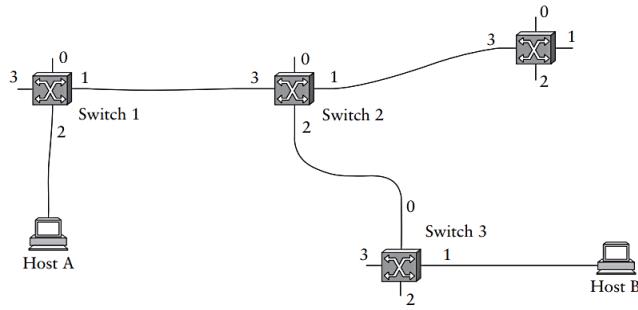
Il problema di come popolare le tabelle di inoltro è noto come **intradamento (routing)** e verrà discusso successivamente.

Virtual Connection (VC)

Una tecnica di commutazione di pacchetto ampiamente utilizzata, che differisce in modo significativo dal modello datagram, utilizza il concetto di circuito virtuale. Questa tecnica richiede che prima di inviare dei dati, venga instaurata una connessione virtuale fra l'host sorgente e l'host destinatario. Esistono quindi due fasi: la **fase di connessione** tra i due host (connection setup) e la **fase di trasferimento** dei dati.

Nella fase di instaurazione della connessione (connection setup) è necessario stabilire uno “stato di connessione” che consiste in un’informazione inserita nella tabella dei “Virtual Circuit” (VC) di ciascun switch attraversato dalla connessione stessa. Ciascuna entry nella tabella contiene:

- Un identificatore del circuito virtuale (**VCI, Virtual Circuit Identifier**) che all’interno dello switch identifica univocamente la connessione e che verrà trasmesso nell’intestazione dei pacchetti che appartengono alla connessione stessa;
- Un’interfaccia di ingresso attraverso cui i pacchetti della VC arrivano allo switch;
- Un’interfaccia di uscita attraverso cui i pacchetti della VC escono dallo switch;
- Un VCI, eventualmente diverso, che verrà usato per i pacchetti uscenti.



La semantica di tali informazioni è la seguente: “se dall’interfaccia di ingresso designata arriva un pacchetto che contiene nella propria intestazione il VCI designato, allora il pacchetto deve essere inviato all’interfaccia di uscita specificata dopo che il VCI specificato per l’uscita è stato inserito nella sua intestazione”.

Si può osservare che, in generale i valori di VCI di ingresso e di uscita non sono gli stessi, per cui il valore di VCI non ha il significato di un identificatore globalmente univoco per la connessione, ma ha significato soltanto su una certa linea di connessione, per cui si dice che ha una visibilità localizzata alla linea (link local scope). Ogni volta che si crea una connessione dobbiamo assegnare un valore di VCI per tale connessione in ciascuna linea che essa attraversa e dobbiamo garantire che il VCI scelto su una certa linea non sia attualmente utilizzato su tale linea da qualche connessione virtuale già instaurata.

Per l’instaurazione dello stato di connessione esistono due grandi categorie di approcci:

- Il primo prevede la presenza di un amministratore di rete che configuri lo stato manualmente. Il circuito virtuale è permanente (fino ad avvenimento di una modifica manuale).
- In alternativa un host può inviare nella rete messaggi che provocano l’instaurazione dello stato: si parla di segnalazioni (signalling) ed i circuiti virtuali che ne risultano vengono detti commutati (SVC, switched virtual circuit). Approccio automatico.

In riferimento all’immagine a pagina successiva, immaginiamo che un admin di rete voglia creare manualmente una nuova connessione virtuale dall’host A all’host B. Per prima cosa deve identificare un percorso valido nella rete tra A e B. Successivamente l’amministratore sceglie per ciascuna linea un valore di VCI per la connessione che sia in quel momento inutilizzato: in questo esempio ipotizziamo venga scelto il valore 5 per la linea che va dall’host A allo switch 1 e il valore 11 per la linea che va dallo switch 1 allo switch 2. In tal caso nella tabella dei VC dello switch 1 deve essere presente una linea di configurazione come quella nell’esempio:

Esempio:

Incoming Interface	Incoming VCI	Outgoing Interface	Outgoing VCI
2	5	1	11

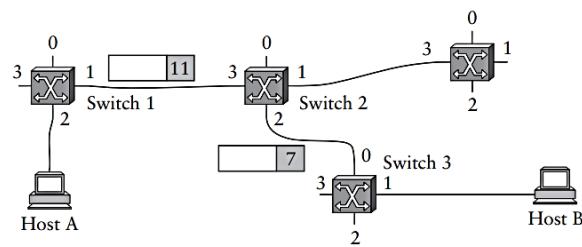
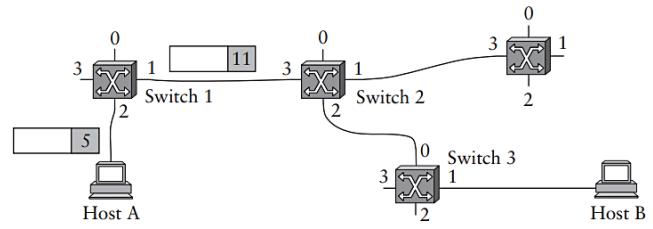
(a)

Incoming Interface	Incoming VCI	Outgoing Interface	Outgoing VCI
3	11	2	7

(b)

Incoming Interface	Incoming VCI	Outgoing Interface	Outgoing VCI
0	7	1	4

(c)



Una volta che sono state popolate correttamente le tabelle dei VC, si può procedere con la fase di trasferimento dei dati. Ogni volta che A vuole inviare un pacchetto all'host B inserisce il valore di VCI uguale a 5 nell'intestazione del pacchetto e lo invia allo switch 1, che riceve il pacchetto dall'interfaccia 2 e utilizza l'appropriata combinazione "interfaccia-input – VCI-input" per determinare leggendo la tabella dei VC l'appropriata combinazione "interfaccia-output – VCI-output".

In una rete reale dalle dimensioni ragionevoli, il compito di configurare le tabelle dei VC in modo corretto in un gran numero di switch diventerebbe rapidamente eccessivo, se si usassero le procedure appena delineate; Per cui si usa quasi sempre qualche forma di segnalazione (signalling) anche per instaurare VC "permanenti".

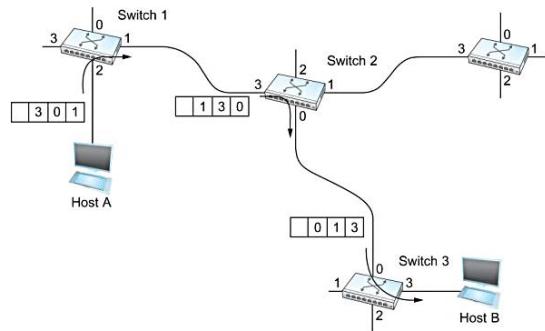
Funzionamento del signalling

Considerando lo schema della rete precedente. Si veda la tabella sopra. Per iniziare:

1. l'host A invia un messaggio di setup nella rete, allo switch 1.
 - a. il messaggio di setup contiene l'indirizzo destinazione di B e deve percorrere una strada verso B completa, per creare una connessione (perché è necessario creare lo stato di connessione per ciascun switch tra A e B facenti parte del percorso).
2. quando lo switch 1 riceve il messaggio di setup, lo inoltra allo switch 2 e crea una entry nella tabella dei virtual circuit con la nuova connessione.
 - a. nel nostro caso: [incoming interface:2 – incoming VCI 5]
 - b. nel nostro esempio lo switch 1 sceglie il valore 5 per la connessione appena creata.
3. quando lo switch 2 riceve il messaggio di setup, esegue le stesse operazioni dello switch 1
 - a. nel nostro esempio sceglie il valore 11 di VCI.
4. similmente lo switch 3 esegue le stesse operazioni.
5. il messaggio di setup arriva all'host B.
6. B accetta la connessione, sceglie un VCI in entrata, che verrà usato per identificare il traffico in arrivo da A.
7. per completare la connessione, ciascun router deve dire al suo vicino "precedente", mediante un ACK quale VCI ha usato per la connessione, così che possano completare le tabelle di VC.
 - a. nel nostro esempio: B invia un ACK allo switch 3 con il valore VCI di 4
 - b. lo switch 3 completa l'entry nella tabella VC e invia un ACK allo switch 2 con il valore di VCI 7. E così via fino ad A.

Source routing (instradamento dalla sorgente)

Un terzo approccio alla commutazione di pacchetto, che non usa né i VC né i datagrammi convenzionali, è noto come instradamento dalla sorgente, il cui nome deriva dal fatto che l'host sorgente fornisce tutte le informazioni relative alla topologia della rete che sono necessarie per inoltrare un pacchetto all'interno della rete stessa. Esistono vari modi per implementare il source routing. Uno di questi modi prevede di assegnare un numero a ciascuna uscita di ciascuno switch e di inserire tale numero nell'intestazione del pacchetto. La funzione di commutazione è quindi molto semplice: ogni volta che arriva un pacchetto ad un ingresso, lo switch legge nell'intestazione del pacchetto il numero della porta e inoltra il pacchetto verso tale uscita. Dato che però in generale vi sarà più di uno switch lungo il percorso tra sorgente e destinazione, l'intestazione deve contenere informazioni per ciascuna porta, un modo di fare ciò sarebbe quello di inserire un elenco ordinato di numero di porta. Attuabile per reti molto piccole di cui si conosce perfettamente la topologia.



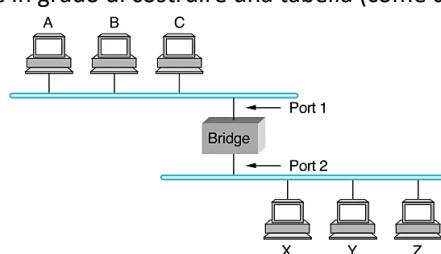
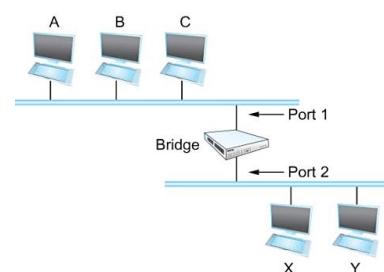
Commutatori per LAN e bridge

Dopo aver discusso alcune idee su cui si fonda la commutazione, concentriamoci ora su alcune specifiche tecnologie di commutazione, iniziando da una categoria di commutatori (switch) che viene usata per inoltrare pacchetti fra reti locali a mezzo fisico condiviso (es: ethernet). Tali commutatori sono a volte chiamati con il nome di LAN switch (commutatori per reti locali).

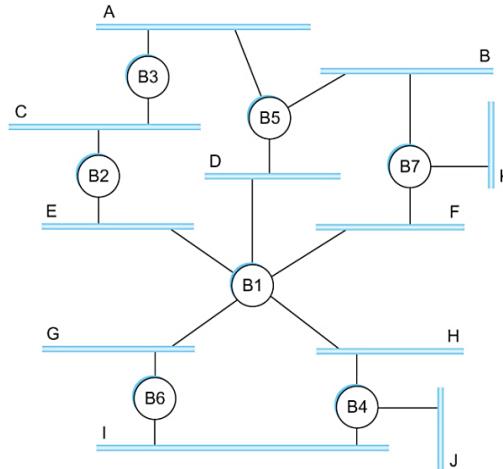
Ipotizziamo di dover interconnettere due reti ethernet. Si potrebbe inserire un nodo fra le due reti ethernet, con il compito di inoltrare i frame da una rete all'altra: questo nodo opererebbe in modalità promiscua, accettando tutti i frame trasmessi su entrambe le reti, in modo da poterli inoltrare da una all'altra. Il nodo appena descritto viene tipicamente chiamato **bridge** (ponte) ed un insieme di reti locali connesse mediante uno o più bridge viene chiamato **rete locale estesa (extended LAN)**. Nella loro forma più semplice i bridge semplicemente accettano i frame delle LAN in entrata e li inoltrano su tutte le LAN collegate alla propria uscita. Questo approccio è stato usato dai primi bridge, ma da allora è stato migliorato per fare dei bridge un meccanismo più efficiente.

Bridge ad apprendimento (learning bridge)

La prima ottimizzazione che possiamo mettere in opera in un bridge consiste nell'osservare che non c'è bisogno che il bridge inoltri tutti i frame che riceve. Considerando il bridge in figura a destra, ogni volta che un frame inviato dall'host A è indirizzato all'host B arriva sulla porta 1. Non c'è bisogno che il bridge lo inoltri sulla porta 2. La domanda, perciò è: come può un bridge apprendere su quale porta risiedono i vari host? L'opzione più semplice prevede un intervento umano. In alternativa, l'idea è che ciascun bridge ispezioni anche l'indirizzo della sorgente di tutti i frame che riceve. In questo modo quando l'host A invia un frame ad un host che si trova su un qualsiasi lato del bridge, il bridge riceve tale frame e memorizza il fatto che il frame proveniente dall'host A è stato ricevuto dalla porta 1. In questo modo il bridge è in grado di costruire una tabella (come quella presentata in figura).



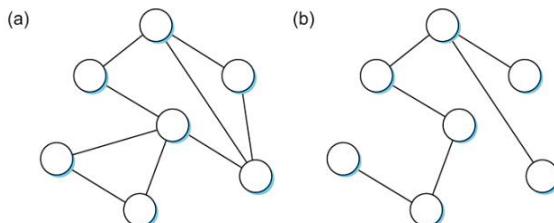
Host	Port
A	1
B	1
C	1
X	2
Y	2
Z	2



La strategia precedentemente illustrata funziona bene fino al momento in cui la rete locale estesa (extended LAN) non presenta anelli (cicli/loop). In quel caso fallisce in modo drammatico: i frame circolano all'infinito nella rete locale.

Esempio: i bridge B1, B4 e B6 formano un anello. Qualunque sia la causa per la quale una rete abbia un loop, i bridge devono essere in grado di gestire correttamente le configurazioni ad anello. Il problema viene risolto facendo eseguire ai bridge un algoritmo distribuito ad albero di copertura (spanning tree). Se si immagina che la LAN estesa venga rappresentata da un grafo che possa avere dei cicli, allora uno spanning tree è un sottografo che ne copre (contiene) tutti i vertici senza avere cicli. In altre parole, uno spanning tree di un grafo, ha gli stessi vertici del grafo originario, ma gli mancano alcuni rami (e non ha cicli).

Esempio: B è un spanning tree di A:



L'algoritmo – spanning tree (RSTP)

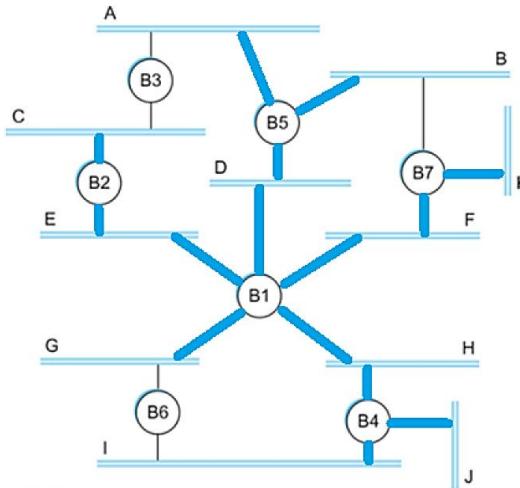
L'algoritmo ad albero di copertura **“Rapid Spanning Tree Protocol - RSTP”** è un protocollo utilizzato da un insieme di bridge per accordarsi su uno spanning tree di una particolare LAN estesa. (la specifica IEEE 802.1 per i bridge di reti locali è basata su questo algoritmo e viene attualmente utilizzata).

In sostanza, ciascun bridge decide verso quali porte inoltrare o non inoltrare frame: la rete locale estesa viene ricondotta ad avere la topologia di un albero aciclico, rimuovendo alcune porte. L'idea principale dello spanning tree consiste nel fatto che sono i bridge a scegliere le porte verso le quali inoltreranno i frame.

L'algoritmo seleziona le porte nel modo seguente:

- ciascun bridge ha un identificatore univoco (nel nostro caso B1, B2, B3 etc.)
- primo passo: l'algoritmo nomina radice dello spanning tree il bridge avente l'identificatore più piccolo. B1 nel nostro esempio.
- ciascun bridge calcola il percorso più breve verso la radice e prende nota di quale delle proprie porte si trova su tale percorso. Tale porta viene selezionata in qualità di percorso preferito dal bridge verso la radice.
- infine, tutti i bridge connessi ad una certa LAN nominano un proprio unico bridge designato, che avrà il compito di inoltrare i frame verso il bridge radice. In ciascuna LAN, il bridge designato è quello più vicino alla radice (il più piccolo identificatore nel caso siano due o più equidistanti).
 - dato che ciascun bridge è connesso a più di una LAN, esso partecipa alla nomina del bridge designato di ciascuna LAN di cui fa parte, in pratica, ciascun bridge decide se essere o meno il bridge designato relativamente a ciascuna delle proprie porte, in modo da inoltrare i frame soltanto verso quelle porte per le quali è il bridge designato.

Esempio: spanning tree della LAN estesa precedentemente illustrata.



NB:

- B1 è la root;
- B5 e B3 sono collegati alla LAN A, ma B5 è il bridge designato all'inoltro dei frame in direzione della LAN A in quanto è il più vicino dei due alla root B1.
- Discorso analogo per B5 e B7 per quanto riguarda la LAN B, B5 è il bridge perché ha l'identificatore più basso tra i due (che si trovano alla stessa distanza).
- Discorso identico al precedente per B4 e B6 per la LAN I.

Problema: mentre una persona può guardare l'intera topologia di una extended LAN, i bridge della rete non possono farlo. I bridge devono scambiarsi messaggi di configurazione e poi decidere basandosi su tali messaggi, di essere radice oppure no e di essere il bridge designato oppure no.

I messaggi scambiati dai bridge contengono tre informazioni:

1. L'identificatore del bridge che invia il messaggio;
2. L'identificatore del bridge radice secondo le informazioni in possesso del bridge che invia il messaggio;
3. La distanza, misurata in hop (segmenti), fra il bridge radice e il bridge che invia il messaggio.

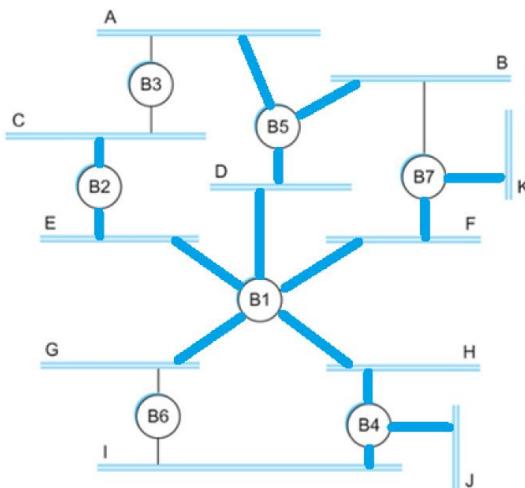
Ciascun bridge memorizza il messaggio di configurazione "migliore" che ha visto su ciascuno delle proprie porte, tenendo conto dei messaggi che ha inviato e ricevuto. All'inizio, ciascun bridge ritiene di essere la radice, per cui invia un messaggio di configurazione in ciascuna delle proprie porte identificandosi come radice e segnalando una distanza dalla radice uguale a 0. Quando riceve un messaggio di configurazione tramite una porta, il bridge verifica se tale messaggio è migliore o no rispetto a quello attualmente considerato come il migliore. Il nuovo messaggio di configurazione è considerato migliore se:

- Identifica una radice con identificatore minore;
- Identifica una radice con lo stesso identificatore ma con una distanza minore,
- L'identificatore della radice e la distanza dalla radice sono uguali, ma il bridge che ha inviato il messaggio ha un identificatore minore.

Se il messaggio ricevuto risulta migliore, il bridge elimina la vecchia informazione e memorizza quella nuova, aumentando di 1 il campo distanza poiché esso si trova a un segmento più distante dalla radice di quanto lo sia il bridge che gli ha inviato il messaggio.

Quando un bridge riceve un messaggio di configurazione da cui deduce di non essere la radice, il bridge smette di generare propri messaggi di configurazione, e invece inoltra soltanto i messaggi di configurazione ricevuti da altri bridge, dopo aver aggiunto 1 al campo distanza. Allo stesso modo, quando un bridge riceve un messaggio di configurazione da cui deduce di non essere il bridge designato per quella porta, il bridge smette di inviare messaggi di configurazione da tale porta.

Il sistema si stabilizzerà, con un solo bridge radice che continua a generare messaggi.

Esempio:

Si consideri l'attività al nodo B3, la sequenza di eventi sarebbe la seguente:

1. B3 riceve (B2,0,B2)
2. Poiché $2 < 3$, B3 accetta B2 come radice.
3. B3 aggiunge 1 alla distanza dichiarata da B2 per cui invia (B2,1,B3) verso B5.
4. Nel frattempo, B2 accetta B1 come radice, perché ha un identificatore minore, e invia (B1,1,B2) verso B3.
5. B5 accetta B1 come radice e invia (B1,1,B5) verso B3.
6. B3 accetta B1 come radice e nota che sia B2 che B5 si trovano più vicini di B3 stesso alla radice, per cui B3 smette di inoltrare messaggi verso le proprie interfacce.
7. Ciò lascia B3 con entrambe le porte non selezionate.

Anche dopo che il sistema si è stabilizzato, il bridge radice continua ad inoltrare tali messaggi di configurazione nel caso che avvengano modifiche alla rete (si guasta uno switch).

Broadcast e multicast

Broadcast: inoltra tutti i frame “broadcast” in tutte le porte, esclusa quella di provenienza.

Multicast: Inoltra tutti i frame “multicast” in tutte le porte, esclusa quella di provenienza, come broadcast.

La discussione precedente si è concentrata sulla modalità di inoltro dei frame di tipo unicast (da una rete LAN ad un'altra). Dato che l'obiettivo di un bridge è quello di estendere in modo trasparente una rete locale coinvolgendo più reti, e poiché la maggior parte delle LAN forniscono supporto ai frame multicast e broadcast, i bridge devono anch'essi fornire supporto per queste due caratteristiche.

Per la trasmissione broadcast è semplice: ciascun bridge inoltra un frame che abbia l'indirizzo broadcast come destinazione verso tutte le proprie porte attive diverse da quella da cui è arrivato il frame.

Per la trasmissione multicast si può realizzare allo stesso modo e ciascun host decide per conto proprio se accettare il messaggio oppure no.

Limiti dei bridge

La soluzione appena vista è adatta ad un utilizzo in situazioni abbastanza limitate, tipicamente per connettere una manciata di reti locali simili. I principali limiti dei bridge diventano evidenti quando si considera il problema della scalabilità e della eterogeneità.

Scalabilità: non è realistico pensare di connettere con bridge più di un numero limitato di reti locali, innanzitutto perché l'algoritmo che calcola lo spanning tree ha costo lineare. Inoltre, è impensabile che in una rete estesa tutti gli host vedano i messaggi broadcast (si pensi a una rete mondiale).

Eterogeneità: ogni parte della rete deve avere indirizzi dello stesso tipo e supportare le stesse features.

Un altro problema può essere quello di due reti con differenti MTUs.

ES: AAL5 è una tecnica per inviare pacchetti di livello 3 su ATM (che è di livello 2). Un pacchetto viene diviso in tanti frame di lunghezza fissa (chiamati celle), ognuno con 48 byte di payload e 5 di intestazione.

L'ultima cella contiene, oltre alla parte finale dei dati, anche 8 byte di informazioni per la ricostruzione del pacchetto. Si calcoli l'overhead (in byte) introdotto da AAL5 per inviare un pacchetto di 1500 byte.

- Risposta:

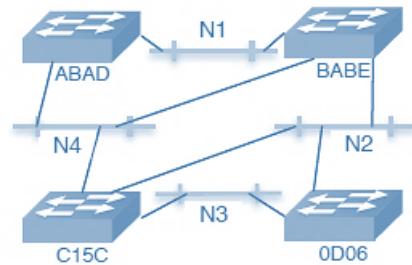
- Numero di celle necessarie: $1500+8/48: 31.416 \rightarrow \mathbf{32}$
- 32 celle significa che ci sono $32*5$ byte di intestazione: **160 byte**
- Ora, considerando che i dati sono 1500 ma abbiamo trasmesso $32*(48+5) = \mathbf{1696 \text{ byte in totale}$ (di cui $31.416 * 48$ di byte utili, 8 di intestazione, $5*32$ di intestazione e $0.584 * 48$ vuoti) di cui **1696-1500 = 196 byte di overhead** (ovvero $8+5*32+0.584 * 48$).

ES: I bridge della rete a lato, i cui id sono scritti in esadecimale, si sono auto configurati con l'algoritmo di spanning tree.

- Qual è il rootbridge?
- Ci sono bridge inattivi (e se sì, quali)?

- Risposta:

- Il rootbridge è **0D06** in quanto l'algoritmo sceglie il root il cui id è di valore minore. Se convertiamo dall'esadecimale gli id, 0D06 risulta il minore.
- ABAD e C15C sono inattivi.** N3 viene raggiunto passando per la root in quanto suo bridge designato. N4 ed N1 vengono raggiunti passando per BABE in quanto loro root designato. Non c'è necessità di ABAD e C15C per raggiungere nessuna rete.



Internetworking

Abbiamo visto come costruire una rete usando linee di collegamento punto-punto, mezzi condivisi e commutatori. Il problema, ora, è far comunicare più reti implementate con diverse tecnologie. Questo capitolo tratta i problemi che nascono dall'interconnessione di reti diverse. L'idea di interconnettere questi diversi tipi di linee di reti è detta **internetworking** (interconnessioni di reti).

Esistono due importanti problemi da risolvere quando si parla di internetworking:

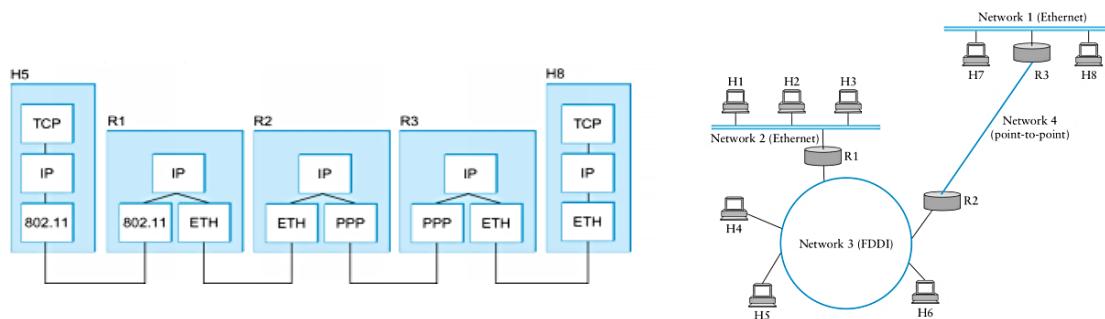
- **Eterogeneità:** consiste nel fatto che gli utenti di una rete di un certo tipo vogliono essere in grado di comunicare con gli utenti di reti di tipi diversi. Per complicare ulteriormente il problema, l'instaurazione di una connessione fra host appartenenti a due reti diverse può richiedere l'attraversamento di parecchie altre reti interposte, che utilizzano tecnologie ancora diverse. Queste diverse reti interposte possono utilizzare schemi di indirizzamento e protocolli diversi. L'obiettivo dell'eterogeneità è quello di fornire un servizio tra host utile e ben prevedibile, usando questo groviglio di reti diverse.
- **Dimensione:** l'instradamento (routing) è uno dei problemi influenzati dalla dimensione di una rete globale. Trovare un percorso efficiente attraverso una rete con miliardi di nodi è un problema molto complesso. A questo segue l'indirizzamento, cioè il compito di fornire identificatori appropriati a tutti questi nodi.

Internet

Con il termine “internetwork”, o a volte soltanto “internet” (i minuscola), ci riferiamo ad un insieme di reti di qualsiasi tipo interconnesse per fornire un servizio di consegna di pacchetti fra host. Con Internet (i maiuscola) ci si riferisce a quella internetwork mondiale, utilizzata da tutti e a cui oggi sono connesse quasi tutte le reti.

Semplice interconnessione di reti (Protocollo IP)

Il protocollo Internet (IP) è lo strumento chiave usato oggi per costruire reti interconnesse scalabili ed eterogenee. Si può immaginare che IP venga eseguito da tutti i nodi (sia host che router) di un insieme di reti e che definisca un'infrastruttura che consente a tali nodi e reti di funzionare dal punto di vista logico come una singola rete interconnessa.



Modello di servizio IP

Un buon punto da cui iniziare quando si progetta una internetwork è la definizione del suo **modello di servizio**: cioè del servizio fra host che si vuol fornire. Si può fornire un certo tipo di servizio fra host soltanto se tale servizio può, in qualche modo, essere fornito da tutte le sottoreti fisiche. La filosofia utilizzata nella definizione del modello di servizio IP, quindi, è stata quella di renderlo così poco esigente da fare in modo che qualsiasi tecnologia di rete che possa comparire all'interno di una internetwork, sia in grado di fornire il servizio necessario.

Il modello di servizio IP può essere immaginato come composto da due parti:

- Uno **schema di indirizzamento** che fornisce il modo di identificare tutti gli host nella rete interconnessa.
- Un **modello datagram** (cioè privo di connessione) per la consegna dei dati.

Questo modello di servizio viene spesso chiamato “best effort” in quanto IP non garantisce alcuna garanzia sulla consegna dei datagrammi, ma ci prova a tutti i costi.

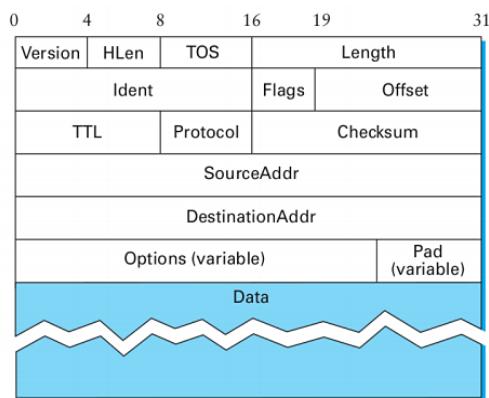
Consegna di datagrammi

Il **datagramma IP** è un elemento fondamentale di IP (Internet Protocol). Il datagramma è un tipo di pacchetto che viene trasmesso all’interno di una rete in modalità priva di connessione. Ogni datagramma porta con sé sufficienti informazioni perché la rete possa inoltrare il pacchetto fino alla sua destinazione corretta. Il pacchetto viene inviato e la rete fa del proprio meglio (best effort) per farlo giungere alla destinazione desiderata.

Concetto di **best effort**: se qualcosa non funziona bene e il pacchetto viene perduto, corrotto, mal indirizzato o per qualsiasi motivo non riesce a raggiungere la propria destinazione, la rete non fa nulla = servizio inaffidabile. La consegna best effort non significa soltanto che i pacchetti possono essere smarriti; a volte vengono consegnati più volte: i protocolli di livello superiore o le applicazioni che operano al di sopra di IP devono essere consapevoli di tutte queste possibilità di malfunzionamento e agire di conseguenza.

Formato del pacchetto (IPv4)

Un ruolo fondamentale nel modello di servizio IP è assunto dal tipo di pacchetti che possono essere trasportati. Il datagramma IP, come la maggior parte dei pacchetti, consiste di un’intestazione seguita da un certo numero di byte di dati. I formati dei pacchetti dello strato di internetwork e degli strati superiori sono quasi tutti progettati per essere allineati su multipli di 32 bit.



Lista dei campi:

- Il campo **Version** specifica la versione di IP, che attualmente è la versione 4 e viene chiamata IPv4.
- Il campo successivo **HLen (header length)**, indica la lunghezza dell’intestazione, misurata in parole di 32 bit. Quando non ci sono opzioni, cosa che accade assai spesso, l’intestazione è lunga 5 parole (20 byte).
- Il campo **TOS (type of service)** di 8 bit ha, come funzione fondamentale, quella di consentire il trattamento differenziato dei pacchetti in base alle necessità delle applicazioni.
- I 16 bit successivi dell’intestazione contengono la **lunghezza (length) del datagramma**, compresa l’intestazione. Diversamente dal campo HLen, il campo Length conta i byte invece delle parole, per cui la dimensione massima di un datagramma IP è di 65535 byte.
- Il byte successivo è il campo **TTL (time to live)**; l’obiettivo di questo campo è di evidenziare quei pacchetti che continuano a viaggiare in percorsi circolari ed eliminarli anziché permettere che consumino risorse indefiniteamente. Il valore assegnato inizialmente a questo campo, se troppo elevato, i pacchetti potrebbero circolare inutilmente per lungo tempo prima di essere eliminati, mentre se troppo basso, potrebbero non riuscire a giungere a destinazione. L’attuale standard è 64.
- Il campo **Protocol** è semplicemente una chiave di demultiplexing che identifica il protocollo di livello superiore a cui va consegnato il pacchetto IP.

- Il valore del campo **Checksum** viene calcolato considerando l'intera intestazione IP come una sequenza di parole di 16 bit, sommandole usando l'aritmetica in complemento a uno e prendendo il complemento a uno del risultato. L'algoritmo ignora qualsiasi pacchetto la cui somma di controllo sia errata. Questo tipo di somma di controllo non ha le stesse robuste proprietà di rilevazione di errore viste in CRC, ma è molto più semplice da calcolare con strumenti software.
- **SourceAddr** rappresenta l'indirizzo del mittente.
- **DestinationAddr** rappresenta il destinatario del pacchetto ed è l'elemento chiave per la consegna in modalità datagram: ogni pacchetto contiene un indirizzo completo della sua destinazione prevista, in modo che ogni router possa prendere le proprie decisioni sull'inoltro.

Frammentazione e ricostruzione

Uno dei problemi da affrontare per fornire un modello di servizio tra host che sia uniforme su un insieme di reti eterogenee consiste nel fatto che ciascuna tecnologia di rete tende ad avere una propria personale opinione relativamente a quella che deve essere la dimensione di un pacchetto. Per IP esistono due possibili scelte:

- garantire che tutti i datagrammi IP siano abbastanza piccoli da poter essere singoli pacchetti in una qualsiasi tecnologia di rete;
- fornire una procedura per mezzo della quale i pacchetti, qualora siano troppo grandi, possano essere frammentati e poi ricostruiti (**scelta adottata da IP**).

La seconda possibilità è quella effettivamente adottata. Questo significa che un host non sarà costretto a inviare pacchetti inutilmente piccoli rispetto alle proprie esigenze, cosa che sprecherebbe banda e consumerebbe risorse di elaborazione degli host.

L'idea fondamentale è che ogni tipo di rete ha una propria unità massima trasmissibile **MTU (maximum transmission unit)**, il più grande datagramma IP che essa può trasportare all'interno di un proprio frame. Quando un host invia un datagramma IP, quindi, può scegliere la dimensione che preferisce. Il valore di MTU della rete a cui l'host è direttamente connesso è una scelta ragionevole: in questo caso la frammentazione sarà necessaria soltanto se il percorso verso la destinazione attraversa una rete con un valore di MTU inferiore. Tipicamente la **frammentazione** avviene in un router, nel momento in cui riceve un datagramma da inoltrare verso una rete avente valore di MTU inferiore alla dimensione del datagramma ricevuto. Per fare in modo che i frammenti possano essere ricostruiti dall'host ricevente, devono avere tutti lo stesso identificatore nel campo Ident, identificatore che viene scelto dall'host sorgente e che deve essere univoco fra tutti i datagrammi che possono arrivare alla destinazione, provenendo da quella sorgente, in un intervallo di tempo ragionevolmente lungo. L'host che deve ricostruire il pacchetto, sarà in grado di riconoscere i frammenti che vanno assemblati insieme; se ne dovesse mancare qualcuno, l'host abbandona il processo di ricostruzione ed elimina i frammenti che sono arrivati: il protocollo IP non tenta di recuperare frammenti mancanti. Vi sono dunque due punti importanti da tenere in considerazione:

- Ciascun frammento è, a sua volta, un datagramma IP completo, che viene trasmesso attraverso una serie di reti fisiche in modo indipendente dagli altri frammenti.
- Ciascun datagramma IP viene nuovamente incapsulato da ogni rete fisica sulla quale viaggia.

La frammentazione produce datagrammi IP più piccoli ma perfettamente validi, che possono essere facilmente sottoposti a un processo di ricostruzione per riottenere il datagramma originario una volta ricevuti, indipendentemente dall'ordine di arrivo. La ricostruzione viene effettuata dall'host destinatario e non dai router. A pagina successiva segue un esempio di frammentazione.

Importante: ciascun frammento (escluso l'ultimo) deve contenere dati con una dimensione multiplo di 8byte.

Si vedano gli esempi sottostanti per una spiegazione più chiara riguardante l'implementazione in IPv4

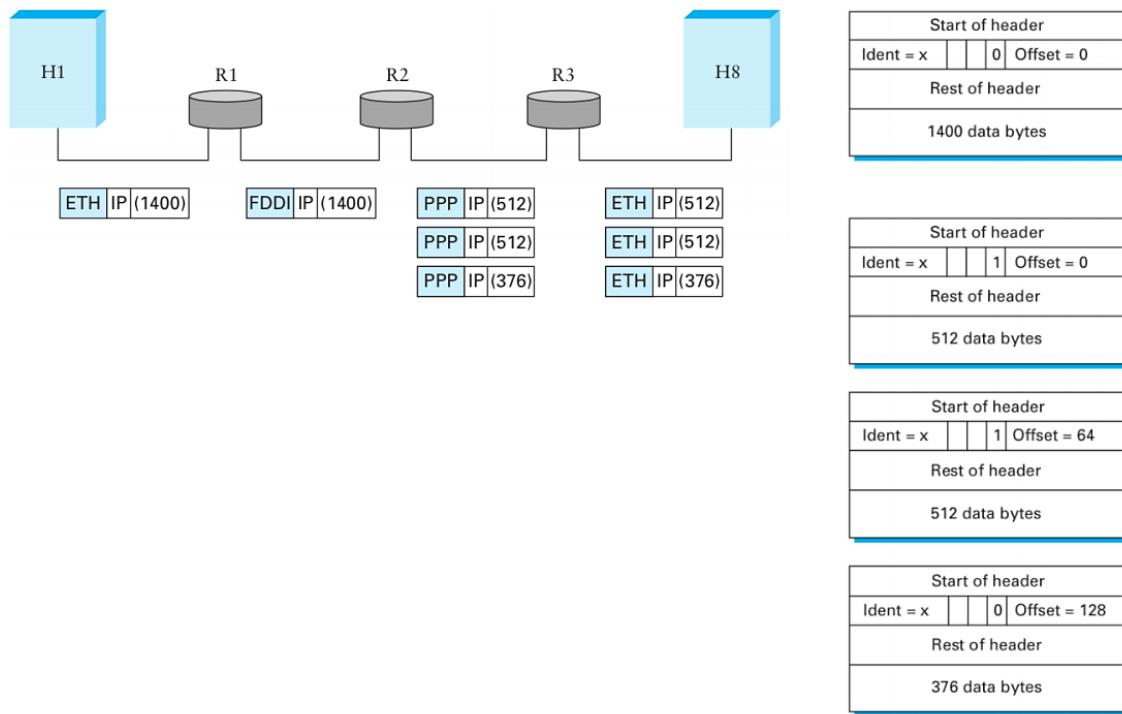
Esempio 1: data: 140byte, intestazione IP: 20byte, MTU: 90, quindi i frammenti (escluso l'ultimo) non conterranno $90-20=70$ byte ma ne conterranno 64 (quindi 64 di dati e 20 di intestazione + 6 byte di scarto/overhead) ovvero il multiplo di 8 più prossimo a 90.

Esempio 2: come si vede dall'immagine in basso, il pacchetto non è frammentato ed è composto da 1400 byte di dati e da un'intestazione IP di 20 byte. Quando il pacchetto arriva al router R2, che ha MTU uguale a 532 byte, deve essere frammentato. Un valore di MTU pari a 532 byte consente di inserire 512 byte di dati dopo l'intestazione IP di 20 byte, per cui il primo frammento contiene 512 byte di dati.

Il router imposta:

- al valore 1 il bit M (more) nel campo Flags, stando a significare che seguiranno ulteriori frammenti;
- al valore 0 il campo Offset, poiché questo frammento contiene la prima parte del datagramma originario.

I dati che vengono inseriti nel secondo frammento iniziano con il 513-esimo byte dei dati originari, per cui il campo Offset di questa intestazione viene impostato al valore 64 (ovvero $512/8$ per scelta dei progettisti di utilizzare multipli di 8 byte). Il terzo frammento contiene gli ultimi 376 byte di dati e ora il campo Offset vale $2 \times 512/8 = 128$. *Dato che si tratta dell'ultimo frammento, il bit M vale 0.*



ES: Un router riceve un pacchetto IP di 1500 byte, compresa l'intestazione di 20 byte, e deve inoltrarlo attraverso una interfaccia che ha una MTU di 520 byte. Quanto è lungo il payload del frammento che ha MoreFragments=0?

- I primi due frammenti portano dei payload di 496 byte, perché 496 è il più grande numero inferiore a $520-20=500$ e divisibile per 8 (perché l'offset va diviso per 8). Quindi il terzo ed ultimo frammento deve avere $1480-(496*2) = 488$

Indirizzi globali

Il modello di servizio di IP deve fornire uno schema di indirizzamento. Se si vuole essere in grado di inviare dati a un qualsiasi host collegato alla rete deve esserci un meccanismo che ci permetta di identificarli. L'unicità globale degli indirizzi è la prima proprietà che dovrebbe essere garantita da un corretto schema di indirizzamento.

Gli indirizzi IP (a differenza di quelli ethernet) sono gerarchici, sono cioè composti da diverse parti, che corrispondono ad una specie di gerarchia all'interno della internetwork. Gli indirizzi IP nello specifico consistono di due parti:

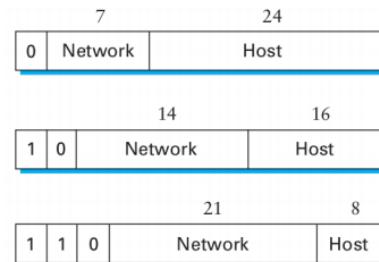
- **un indirizzo di rete:** identifica la rete a cui è connesso l'host, e tutti gli host connessi alla stessa rete hanno lo stesso indirizzo di rete come parte del proprio indirizzo IP.
- **un indirizzo di host:** identifica univocamente ciascun host all'interno della propria rete

NB: se un router è connesso in due reti, deve possedere due indirizzi per ciascuna rete, uno per ciascuna interfaccia.

Come sono fatti questi indirizzi gerarchici? Le dimensioni delle due parti non sono uguali per tutti gli indirizzi; inizialmente gli indirizzi IP erano suddivisi in tre diverse classi; esistono anche indirizzi di classe D per multicast e di classe E attualmente inutilizzati.

La classe di un indirizzo IP viene identificata dai suoi bit più significativi:

- primo bit 0 → classe A
- primi bit 10 → classe B
- primi bit 110 → classe C



Ciascuna classe riserva un certo numero di bit per la parte di rete dell'indirizzo e destina la restante parte dell'indirizzo all'identificazione dell'host nella rete.

- Le reti di **classe A** hanno 7 bit per la parte di rete e 24 bit per la parte di host dell'indirizzo, per cui esistono soltanto 126 reti di classe A (0 e 127 sono riservati ad altro), ma ciascuna di esse può avere $2^{24}-2$ host (circa 16 milioni).
- Gli indirizzi di **classe B** assegnano 14 bit alla parte di rete e 16 bit alla parte di host, per cui ciascuna rete di classe B può avere 65535 host.
- Infine, una rete di **classe C** ha soltanto 8 bit a disposizione per la parte di host dell'indirizzo e 21 bit per la sua parte di rete, quindi può avere soltanto 256 identificatori univoci per gli host e, di conseguenza, può avere solo 254 host connessi.

Per convenzione gli indirizzi IP vengono scritti mediante quattro numeri interi decimali, separati da un punto. Ciascun numero intero rappresenta il valore contenuto in un byte dell'indirizzo, iniziando da quello più significativo.

In una rete, alcuni indirizzi non possono essere utilizzati per gli host:

- tutti 0 nella parte dell'host identifica una rete: ad esempio: 158.110.0.0
- tutti 1 nella parte finale è usato per il broadcast: ad esempio: 158.110.255.255

Inoltro di datagrammi nel protocollo IP

Inoltro: meccanismo di base mediante il quale i router IP inoltrano i datagrammi in una internetwork.
Ricapitolando:

- **Inoltro:** processo che consiste nel ricevere un pacchetto da una porta di ingresso e trasferirlo (inoltrarlo) verso la porta di uscita corretta.
- **Instradamento:** processo di costruzione delle tabelle (di instradamento/routing) che consentono di determinare l'uscita corretta per un certo pacchetto ricevuto in entrata.

Punti principali da tenere a mente nel discutere di inoltro di datagrammi IP:

- Ogni datagramma IP contiene l'indirizzo IP dell'host di destinazione.
- La parte di rete di un indirizzo IP identifica univocamente una singola rete fisica che fa parte di Internet.
- Tutti gli host e tutti i router che condividono la stessa parte di rete del proprio indirizzo sono connessi alla stessa rete fisica e possono, quindi, comunicare tra loro scambiandosi frame su tale rete.
- Ogni rete fisica che faccia parte di Internet ha almeno un router che, per definizione, è anche connesso ad almeno un'altra rete fisica: tale router è in grado di scambiare pacchetti con host e router di entrambe le reti.
- Se un nodo non è connesso alla stessa rete fisica cui è connesso il nodo destinazione, è necessario che invii il datagramma IP ad un router, in generale, ciascun nodo può scegliere tra vari router, per cui deve identificare il migliore. Il router che viene scelto viene chiamato **next hop router**.

L'inoltro di datagrammi IP può quindi avvenire nel modo seguente:

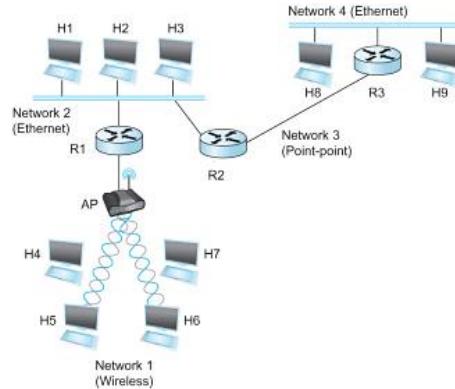
- Un datagramma IP viene inviato da un host sorgente ad un host destinazione, eventualmente attraversando dei router lungo il suo percorso.
- Ogni nodo che sia un router o sia un host, tenta per prima cosa di stabilire se è connesso alla stessa rete fisica della destinazione:
 - per farlo, confronta l'indirizzo di rete contenuto nell'indirizzo di destinazione con l'indirizzo di rete di ciascuna delle proprie interfacce di rete. Se viene verificata la corrispondenza, ciò significa che la destinazione si trova sulla stessa rete fisica cui è connessa l'interfaccia identificata, e il pacchetto può essere consegnato direttamente da tale rete.
- Se il nodo non è connesso alla stessa rete fisica cui è connesso il nodo destinazione, è necessario inviare il datagramma ad un router.
 - Il router che viene scelto prende il nome di "router del salto successivo" (next hop router), il quale troverà il successivo hop consultando la propria tabella di inoltro.

Funzionamento dell'algoritmo

Esempio:

Si supponga che H1 voglia inviare un datagramma a H2

Dato che si trovano sulla stessa rete fisica, H1 e H2 hanno lo stesso numero di rete nel proprio indirizzo IP, per cui H1 deduce di poter consegnare direttamente il datagramma a H2 sulla rete Ethernet. Il problema che rimane da risolvere è come possa H1 trovare l'indirizzo Ethernet corretto per H2? utilizza il meccanismo di risoluzione degli indirizzi descritto successivamente.



Si supponga che H5 voglia inviare un datagramma a H8

Questi host hanno numeri di rete diversi, per cui H5 deduce di dover inviare il datagramma a un router. R1 è l'unica scelta, per cui H5 invia il datagramma a R1 usando la rete wireless. A questo punto R1 sa di non poter consegnare il datagramma direttamente a H8, dato che nessuna delle sue interfacce si trova sulla rete a cui appartiene H8. Si supponga che il router di default di R1 sia R2: R1 invia, quindi, il datagramma a R2 usando la rete Ethernet.

Si ipotizzi, ora, che la tabella di inoltro di R2 sia la seguente:

Numero di rete (NetworkNum)	Salto successivo (NextHop)
1	R1
4	R3

R2 cerca nella tabella il numero di rete di H8 (rete numero 4) e inoltra, conseguentemente, il datagramma a R3, usando la rete costituita dalla linea punto-punto. Infine, R3, sapendo di essere sulla stessa rete di H8, inoltra il datagramma direttamente a H8. È bene notare che si possono inserire nella tabella di inoltro anche le informazioni relative alle reti direttamente connesse. Per esempio, si potrebbero etichettare le interfacce di rete del router R2 come interfaccia 0 per la linea di collegamento punto-punto e interfaccia 1 per la rete Ethernet (rete numero 2). Quindi, R2 avrebbe la tabella di inoltro mostrata di seguito.

Numero di rete (NetworkNum)	Salto successivo (NextHop)
1	R1
2	Interfaccia 1
3	Interfaccia 0
4	R3

In questo modo, per qualunque numero di rete che possa trovare in un pacchetto, il router R2 sa cosa fare. Queste tabelle vengono costruite mediante un protocollo di instradamento che verrà affrontato più avanti. È possibile ora capire come l'indirizzamento gerarchico, con la suddivisione dell'indirizzo nelle sue parti di rete e di host, migliori la scalabilità di una rete di grandi dimensioni. In questo modo i router contengono tabelle di inoltro che elencano soltanto un insieme di numeri di rete.

L'algoritmo in pseudocodice:

```

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) then
    deliver packet to default router
else drop packet

```

Subnetting

Esempio del problema dell'inefficienza nell'assegnamento degli indirizzi: una rete con due soli nodi che usa un'intera rete di classe C spreca 253 indirizzi perfettamente utilizzabili; al pari di una rete di classe B con poco più di 255 host che spreca più di 64000 indirizzi.

Assegnando un numero di rete a ciascuna rete fisica, quindi, si esaurisce lo spazio degli indirizzi IP molto velocemente. Assegnare molti numeri di rete ha anche un altro svantaggio: più numeri di rete vengono utilizzati, più grandi diventano le tabelle di inoltro dei router. Tabelle di inoltro molto grandi rendono più costosi i router e le ricerche al loro interno sono potenzialmente più lente, a parità di tecnologia, rispetto a tabelle di dimensioni minori.

La **suddivisione in sottoreti (subnetting)** è un modo per ridurre la quantità totale di numeri di rete che vengono assegnati. L'idea consiste nel prendere un singolo numero di rete IP ed assegnare gli indirizzi IP aventi quel numero di rete a diverse reti fisiche, che vengono chiamate sottoreti (subnet). Le sottoreti devono essere vicine l'una all'altra: un router sarà in grado soltanto di scegliere un percorso che porti a una qualsiasi delle sottoreti, che quindi è meglio siano localizzate nella stessa direzione all'interno di Internet.

Network number	Host number	
Class B address		
11111111111111111111111111	00000000	
Subnet mask (255.255.255.0)		
Network number	Subnet ID	Host ID
Subnetted address		

Il meccanismo mediante il quale un singolo numero di rete può essere condiviso da più reti richiede la configurazione di tutti i nodi di ciascuna sottorete mediante una **subnet mask** (maschera di sottorete). Quando si usano gli indirizzi IP semplici, tutti gli host che si trovano sulla stessa rete devono avere lo stesso numero di rete. La subnet mask ci consente di introdurre un numero di sottorete: **tutti gli host appartenenti alla stessa rete fisica avranno lo stesso numero di sottorete**, mentre **host che si trovano in reti fisiche differenti possono condividere un singolo numero di rete**.

La maschera di sottorete (subnet mask) quindi, introduce a tutti gli effetti un ulteriore livello di gerarchia negli indirizzi IP.

Esempio: Supponiamo di voler condividere un singolo indirizzo di classe B fra diverse reti fisiche:

Network number	Host number
Class B address	

Supponiamo di utilizzare la seguente subnet mask: Le maschere di sottorete vengono scritte con la stessa notazione degli IP.

11111111111111111111111111	00000000
----------------------------	----------

Subnet mask (255.255.255.0)

I primi 24 bit ora definiscono il numero di rete, mentre gli altri 8 bit vengono usati per il numero dell'host. Dato che i primi 16 bit (dell'indirizzo classe B) identificano la rete, possiamo ora immaginare che l'indirizzo sia composto da tre parti, anziché due:

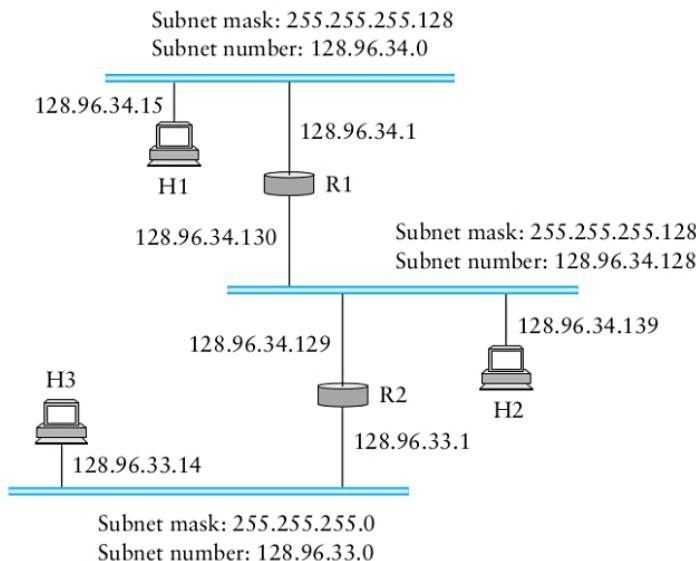
1. Numero di rete;
2. Numero di sottorete;
3. Numero dell'host.

Network number	Subnet ID	Host ID
----------------	-----------	---------

Subnetted address

Per un host la gestione delle sottoreti comporta che la sua configurazione contenga ora un indirizzo IP e una maschera di sottorete, per identificare la sottorete cui è connesso.

Esempio: l'host H1 della figura seguente è configurato con l'indirizzo 128.96.34.15 e la maschera di sottorete 255.255.255.128. (tutti gli host appartenenti alla stessa sottorete sono configurati con la stessa subnet mask).



L'operazione di **AND** bit a bit di questi due numeri definisce il numero di sottorete dell'host e di tutti gli host che fanno parte della stessa sottorete. In questo caso: 128.96.34.15 AND 255.255.255.128 = 128.96.34.0 che è quindi il numero di sottorete.

128.96.34.15:	10000000.01100000.00100010.00001111	AND	
255.255.255.128:	11111111.11111111.11111111.10000000	=	
128.96.34.0:	10000000.01100000.00100010.00000000		

bit identificanti l'id dell'host all'interno della sottorete

bit identificanti la subnet mask

bit identificanti l'id della sottorete

bit identificanti che la rete è di classe B

bit identificanti l'id della rete

conclusioni: **10000000.01100000.00100010.00001111**

128.96.34.15: host **0001111** facente parte della sotto rete **10000000.01100000.00100010.0** della rete di classe B
10000000.01100000

Quando l'host vuole inviare un pacchetto a un certo indirizzo IP, la prima cosa che fa è l'operazione di AND bit a bit fra la propria maschera di sottorete e l'indirizzo IP di destinazione. Se il risultato è uguale al proprio numero di sottorete, allora il mittente sa che l'host destinatario si trova sulla sua stessa sottorete e il pacchetto può essere consegnato direttamente; altrimenti, il pacchetto deve essere inviato a un router per essere inviato a un'altra sottorete.

L'introduzione della suddivisione in sottoreti modifica anche il formato della tabella di inoltro dei router, la quale deve ora contenere dati del tipo: [**NetworkNum**, **SubnetMask**, **NextHop**]

Per trovare nella tabella la riga corretta il router esegue, in successione, l'operazione AND fra l'indirizzo di destinazione del pacchetto e il campo SubnetMask di ogni riga della tabella: se il risultato corrisponde al campo SubnetNumber della riga, allora è stata trovata quella giusta e il pacchetto viene inoltrato al router del salto successivo (NextHop) indicato, appunto, da quella riga. Diverse parti di una internetwork vedono informazioni di instradamento diverse: è un esempio di aggregazione di informazioni di instradamento.

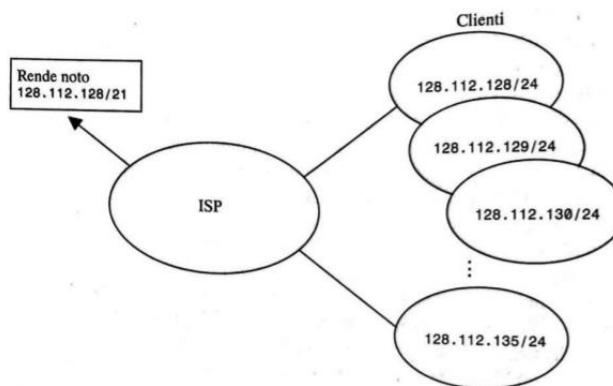
Subnet mask table:

Decimal Mask	Binary Mask	Subnet Bits	Possible Subnets	Hosts Bits	Max Hosts
255	11111111	8	256	0	0
254	11111110	7	128	1*	0*
252	11111100	6	64	2	2
248	11111000	5	32	3	6
240	11110000	4	16	4	14
224	11100000	3	8	5	30
192	11000000	2	4	6	62
128	10000000	1	2	7	126
0	00000000	0	1	8	254

Indirizzamento senza classi (CIDR)

La strategia che prevede l'uso di sottoreti ha una controparte chiamata **instradamento interdominio senza classi (Classless InterDomain Routing - CIDR)**. L'uso delle sottoreti consente solamente la suddivisione di un indirizzo di classe in più sottoreti, mentre CIDR consente di aggregare più indirizzi di classe in un'unica "superrete".

La strategia di CIDR cerca di bilanciare il desiderio di minimizzare il numero di percorsi che un router deve conoscere e la necessità di assegnare gli indirizzi in modo efficiente. Per fare ciò, CIDR consente di usare un'unica riga in una tabella di inoltro per indicare come raggiungere un insieme di reti diverse e realizza questo obiettivo superando i rigid confini posti tra le classi di indirizzi. Per usare CIDR serve un nuovo tipo di notazione per rappresentare dei prefissi. La convenzione è quella di scrivere /X dopo il prefisso, con X lunghezza del prefisso stesso in bit.



Si consideri la figura qui sopra, nell'ipotesi che agli otto clienti serviti dall'ISP siano stati assegnati prefissi di rete a 24 bit tra loro adiacenti (/24), che inizino tutti con gli stessi 21 bit. Dato che tutti i clienti sono raggiungibili tramite il medesimo fornitore di rete, quest'ultimo può rendere noto un unico percorso per tutti, usando semplicemente il prefisso comune condiviso a 21 bit. E questo può funzionare anche se non tutti i prefissi a 24 bit sono già stati assegnati a qualche cliente: è sufficiente che il fornitore abbia acquisito il diritto di assegnare tali prefissi ai suoi clienti. Si può, ad esempio, assegnare anticipatamente al fornitore una porzione dello spazio degli indirizzi, lasciando che sia il fornitore stesso ad assegnare ai propri clienti indirizzi che vi appartengono, quando ne avrà bisogno. Non è necessario che i prefissi di tutti i clienti abbiano la stessa lunghezza.

Rivisitazione dell'inoltro IP

L'uso di CIDR implica che i prefissi possano avere una lunghezza qualsiasi compresa tra 2 e 32 bit. Inoltre, a volte è possibile che alcuni indirizzi possano corrispondere a più di un prefisso [esempio: 171.69 (prefisso a 16 bit) e 171.69.10 (prefisso a 24 bit)]. In questi casi la regola utilizzata è quella della "corrispondenza più lunga" (longest match): il pacchetto viene messo in corrispondenza al prefisso più lungo.

ES: Per ognuna delle seguenti reti, si dica se l'indirizzo 192.168.7.14 vi appartiene o no: (a) 128.0.0.0/1; (b) 192.0.0.0/16; (c) 192.160.0.0/12.

- (a) sì; (b) no; (c) sì.

Traduzione degli indirizzi (ARP)

<https://www.youtube.com/watch?v=cn8Zxh9bPio>

Come i datagrammi arrivano da un particolare host o router dopo aver raggiunto la rete fisica corretta?

I datagrammi IP contengono indirizzi IP, mentre l'interfaccia hardware dell'host o del router al quale vogliamo inviare il datagramma è in grado di interpretare solamente lo schema di indirizzamento della particolare rete a cui appartiene. Di conseguenza dobbiamo tradurre l'indirizzo IP in un indirizzo dello strato di linea di collegamento che abbia senso per quella particolare rete fisica. Possiamo quindi encapsulare il datagramma IP all'interno di un frame che contenga tale indirizzo dello strato di linea di collegamento ed inviarlo alla destinazione finale.

La soluzione più generale è quella di far gestire a ciascun host una tabella di coppie di indirizzi, cioè una tabella che trasformi (traduca) indirizzi IP in indirizzi fisici. L'approccio migliore consiste ne far apprendere dinamicamente i contenuti della tabella a ciascun host tramite la rete stessa, obiettivo che si può raggiungere con il protocollo **ARP (Address Resolution Protocol)**.

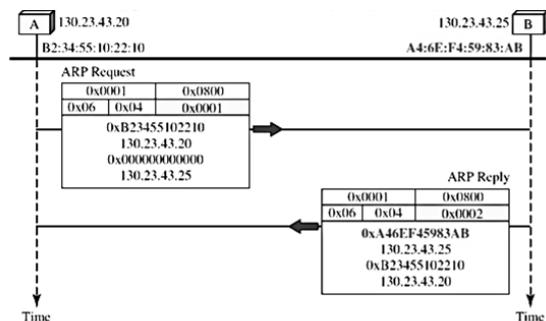
Il compito di ARP è quello di consentire a ciascun host di una rete di costruire tale tabella. Se un host vuole inviare un datagramma IP ad un host (o ad un router) che si trova sulla sua stessa rete, per prima cosa cerca una corrispondenza nella cache (la tabella). Se non la trova, è necessario invocare sulla rete il protocollo ARP, inviando una richiesta mediante un messaggio broadcast nella rete. Tale richiesta contiene l'indirizzo IP in questione (di destinazione) e ciascun host che riceve tale messaggio, controlla se il proprio IP corrisponde. Se corrisponde invia un messaggio di risposta contenente il proprio indirizzo dello strato di linea al richiedente, che può così aggiungere l'informazione alla propria tabella ARP. Ovviamente il messaggio di richiesta contiene anche l'indirizzo IP e indirizzo dello strato di linea dell'host sorgente così che tutti gli host possano aggiornare la corrispondenza nella propria tabella in cache nel caso si ritenga necessario: ovvero quando l'host è l'obiettivo (si ritiene che si instaurerà una comunicazione e che quindi sia necessario conoscere l'indirizzo fisico della sorgente).

Pacchetto ARP

Il pacchetto ARP, oltre agli indirizzi IP e all'indirizzo dello strato di linea della sorgente e della destinazione, contiene:

- Un campo **HardwareType**, che specifica il tipo di rete fisica.
- Un campo **ProtocolType**, che specifica il protocollo dello strato superiore.
- I campi **HLen** (lunghezza dell'indirizzo "hardware") e **PLen** (lunghezza dell'indirizzo "di protocollo"), che indicano, rispettivamente, la lunghezza dell'indirizzo dello strato di linea di collegamento e dell'indirizzo di protocollo di livello superiore.
- Un campo **Operation**, che specifica se si tratta di una richiesta o di una risposta;
- Gli indirizzi hardware (Ethernet) e di protocollo (IP) della sorgente (source) e della destinazione (target).

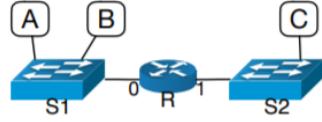
0	8	16	31
Hardware type = 1			ProtocolType = 0x0800
HLen = 48		Operation	
SourceHardwareAddr (bytes 0–3)			SourceProtocolAddr (bytes 0–1)
SourceHardwareAddr (bytes 4–5)		TargetHardwareAddr (bytes 0–1)	
SourceProtocolAddr (bytes 2–3)			TargetHardwareAddr (bytes 2–5)
TargetHardwareAddr (bytes 2–5)			TargetProtocolAddr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



ES: Nella rete a destra, gli indirizzi assegnati alle varie interfacce sono i seguenti:

$R0: 192.168.1.1; R1: 192.168.2.1; A: 192.168.1.50; B: 192.168.1.200;$
 $C: 192.168.2.42.$ (a) Quale può essere il CIDR della rete di A e B? (b) Se A deve inviare un pacchetto IP a C, per quale indirizzo IP deve trovare il corrispondente MAC address, con ARP?

- (a) 24
- (b) Quello dell'interfaccia del router sulla sua rete, ossia 192.168.1.1.



Configurazione di host (DHCP – Dynamic Host Configuration Protocol)

Gli indirizzi IP, oltre a essere unici in una internetwork, devono anche riflettere la struttura della rete stessa. Gli indirizzi IP contengono una componente che identifica la rete e una che identifica l'host all'interno della rete, con la componente di rete che deve essere uguale per tutti gli host di una stessa rete. Di conseguenza, non è possibile che gli indirizzi IP vengano configurati negli host una volta per tutte quando vengono costruiti, perché, per farlo, il costruttore dovrebbe sapere quali host andranno a finire in ciascuna rete esistente al mondo. La maggior parte dei sistemi operativi fornisce uno strumento mediante il quale un utente può configurare manualmente le informazioni IP necessarie all'host; questo processo di configurazione è molto sensibile agli errori. Per questo motivo si rendono necessari metodi di configurazione automatica, il principale dei quali è noto come Dynamic Host Configuration Protocol (DHCP).

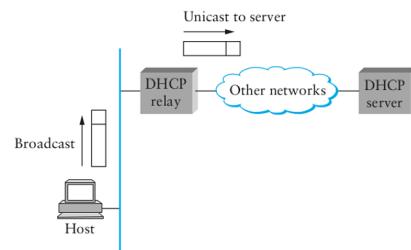
Il **protocollo DHCP** si basa sulla presenza di un server DHCP che ha la responsabilità di fornire agli host le informazioni di configurazione; ha la funzione di banca dati centralizzata per le informazioni di configurazione degli host. DHCP gestisce un insieme di indirizzi disponibili (impostati dall'amministratore), che consegna agli host su richiesta, riducendo sensibilmente il lavoro di configurazione che deve essere svolto dall'amministratore. Dato che il suo obiettivo è quello di minimizzare la quantità di configurazioni manuali necessarie per far funzionare ciascun host, sarebbe sostanzialmente un mancato raggiungimento di tale obiettivo se ogni host dovesse essere configurato con l'indirizzo di un DHCP server, per cui il primo problema che deve essere risolto dal protocollo DHCP è la scoperta di un server.

Per contattare un server DHCP, un host appena acceso o appena connesso invia un messaggio **DHCPDISCOVER** a uno speciale indirizzo IP (255.255.255.255), che è un indirizzo broadcast del protocollo IP; ciò significa che il messaggio verrà ricevuto da tutti gli host e da tutti i router di tale rete.

Nel caso più semplice uno di tali nodi è il server DHCP per la rete, ma non è realistico richiedere che esista un server DHCP su ciascuna rete: il protocollo DHCP utilizza il **concetto di relay agent**:

In ciascuna rete ne esiste almeno uno, che viene configurato con un'unica informazione, ovvero, l'indirizzo IP del server DHCP.

Quando un relay agent riceve un messaggio DHCPDISCOVER, lo inoltra in modalità unicast al server DHCP e attende la risposta, per inoltrarla all'host che aveva inviato la richiesta.



Operation	HType	HLen	Hops
Xid			
Secs			
ciaddr			
yiaddr			
siaddr			
giaddr			
chaddr (16 bytes)			
sname (64 bytes)			
file (128 bytes)			
options			

Un messaggio DHCP viene inviato usando un protocollo chiamato **UDP (User Datagram Protocol)**. Quando cerca di ottenere le informazioni di configurazione, il client DHCP inserisce nel campo chaddr il proprio indirizzo hardware; il server DHCP risponde compilando il campo yiaddr ("il tuo" indirizzo IP) e restituendo il pacchetto al client. Nel campo options può inserire altre informazioni, come l'indirizzo del router di default che deve essere usato dal client.

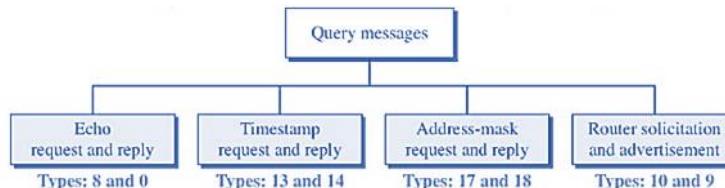
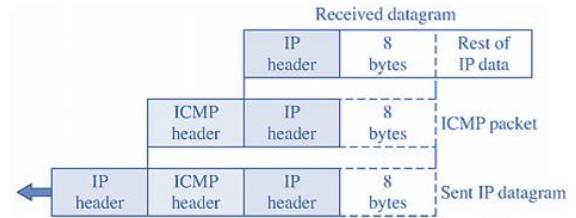
Segnalazione di errori (ICMP - Internet Control Message Protocol)

Gestione degli errori in Internet: il protocollo IP viene sempre affiancato da un protocollo ausiliario, Internet Control Message Protocol (ICMP), che definisce un insieme di messaggi d'errore inviati all'host sorgente quando:

- un nodo (host o router) non è in grado di elaborare con successo un datagramma;
- **un host di destinazione è irraggiungibile;**
- un processo di ricostruzione di un datagramma non è andato a buon fine;
- **il campo TTL di un pacchetto ha raggiunto lo 0;**
- è fallita la somma di controllo (Internet checksum) dell'intestazione di un pacchetto IP;
- c'è un cammino migliore per la destinazione designata.

In tutti i casi solo l'host sorgente sarà avvisato.

Il protocollo ICMP definisce anche messaggi di controllo che un router può restituire a un host sorgente. La parte iniziale del datagramma rifiutato è inclusa nel messaggio ICMP per facilitare l'analisi dell'errore e la scelta di adeguate contromisure.



Il protocollo ICMP viene usato, da due strumenti molto diffusi per il collaudo e l'analisi delle reti:

- **Ping:** il comando ping usa i messaggi echo di ICMP per determinare se un nodo è attivo e raggiungibile;
- **Traceroute:** usa una tecnica per individuare l'insieme di router che si trovano lungo il percorso che porta a una determinata destinazione → usa messaggi di errore ICMP e TTL. Invia un messaggio con TTL=1; il primo router lo rilascia e invia al mittente un errore ICMP, in questo modo il mittente scopre l'indirizzo del primo router e invia un messaggio con TTL=2. Il primo router decrementa TTL e lo inoltra al router successivo, il quale lo rilascia e invia al mittente un errore ICMP. In questo modo il mittente scopre l'indirizzo del secondo router e così via.

Instradamento (Routing)

Fino ad adesso, abbiamo sempre ipotizzato che gli switch e i router avessero sufficienti conoscenze sulla topologia della rete da poter scegliere la porta corretta verso cui inoltrare ciascun pacchetto. Nel caso di circuiti virtuali, l'instradamento è un problema soltanto per i pacchetti che richiedono una connessione, nelle reti a datagramma invece, come le reti IP, l'instradamento è un problema che deve essere risolto per ciascun pacchetto. Uno switch prende questa decisione (di dove inviare un pacchetto) consultando la propria tabella di inoltro, quindi, il problema fondamentale dell'instradamento è: come fanno gli switch e i router ad acquisire le informazioni presenti nelle proprie tabelle di inoltro?

Una riga nella **tabella di inoltro** contiene una corrispondenza tra un numero di rete e un'interfaccia d'uscita, nonché alcune informazioni a livello MAC, come l'indirizzo ethernet della destinazione successiva.

Network Number	Interface	MAC Address
10	if0	8:0:2b:e4:b:1:2

Network Number	NextHop
10	171.69.245.10

La **tabella di instradamento** invece, è la tabella che viene costruita dagli algoritmi di instradamento prima di costruire la tabella di inoltro: in generale, contiene corrispondenze tra numeri di rete e destinazioni successive (next hop), ma può anche contenere informazioni sul modo in cui sono state acquisite tali corrispondenze, per consentire al router di decidere sulla loro eventuale eliminazione.

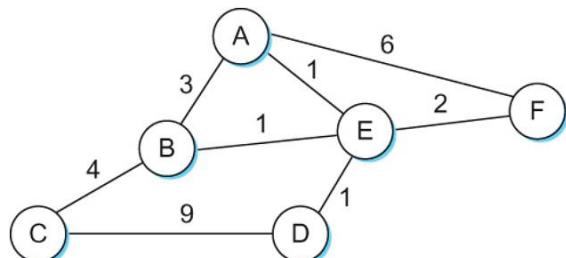
Quindi: la tabella di inoltro è diversa dalla tabella di instradamento.

La tabella di inoltro (**forwarding table**) si crea utilizzando anche la tabella di instradamento (**routing table**). Mantenere distinte queste due tabelle, è una scelta di implementazione, ma vi sono anche buone ragioni per usare strutture dati separate: ad esempio, la tabella di inoltro deve avere una struttura tale da ottimizzare la ricerca di un numero di rete quando si inoltra un pacchetto, mentre quella di instradamento deve essere ottimizzata per calcolare le modifiche nella topologia della rete.

Segue una lista di soluzioni che serviranno come blocco elementare per un'infrastruttura di instradamento gerarchica che viene usata oggigiorno da Internet. I seguenti protocolli vengono chiamati protocolli di instradamento intradominio o **interior gateway protocols (IGP)**.

La rete rappresentata con un grafo

L'instradamento è, sostanzialmente, un problema di teoria dei grafi. I rami nel grafo corrispondono alla linea di collegamento nella rete: a ciascun ramo è associato un costo. I nodi del grafo sono i nodi nella rete. Il problema fondamentale consiste nel trovare il percorso a costo minore fra due nodi qualsiasi. Un approccio statico può essere di provare tutte le combinazioni, ma ha parecchi svantaggi:



- non gestisce i guasti di un nodo o di una linea di collegamento;
- non considera l'inserimento di nuovi nodi o di nuove linee di collegamento;
- implica che i costi associati a ciascun nodo non possano cambiare (surreale).

Per questi motivi, nella maggior parte delle reti si risolve il problema dell'instradamento con protocolli dinamici e distribuiti (distribuiti così da rendere più facile la scalabilità delle reti).

Ipotizziamo che siano noti i costi dei rami nella rete. Prenderemo in esame le due categorie principali di **protocolli di instradamento**:

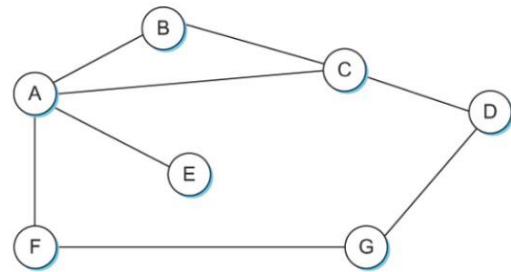
- a vettore di distanza (**distance vector**);
- a stato delle linee (**link state**).

Vettore di distanza – distance vector routing (RIP)

Nell'algoritmo a vettore di distanza (distance vector) ciascun nodo costruisce un array monodimensionale contenente le distanze (i costi) relative a tutti gli altri nodi e distribuisce tale vettore ai suoi più immediati vicini (nell'ipotesi che ciascun nodo conosca il costo delle linee che lo collegano ai vicini a cui è connesso direttamente).

Esempio: ogni nodo è a distanza 1 dai suoi immediati vicini.

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	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0



Inizialmente, ciascun nodo imposta a 1 il costo necessario per raggiungere i vicini a cui è direttamente connesso e a ∞ verso tutti gli altri nodi.

Ad esempio, la tabella relativa al nodo A, all'inizio sarebbe così:

Destination	Cost	NextHop
B	1	B
C	1	C
D	∞	—
E	1	E
F	1	F
G	∞	—

Il passo successivo nell'instradamento a vettore di distanza consiste nell'invio, da parte di ogni nodo ai nodi ad esso direttamente connessi, di un messaggio contenente il proprio elenco di distanze. I vari nodi, analizzando i vettori distanza ricevuti, e possono aggiornare le proprie tabelle eseguendo alcune valutazioni.

Destination	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

In assenza di cambiamenti di topologia, bastano pochi scambi di informazioni fra nodi vicini per fare in modo che tutti i nodi abbiano una tabella di instradamento completa: il procedimento mediante il quale tutti i nodi assumono informazioni di instradamento coerenti viene chiamato **convergenza**. Il risultato finale è mostrato in questa tabella:

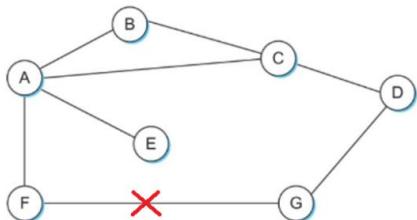
Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

Nell'instradamento a vettore di distanza dobbiamo completare alcuni dettagli: Innanzitutto, notiamo che un nodo decide di inviare un aggiornamento di instradamento i propri vicini in due situazioni diverse.

- Il primo caso avviene in occasione di un **aggiornamento periodico**: ciascun nodo invia un messaggio di aggiornamento automatico ad intervalli regolari, anche se non ci sono state modifiche: questo fa sapere agli altri nodi che il nodo è ancora attivo e consente di continuare a ricevere informazioni che potrebbero essere loro utili nel caso che qualche percorso diventi inagibile. La frequenza di questi aggiornamenti periodici varia di protocollo in protocollo, ma è tipicamente dell'ordine di alcuni secondi o pochi minuti.
- Il secondo meccanismo, a volte chiamato **aggiornamento innescato (triggered)**, entra in gioco quando un nodo riceve da uno dei suoi vicini un aggiornamento che provoca la modifica di uno dei percorsi all'interno della sua tabella di instradamento: in sostanza, ogni volta che viene modificata la tabella di instradamento di un nodo, il nodo invia un aggiornamento ai propri vicini, provocando eventualmente una modifica nelle loro tabelle, con conseguente massaggio di aggiornamento inviato ai loro vicini.

I guasti vengono identificati/monitorati applicando i seguenti accorgimenti:

- Un nodo tiene costantemente sotto controllo una linea di collegamento verso un altro nodo, inviando pacchetti di controllo e osservando se riceve la relativa conferma.
- Un nodo decide che una linea è guasta quando non riceve gli aggiornamenti periodici di instradamento per un determinato intervallo di tempo.



Esempio: segue un esempio di guasto e di come esso venga risolto dal protocollo:

1. si consideri cosa avviene quando il nodo F si accorge che la linea verso G non funziona correttamente;
2. dapprima, F imposta a ∞ la propria distanza verso G e trasmette tale informazione ad A;
3. poiché A sa di poter raggiungere G lungo un percorso di due salti che passa proprio attraverso F, anche A imposta a ∞ la propria distanza verso G;
4. tuttavia, ricevendo il successivo aggiornamento da C, A viene a sapere che C può raggiungere G con un percorso di 2 salti, per cui A calcola di poter raggiungere G passando per C con un numero di salti uguale a 3, che è minore di ∞ , e aggiorna di conseguenza la propria tabella;
5. quando A segnala il cambiamento a F, F apprende di poter raggiungere G tramite A al costo di quattro salti, che è minore di ∞ , e il sistema diviene nuovamente stabile.

Problema del conteggio a infinito: in alcune sfortunate circostanze, può capitare che un guasto alla rete non permetta alla rete di stabilizzarsi nuovamente. Supponiamo, ad esempio, che si guasti la linea di collegamento tra A ed E. Nel successivo ciclo di aggiornamenti, A segnala di trovarsi a distanza infinita da E, ma B e C segnalano di trovarsi a distanza 2 da E. In relazione al preciso succedersi degli eventi, può accadere quanto segue: il nodo B, dopo aver saputo che tramite C si può raggiungere E in 2 salti, deduce di poter raggiungere E in 3 salti e lo segnala ad A, il nodo A, deduce quindi di poter raggiungere E in 4 salti e lo segnala a C che dedurrà di poterlo raggiungere in 5 salti, e via così. Questo ciclo termina soltanto quando le distanze raggiungono un valore sufficientemente elevato da essere considerato infinito; nel frattempo nessun nodo realizza che E è irraggiungibile e le tabelle di instradamento non si stabilizzano. Questo problema viene detto problema del conteggio a infinito (count-to-infinity).

Una soluzione parziale è quella di utilizzare un numero relativamente piccolo per rappresentare l'infinito, ad esempio 16. Una tecnica che migliora il tempo necessario alla stabilizzazione dell'instradamento è detta **divisione dell'orizzonte (split horizon)**.

L'idea consiste nel: quando un nodo invia un aggiornamento di instradamento ai propri vicini, non invia ad un vicino quei percorsi che ha appreso da esso.

Per esempio, se B nella propria tabella ha il percorso (E, 2, A), allora sa di averlo appreso dal nodo A: quando invia un aggiornamento al nodo A, non include il percorso (E, 2) nell'aggiornamento.

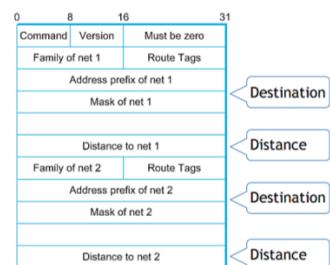
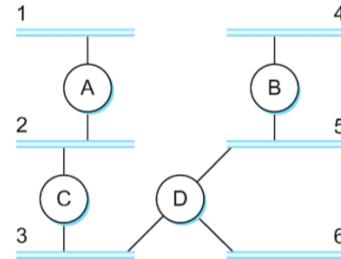
Una variante più robusta è detta **split horizon with poison reverse**, in cui B in realtà invia anche tale percorso ad A, ma con informazioni così negative da impedire che A usi B per raggiungere E (es. (E, ∞)).

Il problema di entrambe le soluzioni è che funzionano soltanto per eliminare i cicli di instradamento che coinvolgono due soli nodi.

Routing Information Protocol (RIP)

RIP è l'esempio canonico di protocollo di instradamento basato sull'**algoritmo a vettore di distanza** appena descritto.

Nelle internetwork, i protocolli di instradamento sono leggermente diversi dal modello di grafo idealmente descritto in precedenza. In una internetwork, l'obiettivo dei router è quello di apprendere come inoltrare pacchetti ad altre reti, per cui, anziché comunicare il costo (la distanza) necessario per raggiungere altri router, i router comunicano il costo per raggiungere altre reti.

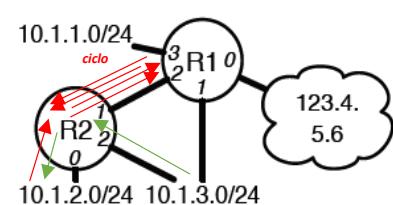


Il protocollo RIP è, a tutti gli effetti, un'implementazione pedissequa dell'instradamento a vettore di distanza. I principi che stanno alla base dell'instradamento non cambiano: se il router A apprende dal router B che la rete X può essere raggiunta tramite B a un costo inferiore rispetto a quanto possibile tramite next hop presente nella propria tabella di instradamento, A aggiorna di conseguenza il costo e le informazioni del next hop relative al quel numero di rete. **I router che eseguono RIP inviano i loro aggiornamenti ogni 30s**, oltre ad inviare un messaggio di aggiornamento ogni volta che un aggiornamento ricevuto da un altro router provoca una modifica nella propria tabella di instradamento. RIP non è limitato agli indirizzi IP, ma supporta più famiglie di indirizzi.

ES: Le interfacce e le tabelle di instradamento dei router in figura sono come seguono:

- R1:if1 = 10.1.3.1
- R1:if2 = 192.168.2.1
- R1:if3 = 10.1.1.1
- R2:if0 = 10.1.2.1
- R2:if1 = 192.168.2.2
- R2:if2 = 10.1.3.2

dest	if	next hop
10.1.1.0/24	3	-
10.1.2.0/23	2	192.168.2.2
/*	0	123.4.5.6
dest	if	next hop
10.1.2.0/24	0	-
/*	2	10.1.3.1



(a) Cosa succede ad un pacchetto inviato da 10.1.3.5 a 10.1.2.5?

(b) E al pacchetto di risposta, da 10.1.2.5 a 10.1.3.5?

NB: /* significa default gateway

- Risposta:

(a) L'host 10.1.3.5 può utilizzare R2 come gateway, il quale inoltrerà il pacchetto a destinazione attraverso l'interfaccia 0. Oppure 10.1.3.5 invia il pacchetto a R1, il quale applica la seconda regola e lo inoltra a R2, che lo consegna. In ogni caso, il pacchetto arriva a destinazione.

(b) R2 inoltra il pacchetto al suo default gateway, ossia 10.1.3.1 (potrebbe fare una consegna diretta, ma la tabella non glielo permette in quanto next-hop sulla regola non è definito). R1 inoltra il pacchetto a R2 in base alla seconda regola, e quindi il pacchetto inizia a ciclare tra R1 e R2, finché il TTL non scende a 0 e viene scartato.

Stato delle linee – link state routing (OSPF)

L'instradamento tramite stato delle linee (link state routing) costituisce la seconda principale categoria dei protocolli di instradamento intradominio. Si assume che ciascun nodo sia in grado di conoscere lo stato delle linee (funzionanti o non funzionanti) che lo collegano ai propri vicini e, di nuovo, vogliamo fornire a ciascun nodo sufficienti informazioni per trovare il percorso di costo minimo verso qualsiasi destinazione.

L'idea che sta alla base del link state routing è la seguente: ciascun nodo sa come raggiungere i vicini a cui è connesso direttamente e, se garantiamo che la totalità di queste informazioni venga trasmessa ad ogni nodo, allora ogni nodo avrà sufficienti informazioni sulla rete da poter costruire una mappa completa della rete stessa.

Quindi, i protocolli di instradamento mediante lo stato delle linee si affidano a due meccanismi:

- la **trasmissione affidabile delle informazioni** relative allo stato delle linee di collegamento;
- il **calcolo di percorsi** a partire dall'insieme delle informazioni accumulate.

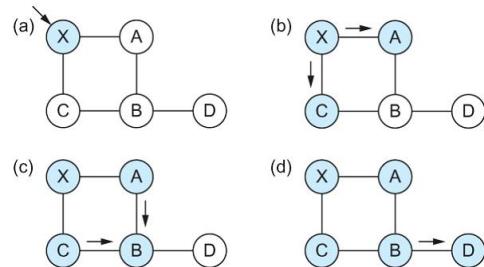
Inondazione affidabile

L'inondazione affidabile (**reliable flooding**) è un processo che garantisce a tutti i nodi che partecipano al protocollo di instradamento di ricevere da tutti gli altri nodi una copia delle informazioni relative allo stato delle linee di collegamento.

L'idea si basa sul fatto che un nodo invii le proprie informazioni verso tutte le linee ad esso connesse. Questo processo continua finché l'informazione non ha raggiunto tutti i nodi della rete. Più precisamente, ciascun nodo crea un pacchetto di aggiornamento (**LSP link state packet**) che contiene:

- l'identificativo (**ID**) del nodo che ha creato il pacchetto;
- un elenco dei vicini direttamente connessi a quel nodo, con il costo della linea di collegamento verso ciascun nodo;
- un numero di sequenza;
- il numero di vita del pacchetto.

L'inondazione funziona nel modo seguente: viene resa affidabile la trasmissione degli LSP fra router adiacenti usando conferme (ACK) e ritrasmissioni. L'affidabilità consiste, anche, nel garantire che ciascun nodo possieda la copia più recente delle informazioni.



Esempio: Si consideri il nodo X, che riceve una copia di un LSP generato da un altro nodo Y. X controlla di aver memorizzato un'altra copia di un LSP proveniente da Y: se non ne ha, memorizza il pacchetto appena ricevuto, altrimenti confronta i numeri di sequenza. Se il pacchetto nuovo ha un numero di sequenza maggiore di quello memorizzato, si assume che sia più recente e viene memorizzato andando a sostituire quello precedente. Un numero di sequenza inferiore (o uguale) contraddistingue, invece, un LSP meno recente di quello memorizzato in precedenza, per cui il pacchetto verrebbe ignorato.

Se il pacchetto ricevuto è il più recente, X invia una copia di tale LSP a tutti i propri vicini tranne al vicino da cui ha ricevuto l'LSP, nel nostro esempio ad A e C, che eseguiranno le stesse operazioni.

Proprio come nel protocollo RIP, ciascun nodo genera nuovi LSP in due occasioni:

- dopo la scadenza di un temporizzatore periodico;
- in seguito ad una modifica della topologia della rete.

L'unica motivazione topologica che induce un nodo a generare un LSP è il guasto di una linea di collegamento a esso direttamente connessa, oppure il guasto di uno dei nodi a esso vicini.

Il guasto di un nodo vicino o la perdita di connessione verso esso si può identificare usando periodici pacchetti di "saluto" ("hello" packets): ciascun nodo li invia a intervalli predefiniti ai propri vicini e la linea di collegamento verso un vicino viene dichiarata guasta se per un tempo sufficientemente lungo non si ricevono pacchetti di saluto dal quel vicino.

Uno degli obiettivi principali nel meccanismo di inondazione è che le informazioni più recenti vengano trasmesse a tutti i nodi il più velocemente possibile, mentre le informazioni obsolete vengano rimosse dalla rete e non fatte più circolare. Altro obiettivo è la minimizzazione della quantità totale di traffico generato nella rete (carico inutile: "overhead").

- Un modo semplice per ridurre l'overhead consiste nell'evitare di generare LSP a meno che non sia assolutamente necessario, usando temporizzatori molto lunghi. Questo è possibile perché quando vi sono modifiche topologiche, il protocollo di inondazione è veramente affidabile.
- Per garantire che le informazioni obsolete vengano sostituite da quelle più recenti, i pacchetti LSP contengono numeri di sequenza. Ogni volta che un nodo genera un nuovo LSP, incrementa di un'unità il numero di sequenza. Non è previsto che tali numeri tornino al valore iniziale, per cui il campo relativo deve essere molto grande.

I pacchetti LSP hanno anche un tempo di vita (TTL → time to live), usato per garantire che le informazioni obsolete relative allo stato delle linee vengano prima o poi eliminate dalla rete. Prima di trasmettere ai propri vicini un LSP ricevuto, un nodo ne decrementa sempre il campo TTL; inoltre ciascun nodo "fa invecchiare" periodicamente i pacchetti LSP memorizzati al proprio interno. Quando il campo TTL di un LSP raggiunge il valore 0, il nodo ritrasmette tale pacchetto con TTL a zero, evento che viene interpretato da tutti i nodi della rete come un segnale di rimozione per quel pacchetto.

Calcolo dei percorsi più corti

Dopo aver ricevuto una copia del pacchetto LSP di tutti gli altri nodi, ogni nodo è in grado di calcolare una mappa completa della topologia della rete e decidere quale sia il percorso migliore verso qualsiasi destinazione. Come vengono calcolati precisamente i percorsi a partire da queste informazioni? Tramite l'algoritmo di Dijkstra sotto descritto.

Dijkstra's algorithm

Immaginando che un nodo esami tutti i pacchetti LSP che ha ricevuto e costruisca una rappresentazione della rete, in cui

- N indica l'insieme dei nodi del grafo;
- $l(i, j)$ indica il costo (non negativo) (peso) associato al ramo che esiste fra i nodi i, j appartenenti a N , con $l(i, j) = \infty$ (se non esiste un ramo che collega i e j).

L'algoritmo gestisce due variabili:

- M che indica l'insieme di nodi finora presi in considerazione dall'algoritmo stesso;
- $C(n)$ che indica il costo del percorso dal nodo s al nodo n .

Pseudocodice:

```

 $M = \{s\}$ 
for each  $n$  in  $N - \{s\}$ 
   $C(n) = l(s, n)$ 
while ( $N \neq M$ )
   $M = M \cup \{w\}$  such that  $C(w)$  is the minimum for all  $w$  in  $(N - M)$ 
  for each  $n$  in  $(N - M)$ 
     $C(n) = \text{MIN}(C(n), C(w) + l(w, n))$ 
```

Descrizione dell'algoritmo: si parte con M contenente il nodo s e si genera la tabella dei costi (i valori C(n)) verso gli altri nodi, usando soltanto i costi noti per raggiungere i nodi direttamente connessi. Successivamente, si cerca il nodo raggiungibile al costo minimo (w) e lo si inserisce in M. Infine, si aggiorna la tabella dei costi considerando il costo necessario per raggiungere nodi tramite w. Nell'ultima linea di codice viene scelto come nuovo percorso verso il nodo n quello che passa per il nodo w se il costo totale necessario per andare dalla sorgente a w è, poi, da w a n è inferiore al costo del vecchio percorso memorizzato verso n.

In pratica, ciascuno switch calcola la propria tabella di instradamento usando un'implementazione dell'algoritmo di Dijkstra chiamato **forward search algorithm**.

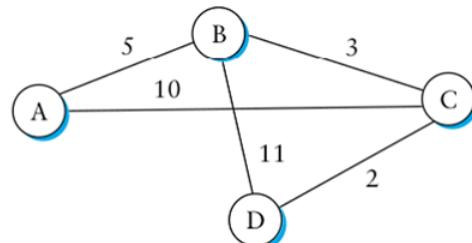
Ciascuno switch utilizza due liste: **Tentative** e **Confirmed**, ognuna contenente un insieme di dati aventi la forma **<Destination, Cost, NextHop>**.

L'algoritmo implementato negli switch funziona nel modo seguente:

1. Inizializzo la lista Confirmed con il dato relativo a me stesso, con costo 0.
2. Seleziono il pacchetto LSP relativo al nodo (chiamato Next) appena inserito nella lista Confirmed nel passo precedente.
3. Per ciascun vicino (Neighbor) di Next, calcolo il costo (Cost) necessario per raggiungere Neighbor come somma del costo da me stesso a Next e del costo da Next a Neighbor.
 - a. Se Neighbor non è nella lista Confirmed né nella lista Tentative, aggiungo alla lista Tentative, dove NextHop è la direzione verso cui devo andare per raggiungere Next.
 - b. Se Neighbor è nella lista Tentative e Cost è minore del costo attualmente associato a Neighbor, sostituisco il dato attualmente memorizzato per Neighbor con, dove NextHop è la direzione verso cui devo andare per raggiungere il Next.
4. Se la lista Tentative è vuota, l'algoritmo termina; altrimenti scelgo dalla lista Tentative il dato avente il costo inferiore, spostandolo nella lista Confirmed e tornando al passo 2.

Esempio:

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.



L'algoritmo di instradamento a stato delle linee giunge rapidamente alla stabilità, non genera molto traffico e risponde con prontezza a modifiche della topologia o a guasti di nodi. Di contro, la quantità di informazioni memorizzata in ciascun nodo (pari a un LSP per ogni altro nodo della rete) può essere molto grande (problema di scalabilità).

Differenza tra gli algoritmi a vettore di distanze e a stato delle linee

- Con il **vettore di distanza** ciascun nodo scambia informazioni solamente con i vicini a cui è direttamente connesso, ma trasferisce loro tutte le informazioni che ha appreso, cioè la distanza verso tutti gli altri nodi.
- Con lo **stato delle linee** ciascun nodo scambia informazioni con tutti gli altri, trasferendo solamente le informazioni di cui è assolutamente certo, cioè soltanto lo stato delle linee a cui è direttamente connesso.

Open Shortest Path First (OSPF)

Uno dei protocolli a stato delle linee più diffusi è OSPF (Open shortest path first). OSPF aggiunge all'algoritmo a stato delle linee descritto in precedenza alcune caratteristiche:

- **Autenticazione dei messaggi di instradamento:** nel caso in cui un host configurato male decide di poter raggiungere qualsiasi altro host a costo zero, questo host trasmette questo tipo di informazione; tutti i router adiacenti aggiornano le proprie tabelle di instradamento, puntando a tale host, di conseguenza esso riceverà grandi quantità di dati che non riuscirà a gestire. Tipicamente li ignora tutti bloccando la rete. Tale fenomeno disastroso può essere molte volte evitato richiedendo che gli aggiornamenti di instradamento vengano autenticati.
- **Ulteriore gerarchia:** la gerarchia è uno strumento fondamentale per rendere scalabili i sistemi. Il protocollo OSPF introduce un ulteriore livello gerarchico nell'instradamento, consentendo la suddivisione di un dominio in aree. Ciò significa che un router non deve necessariamente conoscere come raggiungere ciascuna rete all'interno del dominio, ma può sufficientemente sapere soltanto come raggiungere l'area giusta.
- **Bilanciamento del carico:** il protocollo OSPF consente che a più percorsi diretti verso la medesima destinazione venga assegnato lo stesso costo, distribuendo in maniera paritetica il traffico.

Version	Type	Message length
SourceAddr		
AreaId		
Checksum		Authentication type
Authentication		

Struttura del messaggio OSPF: esistono diversi tipi di messaggi OSPF, ma tutti hanno la medesima intestazione.

- Il campo **Version** indica la versione del protocollo;
- Il campo **Type** può assumere valori compresi tra 1 e 5;
- Il campo **SourceAddr** identifica il mittente del messaggio;
- Il campo **AreaId** è un identificativo a 32 bit per l'area in cui si trova il nodo. L'intero pacchetto è protetto da una somma di controllo (**CheckSum**) a 16 bit;
- Il campo **Authentication type** vale 0 se non viene usata alcuna autenticazione, altrimenti può valere 1 (imponendo l'utilizzo di una semplice password) oppure 2 (che sta a indicare l'uso di somma di controllo). In questi ultimi casi il campo Authentication contiene la password oppure la somma di controllo.

Tra i cinque messaggi OSPF, quello di tipo 1 è il messaggio "hello", un "saluto" che viene inviato da un router ai suoi vicini per segnalare il proprio corretto stato di funzionamento e di connessione, come descritto in precedenza. Gli altri tipi di messaggio vengono usati per chiedere, inviare e confermare la ricezione di

messaggi relativi allo stato delle linee. Il protocollo OSPF deve fornire informazioni in merito a come raggiungere le diverse reti che la compongono. Nello specifico, un router che esegue OSPF può generare pacchetti (relativi allo stato delle linee) che notificano come raggiungibili una o più delle reti direttamente connesse a quel router. Inoltre, un router che sia connesso a un altro router mediante una linea, deve dare informazioni in merito al costo di utilizzo di tale linea per raggiungere quel router.

Il blocco elementare che costituisce i messaggi del protocollo OSPF viene chiamato **LSA (link-state-advertisement – segnalazione dello stato di una linea)**. Segue il formato del pacchetto per un'informazione di tipo 1 sullo stato della linea. I messaggi LSA di tipo 1 informano sul costo delle linee di connessione fra router, mentre quelli LSA di tipo 2 vengono usati per indicare quali reti siano connesse al router che li invia.

LS Age		Options	Type=1
Link-state ID			
Advertising router			
LS sequence number			
LS checksum		Length	
0	Flags	0	Number of links
Link ID			
Link data			
Link type	Num_TOS	Metric	
Optional TOS information			
More links			

- Il campo **LS Age** ha lo stesso significato del campo TTL.
- Il campo **Type** contenente il valore 1 indica che si tratta di un LSA di tipo 1.
- In un **LSA** di tipo 1 i campi Link-state ID e Advertising router sono identici: ciascuno contiene un identificatore a 32 bit del router che ha generato l'informazione (essenziale che sia univoco all'interno del dominio).
- Il campo **LS sequence number** è usato nel modo descritto in precedenza, per identificare LSA obsoleti o duplicati.
- Il campo **LS checksum** viene utilizzato per verificare che i dati non siano stati corrotti.
- Il campo **Length** contiene la lunghezza, in byte, dell'intero LSA.
- Ciascuna linea di collegamento presente in un LSA è rappresentata da
 - un identificativo (**Link ID**)
 - alcuni dati denominati **Link data**
 - una metrica **Metric**: i primi due campi identificano la linea; il campo Metric rappresenta il costo associato alla linea.
- L'informazione sul tipo di servizio (**TOS**) è presente per consentire al protocollo OSPF di scegliere percorsi diversi per i pacchetti IP sulla base del valore presente nel loro campo TOS. Invece di assegnare a una linea un singolo costo, è possibile assegnarle costi diversi in relazione al valore di TOS dei dati.

ES: I router della rete a lato utilizzano RIP e hanno raggiunto la configurazione stabile.

(a) Si dia la tabella di instradamento di B.

(b) Ad un certo punto, il collegamento B-D si interrompe, e subito dopo B riceve il vettore delle distanze di A. Come diventa la tabella di instradamento di B?

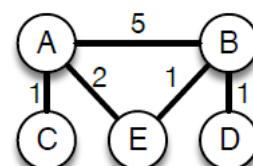
R: (a)

dest	d	n.h.
A	3	E
B	0	-
C	4	E
D	1	D
E	1	E

(b)

dest	d	n.h.
A	3	E
B	0	-
C	4	E
D	9	A
E	1	E

b: B imposta a infinito la distanza da D, ma subito dopo A lo informa che può raggiungere D in 4 passi (A-E-B-D). B allora imposta A come next-hop di D e quindi la rotta B-A-E-B-D ha distanza 9. Nei passi successivi il tutto si stabilizzerà.

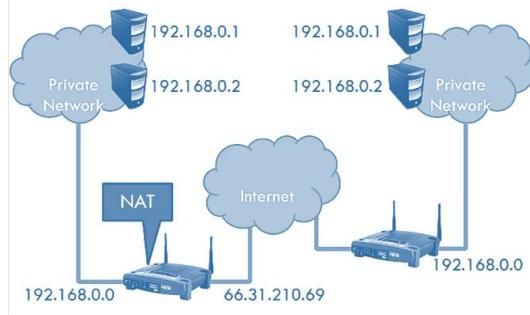


NAT (Network address translation)

<https://www.youtube.com/watch?v=FTUV0t6JaDA>
<https://www.youtube.com/watch?v=QBqPzHEDzvo>

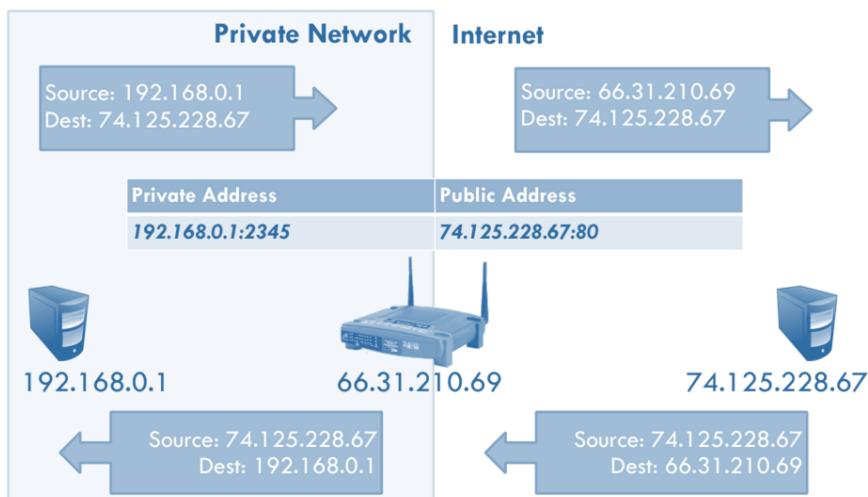
Per risolvere il problema dell'esaurimento dello spazio di indirizzamento dovuto all'aumentato utilizzo del protocollo IP è divenuta popolare la tecnologia di “**traduzione dell'indirizzo di rete**” NAT. L'idea che sta alla base di NAT è il fatto che tutti gli host che potrebbero comunicare tra loro attraverso Internet non hanno bisogno di indirizzi globalmente unici. Al contrario, è possibile assegnare a un host un “indirizzo privato”, che non sia necessariamente unico globalmente ma che lo sia in un ambito più ristretto (per esempio, all'interno della rete aziendale in cui si trova l'host).

Ad esempio, fintanto che un host comunica solamente con altri host interni a una rete aziendale, un indirizzo localmente unico è sufficiente. Se l'host dovesse voler comunicare con un host esterno alla rete aziendale, lo farebbe tramite un “dispositivo di traduzione degli indirizzi di rete” (NAT box), un dispositivo in grado di tradurre l'indirizzo privato utilizzato dall'host in un indirizzo unico su scala globale che è stato assegnato al dispositivo stesso.



Poiché è probabile che un piccolo sottoinsieme degli host dell'azienda richieda contemporaneamente i servizi del NAT, può darsi che il dispositivo di traduzione sia in grado di svolgere il proprio compito usando un piccolo insieme di indirizzi globalmente unici, un numero di indirizzi molto minore di quello che sarebbe necessario. Si può quindi immaginare che un dispositivo NAT riceva pacchetti da un host interno all'azienda e che traduca l'indirizzo IP del mittente da un indirizzo privato (per esempio 10.0.1.5) a un indirizzo unico su scala globale (per esempio 171.69.210.246). Quando i pacchetti tornano dall'host remoto e sono indirizzati a 171.69.210.246, il dispositivo NAT traduce l'indirizzo di destinazione in 10.0.1.5 e inoltra il pacchetto verso l'host.

Il principale svantaggio di tale tecnologia è che rende invalida un'ipotesi chiave di IP: tutti i nodi hanno indirizzi unici su scala globale. Inoltre, rende difficile per un dispositivo esterno alla sottorete l'instaurazione di una connessione diretta a un dispositivo che si trovi nel versante privato del NAT box, dato che, in assenza di una corrispondenza già stabilita all'interno del dispositivo di traduzione, non esiste alcun indirizzo pubblico a cui inviare la richiesta di connessione.

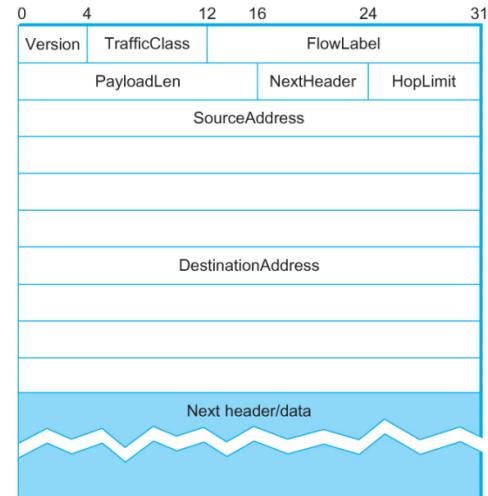


IPv6

IPv6 fornisce uno spazio di indirizzamento a 128bit, mentre IPv4 usa indirizzi a 32bit. Gli indirizzi IPv6 non usano le classi, ma lo spazio di indirizzamento è comunque suddiviso in vari modi, in base ai bit iniziali. Invece di specificare diverse classi di indirizzi, i bit iniziali specificano diversi usi dell'indirizzo IP.

IPv6 porta principalmente le seguenti novità (oltre allo spazio di indirizzamento esteso):

- supporto per servizi in tempo reale;
- supporto per la sicurezza;
- autoconfigurazione, ossia la capacità degli host di configurare autonomamente se stessi con informazioni quali il proprio indirizzo IP e il proprio nome di dominio;
- migliore funzionalità di instradamento, compreso il supporto per gli host mobili.



Il **TrafficClass** è un campo utilizzato per dichiarare la priorità del pacchetto e la **FlowLabel** serve per identificare i flussi all'interno dei router.

ES: I provider di connettività IPv6 assegnano ad ogni utente (anche singoli privati) una rete pubblica /64 (se non addirittura una /48). Supponendo di aver ricevuto un prefisso /64 per una certa rete Ethernet/WiFi, come può ogni host della rete assegnare alla propria interfaccia un indirizzo IPv6 univoco globalmente, senza server DHCP e senza configurazione manuale?

- Dato che un indirizzo IPv6 è di 128 bit, se il prefisso è di 64 bit rimangono 64 bit liberi per la parte di host. Questi 64 bit possono essere costruiti in modo univoco appendendo i 48 bit del MAC address (l'indirizzo Ethernet) e 16 bit a 0 per riempimento. (Questa è la tecnica di autoconfigurazione di IPv6.)

Advanced internetworking

Interdomain routing (BGP)

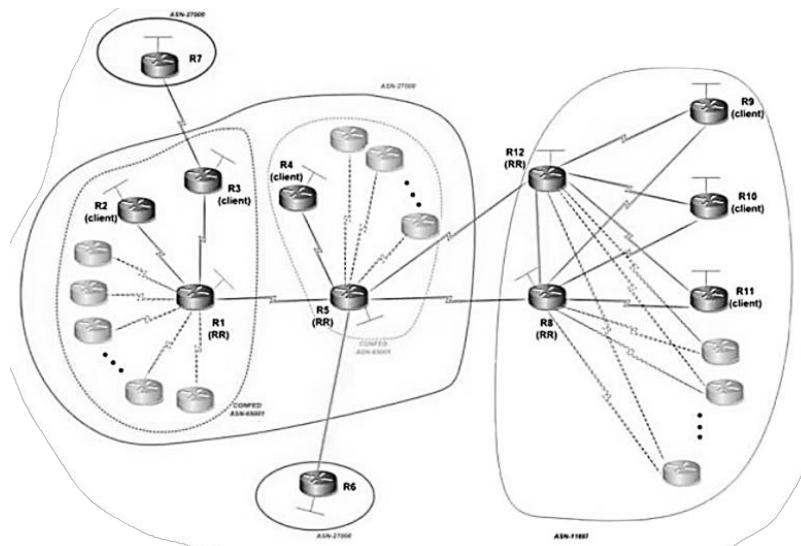
Internet è organizzato in sistemi autonomi, ciascuno dei quali si trova sotto il controllo di una singola entità di gestione amministrativa. Una complessa rete interna ad un'azienda potrebbe essere un unico AS.

AS: Un sistema autonomo di rete (in inglese Autonomous System Network), in riferimento ai protocolli di routing, è un gruppo di router e reti sotto il controllo di una singola e ben definita autorità amministrativa.

L'idea che sta alla base dei sistemi autonomi consiste nel fornire un ulteriore strumento per aggregare gerarchicamente le informazioni di instradamento in una grande internetwork, migliorando così la scalabilità. In Internet, i sistemi autonomi vengono anche chiamati domini; chiameremo quindi le due parti del problema con i nomi di instradamento intradominio e instradamento interdominio.

- **Intradominio:** instradamento all'interno di un singolo sistema autonomo;
- **interdominio:** instradamento tra sistemi autonomi.

Oltre che a migliorare la scalabilità, il modello di sistema autonomo disaccoppia l'intradominio che avviene in un AS da quello che avviene in un altro AS, in modo che ciascun AS possa eseguire il protocollo intradominio che preferisce.



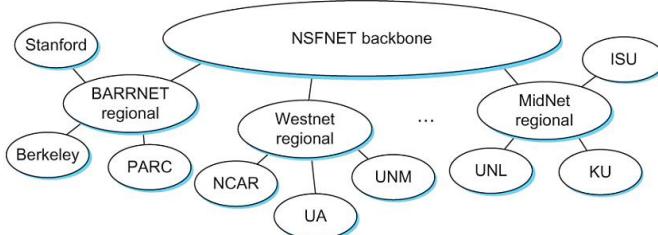
Il problema dell'intradominio si occupa, invece, di fare in modo che diversi AS condividano l'uno con l'altro le informazioni di raggiungibilità. Una caratteristica dell'idea dei sistemi autonomi consiste nel consentire ad alcuni AS di ridurre drasticamente la quantità di informazioni di instradamento di cui si devono far carico mediante l'utilizzo di percorsi di default. Ad esempio, se una rete aziendale è connessa al resto di Internet mediante un unico router, allora diviene molto semplice per un host o un router interni al sistema autonomo scoprire dove inviare i pacchetti aventi destinazione esterna all'AS: per prima cosa devono andare al router di confine dell'AS, che diviene appunto il percorso di default.

Nella storia recente di Internet sono stati utilizzati due principali protocolli di instradamento interdominio:

- **EIGRP (Exterior Gateway Protocol);**
- **BGP (Border Gateway Protocol).**

EGP forza una topologia ad albero, che non rispecchia la topologia attuale di Internet.

BGP (BGP-4) si basa sull'ipotesi che Internet è un insieme di AS arbitrariamente interconnessi.



L'immagine mostra la struttura indotta dall'utilizzo di EGP: gerarchia ad albero.

Si definiscono:

- **Traffico locale:** il traffico che ha origine e termina in nodi che si trovano all'interno di un unico AS.
- **Traffico transito:** il traffico che attraversa un AS.

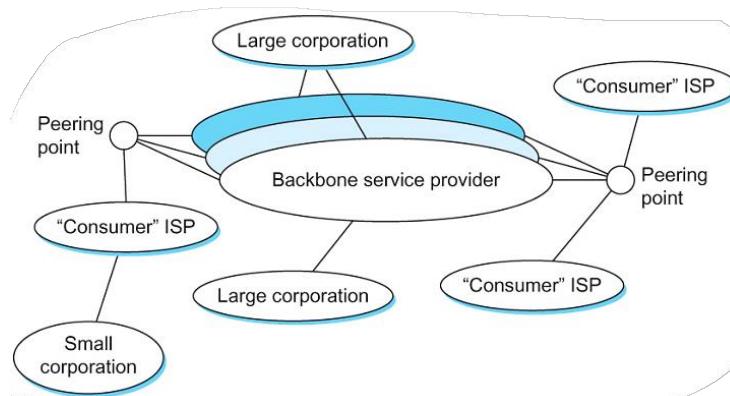
Grazie a queste definizioni, è possibile classificare i sistemi autonomi in tre grandi categorie:

- **AS stub:** ha una sola connessione verso un altro AS e, quindi, trasporta soltanto traffico locale;
- **AS multihomed:** ha connessioni con più AS ma rifiuta di trasportare traffico in transito;
- **AS di transito:** ha connessioni con più AS ed è progettato per trasportare sia traffico locale che in transito.

BGP: l'instradamento interdominio pubblicizza soltanto la raggiungibilità, cioè in sintesi, l'affermazione che "tramite questo AS si può raggiungere quella rete". Ciò significa che, per l'instradamento interdominio, scegliere un percorso ottimale è sostanzialmente impossibile. In sostanza, siamo più interessati alla raggiungibilità che all'ottimizzazione. L'identificazione di un percorso che sia in qualche modo prossimo all'ottimale e privo di cicli è considerato un grande risultato.

I principali tre motivi che rendono difficili l'instradamento interdominio sono:

- **Scalabilità:** un router di una rete backbone in Internet deve essere in grado di inoltrare pacchetti destinati a qualsiasi punto nella rete.
- **Autonomia:** la natura autonoma dei domini fa sì che ciascun dominio può eseguire al proprio interno, protocolli di instradamento diversi, e questo causa l'impossibilità di calcolare facilmente costi significativi per percorsi che attraversano più AS.
- **Fiducia:** Il fornitore A potrebbe essere scettico nel credere ad alcune informazioni pubblicate dal fornitore B.



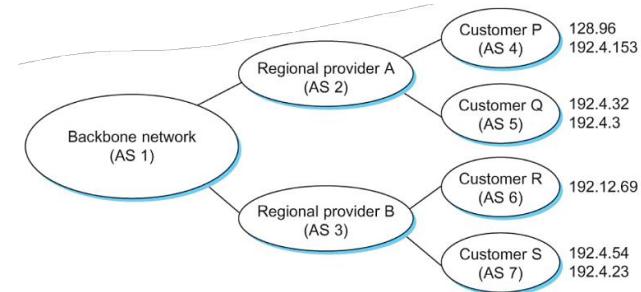
Per configurare il protocollo BGP, il gestore di ciascun AS sceglie almeno un nodo e gli assegna la funzione di "annunciatore BGP" (**BGP speaker**), che sostanzialmente effettua gli annunci per l'intero AS. Tale annunciatore BGP stabilisce sessioni con gli annunciatori BGP di altri AS, per scambiare informazioni in merito alla raggiungibilità. Oltre agli annunciatori BGP, ogni AS ha uno o più **gateway** di confine, che non coincidono necessariamente con gli speaker. I gateway di confine (border gateway) sono i router mediante i quali i pacchetti entrano ed escono dal sistema autonomo. Nel contesto dell'instradamento interdominio, un gateway è un router IP che ha quindi il compito di inoltrare pacchetti tra AS diversi.

BGP non è né un protocollo a vettore di distanza né a stato delle linee. Diversamente **BGP pubblicizza percorsi completi, sotto forma di elenchi di AS tramite i quali si può raggiungere una particolare rete.**

BGP rileva i loop con estrema facilità: se un determinato AS compare due volte nel percorso, il percorso viene scartato.

Esempio: riferendosi all'immagine precedente:

- lo speaker di AS2 pubblicizza la raggiungibilità verso P e verso Q
 - Le reti 128.96/192.4.153/192.4.32/192.4.3 possono essere direttamente raggiunte da AS2
- lo speaker per la rete backbone pubblicizza che
 - Le reti 128.96/192.4.153/192.4.32/192.4.3 sono raggiungibili attraverso il percorso <AS1, AS2>.



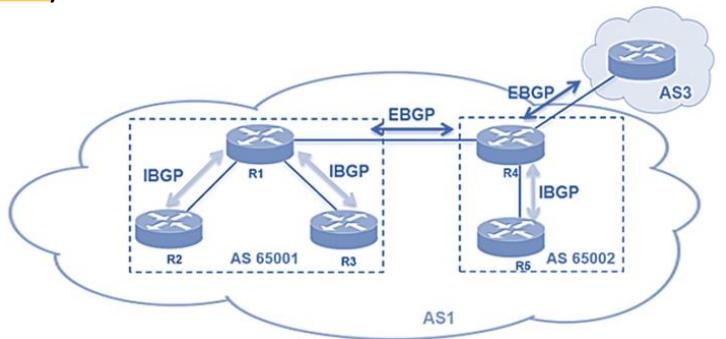
Nel protocollo BGP, tutti gli AS devono essere identificati univocamente: ad esempio AS2 può riconoscersi solo se nel percorso è l'unico a chiamarsi in quel modo. I numeri identificativi degli AS (ASN) sono a 32bit e vengono assegnati e registrati da autorità centrali (ad esempio dalla IANA - Internet Assigned Numbers Authority).

Un AS pubblicherà soltanto quei percorsi che considera sufficientemente validi per sé; se un annunciatore BGP ha una scelta di diversi percorsi verso una destinazione, sceglierà il migliore in base alle politiche locali, e questo sarà l'unico percorso che verrà pubblicizzato. Oltre a pubblicizzare percorsi, gli annunciatori BGP devono essere in grado di cancellare percorsi precedentemente pubblicizzati, nel caso in cui lungo il percorso sia guasta una linea o un nodo. Ciò avviene con una specie di informazione negativa, nota come **withdrawal route (ritiro di percorso)**.

eBGP e iBGP

Il protocollo BGP (**border gateway protocol**) può essere utilizzato sia per il routing intradominio che interdominio:

- **External BGP (eBGP):** usato tra router appartenenti a diversi AS.
- **Internal BGP (iBGP):** usato tra router dello stesso AS.



Internet Multicast

Come abbiamo visto precedentemente, le reti ad accesso multiplo come ethernet realizzano il multicast in hardware. Ora vedremo come estendere il multicast a livello software per l'interconnessione di reti. La motivazione per lo sviluppo del multicast consiste nel fatto che esistono applicazioni che vogliono inviare un pacchetto a più di un host di destinazione. Invece di costringere l'host mittente ad inviare pacchetti diversi a ciascun host di destinazione, vogliamo consentire al mittente di inviare un unico pacchetto ad un indirizzo multicast, lasciando alla rete (internetwork) il compito di consegnare una copia del pacchetto a ciascun host appartenente ad un gruppo. I singoli host possono decidere di entrare a far parte di un gruppo multicast o di uscirne autonomamente. Le funzioni multicast vengono realizzate estendendo le funzioni di instradamento svolte dai router che connettono queste reti:

- soluzioni basate sull'instradamento a vettore di distanza (es: DVMRP);
- soluzioni basate sull'instradamento a stato delle linee;
- soluzione realizzabile con qualsiasi protocollo di instradamento sottostante, denominata Protocol Indipendent Multicast (PIM).

Inoltre, tipologie di multicast:

- One-to-many: Source Specific Multicast (SSM)
- Many-to-many: Any Source Multicast (ASM)

Gli approcci qui descritti fanno riferimento all'attuale implementazione IP del multicast (IPv4).

Multicast in IP

Il modello di servizio del multicast in IP: sfrutta l'idea di un gruppo multicast a cui i riceventi si aggregano. A ciascun gruppo viene assegnato uno speciale indirizzo, che viene utilizzato come indirizzo di destinazione da chi voglia inviare pacchetti a tale gruppo. Gli host entrano in gruppo multicast usando un protocollo denominato **Internet Group Management Protocol (IGMP)**, che usano (in IPv4) per avvertire un router sulla propria rete locale che desiderano ricevere pacchetti inviati ad un certo gruppo multicast.

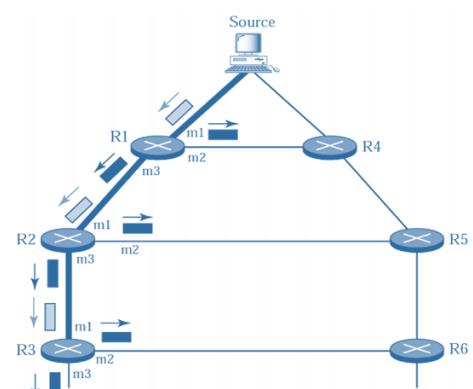
I protocolli descritti in seguito si occupano di come i pacchetti vengano distribuiti ai router giusti, mentre la consegna dei pacchetti dal router "last hop" all'host viene gestita dalle capacità multicast della rete sottostante, come descritto nei capitoli precedenti.

Multicast a vettore di distanza (DVMRP)

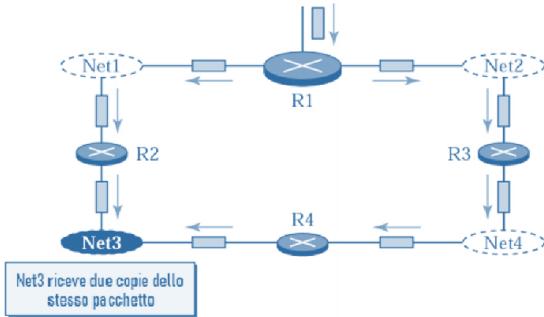
Il DVMRP (Distance Vector Multicast Routing Protocol) estende il routing a vettore di distanza per implementare il multicast. Adotta una strategia **"flood and prune"**: inonda la rete con traffico multicast e nel frattempo rimuove i rami che dichiarano di non essere interessati al traffico.

Il protocollo è quindi strutturato in due fasi:

- **Inondazione della rete: Reverse Path Flooding (RPF)**
 - dalla tabella di routing unicast, ciascun router conosce già quale sia il router "next-hop" facente parte del percorso più breve verso la sorgente S.
 - quando il router riceve un pacchetto multicast, guarda all'indirizzo della sorgente S e inoltra il pacchetto a tutti i collegamenti (escluso da quello che glielo ha inviato) sse il pacchetto è arrivato dal router "next-hop" facente parte del percorso più breve verso la sorgente S.
 - in questo modo non ci sono modi di creare loop.



Problema: le reti con più di un router possono ricevere pacchetti duplicati.



Per eliminare i pacchetti duplicati si applica il seguente approccio: si imposta un router genitore relativo ad S per ciascuna rete locale, e si permette solo ai genitori di inoltrare i pacchetti. I router genitori vengono scelti facendo in modo che siano quelli a distanza minima da S (nell'esempio sopra, R2 risulterebbe essere il genitore della Net3).

Questa strategia, chiamata **Reverse Path Broadcast (RPB)** garantisce che ciascuna rete locale riceva esattamente una copia di ogni pacchetto tramite un percorso più breve (source-based shortest path tree).

- **Pruning della rete:** si escludono le reti che non hanno host interessati al gruppo multicast. Due fasi:
 - **fase 1:** determinare se una rete locale è una foglia della distribuzione con membri del gruppo multicast. Una LAN è una foglia se il suo genitore è l'unico router nella rete locale. Gli host nella LAN devono comunicare al router di far parte del gruppo multicast. Quindi, il router sa se nella sua LAN ci sono membri del gruppo multicast o no.
 - **fase 2:** propagare l'informazione di assenza di membri del gruppo multicast.

Quindi, quando una trasmissione multicast inizia per un gruppo G:

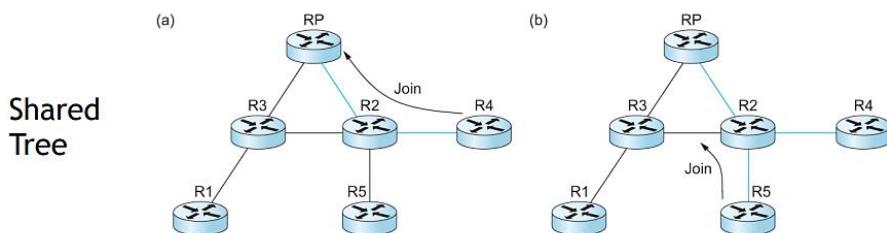
- Inizialmente inonda la rete, attraverso RPF(Reverse Path Flooding)/RPB(Reverse Path Broadcast), costruendo un albero di distribuzione ricoprente tutta la rete.
- L'albero viene “potato” rimuovendo i router non interessati a G, che notificano di interrompere la trasmissione dal “basso verso l’alto”.

Multicast routing (PIM)

La strategia PIM è stata sviluppata per rispondere ai problemi di scalabilità degli esistenti protocolli di instradamento (ad esempio il DVMRP appena visto). Ad esempio, inviare il traffico in modalità broadcast a tutti i router finché non chiedono esplicitamente di essere eliminati da tale distribuzione non è una buona scelta di progetto nel caso in cui la maggior parte dei router non voglia ricevere il traffico fin dall'inizio. La strategia PIM suddivide il problema in due casi: **modalità sparsa** e **modalità densa**.

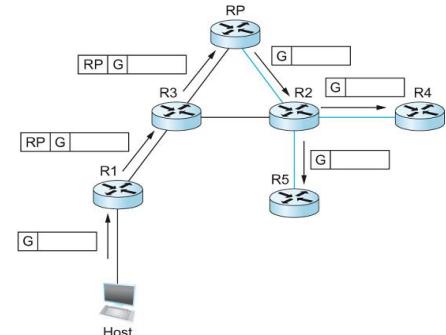
Nella **modalità sparsa** di PIM (PIM-SM), i router entrano ed escono esplicitamente dai gruppi multicast usando, rispettivamente, i messaggi del protocollo PIM noti come **Join** e **Prune**. Questi messaggi vanno inviati a un **gruppo d'incontro** (RP – rendezvous point) assegnato dal protocollo PIM a ciascun gruppo multicast.

Come risultato dell'invio di messaggio Join all'RP da parte dei vari router, viene costruito un albero per l'inoltro multicast. La strategia PIM-SM consente la costruzione di due tipi di alberi: un albero condiviso, che viene usato da tutti i mittenti, e un albero specifico per una sorgente, che può essere usato da un unico host nel ruolo di mittente.



Ogni router analizza il messaggio di join e aggiunge nella sua tabella la regola per inoltrare il traffico del gruppo G attraverso la porta da cui è arrivato il messaggio (di join). Se il router non stava partecipando all'albero, allora inoltra il pacchetto di join verso RP, e registra l'interfaccia corrispondente come l'unica da cui il traffico può arrivare; in caso contrario non fa nulla. Una volta che l'albero è costruito:

1. L'host invia un pacchetto al gruppo G attraverso la sua rete locale.
2. Il pacchetto viene ricevuto dal router designato.
3. Il router designato inserisce il pacchetto in un pacchetto IP di tipo unicast.
4. RP (gruppo d'incontro) riceve ed apre il pacchetto e lo inoltra attraverso l'albero condiviso.



La **modalità densa** invece (PIM-DM) è molto simile al DVMRP (multicast a vettore di distanza) ma è indipendente dal protocollo unicast sottostante (a differenza di DVMRP che è vincolato dal protocollo a vettore di distanza). Nella modalità densa, ciascun nodo (router) mantiene una copia dell'albero di distribuzione multicast, e usa l'approccio flood and prune: RPF per distribuire il traffico e poi in una fase successiva di pruning vengono scartati i router che non desiderano ricevere il traffico multicast.

Protocolli end-to-end

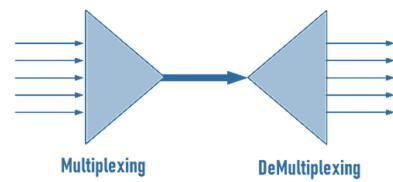
Nei capitoli precedenti sono state descritte diverse tecnologie che possono essere utilizzate per connettere insieme più calcolatori: linee di collegamento dirette (LAN come l'ethernet), reti a commutazione di pacchetto (eLAN ad esempio) e internetwork come Internet. Il problema successivo consiste nella trasformazione di questo servizio di consegna di pacchetti tra host in un canale di comunicazione tra processi: questo ruolo è svolto dal **livello di trasporto** dell'architettura di rete (in riferimento all'architettura ISO OSI il livello 4) che viene anche detto protocollo end-to-end, dal momento che fornisce il supporto alla comunicazione tra programmi applicativi posti ai due estremi del canale. Un protocollo di trasporto end-to-end può essere schematizzato con alcune delle sue caratteristiche più importanti:

- garanzia di consegna del messaggio;
- consegna dei messaggi nello stesso ordine in cui vengono inviati;
- consegna di una sola copia di ciascun messaggio;
- supporto per messaggi di dimensione arbitraria;
- supporto per la sincronizzazione fra mittente e destinatario;
- possibilità per il ricevente di applicare un controllo di flusso nei confronti del mittente;
- supporto per più processi applicativi in ciascun host.

NB: questo elenco non contiene tutte le funzionalità che un processo applicativo può richiedere alla rete: ad esempio la sicurezza, tipicamente fornita da protocolli di livello più alto.

Tipicamente la rete su cui il protocollo di trasporto (end-to-end) si trova a operare può avere delle limitazioni, come ad esempio:

- possibilità di perdere messaggi;
- modificare l'ordine dei messaggi;
- consegnare più copie di uno stesso messaggio;
- impostare un limite finito alla dimensione dei messaggi;
- consegnare i messaggi con un ritardo indefinitamente lungo.

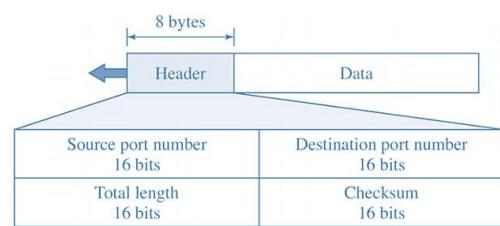


Quindi, la sfida dei protocolli di trasporto è quella dello sviluppo di tecniche e algoritmi che trasformino le proprietà assai poco lusinghiere della rete sottostante nel servizio di alto livello richiesto dai programmi applicativi, ovvero un servizio di flusso di messaggi affidabile che rispetti le proprietà sopra elencate.

UDP (User Datagram Protocol) – semplice demultiplexing

Il più semplice protocollo di trasporto estende il servizio di consegna fra host svolto dalla rete sottostante in un servizio di comunicazione tra processi. Essendo molto probabile che vi siano molti processi in esecuzione in un host qualsiasi, il protocollo deve aggiungere un livello di demultiplexing, consentendo così la condivisione della rete fra più processi applicativi presenti in ciascun host. Oltre a questo, il protocollo di trasporto non aggiunge alcuna funzionalità al servizio best-effort fornito dalla rete sottostante: il protocollo **User Datagram Protocol (UDP)** di Internet è un esempio di questo protocollo.

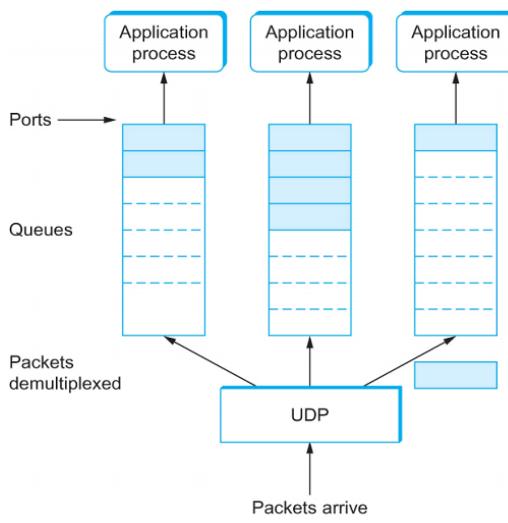
UDP prevede che i processi si identifichino l'un l'altro indirettamente, usando un localizzatore astratto, spesso chiamato **porta** o casella postale (mailbox): un processo sorgente invia un messaggio ad una porta e il processo destinatario riceve il messaggio da una porta (a differenza di un approccio più diretto che prevede di identificare i processi dal PID assegnato da sistema operativo dell'host).



L'intestazione per un protocollo di trasporto (end-to-end) che realizzi questa funzione di demultiplexing contiene tipicamente un identificativo (porta) sia per il mittente che per il destinatario. In UDP le porte non assumono significato sull'intera rete (Internet in questo caso) ma su un singolo host: in sostanza un processo viene effettivamente identificato su un certo host da una coppia **<porta, host>** che a tutti gli effetti costituisce la chiave di demultiplexing per il protocollo UDP.

Come fa un processo ad apprendere il valore della porta del processo con cui vuole comunicare? Tipicamente, un processo con funzioni di **client** inizia uno scambio di messaggio con un processo di tipo **server** (modello client - server). Una volta che il client ha contattato il server, il server conosce la porta del client (che era contenuta nell'intestazione del messaggio) e può rispondere e far così apprendere al client la porta. Un approccio assai diffuso consiste nel fatto che il server accetti messaggi da una qualche porta **ben nota (well-known port)**, cioè ogni server riceve i propri messaggi da una porta prefissata ampiamente pubblicizzata. La porta ben nota viene quindi utilizzata per accordarsi su quale porta usare per le comunicazioni successive, lasciando la possibilità che essa sia poi libera per altri client che vogliono instaurare una nuova comunicazione con il server attraverso essa.

		32-bit source IP address	
		32-bit destination IP address	
All 0s	8-bit protocol (17)	16-bit UDP total length	
Source port address 16 bits	Destination port address 16 bits		
UDP total length 16 bits	Checksum 16 bits		
Data		(Padding must be added to make the data a multiple of 16 bits)	



Tipicamente, una porta viene implementata mediante una coda di messaggi. Quando arriva un messaggio, il protocollo UDP (ad esempio) accoda il messaggio al termine della coda: se la coda fosse piena, il messaggio verrebbe ignorato. Quando un processo vuole ricevere un messaggio ne preleva uno dalla fronte della coda.

Infine, anche se UDP non implementa il controllo di flusso, né la consegna affidabile o in sequenza corretta, fa qualcosa di più del semplice demultiplexing: assicura anche la correttezza del messaggio, per mezzo di una somma di controllo (attualmente facoltativa in IPv4 ma obbligatoria in IPv6). Il protocollo UDP calcola la somma di controllo sull'intera intestazione UDP, sul contenuto del messaggio e su una parte chiamata pseudointestazione, composta da tre campi dell'intestazione IP: numero di protocollo, indirizzo IP di mittente e destinatario.

ES: Un server sta trasmettendo un flusso di datagrammi UDP ad un client, che li consuma ad una velocità di dieci datagrammi al secondo. Ogni datagramma ha un payload di 1KB. A quale velocità (in KB/s) deve inviare dati il server per saturare in 5 secondi il buffer di input, se questo è da 16KB?

- Siano rispettivamente I i datarate di input e output (in KB/s) del buffer di lunghezza L (in KB). Per saturare il buffer in un tempo t , deve essere:

$$(I - O) * t = L$$

da cui:

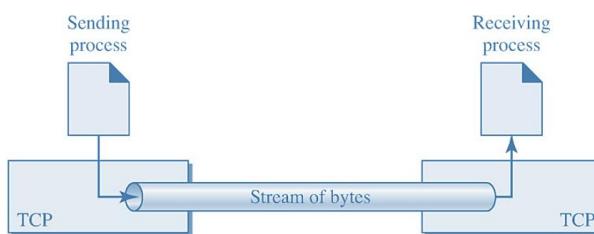
$$I = O + L/t$$

Sostituendo, abbiamo $I = 10 + 16/5 = 13,2 \text{ KB/s}$.

TCP (Transport Control Protocol) – flusso affidabile di byte

Diversamente da un semplice protocollo di demultiplexing come UDP, un protocollo di trasporto più sofisticato offre un servizio di flusso di byte affidabile e orientato alla connessione. Tale servizio si è dimostrato molto utile in quanto libera l'applicazione dal compito di gestire i dati fuori ordine o mancanti. Il **Transport Control Protocol (TCP)** di Internet, è probabilmente il protocollo, di questo tipo, attualmente più utilizzato e ottimizzato. In termini delle proprietà enunciate all'inizio di questo capitolo, TCP garantisce consegna affidabile e in sequenza di un flusso di byte. È un **protocollo full-duplex**, per cui ciascuna connessione TCP fornisce una coppia di flussi di byte, una in ciascuna direzione.

Il protocollo TCP prevede anche un meccanismo di **controllo del flusso**, per ciascuno di questi flussi di byte, consentendo al ricevitore di limitare la quantità di dati che il mittente può inviare in un certo istante. Infine, come UDP, TCP fornisce il supporto ad un meccanismo di demultiplexing, che consente a più programmi applicativi di coesistere e utilizzare la rete in un host.



Inoltre, TCP realizza anche un meccanismo di **controllo della congestione**. L'idea su cui si basa questo meccanismo è quella di controllare la velocità con cui TCP invia i dati, non per evitare che il mittente sovraccarichi il ricevente, ma per impedire al mittente di sovraccaricare la rete. Quindi, ricapitolando:

- **controllo di flusso**: si occupa di evitare che il mittente esaurisca la capacità dei ricevitori;
- **controllo di congestione**: si occupa di evitare che vengano trasmessi troppi dati sulla rete.

Problemi relativi alle connessioni end-to-end

Nel cuore di TCP si colloca l'algoritmo a finestra scorrevole detto **sliding window**. Nonostante TCP si tratti fondamentalmente di un implementazione di questo algoritmo, esistono molte differenze importanti, causate dal fatto che esso opera in Internet piuttosto che in una semplice connessione punto-punto.

Innanzitutto, mentre l'algoritmo sliding window presentato nella sezione precedente viene eseguito su una singola linea fisica di connessione che connette sempre gli stessi due calcolatori in point-to-point, TCP consente connessioni logiche fra processi che sono in esecuzione su due calcolatori qualsiasi in Internet. Ciò significa che TCP necessita di un'esplicita fase di instaurazione della connessione, durante la quale le due parti coinvolte nella connessione si accordano per scambiarsi dati reciprocamente. TCP ha anche un'esplicita fase di terminazione della connessione.

La seconda differenza consiste nel fatto che, mentre una singola rete fisica che connette continuamente due calcolatori ha un prefissato valore di **RTT (Round Trip Time** – il tempo che un segnale ci mette ad arrivare a destinazione più il tempo che il relativo ACK ci mette ad arrivare al mittente), le connessioni TCP hanno molto probabilmente valori molto diversi. Il protocollo TCP deve essere in grado di consentire il funzionamento di connessione a basso RTT e ad alto RTT. Ciò significa che per l'algoritmo sliding window, il meccanismo di timeout deve essere adattivo in base alla rete su cui si trova ad operare.

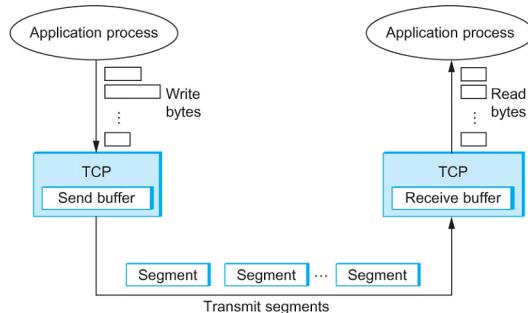
Una terza differenza consiste nel fatto che i pacchetti possono essere riordinati mentre attraversano Internet, mentre questo non è possibile in una linea punto-punto. Pacchetti che siano fuori ordine di poco non causano errori in quanto l'algoritmo sliding window è in grado di riordinarli correttamente usando il numero di sequenza. Il vero problema è quanto possano essere fuori ordine i pacchetti, oppure, detto alternativamente, quanto possono arrivare in ritardo a destinazione. Nel caso peggiore, un pacchetto può venire ritardato da

Internet finché il suo campo TTL del pacchetto IP non scade (TTL = 0). TCP ipotizza che ciascun pacchetto abbia un tempo di vita massimo chiamato **tempo di vita massimo per un segmento (MSL, maximum segment lifetime)**.

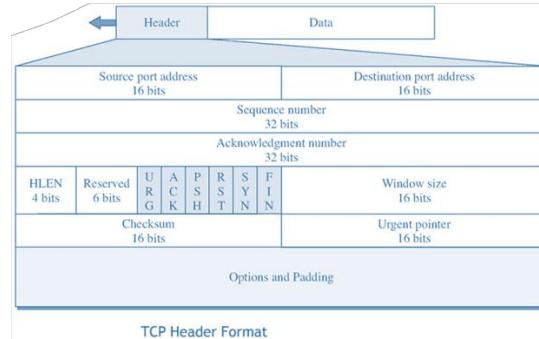
Quarta differenza: i calcolatori connessi alle linee punto-punto sono solitamente progettati per gestire quella particolare linea, d'altra parte, ciascun tipo di calcolatore può essere collegato a Internet, rendendo assai variabile la quantità di risorse dedicate alla connessione TCP; ciò significa che TCP deve prevedere un meccanismo che ciascuna parte coinvolta nella comunicazione possa usare per apprendere quante risorse e quali risorse possano essere destinate alla connessione dall'altra entità (controllo di flusso).

Formato del segmento

Il protocollo TCP è di tipo byte-oriented: ciò significa che il mittente invia byte in una connessione di tipo TCP e il destinatario riceve byte. Anche se il “flusso di byte” descrive il servizio offerto da TCP ai processi applicativi, il protocollo TCP non trasmette singoli byte su Internet, ma memorizza un numero sufficiente di byte ricevuto dal processo applicativo sull'host mittente finché non ha riempito un pacchetto (IP) di dimensioni ragionevoli, dopodiché invia tale pacchetto all'host destinatario. Sull'host di destinazione, il protocollo TCP svuota quindi all'interno di un buffer il contenuto del pacchetto ricevuto, dopodiché il processo ricevente leggerà i byte da questo buffer quando vorrà.



I pacchetti scambiati fra pari entità del protocollo TCP sono detti **segmenti** perché ognuno di essi trasporta un segmento del flusso di byte. Ciascun segmento TCP contiene l'intestazione schematizzata nell'immagine sottostante:



I campi **destination port** e **source port** identificano rispettivamente le porte della sorgente e della destinazione, proprio come nel protocollo UDP. Questi due campi si combinano con gli indirizzi di sorgente e di destinazione per identificare univocamente ciascuna connessione TCP. La chiave di demultiplexing di TCP è quindi data dalla quaterna: <**SrcPort**, **SrcIPAddr**, **DstPort**, **DstIPAddr**>

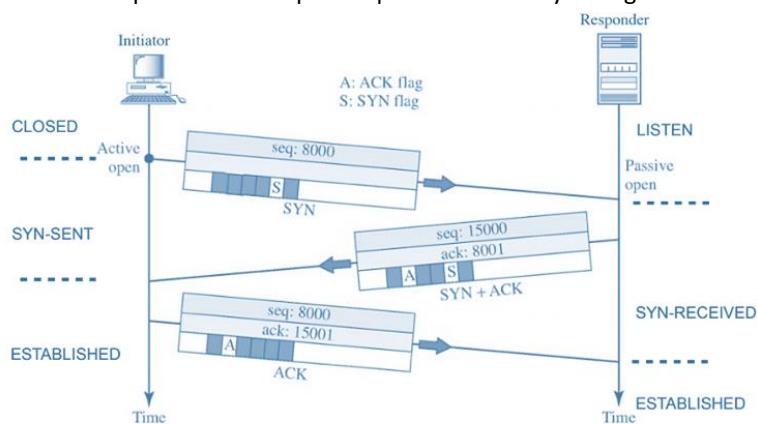
In TCP ogni byte ha un numero di sequenza, il campo **SequenceNum** contiene il numero di sequenza del primo byte di dati trasportato da quel segmento.

Instaurazione e terminazione di una connessione

Una connessione TCP inizia quando un client (il chiamante) effettua l'apertura attiva di un canale di comunicazione verso un server (il chiamato). Le due parti iniziano ad inviare dati soltanto quando la fase di instaurazione della connessione risulta terminata. Analogamente, quando una delle due parti ha terminato di inviare dati, chiude la connessione nella direzione corrispondente, spingendo TCP ad iniziare uno scambio di messaggi per la chiusura della connessione. È possibile che un'entità abbia chiuso la connessione e smesso di inviare dati, mentre l'altra continui a tenere aperta l'altra metà e ad inviare dati.

Three-way handshake

L'algoritmo utilizzato dal TCP per instaurare una connessione viene chiamato **three-way handshake (stretta di mano in tre fasi)**. L'handshake in tre fasi richiede lo scambio di tre messaggi fra il client e il server. Lo schema di base prevede che le due parti si accordino su un insieme di parametri, che, nel caso di TCP sono i numeri di sequenza che le due parti useranno per i rispettivi flussi di byte. Seguono le tre fasi:



NB: client: entità attiva, server: entità passiva.

1. il client invia un segmento al server annunciando il numero iniziale di sequenza che pensa di usare (**Flags = SYN, SequenceNum = x**);
2. Il server risponde con un singolo segmento che svolge due funzioni:
 - a. conferma il numero di sequenza del client (**Flags = ACK, Ack = x+1**);
 - b. annuncia il proprio numero iniziale di sequenza (**Flags = SYN, SequenceNum = y**).
3. Il client risponde con un terzo segmento che conferma il numero di sequenza del server (**Flags = ACK, Ack = y + 1**).

Diagramma delle transizioni di stato

Il protocollo TCP è sufficientemente complesso da contenere, nella propria specifica, un diagramma delle transizioni di stato.

TCP è “stateful” perciò si trova sempre in un determinato “stato”. Le transizioni tra uno stato e un altro sono causate da eventi.

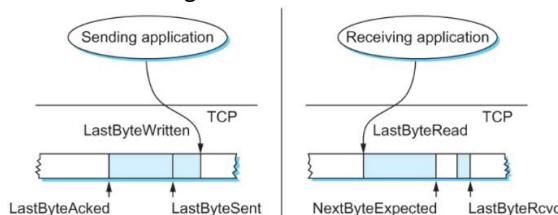
Algoritmo sliding window rivisitato

Come enunciato in precedenza, TCP implementa una variante dell'algoritmo sliding window che svolge diverse funzioni aggiuntive:

- consegna affidabile dei dati;
- corretta sequenza di consegna;
- controllo di flusso.

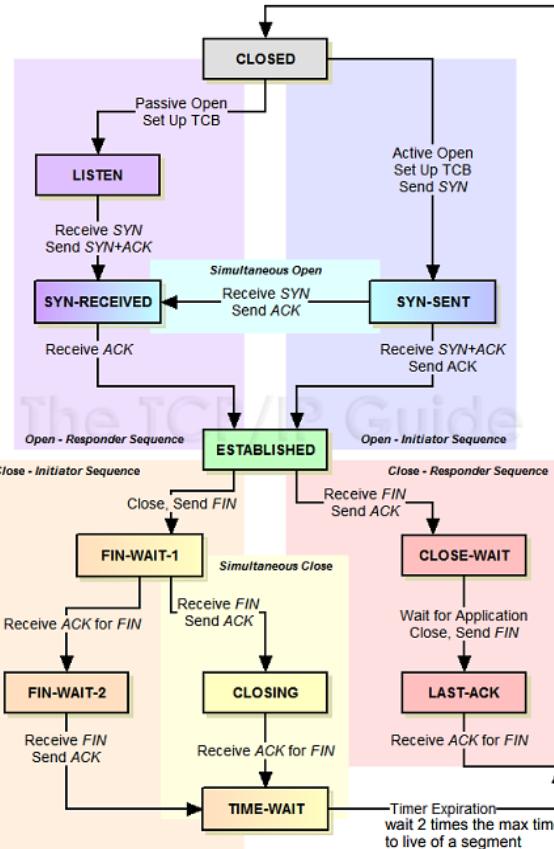
Consegna affidabile e in sequenza

Si consideri la seguente illustrazione:



Sul lato del mittente, TCP gestisce un buffer di spedizione, che viene usato per memorizzare i dati inviati ma non ancora confermati, oltre ai dati consegnati dall'applicazione mittente ma non ancora trasmessi. Sul lato ricevente, TCP gestisce un buffer di ricezione, che contiene i dati che arrivano fuori ordine, oltre ai dati che sono in ordine corretto ma che non sono ancora stati letti dal processo applicativo nel ricevente. Esaminiamo ora i due lati della comunicazione:

- **Mittente:**
 - vengono gestiti tre puntatori per il buffer di invio, ciascuno con un significato ovvio:
 - **LastByteAcked**
 - **LastByteSent**
 - **LastByteWritten**
 - chiaramente:
 - **LastByteAcked <= LastByteSent** perché il ricevitore non può aver confermato un byte che non è ancora stato inviato
 - **LastByteSent <= LastByteWritten** perché TCP non può inviare un byte che non è ancora stato scritto nel buffer.
- **Ricevente:**
 - viene gestito un insieme di puntatori (numeri di seq.) analogo:
 - **LastByteRead**
 - **NextByteExpected**
 - **LastByteRcvd**
 - chiaramente:
 - **LastByteRead < NextByteExpected** perché un byte non può essere letto dall'applicazione finché non è stato ricevuto e sono stati ricevuti anche tutti i byte precedenti.
 - **NextByteExpected <= LastByteRcvd + 1**



Controllo di flusso

In un protocollo sliding window la dimensione della finestra determina la quantità di dati che possono essere inviati senza aspettare la conferma del ricevente.

Di conseguenza, il ricevente controlla la velocità del mittente comunicando una finestra che non sia più grande della quantità di dati che può memorizzare. Osservare che dal lato del ricevente, il protocollo TCP deve fare in modo che sia sempre valida:

$$\text{LastByteRcvd} - \text{LastByteRead} \leq \text{MaxRcvBuffer}$$

per evitare che il buffer si riempia. Per far ciò comunica una dimensione di finestra uguale a:

$$\begin{aligned} \text{AdvertisedWindow} = \\ \text{MaxRcvBuffer} - (\text{NextByteExpected} - 1) - \text{LastByteRead} \end{aligned}$$

che rappresenta la quantità di spazio rimasto libero nel buffer di ricezione. A mano a mano che i dati arrivano, il ricevitore li conferma, insieme a tutti i byte già arrivati in precedenza, e sposta verso destra (incrementa) LastByteRcvd, provocando potenzialmente, una contrazione della finestra disponibile. Il fatto che la finestra si contragga oppure no, dipende da quanto velocemente il processo applicativo locale utilizza i dati. Se il processo è più veloce, la finestra si ingrandisce, se è più lento si rimpicciolisce.

Dal lato del mittente, il protocollo TCP deve utilizzare la finestra che viene comunicata dal ricevente, per cui ad ogni istante deve garantire che:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow}$$

Ovvero: il mittente calcola la finestra effettiva che limita la quantità di dati che possono essere inviati:

$$\text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked})$$

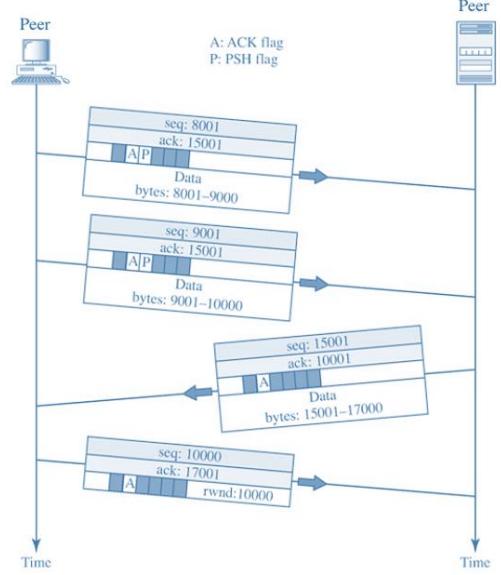
Mentre accade tutto ciò, il lato trasmittente deve assicurarsi che il processo applicativo locale non faccia traboccare il buffer di trasmissione, cioè che:

$$\text{LastByteWritten} - \text{LastByteAcked} \leq \text{MaxSendBuffer}$$

Se risulta che il processo tenta di inviare byte al protocollo TCP ma invaliderebbe la relazione sopra, TCP ferma il processo (non gli permette di scrivere byte sul buffer).

Ritorno al valore iniziale (wraparound)

L'importanza di avere uno spazio a 32 bit per i numeri di sequenza consiste nel fatto che i numeri di sequenza usati per una connessione potrebbero tornare al valore iniziale (**wraparound**): in un certo momento potrebbe essere inviato un byte avente il numero di sequenza x, e poi, più tardi, potrebbe essere inviato un secondo byte con lo stesso numero. Considerando il tempo di vita di segmento TCP in Internet, dobbiamo avere la garanzia che il numero di sequenza non torni al proprio valore iniziale in un periodo ragionevole di tempo per evitare che ci siano due segmenti TCP con lo stesso sequenceNum. Ciò dipende dalla velocità a cui si possono trasmettere i dati. La tabella mostra come in connessioni molto veloci questo possa accadere, considerando di avere un **Maximum Segment Lifetime (MSL)** di 120 secondi (il default in TCP). Il problema viene risolto con le recenti implementazioni con più bit indicanti il sequenceNum, che ci permetteranno di utilizzare TCP in reti molto più veloci.



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 pieno il canale di trasmissione

L'importanza del campo AdvertisedWindow a 16 bit consiste nel fatto che esso deve essere sufficientemente grande da consentire al mittente di tenere pieno il canale di trasmissione.

Sappiamo che il ricevitore è libero di non aprire la finestra fino al valore massimo del campo AdvertisedWindow, ma qui siamo interessati alla situazione in cui il ricevitore ha spazio di memorizzazione a sufficienza per gestire tanti dati quanti ne sono consentiti da AdvertisedWindow.

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

Il campo AdvertisedWindow del protocollo TCP si trova in una situazione ancora peggiore del campo SequenceNum: non è grande a sufficienza per connessioni con finestra di dimensione massima maggiori di 64KB.

TCP risolve questo problema implementando una estensione che consente di utilizzare una finestra più grande. Questa estensione prevede un'opzione che definisce un **fattore di scala** per la finestra che viene comunicata, cioè, invece di interpretare il numero che appare nel campo AdvertisedWindow come un'indicazione del numero di byte che possono essere inviati dal mittente senza averne conferma, questa opzione consente alle due parti coinvolte di accordarsi sul fatto che il campo AdvertisedWindow rappresenti un conteggio di spezzoni maggiori di dimensioni (ad esempio unità di 16byte).

Stimolare la trasmissione

Come fa il protocollo TCP a decidere di trasmettere un segmento? Come abbiamo visto, TCP fornisce supporto all'astrazione di flusso di byte: i processi applicativi scrivono byte nel flusso, e sta al protocollo TCP decidere di avere byte a sufficienza per inviare un segmento.

TCP gestisce una variabile, tipicamente denominata **dimensione massima del segmento (MSS, Maximum Segment Size)** ed invia un segmento non appena ha raccolto dal processo mittente un numero di byte uguale a MSS: che tipicamente viene impostato al valore del più grande segmento TCP che può essere inviato senza provocare la frammentazione locale del corrispondente pacchetto IP (cioè MSS = MTU).

La seconda cosa che provoca la trasmissione di un segmento è l'esplicita richiesta di invio da parte del processo mittente tramite l'operazione di **push**. Infine, l'ultima causa che provoca la trasmissione di un segmento è lo scadere di un timer: il segmento che viene così prodotto contiene tanti byte quanti ne sono presenti (se si aspetta che si riempia il segmento troppo a lungo, si ha una latenza troppo elevata).

Sindrome della finestra futile (silly window syndrome)

L'algoritmo sliding window alla base di **TCP non supporta una dimensione minima sui segmenti trasmessi**. La sindrome della finestra futile è una situazione in cui molti piccoli segmenti inefficienti vengono inviati (si ha così un elevato overhead).

Questo può capitare quando vengono comunicate dimensioni per la finestra troppo piccoli, o il trasmittente è troppo aggressivo e invia subito piccole quantità di dati, piuttosto che farle accumulare nel buffer. Questo problema non è da considerarsi un fallimento dell'algoritmo sliding window ma piuttosto un'inefficienza.

Algoritmo di Nagle

Si consideri il mittente in TCP, se ci sono dati da inviare ma la finestra è aperta per una dimensione inferiore al valore di MSS possiamo voler attendere un po' di tempo prima di inviare i dati, ma la domanda è: quanto a lungo? Se aspettiamo troppo, danneggiamo le applicazioni interattive, se aspettiamo poco, allora rischiamo di spedire un sacco di pacchetti e ricadiamo nella sindrome della finestra futile appena descritta.

La risposta consiste nell'utilizzo di un temporizzatore e nel trasmettere quando questo scade. L'idea alla base dell'algoritmo di Nagle sfrutta il fatto che, mentre il protocollo TCP ha dati in viaggio, il mittente può ricevere una conferma che può venire considerata come un temporizzatore che scade, provocando la trasmissione di ulteriori dati. L'algoritmo di Nagle fornisce una regola semplice e uniforme quando sia il caso di trasmettere:

L'algoritmo in pseudocodice:

```

while (applicazione produce dati da inviare) {
    if (i dati disponibili e la finestra sono >= MSS) then
        invia un segmento completo
    else if (ci sono dati non confermati in viaggio) then
        memorizza i dati nel buffer finché non arriva una conferma
    else invia ora tutti i pacchetti
}

```

In altre parole, se la finestra lo consente, l'invio di un segmento completo è sempre una scelta valida, così come è permesso inviare immediatamente una piccola quantità di dati se non ci sono segmenti in viaggio, ma se ci sono dati in viaggio, il mittente deve attendere una conferma prima di trasmettere il segmento successivo.

Ritrasmissione adattiva

Dato che TCP garantisce la consegna affidabile dei dati, ogni segmento per il quale non sia stata ricevuta conferma entro un certo periodo di tempo viene ritrasmesso. Questo temporizzatore viene impostato da TCP in funzione del valore di RTT che ci si attende fra i due estremi della connessione. Per scegliere un valore RTT adeguato, il protocollo TCP usa un **meccanismo di ritrasmissione adattivo**.

RTT: Round Trip Time: è il tempo che intercorre tra l'invio di un segnale più il tempo necessario per la ricezione della conferma di quel segnale.

Algoritmo originario

L'algoritmo originariamente descritto nelle specifiche di TCP si basa sulla seguente idea: valutare continuamente il valore medio di RTT e di calcolare il valore della temporizzazione in funzione di tale valore. In particolare, ogni volta che TCP invia un segmento di dati, memorizza l'orario attuale e quando poi riceve la conferma dell'arrivo di tale segmento, legge nuovamente l'orario e considera la differenza tra questi due orari come nuovo campione dei valori RTT, SampleRTT. Il protocollo calcola poi un valore stimato di RTT, EstimatedRTT, come media pesata fra la stima precedente e il nuovo campione:

$$\text{EstimatedRTT} = a * \text{EstimatedRTT} + (1 - a) * \text{SampleRTT}$$

Il parametro **a** viene scelto per rallentare le modifiche al valore EstimatedRTT. La specifica originaria di TCP settava **a** tra 0.8 e 0.9. TCP poi usa EstimatedRTT per calcolare il valore della temporizzazione TimeOut nel modo seguente:

$$\text{TimeOut} = 2 * \text{EstimatedRTT}$$

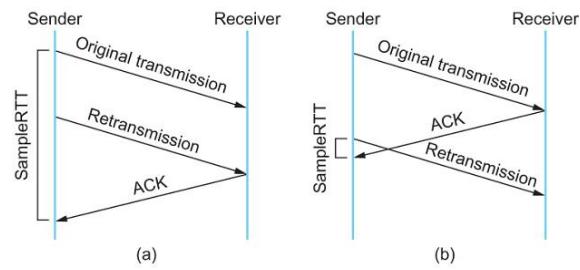
Un problema però è il seguente: si consideri l'illustrazione a destra: quando un segmento viene ritrasmesso, come facciamo a decidere se l'ACK che arriva dopo la ritrasmissione fa riferimento al primo segmento o a quello ritrasmesso? non si può sapere!

Seguono possibili miglioramenti:

Algoritmo di Karn/Partridge

L'algoritmo di Karn/Partridge è un miglioramento dell'algoritmo originale:

- non campiona l'RTT quando avviene una ritrasmissione;
- duplica il TimeOut dopo ogni ritrasmissione.



Algoritmo di Jacobson/karels

L'algoritmo di Jacobson-Karels propone un nuovo schema di ritrasmissione:

- Set Difference = SampleRTT – EstimatedRTT;
- EstimatedRTT = EstimatedRTT + ($\delta \times$ Difference);
- Deviation = Deviation + (|Difference| - Deviation);
- TimeOut = $\mu \times$ EstimatedRTT + $\Phi \times$ Deviation;
 - dove μ è tipicamente settata a 1 e Φ a 4;
 - dove $\delta = 1 - \alpha$.

ES: Durante una connessione TCP tra A e B, A ha appena inviato un frame con FIN=1.

- (a) Se A riceve un segmento con ACK=1 e FIN=0, in che stato si è portato B?
- (b) A potrebbe ricevere un segmento con ACK=1 e FIN=1? In che stato si porterebbe?

- Risposta:

- (a) Vuol dire che B ha ricevuto il FIN ma non ha ancora deciso di chiudere la comunicazione, quindi si è portato in CLOSE_WAIT (deve aspettare il close da parte della sua applicazione).
- (b) Vuol dire che B ha ricevuto il FIN e praticamente nello stesso momento la sua applicazione ha deciso di chiudere; quindi nel segmento di risposta B manda sia l'ACK del FIN che ha ricevuto, sia il FIN della sua chiusura. In tale situazione A si porta in TIME_WAIT.

NB: FIN se impostato a 1 indica che l'host mittente del segmento vuole chiudere la connessione TCP aperta con l'host destinatario. Il mittente attende la conferma dal ricevente (con un FIN-ACK). A questo punto la connessione è ritenuta chiusa per metà: l'host che ha inviato FIN non potrà più inviare dati, mentre l'altro host ha il canale di comunicazione ancora disponibile. Quando anche l'altro host invierà il pacchetto con FIN impostato la connessione, dopo il relativo FIN-ACK, sarà considerata completamente chiusa.

ES: Durante una connessione TCP, una delle due parti ha ricevuto AdvertisedWindow=0. In quali modi può sapere se la finestra si riapre?

- In tre modi:

1. se la parte ha inviato dei dati di cui non ha ricevuto l'ACK, aspetta tali ACK perché potrebbero avere AdvertisedWindow > 0;
2. altrimenti, se ha dei dati da inviare, la parte invia periodicamente un piccolo segmento (di 1 byte) per stimolare la controparte a comunicare la sua AdvertisedWindow;
3. infine, la controparte potrebbe mandare spontaneamente un segmento di dati, e nella cui intestazione c'è la nuova AdvertisedWindow (in piggyback).

Controllo della congestione e allocazione delle risorse

Rivolghiamo ora la nostra attenzione ad un problema che si estende sull'intera pila di protocolli: come **allocare risorse in modo efficace ed equo** ad un insieme di utenti in competizione. Le risorse da condividere sono:

- l'ampiezza di banda delle linee di collegamento;
- i buffer interni ai router o agli switch dove i pacchetti vengono accodati in attesa di essere trasmessi.

I pacchetti competono in un router per l'utilizzo di una linea di collegamento, mentre si trovano in una coda ad aspettare il proprio turno per essere trasmessi sulla linea stessa. Se troppi pacchetti competono, il buffer si riempie e i pacchetti in arrivo vengono scartati. Quando questo fenomeno accade frequentemente si dice che una linea è **congestionata**. La maggior parte delle reti ha un meccanismo di **controllo della congestione**.

Il controllo della congestione e l'allocatione delle risorse coinvolgono sia gli host sia gli apparati di rete, come i router. Il controllo della congestione indica il fatto che la rete non fa nulla per evitare casi di congestione, ma si limita a risolverli una volta che accadono. L'allocatione delle risorse invece è un insieme di accorgimenti tali da cercare di rendere impossibile il verificarsi di casi di congestione, ad esempio assegnando a ciascun host un determinato circuito virtuale da utilizzare. Riassumendo:

- **Allocazione di risorse:** processo mediante il quale gli apparati di una rete cercano di soddisfare le richieste delle applicazioni in merito alle risorse di rete disponibili cercando di prevenire il fenomeno della congestione.
- **Controllo congestione:** gli sforzi compiuti dai nodi della rete per reagire alle condizioni di sovraccarico.

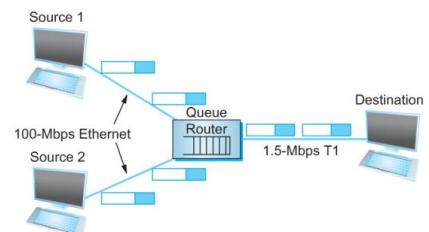
Problemi nell'allocatione delle risorse

Il principale fattore che rende complesso il problema dell'allocatione risorse e il controllo della congestione è il fatto che non sono problemi localizzati in un unico strato della gerarchia dei protocolli. L'allocatione delle risorse viene realizzata in parte dai router o dagli switch interni alla rete e in parte dal protocollo di trasporto che viene eseguito sugli host terminali. I sistemi che si trovano agli estremi di una comunicazione (end systems) usano protocolli di segnalazione per veicolare le proprie richieste di risorse ai nodi della rete, i quali rispondono con informazioni relative alla disponibilità di risorse.

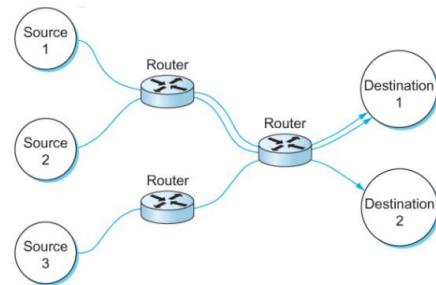
Modello della rete

Definiamo tre caratteristiche dell'architettura di rete che hanno interesse in relazione al problema dell'allocatione delle risorse:

- **Rete a commutazione di pacchetto:** in una rete a commutazione di pacchetto (es: Internet) può darsi che una sorgente sia collegata ad una linea con capacità più che sufficiente per potere inviare un pacchetto, ma i suoi pacchetti possono attraversare, da qualche parte all'interno della rete, una linea di collegamento (ipoteticamente meno performante) utilizzata da molte altre sorgenti di traffico.
- **Flussi privi di connessione:** considereremo durante il capitolo, reti prive di connessione, come Internet (protocollo IPv4). Le reti basate ad esempio su circuito virtuale, in un certo senso applicano già un controllo delle risorse, in quanto alla creazione di un circuito, ciascun router riserva un insieme di buffer per gestire quella connessione, ma questo ha uno svantaggio: sottoutilizzazione delle risorse, ovvero, i buffer riservati ad un particolare circuito non sono disponibili per altro traffico, nemmeno nel caso in cui non risultino utilizzati. Quando si parla di controllo di flusso, categorizzare le proprie reti in "prive di connessione" o "orientate alla connessione" è una suddivisione troppo marcata, esiste



infatti una via di mezzo: i datagrammi, in IP, vengono commutati indipendentemente ma solitamente un flusso di datagrammi fra una particolare coppia di host segue lo stesso percorso attraverso un insieme di router. Questa idea di un **flusso** come di una sequenza di pacchetti inviati da una sorgente ad una destinazione e che seguono lo stesso percorso è un'importante e non trascurabile astrazione da tenere in considerazione nel contesto di allocazione risorse. Un flusso può essere definito a diversi livelli di dettaglio, ad esempio host-to-host, process-to-process etc.



- **Modello di servizio:** con il servizio best-effort di IP, tutti i pacchetti sono gestiti allo stesso modo e agli host terminali non viene dato alcuno strumento per chiedere alla rete di fornire garanzie ad uno dei propri flussi. Un modello di servizio di quel tipo fornisce diverse **qualità di servizio (QoS, quality of service)**.

Gerarchia

Esistono innumerevoli dettagli per i quali le strategie di allocazione di risorse differiscono l'una dall'altra, la definizione di una gerarchia completa è un proposito arduo. Per il momento descriviamo tre dimensioni lungo le quali possono essere caratterizzate le strategie di allocazione delle risorse:

- **Focalizzata sui router o focalizzata sugli host:** le strategie di allocazione di risorse possono essere classificate in due grandi categorie: quelle che risolvono il problema all'interno della rete (cioè nei router o negli switch, **router-centric**) e quelle che lo risolvono ai confini della rete (cioè gli host, probabilmente nel protocollo di trasporto, **host-centric**).
- **Basata sulla prenotazione o basata sui segnali ricevuti:** un secondo modo per catalogare le strategie di allocazione delle risorse si basa sul fatto che usino un meccanismo di prenotazione (**reservation based**) o che si basino sui segnali ricevuti (**feedback based**). In un sistema basato sulla prenotazione, un host terminale richiede alla rete una certa capacità nel momento in cui viene creato un flusso e di conseguenza ciascun router alloca risorse sufficienti per soddisfare tale richiesta (se possibile). In un approccio basato sui segnali ricevuti, un host terminale inizia ad inviare dati senza prenotare anticipatamente alcuna capacità nella rete: in seguito, modifica la propria velocità di trasmissione in base ai segnali (feedback) che riceve.
- **Basata su finestra o basata sulla velocità:** l'approccio basato su finestra consiste nel pubblicizzare una finestra il cui spazio deve essere riservato dai buffer dei router, ad esempio nel protocollo TCP. L'altro approccio si basa sul controllare il comportamento di una sorgente usando un valore di velocità, cioè indicando quanti bit al secondo possono essere assorbiti dal ricevitore o dalla rete.

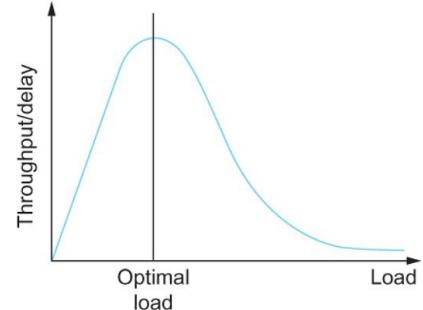
Allocazione di risorse efficace

Un buon punto di partenza per la valutazione dell'efficacia di uno schema di allocazione di risorse è l'esame delle due metriche principali alle reti di calcolatori: il **throughput** e il **ritardo**. Chiaramente vogliamo avere il massimo throughput e il minimo ritardo. Un maggiore throughput potrebbe (non intuitivamente) aumentare il ritardo dei pacchetti, in quanto si creano code più lunghe nei buffer dei router.

Per descrivere questa relazione si utilizza il rapporto tra throughput e il ritardo come metrica di valutazione dell'efficacia di uno schema di allocazione delle risorse. Questo rapporto viene a volte chiamato **potenza (power)** della rete.

$$\text{Power} = \text{Throughput}/\text{Delay}$$

L'obiettivo è quello di rendere massimo questo rapporto, che è funzione del carico della rete, che, a sua volta, viene impostato dal meccanismo di allocazione delle risorse. L'immagine presenta un grafico rappresentativo della potenza, dove, idealmente, il meccanismo di allocazione delle risorse opererebbe vicino al picco di questa curva. A sinistra del picco, la strategia si comporta in modo troppo cauto, cioè non consente di inviare pacchetti a sufficienza per tenere impegnate le linee. Alla destra del picco, si consente di inviare nella rete un numero troppo elevato di pacchetti e gli aumenti di ritardo, dovuti alla permanenza nelle code, iniziano a dominare qualsiasi piccolo guadagno nel valore del throughput.



Allocazione di risorse equa

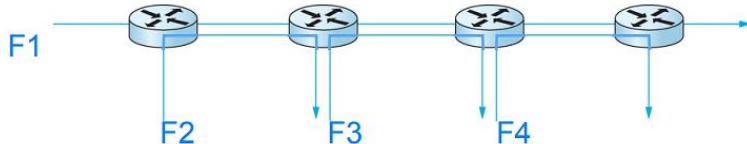
L'utilizzo efficace delle risorse di rete non è l'unico criterio per valutare uno schema di allocazione di risorse: dobbiamo considerarne anche l'equità. Quando diversi flussi condividono una linea, vorremmo che ogni flusso ricevesse la stessa frazione di banda (in assenza di indicazioni aggiuntive).

Ipotizzando che equità implichia egualanza di banda e che tutti i percorsi abbiano la stessa lunghezza, esiste una metrica che può essere utilizzata per quantificare l'equità di una strategia di controllo della congestione. L'indice di equità di Jain (dall'ideatore Raj Jain) è così definito: da un insieme di throughput ($x_1, x_2, x_3, \dots, x_n$), misurati in unità coerenti (ad es: bit/sec) la seguente funzione assegna ai flussi un indice di equità:

$$f(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n x_i^2}$$

L'indice di equità è sempre un numero compreso tra 0 e 1, con il valore 1 che rappresenta la massima equità. Per capire l'intuizione che sta alla base di questa metrica, si consideri il caso in cui tutti gli n flussi ricevono un throughput di 1 unità di dati al secondo, possiamo vedere che l'indice di equità vale: $n^2/n^2=1$.

Esempio: 0.867



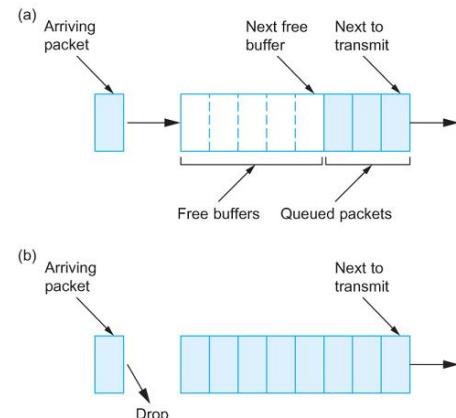
Nel caso invece venga scelto un flusso degli n , a caso con probabilità p ciascuno; la formula di equità diventa la stessa dove x_i diventa $p_i * x_i$

Politiche di gestione delle code

Indipendentemente da quanto sia semplice o complesso il meccanismo di allocazione delle risorse, ogni router deve implementare una politica di gestione delle code che stabilisca come vengono memorizzati nei buffer i pacchetti mentre attendono di essere trasmessi. Si può pensare che l'algoritmo di gestione delle code allochi sia banda (in base alla quale i pacchetti vengono trasmessi) sia spazio nei buffer (in base al quale i pacchetti vengono eliminati). Inoltre, influenza anche, in modo diretto, la latenza di un pacchetto, in quanto determina il tempo di attesa di un pacchetto prima della propria trasmissione.

FIFO (First-In, First-Out) con tailldrop

In FIFO, il primo pacchetto che arriva ad un router è il primo pacchetto ad essere trasmesso da quel router. Dato che la quantità di memoria (buffer) in ciascun router è limitato, se un pacchetto arriva quando la coda è piena, allora il router ignora il pacchetto, come evidenziato in figura b, senza considerare a quale flusso appartenga il pacchetto o quanto importante sia il pacchetto stesso. In questi casi si parla anche di **tail drop** ("eliminazione terminale"), perché vengono eliminati i pacchetti che arrivano all'estremo terminale ("tail") della coda.



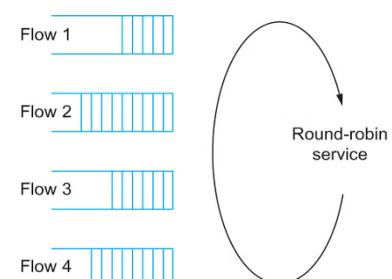
NB: FIFO è una **disciplina di pianificazione** mentre l'eliminazione terminale è una **strategia di eliminazione**. FIFO con tailldrop, essendo il più semplice fra tutti gli algoritmi di gestione delle code è quello più diffusamente utilizzato nei router di Internet in IPv4. Questo semplice approccio al problema della gestione delle code sposta verso i confini della rete tutta la responsabilità per il controllo della congestione e l'allocazione delle risorse. Di conseguenza, la forma di controllo di congestione più utilizzata oggi in Internet ipotizza che non vi sia alcun aiuto da parte dei router: il protocollo TCP si assume la responsabilità di individuare e di gestire la congestione (nel modo che esamineremo più avanti).

FIFO con priorità

Le code a gestione prioritaria sono una semplice variante del meccanismo basilare delle code FIFO. L'idea consiste nel contrassegnare ciascun pacchetto con una priorità e il contrassegno potrebbe trovare posto, ad esempio, nel campo IP denominato **Type of Service (TOS)**. I router, poi, implementano più code FIFO, una per ciascuna classe di priorità: un router trasmette poi, sempre pacchetti estratti dalla coda a priorità più elevata. Il problema delle code prioritarie, ovviamente, sta nel fatto che la coda a priorità potrebbe bloccare tutte le altre nel caso non si svuoti mai. In Internet vengono utilizzate le code prioritarie solo per proteggere i pacchetti più importanti, come gli aggiornamenti di instradamento.

FQ (Fair Queueing)

Il problema principale di FIFO è che non fa distinzione tra diverse sorgenti di traffico: non separa i pacchetti in base al flusso a cui appartengono. Questa situazione pone problemi a due livelli diversi. Innanzitutto, non è chiaro se un algoritmo di controllo della congestione che venga implementato totalmente nella sorgente possa effettuare un controllo adeguato con un aiuto così minimo da parte dei router. Inoltre, come secondo problema, dato che l'intera strategia di controllo della congestione viene implementata alle sorgenti e la gestione FIFO delle code non fornisce alcuna possibilità per stabilire quanto le sorgenti siano efficienti nel realizzare tale strategia, è possibile che una sorgente (o un flusso) che si comporta male possa impegnare una frazione arbitrariamente grande della capacità della rete. Considerando Internet, è possibile che un'applicazione non utilizzi il protocollo TCP e, di conseguenza, sia



esente dal relativo meccanismo di controllo della congestione. La gestione equa delle code (**FQ, fair queueing**) è un algoritmo che è stato proposto per risolvere questo problema. L'idea di FQ consiste nella gestione di una diversa coda per ogni flusso gestito dal router in un certo istante: il router quindi, serve queste code a rotazione (in modalità **round-robin**). Quando un flusso invia pacchetti troppo rapidamente, la sua coda si riempie, quando una coda raggiunge una certa lunghezza, ulteriori pacchetti in arrivo che appartengono a quella coda vengono eliminati. La strategia FQ è stata progettata per un utilizzo congiunto ad un meccanismo di controllo della congestione end-to-end: semplicemente, suddivide il traffico in modo che sorgenti che non si comportano correttamente non interferiscono con quelle che implementano fedelmente l'algoritmo di controllo della congestione end-to-end. FQ assicura anche l'equità di trattamento di un insieme di flussi gestiti da un algoritmo di controllo della congestione.

Nonostante l'idea sia molto semplice, ci sono ancora alcuni dettagli da evidenziare:

- i pacchetti elaborati da un router non hanno necessariamente tutti la stessa lunghezza, per cui, per assegnare in modo equo l'ampiezza di banda di una linea di uscita è necessario prendere in considerazione la lunghezza dei pacchetti.

La strategia Fair Queue, quindi, simula il fatto di inviare bit per bit in modalità round robin e poi usando questo intervallo di tempo per porre in sequenza i pacchetti da trasmettere.

Per un determinato flusso in esame dall'algoritmo, sia:

- P_i la lunghezza del pacchetto i in bit;
- sia S_i l'istante in cui il router inizia la trasmissione del pacchetto i ;
- sia F_i l'istante in cui termina la trasmissione del pacchetto i ;

Se P_i viene espresso come il numero di tick dell'orologio necessarie per trasmettere il pacchetto i (1 tick per servire 1 bit), allora è facile verificare che $F_i = S_i + P_i$. Quando iniziamo la trasmissione del pacchetto i ? Dipende dal fatto che il pacchetto i arrivi prima o dopo l'istante in cui il router ha terminato la trasmissione del pacchetto $i-1$ del flusso in esame.

- sia A_i l'istante in cui il pacchetto i arriva al router.

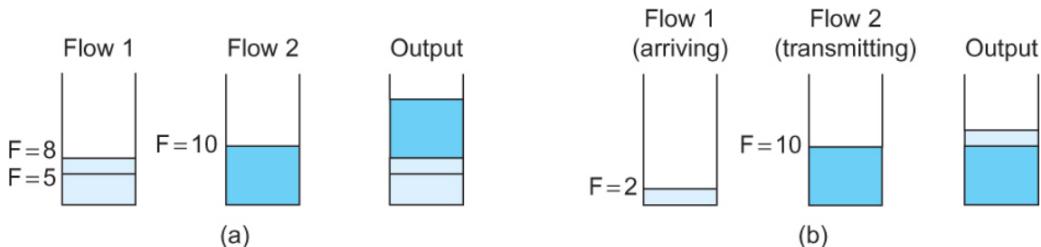
Allora $S_i = \max(F_{i-1}, A_i)$, di conseguenza possiamo calcolare: $F_i = \max(F_{i-1}, A_i) + P_i$

L'algoritmo:

- per ciascun flusso, calcoliamo il timestamp F_i di ciascun pacchetto che arriva;
- abbiamo il timestamp di F_{i-1} , il timestamp dell'ultimo pacchetto, l'arrivo e la sua lunghezza;
 - calcoliamo $F_i = \max(F_{i-1}, A_i) + P_i$
- il prossimo pacchetto x da trasmettere è sempre quello con il timestamp F_i più basso tra tutti i flussi;

Garantisce un'elevata equità tra i flussi, e una piccolissima differenza tra di essi.

Esempio: (a): pacchetti che finiscono prima vengono inviati prima; (b): l'invio di un pacchetto già in corso è completato.



ES: Un router sta servendo tre flussi A, B, C secondo la politica di accodamento equo (Fair Queueing). I pacchetti in coda, con il tempo necessario per trasmetterli (in ms), sono i seguenti:

A1 = 5, A2 = 7, A3 = 2, B1 = 4, B2 = 11, C1 = 9, C2 = 6.

- (a) In che ordine vengono trasmessi i pacchetti?
- (b) A che istante termina la trasmissione del pacchetto A3?

- Risposta

(a) Calcolo l'ordine di conclusione dell'invio dei pacchetti per ciascun flusso:

A1 = 5, A2 = 12, A3 = 14;

B1 = 4, B2 = 15;

C1 = 9; C2 = 15.

Quindi l'ordine di trasmissione è il seguente: **B1, A1, C1, A2, A3, B2, C2.**

NB: L'algoritmo FQ, prima calcola il timestamp di fine trasmissione di un pacchetto (in questo caso si ipotizza che siano tutti arrivati a t=0) e successivamente il pacchetto da trasmettere è sempre quello con timestamp di fine trasmissione più piccolo.

(b) Sommando i tempi necessari per la trasmissione, A3 termina a 27ms.

Controllo di congestione in TCP

La strategia basilare di TCP consiste nell'invio di pacchetti nella rete senza effettuare prenotazioni e, in seguito, reagire agli eventi osservabili che accadono. Il protocollo TCP ipotizza che i router della rete usino, semplicemente, una gestione FIFO delle code, ma può funzionare anche con una gestione FQ. In generale l'idea del controllo di congestione in TCP affida ad ogni sorgente la responsabilità di determinare quanta capacità sia disponibile nella rete, in modo da sapere quanti pacchetti in transito sia possibile avere senza provocare problemi. Quando una sorgente sa di avere in transito un tale numero di pacchetti, usa l'arrivo di un ACK come segnalazione del fatto che uno dei propri pacchetti è uscito dalla rete, per cui ne può inserire in rete un altro senza timore di aumentare il livello di congestione. Dato che usa i messaggi ACK per controllare la velocità di trasmissione dei pacchetti, si dice che **TCP è self-clocking** (auto temporizzato). Ovviamente, determinare prima di tutto la capacità disponibile non è un compito facile e, per rendere peggiore la situazione, la banda disponibile cambia nel tempo, a causa di altre connessioni che vengono instaurate e terminate.

Aumento additivo/diminuzione moltiplicativa (AIMD)

Il protocollo TCP gestisce, per ogni connessione, una nuova variabile di stato, chiamata **CongestionWindow (finestra di congestione)**, che viene usata dalla sorgente per limitare la quantità di propri dati in transito nella rete in un certo istante. La finestra di congestione è, per il controllo di congestione, la controparte della finestra annunciata (AdvertisedWindow) nel controllo di flusso. Il protocollo TCP viene modificato in modo che il numero massimo consentito di byte di dati non confermati sia, ora, il minimo fra la finestra di congestione e la finestra annunciata. La finestra TCP diventa così calcolata:

$$\text{MaxWindow} = \min(\text{CongestionWindow}, \text{AdvertisedWindow})$$

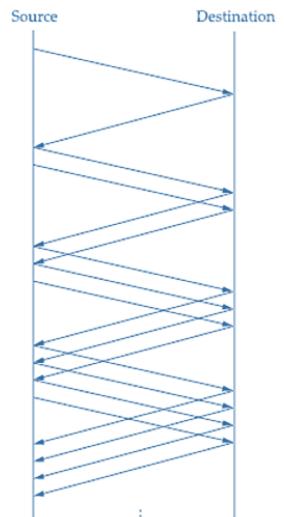
$$\text{EffectiveWindow} = \text{MaxWindow} - (\text{LastByteSent} - \text{LastByteAcked})$$

In sostanza, MaxWindow sostituisce AdvertisedWindow nel calcolo di EffectiveWindow. In questo modo una sorgente TCP non può spedire più velocemente di quanto sia accettabile per il componente più lento: la rete o l'host di destinazione. Il problema è: come TCP apprende il valore appropriato per CongestionWindow? La sorgente TCP impone il valore CongestionWindow basandosi sul livello di congestione di rete da essa percepito: questo richiede la diminuzione della finestra di congestione quando il livello di congestione percepito sale e viceversa. Considerato nella sua globalità, questo meccanismo viene comunemente chiamato aumento additivo/diminuzione moltiplicativa (**AIMD – Additive Increase, Multiplicative Decrease**).

Come può la sorgente determinare che la rete è congestionata? la risposta si basa sull'osservazione che il motivo principale per cui i pacchetti non vengono consegnati, provocando la loro scadenza è proprio la congestione. Il protocollo TCP interpreta la scadenza delle temporizzazioni dei pacchetti come un segnale di congestione e riduce la velocità di trasmissione. Ogni volta che scade un pacchetto, TCP impone CongestionWindow alla metà del suo valore precedente (**diminuzione moltiplicativa** di AIMD). Dobbiamo ovviamente essere anche in grado di aumentare la finestra quando la rete non è congestionata. Ogni volta che una sorgente invia una quantità di pacchetti pari al valore di CongestionWindow con successo, incrementa il valore di CongestionWindow per l'equivalente di un pacchetto (andamento lineare). Questo meccanismo è l'**aumento additivo**. In pratica TCP non attende la conferma per un'intera finestra di pacchetti prima di aggiungere un pacchetto alla CongestionWindow ma incrementa un po' le sue dimensioni ogni volta che arriva un ACK come segue:

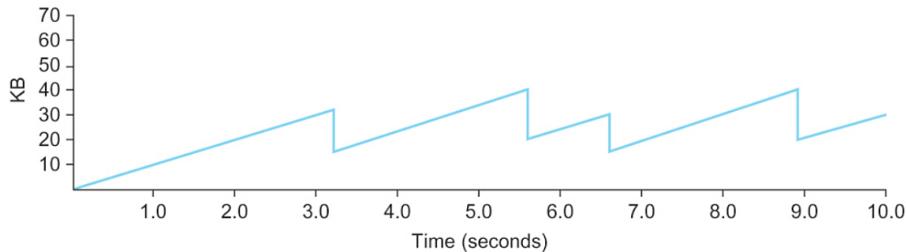
$$\text{Increment} = \text{MSS} \times (\text{MSS}/\text{CongestionWindow})$$

$$\text{CongestionWindow} += \text{Increment}$$



In sostanza, invece di aumentare CongestionWindow di un intero MSS per ogni RTT, lo incrementa per una frazione di MSS ogni volta che viene ricevuto un ACK, nell'ipotesi che ciascun ACK confermi la ricezione di un numero di byte uguale a MSS. Tale frazione vale $MSS/CongestionWindow$.

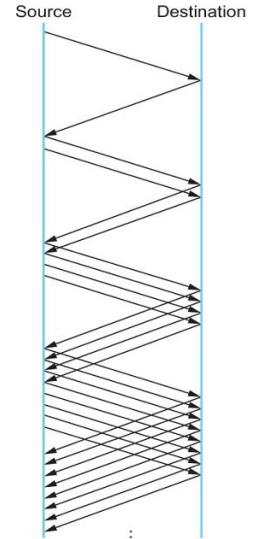
Il concetto importante di AIMD è che la sorgente è disposta a ridurre la propria finestra di congestione molto più rapidamente di quanto la aumenti, in netto contrasto con la strategia ad aumento additivo.



Partenza lenta

Il meccanismo di aumento additivo appena descritto è l'approccio corretto da usare quando la sorgente si trova ad operare in condizioni prossime alla capacità disponibile della rete, ma richiede troppo tempo per fare decollare la connessione quando essa inizia la trasmissione. Il protocollo TCP, quindi, dispone di un secondo meccanismo chiamato **partenza lenta (slow start)**, che viene usato per aumentare rapidamente la dimensione della finestra di congestione all'inizio della connessione: la partenza lenta aumenta la finestra di congestione esponenzialmente, anziché linearmente. In particolare, la sorgente inizia impostando la CongestionWindow al valore corrispondente ad un solo pacchetto. Quando arriva la conferma ACK per tale pacchetto, TCP aggiunge 1 al valore di CongestionWindow. Dopo aver ricevuto le due conferme, TCP aumenta la CongestionWindow di 2, e così via. Il risultato finale è che TCP, in pratica, raddoppia il numero di pacchetti in transito ad ogni intervallo di tempo uguale a RTT. Esistono due particolari situazioni in cui la partenza lenta viene utilizzata:

- All'inizio della connessione, in cui la fonte di trasmissione non ha idea di quanti pacchetti sarà in grado di trasmettere senza congestionare la rete.
 - In questo caso la partenza lenta continua a moltiplicare per due la CongestionWindow finché non avviene una perdita di pacchetto che causerà il dimezzamento della CongestionWindow.
- All'occorrenza di una caduta di connessione.



Ritrasmissione veloce

Il meccanismo TCP denominato **ritrasmissione veloce (fast retransmit)** tenta di risolvere il problema di lunghi periodi di tempo durante i quali la connessione va in contatto a fenomeni di stagnazione in attesa che scada un temporizzatore di un pacchetto. Questo meccanismo non cambia la normale temporizzazione dei pacchetti ma ne va solo a migliorare la funzionalità. L'idea della ritrasmissione veloce è molto semplice: ogni volta che un pacchetto di dati arriva a destinazione, il ricevente risponde con una conferma, anche se con numero di sequenza che è già stato confermato. In questo modo, quando arriva un pacchetto fuori sequenza, che non può quindi essere confermato da TCP perché non sono ancora arrivati i pacchetti precedenti, il protocollo rispedisce la stessa conferma spedita la volta precedente: questa seconda trasmissione della stessa conferma viene chiamata **ACK duplicato**. Quando il mittente vede un ACK duplicato, sa che il ricevente ha certamente ricevuto un pacchetto fuori sequenza, da cui si deduce che un pacchetto precedente potrebbe essere andato perduto. Dato che tale pacchetto precedente potrebbe essere stato semplicemente ritardato dalla rete, anziché perduto, il mittente attende di vedere un certo numero di ACK duplicati, dopodiché

ritrasmette il pacchetto mancante. In pratica il protocollo TCP attende di vedere tre ACK duplicati prima di ritrasmettere il pacchetto.

Recupero veloce

Possiamo realizzare un ultimo miglioramento. Quando il meccanismo di trasmissione veloce segnala una congestione, invece di riportare la finestra di congestione al suo valore iniziale unitario ed eseguire nuovamente la partenza lenta, si possono usare i pacchetti di conferma, ACK, che si trovano ancora in transito per pianificare l'invio dei pacchetti. Questo meccanismo è detto **recupero veloce (fast recovery)**. Questo meccanismo elimina la fase di partenza lenta che avviene tra il momento in cui la ritrasmissione veloce rileva la perdita di un pacchetto e quello in cui inizia l'aumento additivo.

Meccanismi di prevenzione della congestione

È importante ribadire che la strategia di TCP è quella di controllare la congestione quando questa si verifica e non di prevenirla. Una alternativa valida però, è appunto, quella di attuare dei meccanismi di prevenzione della congestione, ovvero di valutare quando questa possa accadere e agire di conseguenza. Questo approccio non è utilizzato da TCP, e viene detto **collision avoidance**.

Randomly Early Detection (RED)

Nel meccanismo di rilevazione casuale anticipata, ogni router viene programmato perché tenga sotto controllo la lunghezza delle proprie code e, quando si accorge che una congestione è imminente, segnali alla sorgente di modificare la propria finestra di congestione. L'invio di un messaggio di congestione ad una sorgente è tutt'altro che esplicito, in quanto solitamente RED viene realizzato in modo da segnalare implicitamente alla sorgente la congestione, eliminando volutamente uno dei suoi pacchetti. La sorgente, quindi, riceve a tutti gli effetti la segnalazione nel momento della scadenza del temporizzatore di quel pacchetto.

Come fa il meccanismo RED a decidere quando scartare un pacchetto?

1. RED calcola la lunghezza media della coda, chiamata **AvgLen** (quanto è piena)
2. Inoltre, RED ha due variabili: **MinThreshold** e **MaxThreshold**
 - Quando un pacchetto arriva, 1 compara AvgLen con questi due valori e:
 - se $\text{AvgLen} \leq \text{MinThreshold}$: metti in coda il pacchetto.
 - se $\text{MinThreshold} < \text{AvgLen} < \text{MaxThreshold}$: calcola la probabilità P e droppa un pacchetto con la probabilità P dove P è: $(\text{AvgLen} - \text{MinThreshold}) / (\text{MaxThreshold} - \text{MinThreshold})$
 - se $\text{MaxThreshold} \leq \text{AvgLen}$: droppa il pacchetto.

Impedire la congestione alla sorgente

L'idea generale dei meccanismi di impedimento della congestione alla sorgente si basano sull'idea di osservare segnali dalla rete che indicano che il buffer di qualche router si sta riempiendo e una congestione sta per verificarsi. Per esempio, il mittente si accorge di un aumento misurabile del valore di RTT per ogni successivo pacchetto che invia, a mano a mano che le code di pacchetti si riempiono nei router della rete. Un particolare algoritmo può sfruttare questa osservazione.

Network Security

Introduzione: nell'ambito della sicurezza informatica la sicurezza delle reti è una problematica che nasce nel momento in cui si hanno più computer interconnessi fra loro, cioè in una rete di calcolatori: essi, infatti, offrono diverse vulnerabilità sfruttabili, più o meno facilmente, da terzi per intromettersi nel sistema ed intercettarne i dati. Quando la rete è aperta a Internet, un'importante aggravante deriva dal fatto che Internet è nata come rete didattica in un ambiente universitario e le sue regole non prevedono metodi di sicurezza intrinseci alla struttura: le difese devono essere messe in atto sulle macchine stesse o creando strutture di rete particolari. La sicurezza della rete comprende quindi tutte quelle politiche e pratiche adottate per prevenire e monitorare l'accesso non autorizzato, l'uso improprio, la modifica o la causa di denial of service di una rete di computer e risorse accessibili alla rete.

Il costo di un servizio di sicurezza deve sempre essere confrontato con il valore di un danno provocato; a rigor di logica si vuole che:

$$\text{Costo sicurezza} < \text{Costo del danno} \times P(\text{danno})$$

Com'è possibile diminuire la vulnerabilità di una rete?

- Evitando che ci siano falliche di sicurezza; quindi concerne tutti gli aspetti di progettazione e implementazione.
- Diminuendo la probabilità degli attacchi tramite servizi che rendono un attacco difficile, costoso e non conveniente per l'attaccante.

Un buon approccio è quindi impiegare abbastanza sforzo da rendere non vantaggioso l'attacco. La sicurezza è sempre un compromesso tra costi, complessità e funzionalità. Un modo facile per avere un sistema robusto è avere un sistema che non faccia nulla (o il minimo indispensabile); solitamente sistemi sicuri che offrono molte funzionalità sono più difficili da realizzare e più complessi da utilizzare.

Obiettivi della sicurezza

La sicurezza delle reti si pone tre obiettivi principali: la triade **CIA: Confidentiality, Integrity and Availability**.

- **Riservatezza:** dobbiamo proteggere le nostre informazioni riservate. Un'organizzazione deve proteggersi da quelle azioni dannose che ne mettono in pericolo la riservatezza delle proprie informazioni.
- **Integrità:** le informazioni devono essere costantemente modificate. Le modifiche devono essere fatte solo da entità autorizzate e attraverso meccanismi autorizzati.
- **Disponibilità:** le informazioni create e archiviate da un'organizzazione devono essere disponibili per le entità autorizzate. Le informazioni devono essere costantemente modificate, devono essere accessibili alle entità autorizzate.

Non è detto che questi tre aspetti debbano per forza coesistere.

Tipologie di attacco

Distinguiamo prima il concetto di minaccia e attacco.

- una **minaccia** è una potenziale violazione della sicurezza; un rischio.
- un **attacco** è un'effettiva violazione o tentativo di violazione della sicurezza.

Seguono alcune definizioni utili:

- **Attacco alla sicurezza:** qualsiasi azione che comprometta la sicurezza di informazioni di un'organizzazione.

- **Servizio di sicurezza:** insieme di funzionalità (meccanismi) che mira a neutralizzare attacchi alla sicurezza tramite meccanismi di sicurezza.
- **Meccanismo di sicurezza:** meccanismo progettato per prevenire o riparare danni causati da attacchi alla sicurezza.

Esistono varie tipologie di attacchi alla sicurezza, essi possono essere suddivisi in:

- Attacchi alla confidenzialità;
- Attacchi all'integrità;
- Attacchi alla disponibilità;

Seguono ora esempi di attacchi per ciascuna categoria:

- ❖ Attacchi alla confidenzialità:
 - **Snooping:** consiste nell'accesso non autorizzato a dati o computer e nell'intercettazione di dati durante le trasmissioni (in questo secondo caso si parla di **sniffing**).
 - **Analisi del traffico:** mira all'ottenimento di certi tipi di informazione monitorando il traffico online, per esempio metadati quali: la dimensione dei pacchetti, il tempo di trasmissione, il mittente, etc.
- ❖ Attacchi all'integrità:
 - **Modifica:** modifica di dati da parte di un attaccante che intercetta il messaggio.
 - **Masquerading o spoofing:** quando l'attaccante si spaccia per qualcun altro, magari il mittente o il destinatario o entrambi (man-in-the-middle).
 - **Replaying:** quando un attaccante ottiene una copia di un messaggio inviato da un utente e cerca di inviarlo di nuovo senza avervi apportato modifiche. Ciò altera il flusso corretto della comunicazione.
 - **Attacchi di ripudio:** quando una delle due parti della comunicazione nega di aver preso parte alla comunicazione stessa.
- ❖ Attacchi alla disponibilità:
 - **Denial of service (DoS):** attacchi che mirano a rallentare o interrompere totalmente il servizio di un sistema. Questo può avvenire, per esempio, eliminando le risposte del server per forzare il client a inviare nuovamente le richieste o eliminare le richieste stesse per forzare il client a inviarle di nuovo.

Gli attacchi possono inoltre essere suddivisi in **attacchi attivi e passivi**.

- **Attacchi attivi:** alterano le risorse del sistema o le loro operazioni/servizi;
- **Attacchi passivi:** apprendono o fanno uso di informazioni del sistema senza intaccarne il suo funzionamento e/o le sue risorse.

Attacks	Passive/Active	Threatening
Snooping Traffic analysis	Passive	Confidentiality
Modification Masquerading Replaying Repudiation	Active	Integrity
Denial of service	Active	Availability

Queste due tipologie di attacco richiedono due approcci diversi:

- Gli attacchi attivi, poiché alterano il sistema, sono molto più facili da identificare ma difficili da prevenire completamente. Ci si focalizza quindi sul rilevamento.
- Gli attacchi passivi sono molto difficili da rilevare, per cui il modo migliore di affrontarli è tramite adottamento di tecniche di prevenzione.

Servizi di sicurezza

Un servizio di sicurezza è un insieme di funzionalità che mirano a neutralizzare attacchi alla sicurezza tramite meccanismi di sicurezza.

Seguono alcuni dei principali servizi di sicurezza:

- **Autenticazione:** garanzia che l'entità che comunica sia quella che dichiara di essere.
- **Controllo degli accessi:** prevenzione di uso non autorizzato (lettura, scrittura, modifica etc.) di una risorsa. Il Denial of Service non rientra in questa categoria.
- **Confidenzialità dei dati:** protezione dei dati dalla divulgazione non autorizzata.
- **Integrità dei dati:** garanzia che i dati ricevuti siano stati inviati da un'entità autorizzata.
- **Non ripudio:** protezione dal ripudio di una delle parti della comunicazione:
 - **dell'origine:** provare che il messaggio sia stato inviato dalla parte specificata.
 - **della destinazione:** provare che il messaggio sia stato ricevuto dalla parte specificata.
- **Disponibilità:** garanzia che una risorsa sia disponibile per l'uso destinato.

I servizi di sicurezza sono composti da vari meccanismi di sicurezza; essi sono funzioni di base per costruire i servizi di sicurezza. È quindi necessario combinarne più insieme. Segue una lista dei principali meccanismi di sicurezza:

- **Cifratura:** nascondere o mascherare i dati;
- **Firma digitale:** garantisce il ricevente sul mittente del messaggio;
- **Scambio di autenticazione:** le parti dimostrano la propria identità l'un l'altro;
- **Traffic Padding:** inserire dati fasulli nel traffico dati reale per evitare l'analisi del traffico;
- **Controllo del routing:** selezione e modifica dei percorsi del traffico per evitare intercettazioni;
- **Autenticazione notariale:** una terza parte fidata controlla le comunicazioni (ad es. per prevenire il ripudio);
- **Controllo dell'accesso:** metodi per dimostrare che un'entità ha diritti di accesso alle risorse.

Non tutti i meccanismi sono adatti per implementare ciascun servizio di sicurezza:

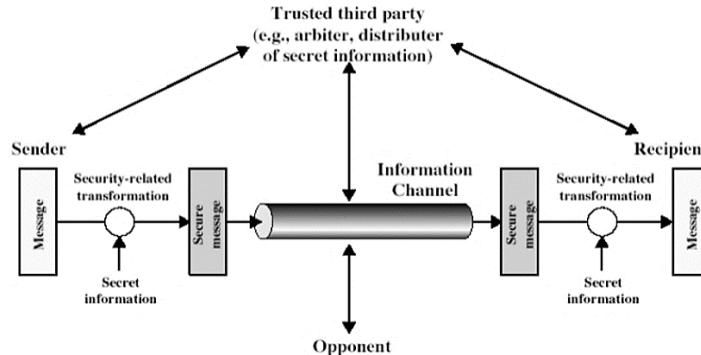
Service	Mechanism							
	Encipher- ment	Digital signature	Access control	Data integrity	Authenti- cation exchange	Traffic padding	Routing control	Notari- zation
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Non-repudiation		Y		Y				Y
Availability				Y	Y			

Modelli per la network security

Un modello in questo ambito è un'astrazione che ci fornisce delle linee guida per progettare servizi di sicurezza e per scegliere le corrette strategie da utilizzare. Esistono due principali modelli:

- modello del **canale insicuro**
- modello dell'**accesso di rete**

Segue il modello del canale insicuro (Dolev-Yao):



L'idea è quella di un canale (di qualsiasi tipo) che permette di trasmettere dati. Il **sender** invia dati attraverso il canale e il **recipient** li riceve. Durante una comunicazione, un attaccante (l'**opponent**) può leggere, alterare, replicare i messaggi che transitano sul canale, questo accade quando si scambiano dati sulla rete Internet.

Si possono applicare sistemi di sicurezza mediante trasformazioni di messaggi prima e dopo l'invio in maniera da poter sventare un attacco passivo o potersi accorgere di un attacco attivo. Le trasformazioni di sicurezza usano delle **informazioni segrete** che, in quanto tali, non sono note all'opponent: sono l'unica cosa che l'attaccante non sente.

Per poter usare questo modello è quindi necessario decidere:

- che trasformazioni di sicurezza applicare, serviranno quindi due funzioni: f e f^{-1} ;
- e generare le **informazioni segrete** (dette chiavi);
- come trasmettere tali informazioni dall'altra parte.

Implementazione dei servizi di sicurezza

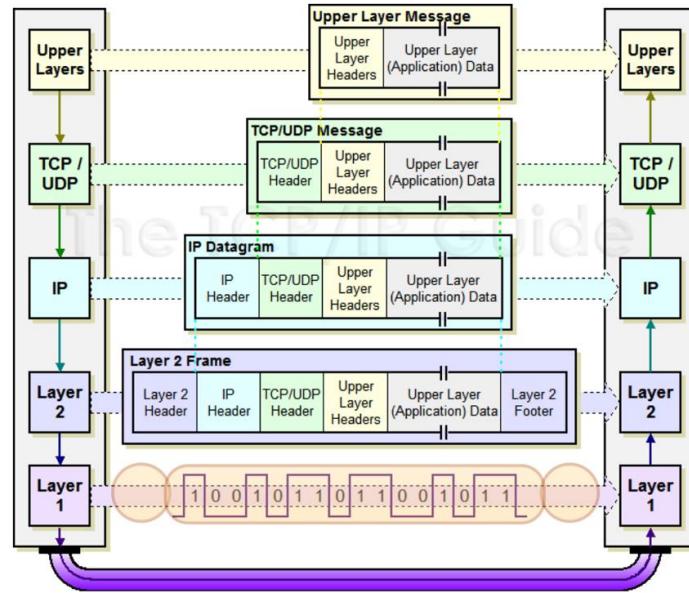
Dove vanno implementati i servizi di sicurezza?

Ogni livello del modello OSI o del modello TCP/IP implementa un canale (virtuale), i servizi di sicurezza possono essere implementati su tutti (quasi) questi canali in vari modi; uno di questi prevede di implementare il modello Dolev-Yao canale per canale: su ogni link si attua una trasformazione; switch e router devono possedere le chiavi di cifratura perché hanno bisogno di decifrare i dati per inoltrarli.

Router e switch devono essere dunque affidabili. In questo modo si interpreta l'intera rete come se fosse un canale insicuro; questo avviene dal livello tre in su perché la quantità di dati coperti è minore, quindi più facile da proteggere.

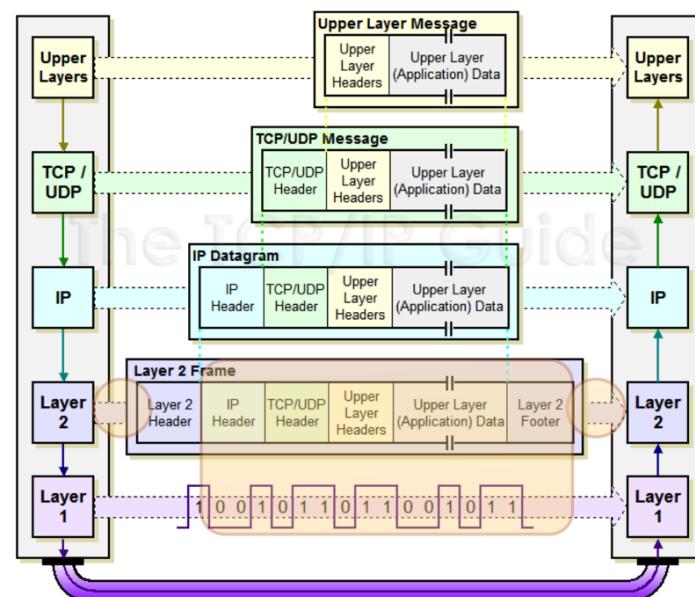
1. Sicurezza a livello fisico

La sicurezza a livello fisico consiste nella mera protezione dei cavi tramite lucchetti, sportelli, schermature etc. in modo tale che tutti i bit trasmessi siano protetti.



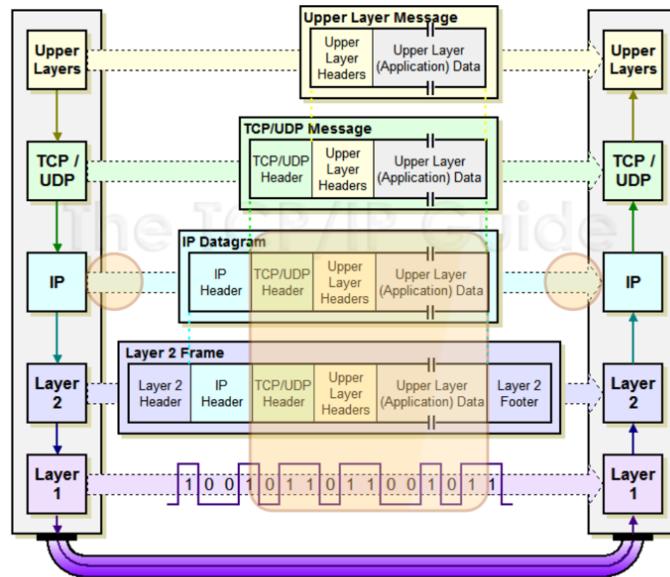
2. Sicurezza a livello data link

Il lavoro avviene all'interno dell'interfaccia di rete dove la trasformazione mette in sicurezza il payload ma non l'intestazione. Quest'ultima non può essere cifrata perché ciò impedirebbe al destinatario di capire quali sono i messaggi a lui destinati. Per un attaccante è quindi possibile vedere chi parla con chi. Inoltre, nei router il messaggio viene decapsulato e incapsulato per essere trasmesso al livello 3, per fare questo il payload viene scoperto e quindi l'intero messaggio diviene visibile.



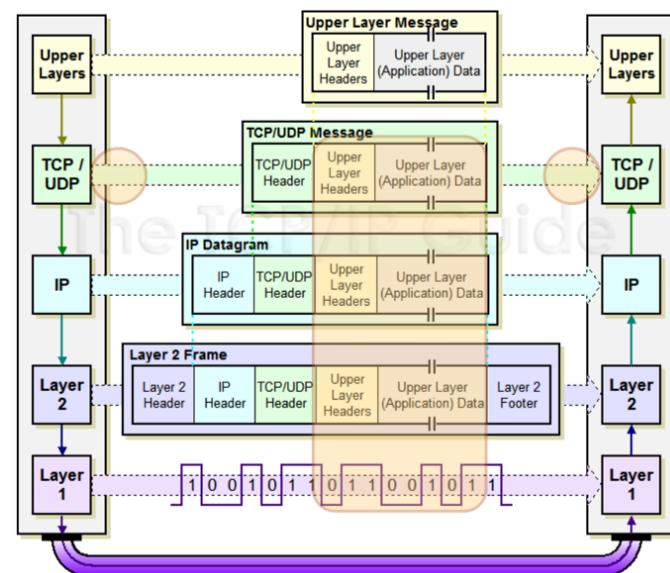
3. Sicurezza a livello di rete

Considerando la comunicazione host-to-host, è possibile mettere in sicurezza tutto quello che avviene all'interno di un host e nella comunicazione con un altro host. Ciò che viene messo in sicurezza è sempre il payload, per i router questo va bene perché per funzionare basta loro l'intestazione di livello 3; però così è ancora possibile capire quale macchina parla con quale macchina. Per implementare la sicurezza a livello 3 è necessario modificare IP (o implementare IPSec sopra di esso) e quindi il sistema operativo. Tutti i segmenti UDP e TCP viaggiano incapsulati quindi i programmati non si accorgono di nulla.



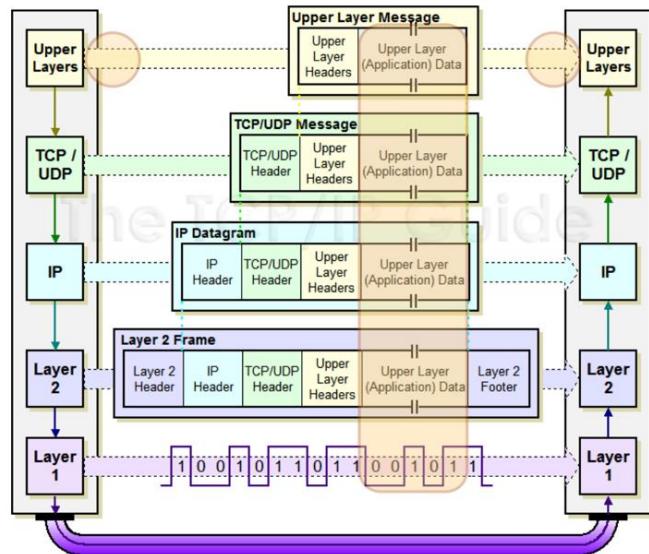
4. Sicurezza a livello di trasporto

Si offrono al programmatore socket speciali (SSL). Rimane scoperto l'header per cui si possono vedere l'indirizzo e la porta dei processi che comunicano. A livello 4 si implementa la sicurezza senza intervenire sul sistema operativo: bisogna avere e saper compilare il codice per poter usare queste socket.



5. Sicurezza a livello di applicazione

Per quel che riguarda i livelli superiori, vengono implementati sistemi di sicurezza ad hoc per le specifiche applicazioni. Copre solo i dati dell'applicazione.



Level	Services	Protocols/applications
7 Application	End-to-end services	PGP, S/MIME, SAML, SSH, EMV, SET, ...
6 Presentation		
5 Session	Authentication encryption traffic analysis...	SSL/TLS, Secure UDP
4 Transport		
3 Network	Authentication, encryption	IPSec
2 Datalink	Authentication for link access, WEP, WPA, 802.1X confidentiality	
1 Physical		

Cifratura

Segue un'introduzione al meccanismo di cifratura:

La cifratura è un **meccanismo di sicurezza** fondamentale usato in svariati servizi. Ne esistono due tipologie:

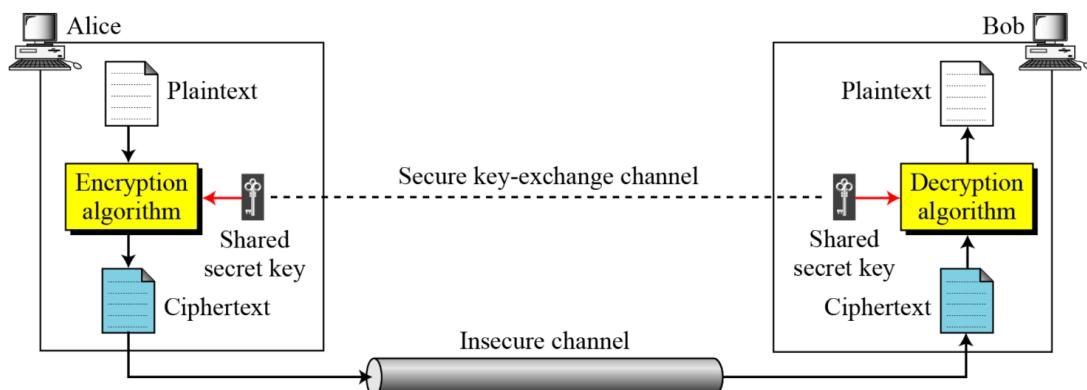
1. Cifratura **simmetrica** (o cifratura a chiave privata)
2. Cifratura **asimmetrica** (o cifratura a chiave pubblica)

Glossario per la terminologia di base:

Termino	Definizione
Plaintext	messaggio originale o "testo in chiaro"
Ciphertext	messaggio codificato/cifrato
Cipher	algoritmo che trasforma il <i>plaintext</i> in <i>ciphertext</i>
Key (chiave)	informazione usata dal <i>cipher</i> nota solo al mittente e al ricevitore della comunicazione per cifrare il <i>plaintext</i>
Encipher (encrypt)	conversione del <i>plaintext</i> in <i>ciphertext</i>
Decipher (decrypt)	recupero del <i>plaintext</i> dal <i>ciphertext</i>
Cryptography (crittografia)	studio dei principi e dei metodi di cifratura
Cryptoanalysis (codebreaking)	studio dei principi e metodi per decifrare un <i>ciphertext</i> senza conoscere la chiave
Cryptology (crittologia)	campo che comprende sia la <i>crittografia</i> sia la <i>criptoanalisi</i>

Cifratura simmetrica

In questo particolare tipo di cifratura il sender e il recipient condividono le stesse informazioni segrete (key) che, ovviamente, non sono note all'attaccante. Viene anche chiamata cifratura a chiave privata.



Richiede due requisiti fondamentali per funzionare correttamente:

- Un buon algoritmo di cifratura per cui, senza conoscere la chiave, sia impossibile ottenere il *plaintext* dal *ciphertext* (il testo in chiaro dal testo cifrato).
- Una chiave segreta nota solo al sender e al recipient.

Matematicamente si hanno due funzioni tali che:

$$\text{ciphertext: } Y = E_K(X) \rightarrow \text{cifratura}$$

$$\text{plaintext: } X = D_K(Y) \rightarrow \text{decifrazione}$$

$$D_K(E_K(X)) = X$$

$$D_K(E_H(X)) = \text{errore se } H \neq K$$

Si assume inoltre che l'algoritmo usato per la cifratura sia noto anche all'attaccante, infatti, secondo il **principio di Kerckhoff**, la sicurezza del messaggio deve dipendere solo ed esclusivamente dalla segretezza della chiave usata (non quindi dalla segretezza dell'algoritmo). L'**unica** informazione non nota all'attaccante è la chiave. Infatti, tutti gli algoritmi di cifratura, tranne quelli militari, sono open source.

I sistemi crittografici si caratterizzano per:

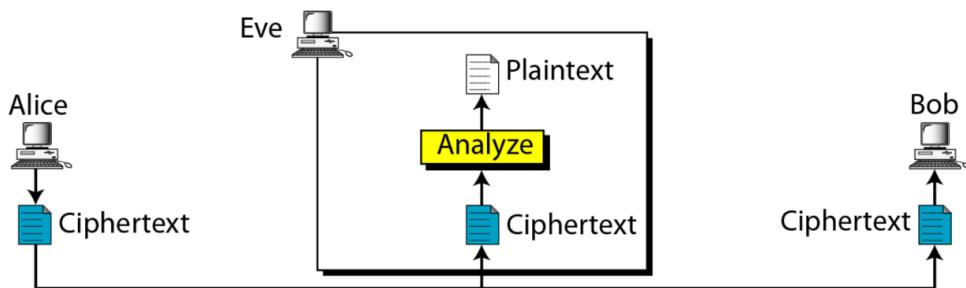
- **Tipo di operazioni di cifratura usate:**
 - sostituzione (di bit, bytes, caratteri, ... o anche dell'intero plaintext)
 - trasposizione (riadattamento del plaintext)
 - prodotto (una composizione delle due operazioni precedenti)
- **Numero di chiavi utilizzate:**
 - singola chiave simmetrica/privata/convenzionale
 - due chiavi asimmetriche/pubbliche
- **Modalità con cui il plaintext viene elaborato:**
 - a blocchi → i dati vengono elaborati in blocchi di misura prefissata
 - a flusso → i dati vengono processati un elemento per volta

Crittoanalisi

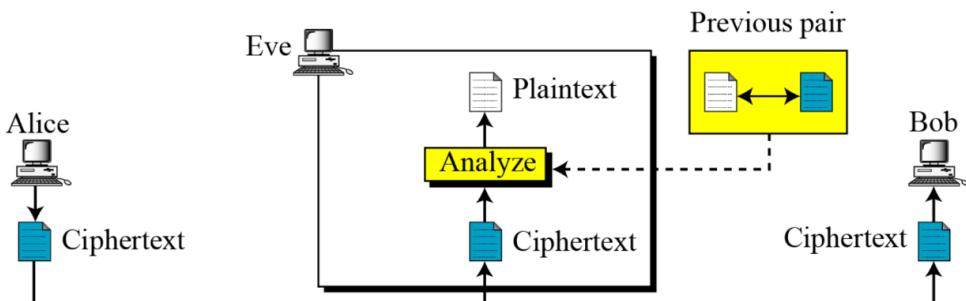
La crittoanalisi è il processo che tenta di scoprire il testo in chiaro corrispondente al testo cifrato, o meglio, di scoprirne la chiave. Prevede diverse tipologie di approcci (attacchi):

Segue una descrizione delle tipologie di attacco principali:

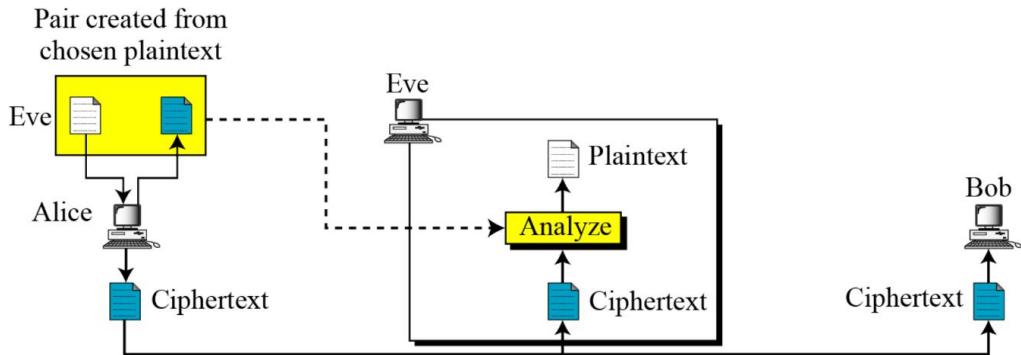
Ciphertext-Only → il crittoanalista/attaccante, conoscendo solo l'algoritmo di cifratura e il testo cifrato (ciphertext), tramite lo sniffing dei dati cerca di leggere il messaggio o di trovare la chiave.



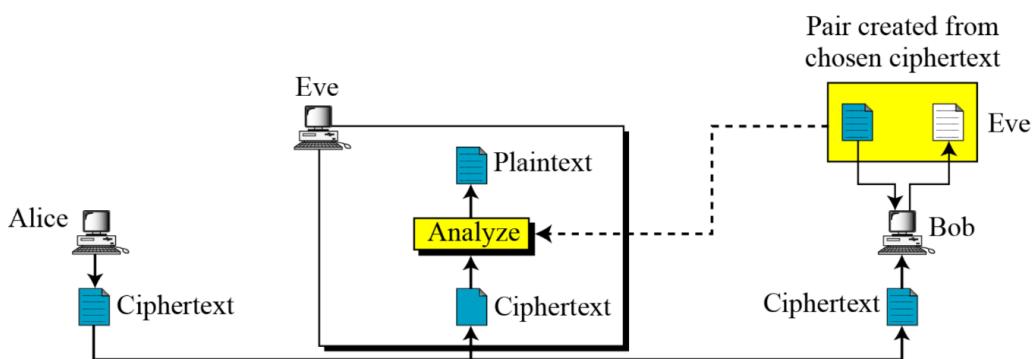
Known-Plaintext → il crittoanalista/attaccante conosce il testo in chiaro e il corrispondente testo cifrato (M ed M') e, tramite lo sniffing di altri messaggi cifrati cerca di scoprire la chiave di cifratura.



Chosen-Plaintext → l'attaccante riesce a fare in modo che il messaggio da cifrare sia quello che vuole lui forzando il mittente a inviare uno specifico messaggio con lo scopo di analizzarne il risultato cifrato e di ottenere la chiave.



Chosen-Ciphertext → l'attaccante è in grado di scegliere uno o più ciphertext e ottenerne il testo in chiaro



Dei buoni algoritmi di cifratura, per essere considerati tali, devono essere in grado almeno di affrontare attacchi di tipo Known-Plaintext.

Come fa un attaccante a sferrare un attacco? In generale può adottare due tipi di strategie:

- Crittoanalitiche
- Brute-force

Le strategie **crittoanalitiche** cercano falle e debolezze nella struttura matematica dell'algoritmo di cifratura.

Esempio

$$E_k(M) = M \oplus K$$

$$C = E_k(M) = M \oplus K$$

$$D_k(E_k(M)) = C \oplus K \text{ quindi:}$$

$$D_k(E_k(M)) = (M \oplus K) \oplus K = M \oplus (K \oplus K) = M \oplus 0 = M$$

\oplus	0	1
0	0	1
1	1	0

$$E_k(M) = M \oplus K$$

$$C = E_k(M) = M \oplus K \text{ quindi:}$$

$$C \oplus M = M \oplus K \oplus M = K \oplus 0 = K$$

Le strategie **brute-force** prevedono di provare tutte le possibili chiavi fino a trovare quella corretta. Necessitano però di saper riconoscere quando una chiave è errata.

Sicurezza incondizionata e sicurezza computazionale

È possibile ottenere due tipi di sicurezza dal processo di cifratura:

- Sicurezza incondizionata:** non importa quanta potenza di calcolo o tempo si abbia a disposizione, l'algoritmo non potrà mai essere violato perché il testo cifrato non fornisce abbastanza informazioni per determinare univocamente il testo in chiaro corrispondente.
- Sicurezza computazionale:** fare in modo che il costo necessario per rompere l'algoritmo sia superiore al valore delle informazioni cifrate.

La strategia brute-force, a livello teorico, è sempre possibile e il tempo stimato per il completamente dipende dalla dimensione della chiave e dalla potenza di calcolo a disposizione.

Key Size (bits)	Number of Alternative Keys	Time required at 1 decryption/ μ s	Time required at 10^6 decryptions/ μ s
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)	$26! = 4 \times 10^{26}$	$2 \times 10^{26} \mu\text{s} = 6.4 \times 10^{12} \text{ years}$	$6.4 \times 10^6 \text{ years}$

Cifrario di Vigenere e One-Time Pad

Si tratta di un semplice algoritmo di sostituzione polialfabetico in cui ogni lettera della chiave indica il numero di spostamenti in avanti del corrispondente carattere nel messaggio originale. Per la decodifica si attuano spostamenti all'indietro.

Esempio:

key:	deceptivedeceptivedeceptive
plaintext:	wearediscoveredsaveyourself
ciphertext:	ZICVTWQNGRZGVTWAVZHCOYGLMGJ

È vulnerabile ad attacchi crittoanalitici in quanto la frequenza delle lettere non viene cancellata dall'algoritmo di cifratura (usa la chiave "deceptive" che si ripete).

Per questo motivo è stato tentato un miglioramento: **One-Time Pad** o **cifrario di Vernam**.

Si supponga di avere una chiave di cifratura lunga come il testo da cifrare ma che sia una sequenza di caratteri **totalmente casuale**. Questo tipo di cifratura risulta robusto perché il testo cifrato non fornisce nessuna relazione statistica con il testo in chiaro; inoltre, per ogni messaggio in chiaro esiste più di una chiave che permette di ottenere un messaggio di senso compiuto ma non è possibile sapere quale sia quello corretto.

Per esempio, dato il testo cifrato EQNVZ usando la chiave XMCKL si ottiene HELLO mentre usando la chiave EFZOZ si ottiene ALOAH e non è possibile identificare quale sia l'alternativa corretta.

È fondamentale per questo tipo di cifrari che una chiave venga usata **una sola volta**, altrimenti si perde il vantaggio della casualità; inoltre, le chiavi presentano l'inconveniente di dover avere la stessa lunghezza del plaintext. Per di più la chiave deve essere nota sia al mittente sia al destinatario → come trasmettere questa informazione?

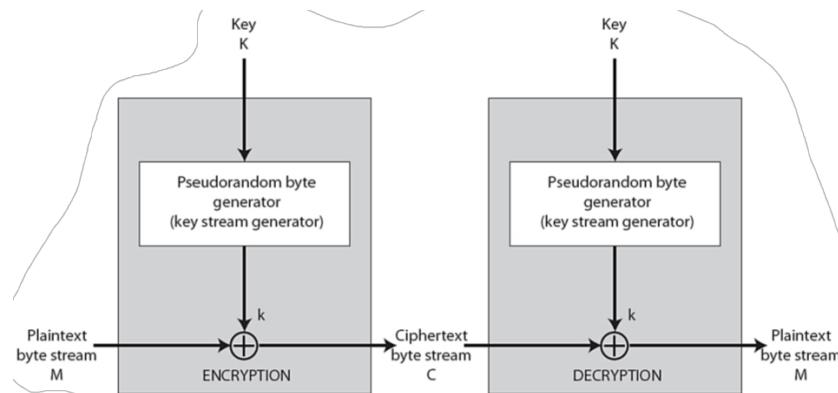
Cifrari a flusso

Elaborano il messaggio bit per bit costruendo uno pseudo random number generator per creare un **keystream** (flusso usato come chiave): data la stessa chiave verrà restituito sempre lo stesso messaggio (quindi non è totalmente casuale).

Viene eseguito lo XOR bit a bit tra il plaintext e il keystream:

$$C_i = M_i \oplus \text{keystream}_i$$

La casualità del keystream elimina le proprietà statistiche del messaggio.



Non bisogna però **mai usare la stessa chiave per messaggi diversi**, altrimenti si otterrà la seguente situazione vulnerabile mostrata con questo esempio:

$$C = M \oplus KS$$

$$C' = M' \oplus KS$$

$$C \oplus C' = (M \oplus KS) \oplus (M' \oplus KS) = M \oplus M'$$

Facendo lo XOR (simbolo \oplus) dei due messaggi cifrati con la stessa chiave è possibile ottenere lo XOR dei due testi in chiaro e quindi i testi in chiaro singoli.

Alcune considerazioni:

- il periodo del keystream (ogni quanto si ripete uno stesso numero) deve essere lungo, senza ripetizioni;
- la complessità del keystream dipende dalla sua stessa lunghezza.

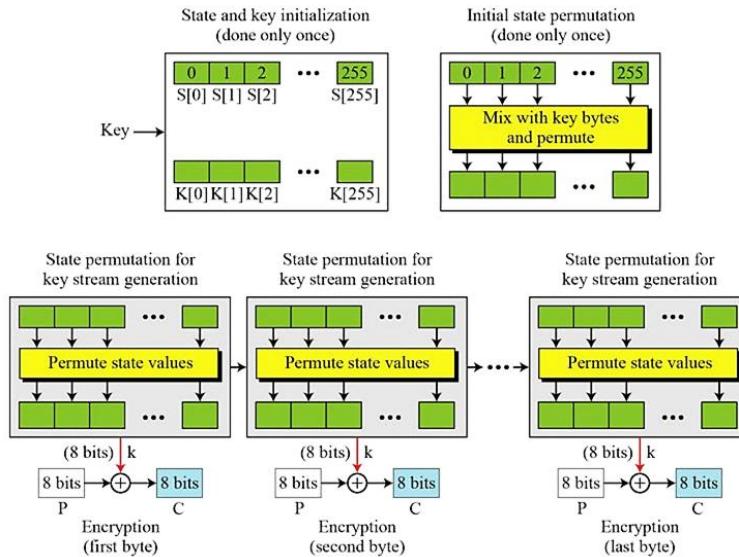
RC4

Si tratta di un cifrario a flusso orientato ai byte che esegue l'operazione di XOR tra un byte del testo in chiaro e un byte della chiave per produrre un byte di testo cifrato. La chiave segreta, da cui il key stream di chiavi mono byte viene generato, può avere dimensione variabile tra 1 e 256 byte.

RC4 è basato sul concetto di stato. In ogni momento, uno stato di 256 byte è attivo, dal quale viene selezionato casualmente un byte per essere usato come chiave di cifratura. La chiave può essere rappresentata come un array di byte: $S[0] S[1] S[2] \dots S[255]$

vedi: <https://www.youtube.com/watch?v=rilp6EQQJ0g>

Mai riusare una chiave in RC4, essendo un cifrario a flusso!



Il contenuto di ciascun elemento è quindi un byte (8 bit) che può essere interpretato come un intero compreso tra 0 e 255. La permutazione per creare il key stream viene ripetuta fin quando ci sono byte di testo in chiaro da cifrare. L'**inizializzazione** avviene in due step:

- 1) Lo stato s viene inizializzato ai valori 0, 1, ..., 255; viene anche creato un key array $K[0], K[1], \dots, K[255]$. Se la chiave segreta fornita (lunga max 256) ha esattamente 256 byte, i byte vengono copiati nel key array, altrimenti vengono ripetuti fino a riempire l'array.

```
for (i = 0 to 255) {
    S[i] ← i
    K[i] ← Key[i mod KeyLength]}
```

- 2) Nel secondo step, lo stato inizializzato passa attraverso una permutazione (swap di elementi) basata sul valore del byte in $K[i]$. Il byte-chiave viene utilizzato solo in questo step per definire quale elemento deve essere swappato.

```
j ← 0
for (1 = 0 to 255) {
    j ← (j + S[i] + K[i]) mod 256
    swap(S[i], S[j])}
```

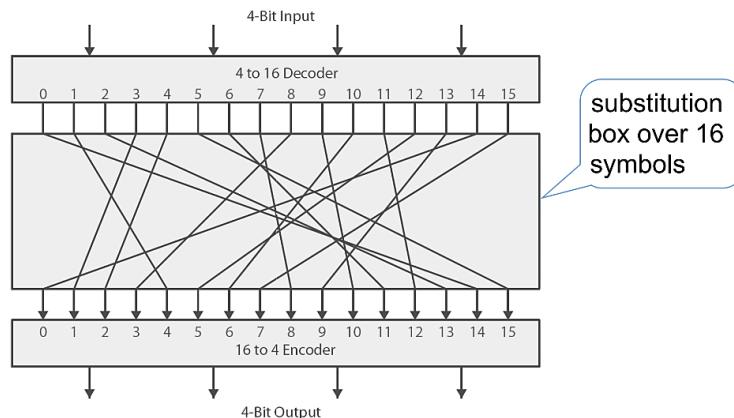
Generazione del key stream:

Le chiavi k del key stream vengono generate una per una. Per prima cosa lo stato viene permutato in base al valore degli elementi dello stato e al valore di due variabili individuali i e j . Successivamente, il valore dei due elementi dello stato in posizione i e j vengono usati per definire l'indice dell'elemento dello stato che servirà da chiave k . Il codice di seguito viene ripetuto per ogni byte del testo in chiaro per creare una nuova chiave nel key stream. Le variabili i e j sono inizializzate a 0 prima della prima iterazione, ma i valori vengono copiati da un'iterazione alla successiva.

```
i ← (i + 1) mod 256
j ← (j + S[i]) mod 256
swap(S[i], S[j])
k ← S[S[i] + S[j]] mod 256
```

Cifrari a blocco

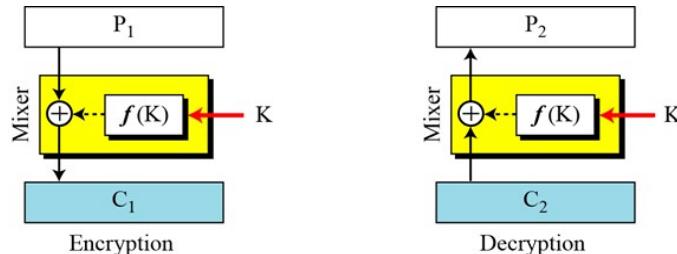
Lavorano con blocchi di dati di misura prefissa. È possibile pensare a un algoritmo di cifratura a blocchi come a una scatola di sostituzione: a un determinato input corrisponde un determinato output random (**funzione biettiva**).



Scegliere la chiave vuol dire scegliere la funzione biettiva che andrà a eseguire lo scambio. I blocchi rendono impossibili le analisi statistiche sulla frequenza delle lettere: più il blocco è grande più le frequenze vengono sparpagliate. Molti cifrari a blocchi simmetrici sono basati sulla struttura del **cifrario di Feistel** illustrata di seguito.

Feistel Cipher Structure

Il cifrario di Feistel si basa sul concetto di cifrario a **prodotto invertibile**; divide i blocchi in input in due metà. Il cifrario combina tutti gli elementi non-invertibili in un'unità e usa la stessa unità sia nell'algoritmo di cifratura sia in quello di decifrazione. Gli effetti di una componente non invertibile nell'algoritmo di cifratura possono essere cancellati nell'algoritmo di decifrazione usando l'operatore OR esclusivo (XOR).



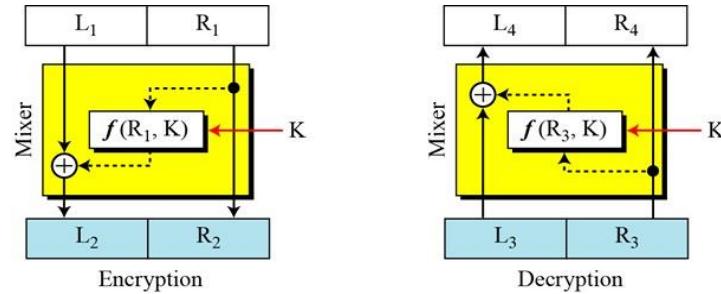
Nel processo di cifratura, una funzione non invertibile $f(K)$ accetta la chiave come input. L'output di questa componente viene messo in XOR con il testo in chiaro per ottenere il testo cifrato. La combinazione di XOR e della funzione f viene chiamato **mixer**. La chiave è la stessa per cifratura e decifrazione, quindi è possibile dimostrare che i due algoritmi eseguono operazioni inverse tra loro (se $C_2 = C_1$ allora $P_2 = P_1$).

- **Cifratura:** $C_1 = P_1 \oplus f(K)$
- **Decifrazione:** $P_2 = C_2 \oplus f(K) = C_1 \oplus f(K) = P_1 \oplus f(K) \oplus f(K) = P_1 \oplus 0 = P_1$

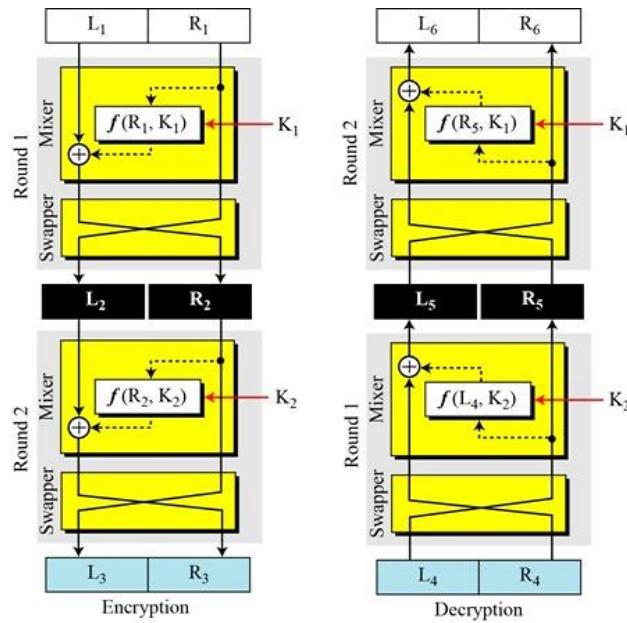
Feistel apporta un miglioramento a questa tecnica. Si vuole che l'input della funzione sia parte del testo in chiaro nella cifratura e parte del testo cifrato nella decifrazione: la chiave può essere usata come secondo input della funzione.

Per fare ciò, il testo in chiaro e il testo cifrato vengono divisi in due blocchi di uguale dimensione: una metà di destra e una di sinistra (R, L). Si supponga che il blocco di destra venga usato come input della funzione e il blocco di sinistra sia messo in XOR con l'output della funzione. C'è da ricordare un dettaglio importante: **l'input della funzione deve essere esattamente lo stesso sia in fase di cifratura sia in fase di decifrazione**.

Questo vuol dire che la metà di destra del testo in chiaro nella fase di cifratura e la metà di destra del testo cifrato nella decifrazione devono essere uguali.



Questo miglioramento ha però un difetto: la metà destra del testo in chiaro non cambia mai. Sono stati quindi necessari ulteriori miglioramenti. Per prima cosa l'aumento del numero di round e anche l'aggiunta di un nuovo elemento in ogni round: uno **swapper**. L'effetto dello swapper nel round di codifica viene cancellato dall'effetto dello swapper del round di decodifica. Ad ogni modo, questo elemento permette di scambiare la metà destra con quella sinistra in ogni round. Da notare la presenza di più chiavi di round (2 nel caso della figura d'esempio) che vanno usate in ordine inverso in cifratura e decifrazione.



more info: https://www.tutorialspoint.com/cryptography/feistel_block_cipher.htm

Data Encryption Standard (DES)

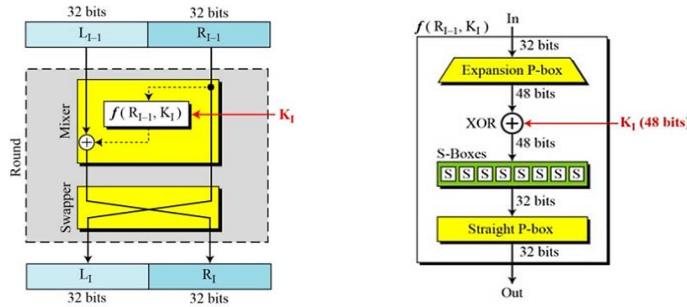
DES è un altro cifrario basato sul cifrario di Feistel che cifra blocchi di dati da 64 bit usando chiavi da 56 bit. La **fase di codifica** è composta da due permutazioni (P-boxes), chiamate permutazione iniziale e finale, e 16 round di Feistel. Ogni round usa una diversa chiave di 48 bit, detta chiave di round, generata dalla chiave di codifica di 56 bit.

Le due permutazioni prendono entrambe l'intero testo di 64 bit ed eseguono la permutazione di ogni bit in base alla regola descritta in una particolare tabella (fissa). Queste permutazioni non hanno alcun significato crittografico in DES in quanto sono entrambe prive di chiave e predeterminate; le ragioni per cui sono state incluse non sono chiare e non sono mai state rivelate dagli inventori di DES.

Come detto, DES usa 16 round, ognuno dei quali è un cifrario di Feistel. È possibile assumere che ogni round sia composto da due elementi di codifica: un **mixer** e uno **swapper**.

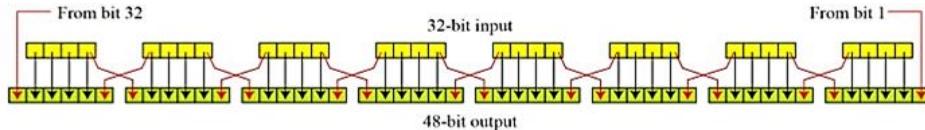
- Lo swapper cambia di posizione la metà di sinistra del testo con quella di destra;
- Il mixer esegue l'operazione di XOR; sono entrambi elementi invertibili.

Tutti gli elementi NON invertibili fanno parte, invece, della funzione DES.



La funzione DES è il cuore di DES e applica una chiave di 48 bit sulla metà più a destra di 32 bit per produrre un output di 32 bit. Si compone di quattro sezioni, esposte di seguito:

1) Espansione (P-box): poiché la metà di destra (R_{l-1}) è composta da 32 bit mentre la chiave è da 48 bit, è necessario espandere la prima per renderla anch'essa di 48 bit. R_{l-1} viene divisa in 8 sezioni di 4 bit ciascuna, successivamente ogni sezione viene espansa a 6 bit: il bit 4 di una sezione diviene il bit 1 della sezione successiva e il bit 1 diviene il bit 6 della sezione precedente



In questo modo si ottiene un output di 48 bit. Si noti che il numero di bit in output è 48 ma il range di valori va solo da 1 a 32 e alcuni bit in input hanno più output.

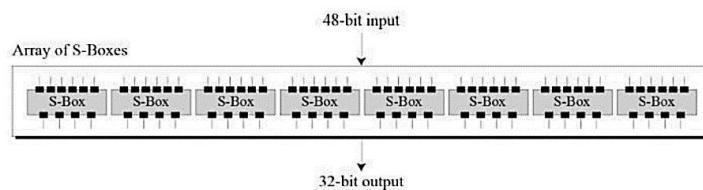
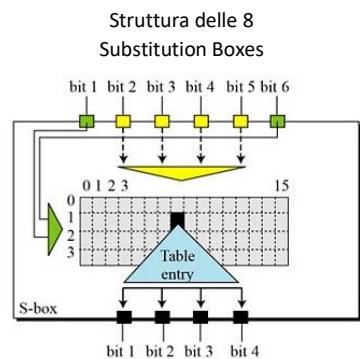
Expansion table:

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

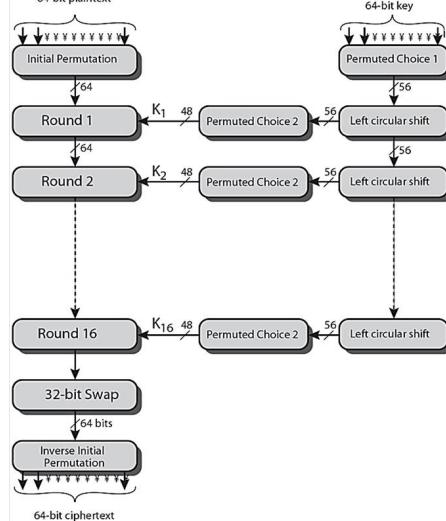
2) Withener (XOR): dopo l'espansione, DES applica l'operazione XOR tra la metà di destra espansa e la chiave di round, le quali sono entrambe di 48 bit. **La chiave di round viene usata solamente per questa operazione.**

3) **Substitution (S-boxes)**: questa è la sezione che crea la vera confusione.

- 8 S-boxes ognuna con 6 bit in input e 4 bit in output.
 - I 48 bit di dati ottenuti dopo lo XOR vengono divisi in 8 chunk da 6 bit ciascuno e ogni chunk viene inserito in una S-box.
 - I risultati di queste box sono chunk di 4 bit che vengono poi combinati in un testo di 32 bit.
 - La sostituzione in box segue una regola precisa basata su una tabella di 4 righe e 16 colonne: la combinazione del bit 1 e del bit 6 dell'input stabilisce una delle 4 righe, la combinazione dei bit da 2 a 5 stabilisce una delle 16 colonne.
 - Ogni S-box ha la sua propria tabella, quindi sono necessarie otto tabelle



4) Permutazione diretta: si tratta di una semplice permutazione con 32 bit in input e 32 bit in output basata su una tabella predefinita.



La decodifica si svolge sugli step della computazione dei dati.

Gli step della codifica vengono svolti usando le sottochiavi (SK16 ... SK1) in ordine inverso:

- IP “disfa” l’operazione FP dello step finale della codifica
 - Il primo round con sottochiave SK16 disfa il 16esimo round della codifica
 - ...
 - Il 16esimo round con sottochiave SK1 disfa il primo round di codifica
 - Infine, FP disfa l’iniziale codifica IP recuperando così i dati originali.

Si tratta di un sistema molto robusto, in quanto implementa l'**effetto valanga**, ovvero il cambiamento di un singolo bit in input o della chiave provoca la modifica di circa la metà dei dati in output. Le chiavi di 56 bit hanno $2^{56} = 7,2 \times 10^{16}$ possibili valori, per cui una ricerca brute-force risulta assai difficile.

more info: https://www.tutorialspoint.com/cryptography/data_encryption_standard.htm
https://www.youtube.com/watch?v=Y61qn_SQJ40&t=0s

Attacchi side channel

Attaccano l'attuale implementazione dell'algoritmo usando conoscenze sull'esecuzione per ottenere informazioni sulle sottochiavi. In particolare, utilizzano il fatto che i calcoli possono richiedere tempi variabili a seconda dei valori degli input: osservando i tempi di esecuzione è possibile indovinare (o restringere il campo di ricerca) la dimensione della chiave. Tutto questo risulta particolarmente problematico per dispositivi lenti come le smartcard. Le attuali implementazioni di DES sono abbastanza resistenti a questo genere di attacchi. Nonostante ciò è stato necessario sostituire DES a causa di attacchi teorici che possono romperlo e di attacchi di ricerca della chiave particolarmente efficaci.

AES è una nuova alternativa, prima della quale veniva usata la crittografia multipla con varie implementazioni di DES (triple-DES).

Triple-DES

Usa tre stage di DES per codifica e decodifica. Attualmente sono in uso due versioni di triple-DES.

Triple-DES a due chiavi

Nel triple-DES a due chiavi ci sono solo due chiavi: k_1 e k_2 . Il primo e il terzo stage usano la chiave k_1 , mentre il secondo usa k_2 . Per rendere il triple-DES compatibile con il DES singolo, lo stage di mezzo usa decryption (reverse cipher) nel campo della cifratura ed encryption (cipher) nel campo della decodifica.

In questo modo, un messaggio cifrato con DES singolo con chiave k può essere decodificato con triple-DES se $k_1 = k_2 = k$. Sebbene triple-DES a due chiavi sia vulnerabile ad attacchi known-plaintext è più forte del double-DES.

Triple-DES a tre chiavi

Usa tre chiavi per evitare i possibili attacchi cui è esposta l'altra versione.

$$C = E_{k3}(D_{k2}(E_{k1}(P)))$$

Le tre chiavi sono generate allo scopo, quindi non c'è bisogno di salvarle da qualche parte.

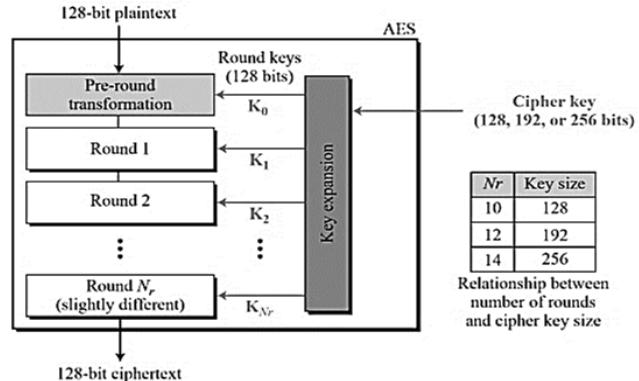
Advanced Encryption Standard (AES – ex Rijndael)

AES è un cifrario non-Feistel a blocchi con chiave simmetrica che codifica e decodifica blocchi di dati di 128 bit. La chiave può essere di 128, 192 o 256 bit a seconda del numero di round. Usa 10, 12 o 14 round.

La figura mostra l'algoritmo di codifica di AES, quello di decodifica è simile con la differenza che le chiavi di round vengono applicate in ordine inverso.

AES usa cinque diverse unità di misura per i dati:

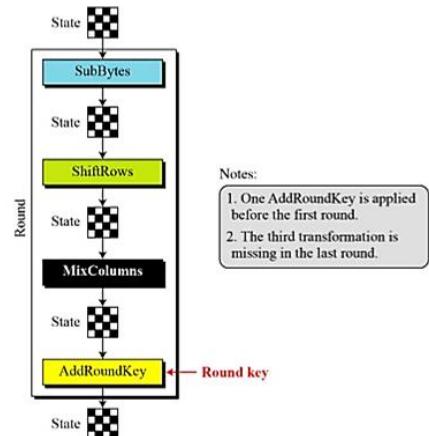
1. **bit**
2. **byte**
3. **words (parole)** = 32 bit
4. **blocchi** = 128 bit
5. **stati** = 128 blocchi



Struttura dei Round

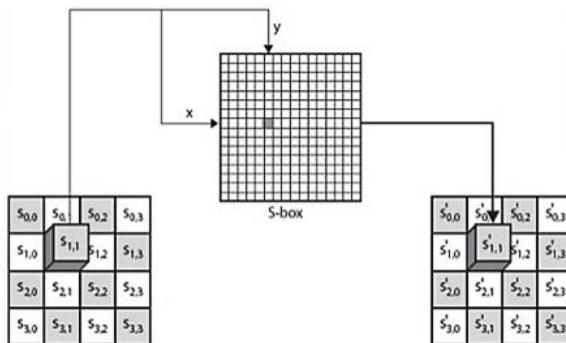
Ogni round, fatta eccezione per l'ultimo, fa uso di quattro trasformazioni invertibili. L'ultimo round ne usa solo tre. Ogni trasformazione prende uno stato e ne crea un altro che verrà usato nella trasformazione successiva oppure nel round successivo.

Vi sono più trasformazioni, analizziamole una ad una:



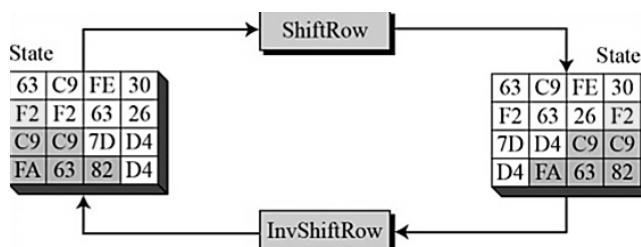
Sostituzione:

AES come DES usa la sostituzione; il meccanismo però è differente: innanzitutto la sostituzione viene fatta su ogni byte e, inoltre, viene usata una sola tabella per trasformare ogni byte; questo vuol dire che se due byte sono uguali, il risultato della trasformazione sarà lo stesso. La trasformazione **SubBytes** viene usata nella fase di codifica: un byte viene interpretato come composto da due digit esadecimale. Il digit di sinistra definisce la riga e il digit di destra definisce la colonna della tabella di sostituzione. Lo stato viene trattato come una matrice di byte 4×4 . Ogni byte viene trasformato indipendentemente; quindi l'**operazione SubBytes prevede 16 trasformazioni byte-a-byte**. La trasformazione **InvSubBytes** è l'operazione inversa eseguita in fase di decodifica usando una tabella apposita.



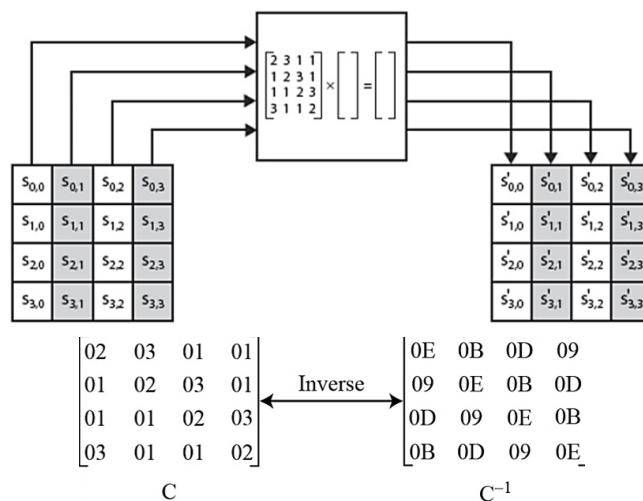
Permutazione:

Le operazioni di shifting eseguono permutazioni tra byte (a differenza di DES che permuta bit-a-bit) senza cambiare l'ordine dei bit all'interno dei byte. In fase di codifica, l'operazione dedicata è chiamata **ShiftRows** e lo shift avviene verso sinistra. Il numero di posizioni dello shift dipende dalla riga della matrice-stato (0, 1, 2, 3). L'operazione inversa che esegue lo shift a destra in fase di decodifica prende il nome di **InvShiftRows**.



Mixing:

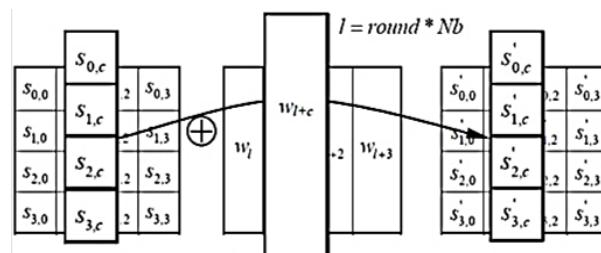
La sostituzione effettuata da SubBytes cambia il valore del byte solo in base a un valore fisso e a una entry della tabella: il processo non include i byte vicini (trasformazione intrabyte). La permutazione effettuata da ShiftRows scambia i byte senza toccare i bit dei singoli bytes (trasformazione byte-exchange). C'è quindi bisogno di una trasformazione **interbyte** che cambi i bit all'interno dei byte basata sui bit dei byte vicini (neighboring bytes). La trasformazione mixing cambia il contenuto di ciascun byte prendendo 4 byte alla volta e combinandoli tra loro per creare 4 nuovi byte. Per garantire che tutti i nuovi byte siano differenti, per prima cosa il processo moltiplica ciascun byte per una diversa costante e solo poi esegue il mixing. L'operazione prende il nome di **MixColumns**, l'inversa invece, **InvMixColumns**.



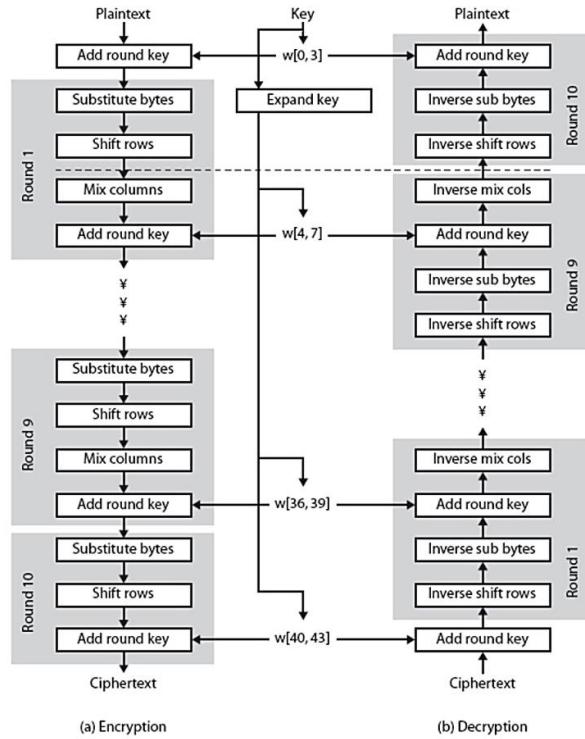
Key adding

La trasformazione più importante è quella che include la chiave di cifratura. La chiave di cifratura è l'unico segreto tra chi comunica. AES usa un processo chiamato **key expansion** che crea $N_r + 1$ chiavi di round usando come base la chiave di cifratura (N_r = numero di round). Ogni chiave è grande 128 bit e viene considerata come formata da 4 parole di 32 bit e ogni parola è trattata a sua volta come una colonna di una matrice.

L'operazione **AddRoundKey** processa una colonna alla volta e aggiunge (esegue lo XOR) una parola della chiave di round ad ogni colonna della matrice-stato = matrix addition (addizione tra matrici); l'operazione inversa esegue invece una sottrazione. AddRoundKey è l'inversa di sé stessa.

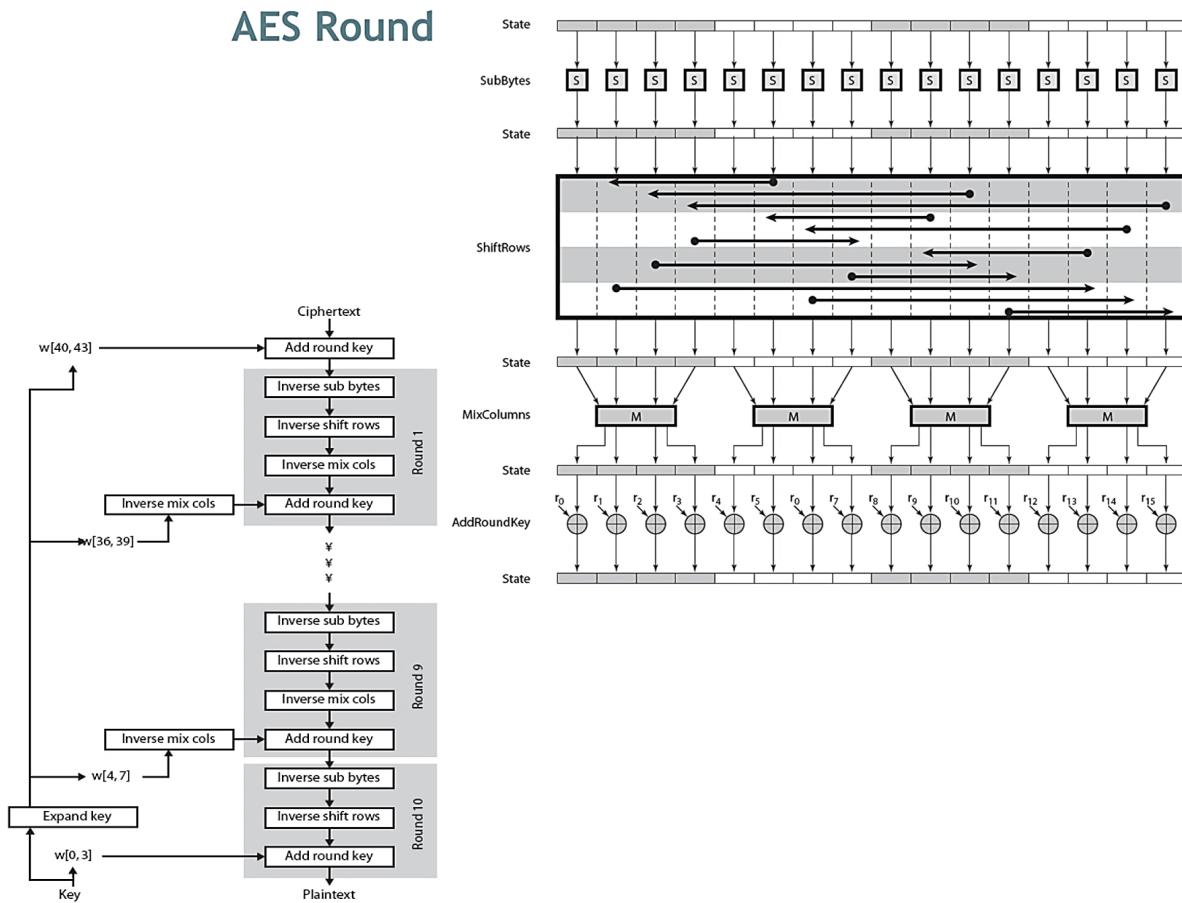


Rijndael



La **decodifica** in AES non è identica alla codifica in quanto i vari step vanno eseguiti in ordine inverso.

AES Round

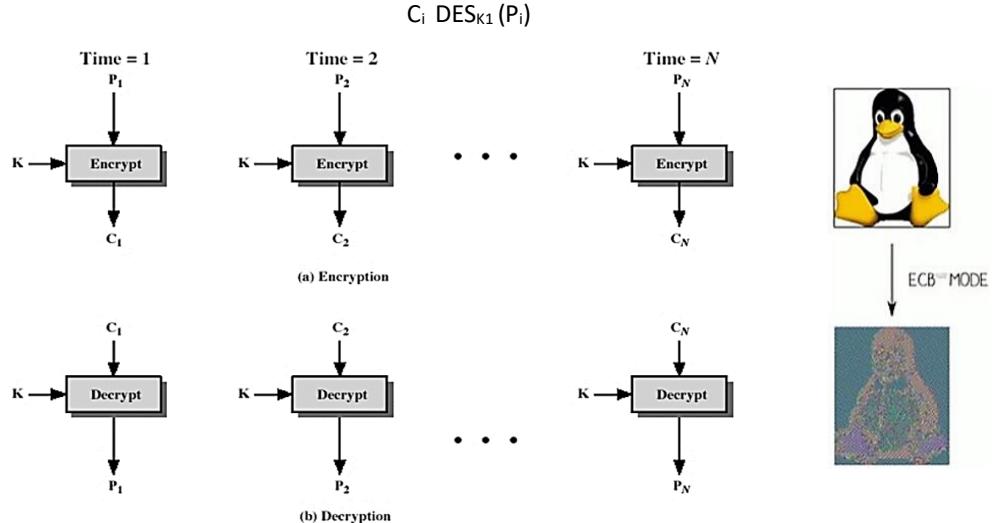


Utilizzo dei cifrari a blocchi

Esistono diversi modi di utilizzare un cifrario a blocchi:

Electronic CodeBook (ECB)

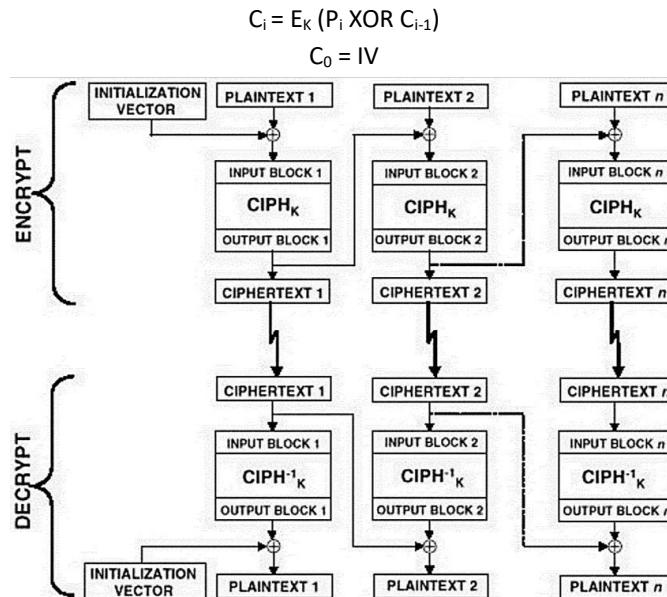
I messaggi sono divisi in blocchi indipendenti. Ogni blocco è un valore sostituito, cifrato indipendentemente dagli altri blocchi:



Una delle principali limitazioni di questo metodo consiste nel fatto che se vi sono ripetizioni nel testo in chiaro, queste si presenteranno anche nel testo cifrato, dunque **la struttura del file viene preservata nella cifratura**. Ciò rende intuibile il messaggio originale.

Cipher Block Chaining (CBC)

Il messaggio è sempre suddiviso in blocchi. Questa volta, però, vengono concatenati tra loro nell'operazione di codifica: ogni blocco precedentemente cifrato viene concatenato con il blocco corrente contenente il plaintext. Per iniziare il processo utilizza un vettore di inizializzazione (initialization vector IV) non segreto:



Decodifica:

Si prende ciphertext1, lo si decodifica e lo si mette in XOR con l'IV per ottenere il testo in chiaro del primo blocco, si procede poi alla stessa maniera per ogni blocco i usando però, al posto dell'IV, il ciphertext(i – 1).

Un blocco di testo cifrato dipende da **TUTTI** i blocchi che lo precedono, quindi **ogni modifica in un blocco influisce su tutti i blocchi successivi**.

Questo metodo necessita di un initialization vector che deve essere noto sia al sender sia al recipient. Se questo viene spedito in chiaro, un attaccante potrebbe modificare dei bit nel primo blocco e modificare l'IV per compensare; quindi l'IV deve o essere un valore fisso oppure deve essere spedito criptato tramite ECB prima del resto del messaggio.

- In ECB qualora ci sia un bit di errore durante la trasmissione (per esempio in P_1), durante la cifratura, il 50% del blocco successivo (P_2) potrebbe essere sbagliato: un bit sbagliato annulla un intero blocco.
- In CBC la situazione precedente porterà all'annullamento di tutto il blocco P_2 più un singolo bit del blocco P_3 .

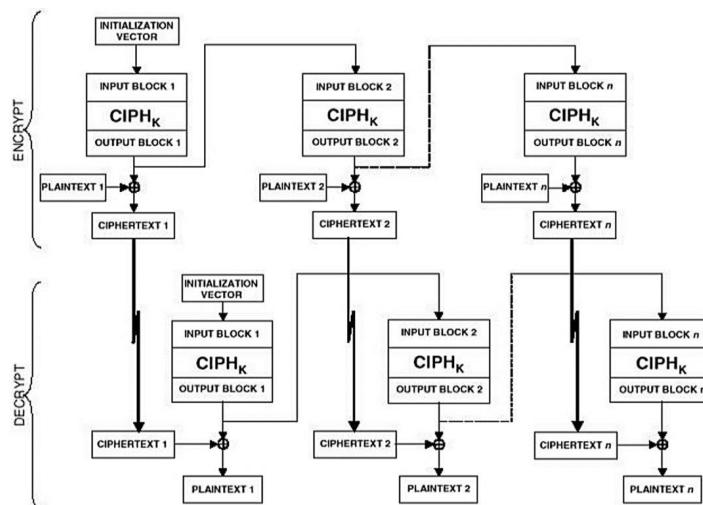
Output FeedBack (OFB)

Un cifrario a blocchi può essere usato anche a flusso dove si usa uno pseudo random number generator per generare un flusso da mettere in XOR con i dati in chiaro: il messaggio viene trattato come un singolo flusso di bit. L'output della cifratura viene aggiunto al messaggio.

$$C_i = P_i \text{ XOR } O_i$$

$$O_i = E_K(O_{i-1})$$

$$O_0 = IV$$



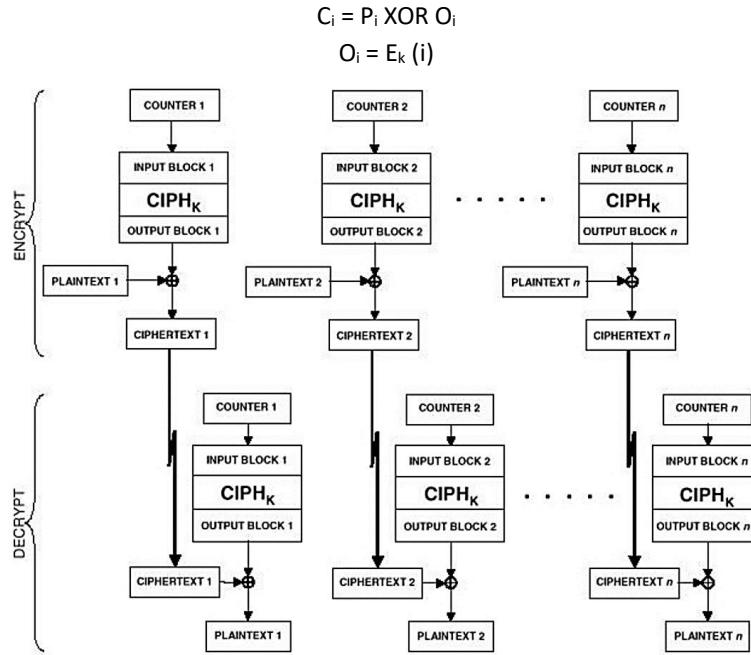
L'algoritmo di cifratura e di decodifica sono lo **stesso algoritmo** = non bisogna implementare l'algoritmo di decodifica.

Non bisogna MAI usare lo stesso flusso più volte = mai usare stesso IV e stessa chiave insieme

Gli errori sui bit non si propagano = un singolo errore in input darà un singolo bit errato in output.

Counter (CTR)

È simile a OFB ma ciò che viene cifrato è il valore di un contatore. Questo metodo ha bisogno di una chiave e di un contatore diversi per ogni blocco di testo in chiaro:



Ogni blocco viene cifrato indipendentemente dagli altri (come ECB ma più robusto). CTR può pre-processare in anticipo. È possibile accedere a qualsiasi parte del file e cifrarla in base alla posizione.

Distribuzione della chiave simmetrica

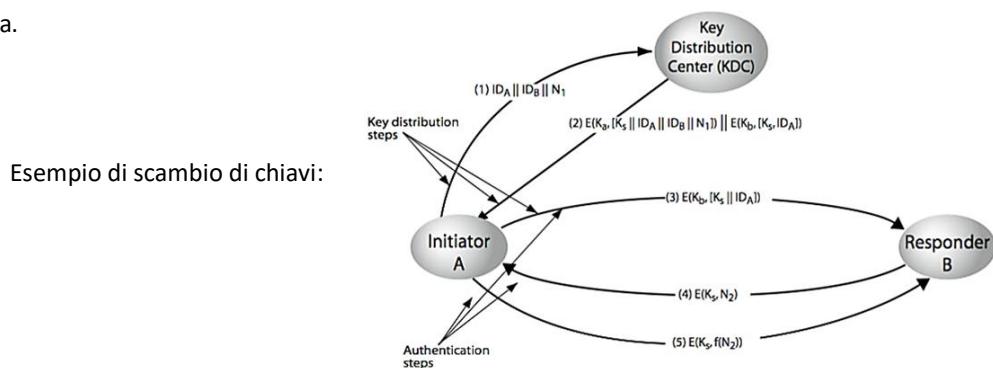
Esistono quattro modi possibili per la distribuzione di una chiave simmetrica (chiave privata):

1. A genera la chiave K e la consegna fisicamente a B → non sempre applicabile.
2. Una terza parte (Key Distribution Center) può selezionare e consegnare la chiave ad A e B.
3. Se A e B hanno comunicato precedentemente possono usare la vecchia chiave per criptare una nuova chiave.
4. Se A e B hanno un canale sicuro con una terza parte C, C può trasmettere la chiave tra A e B.

Gerarchia di chiavi

Premessa: meno dati si cifrano con la stessa chiave meglio è.

- Chiave **master**: viene cambiata raramente ed è usata per spedire pochi dati, solitamente le chiavi di sessione. Può essere condivisa dall'utente e il Key Distribution Center o pre-condivisa tra tutti i partecipanti alla comunicazione. **Viene creata e distribuita al di fuori della sessione**.
- Chiave **di sessione**: chiave temporanea generata a random e usata per una singola sessione; poi viene scartata.



1. A → KDC : $ID_A \parallel ID_B \parallel N_1$
2. KDC → A : $E(K_A, [K_S \parallel ID_A \parallel ID_B \parallel N_1]) \parallel E(K_B, [K_S \parallel ID_A])$
3. A → B : $E(K_B, [K_S \parallel ID_A])$
4. B → A : $E(K_S, N_2)$
5. A → B : $E(K_S, f(N_2))$

Tre aspetti di sicurezza da garantire:

1. confidenzialità della chiave
2. autenticazione A per B e autenticazione B per A
3. freschezza delle chiavi

I primi tre step riguardano la distribuzione della chiave; N_1 è una **nonce** per evitare attacchi reply (evitare che una vecchia chiave venga riutilizzata); gli step 4 e 5 servono per l'autenticazione di A e B e per evitare attacchi reply.

In assenza della nonce:

1. A → KDC: $ID_A \parallel ID_B$
2. KDC → A: $E(K_A, [K_S \parallel ID_A \parallel ID_B]) \parallel E(K_B, [K_S \parallel ID_A])$

In questa situazione un attaccante, intercettando la comunicazione tra A e il KDC potrebbe re-inviare lo stesso messaggio del punto 2 ad A inducendolo a usare la chiave K_S già precedentemente usata (cioè una vecchia K_S).

L'uso della nonce garantisce ad A di ricevere una risposta nuova (e quindi una chiave nuova) ad ogni richiesta e non una risposta "riciclata" (Attacco-REPLY) inserita da un attaccante.

Servizio di integrità e autenticazione di messaggi

È importante ricordare che un attacco all'integrità è un attacco attivo; per contrastarlo vengono aggiunte delle informazioni aggiuntive al messaggio.

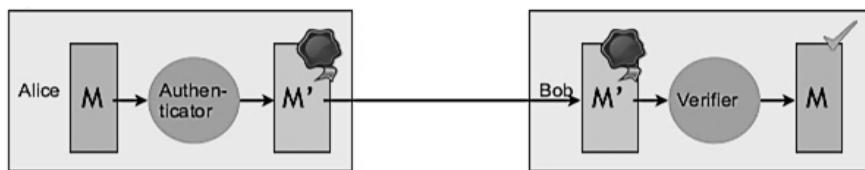
L'integrità e l'autenticazione di un messaggio si occupano di:

- proteggere l'integrità del messaggio
- validare l'identità del mittente

Ovvero: il messaggio inviato da A a B deve essere integro e si deve essere certi che sia stato effettivamente inviato da A: l'integrità implica quindi autenticità.

Meccanismi di autenticazione

Ogni meccanismo di autenticazione è composto da un **autenticatore** e da un **verificatore**, i quali lavorano separatamente.



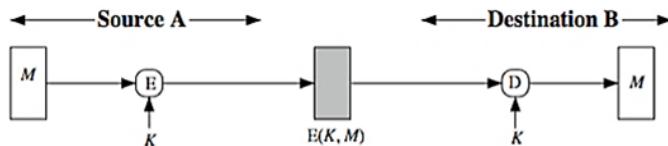
È possibile usare l'algoritmo di cifratura come **autenticatore** e quindi la funzione di decifrazione come **verificatore**. Questo garantisce l'integrità perché se qualcuno ha modificato il messaggio, esso perde di significato in quanto la chiave di sessione K è nota solo ad A e B.

In questo modo non viene però implementato il non ripudio.

Tuttavia, un algoritmo di cifratura può essere usato in modo sbagliato (implementazione sbagliata): è bene implementare i servizi in maniera minimalistica.

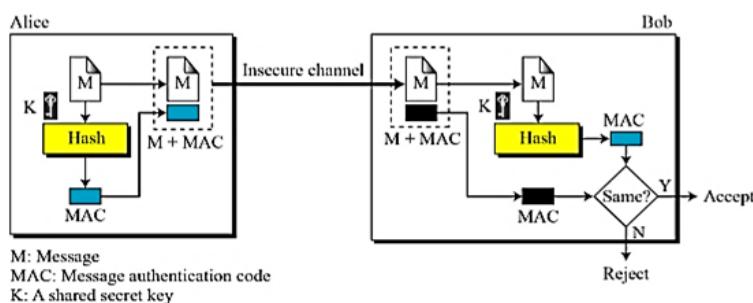
Il non ripudio non viene implementato perché il messaggio cifrato con K non dà nessuna informazione sull'effettivo mittente (A), dice solo che è stato cifrato con K . A e B in questo caso sono **simmetrici**, hanno conoscenze equivalenti = non c'è modo di distinguere A da B all'interno del messaggio.

Autenticazione di messaggi tramite crittografia simmetrica



Ricapitolando: in caso di uso di crittografia simmetrica il destinatario sa che il mittente deve aver creato il messaggio poiché solo il mittente e il destinatario conoscono la chiave utilizzata. Il destinatario sa anche che il contenuto non può essere stato modificato se questo ha struttura, ridondanza o checksum adeguati a rilevare eventuali modifiche. Non viene evitato il ripudio da parte di A o la falsificazione da parte di B.

Codice di autenticazione del messaggio:



Uso della funzione di hash: aggiunge una piccola informazione aggiuntiva al messaggio (**MAC – Message Authentication Code**). La funzione viene creata sulla base del messaggio e sulla chiave di sessione.

$$f_{\text{hash}}(K + M) \rightarrow \text{MAC}, \text{ ciò che viene inviato è } M + \text{MAC}$$

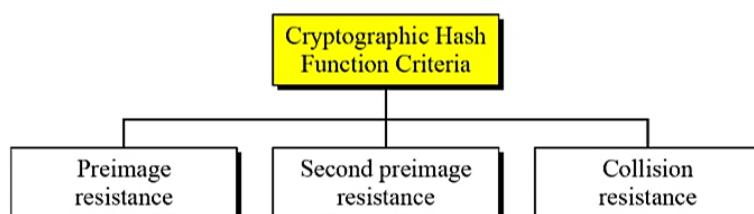
Chi riceve ricalcola il MAC usando il messaggio ricevuto e la chiave; se questo coincide con il MAC ricevuto deduce che il messaggio è integro, altrimenti che il messaggio è stato modificato. Il messaggio gira in chiaro: non viene garantita la confidenzialità e nemmeno il non ripudio.

Funzioni di Hash

Le funzioni di hash **non sono funzioni invertibili** (non sono biiettive): l'output della funzione H sarà un messaggio di dimensioni fisse, non dipendenti dalla misura dell'input. Per esempio, una funzione da 128 bit avrà 2^{128} output possibili, questo vuol dire che gli output sono finiti ma gli input sono infiniti.

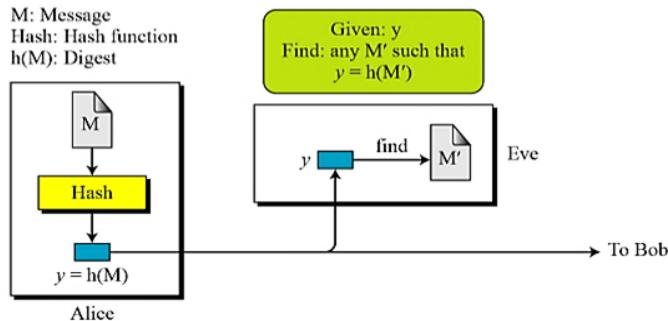
Una buona funzione di hash deve soddisfare i seguenti requisiti:

- deve poter essere applicabile a un messaggio M di qualsiasi dimensione;
- produrre un output h di dimensione fissata;
- deve essere facile da calcolare $h = H(M)$ per ogni messaggio M;
- deve essere resistente in tre modi:



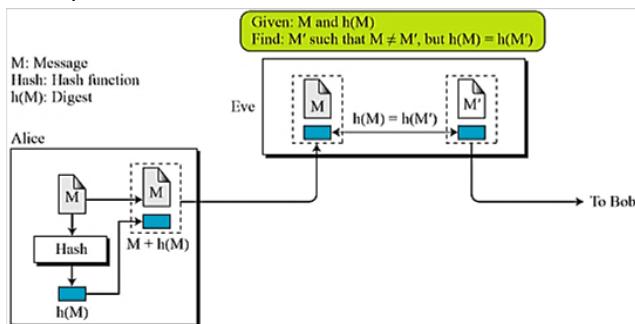
Resistenza alla preimmagine:

Dato h deve essere infattibile trovare x tale che $H(x) = h$. Nel caso fosse possibile, un attaccante potrebbe fare finta di essere una delle parti della comunicazione autentiche.



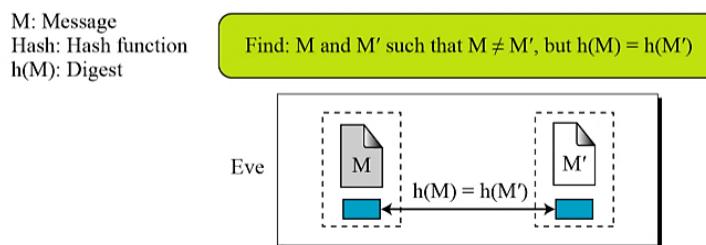
Resistenza alla seconda immagine:

Dato x deve essere impossibile trovare y tale che $H(y) = H(x)$. Nel caso fosse possibile, un attaccante potrebbe intercettare x e sostituirlo con y .



Resistenza alla collisione:

Deve essere impossibile trovare x, y tali che $H(y) = H(x)$. Nel caso fosse possibile, un attaccante potrebbe trovare tali x, y e presentare x ad A, ottenere il MAC e usare questo MAC con y per comunicare con B.



Secure Hash Algorithm (SHA)

Il Secure Hash Algorithm SHA è uno standard sviluppato dalla National Institute on Standards and Technology (NIST). È uno standard fortemente basato su MD5. Ne esistono più versioni: SHA-1, SHA-224, SHA-256, SHA-384 e SHA-512.

Funzioni di hash e sicurezza MAC

- Sfruttamento di attacchi brute-force
 - funzioni di hash robuste resistenti alle collisioni hanno costo $2^{m/2}$
 - hash da 128 sembrano vulnerabili, meglio quelle da 160
- MAC con coppie messaggio-MAC note
 - può essere attaccato il keyspace o il MAC
 - per avere sicurezza sono necessari almeno MAC a 128 bit
- Attacchi criptoanalitici che sfruttano la struttura
 - come i cifrari a blocchi vogliono che gli attacchi brute-force siano la migliore alternativa

HMAC

Modo con cui si usano le funzioni di hash per generare i MAC:

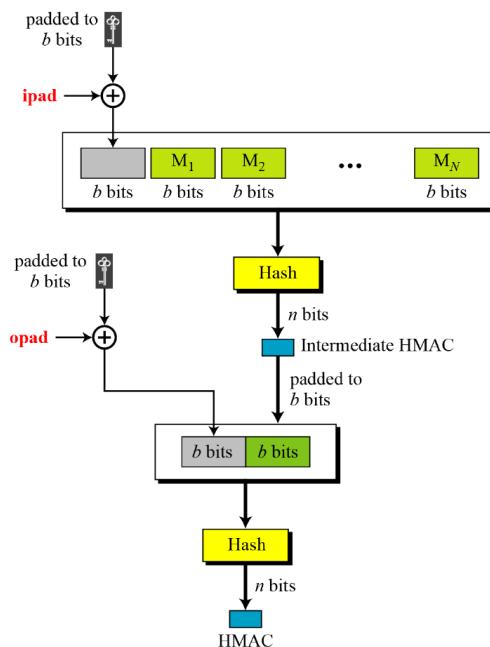
$$\text{HMAC}_K = \text{Hash}[(K^+ \text{ XOR } \text{opad}) \parallel \text{Hash}[(K^+ \text{ XOR } \text{ipad}) \parallel M]]$$

dove K^+ è la chiave con aggiunta di padding e $\text{opad} = 0x5C = '\backslash'$, $\text{ipad} = 0x36 = '6'$ sono specifiche costanti di padding; l'overhead è di soli 3 calcoli di hash in più rispetto a quelli che il messaggio da solo avrebbe bisogno. Può essere usata una qualsiasi funzione di hash.

La sicurezza di HMAC è strettamente dipendente da quella della hash usata dall'algoritmo.

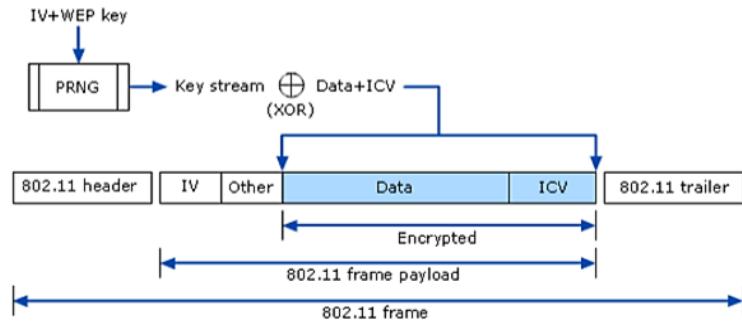
Attaccare HMAC richiede:

- un attacco brute-force sulla chiave usata;
- o un birthday attack.

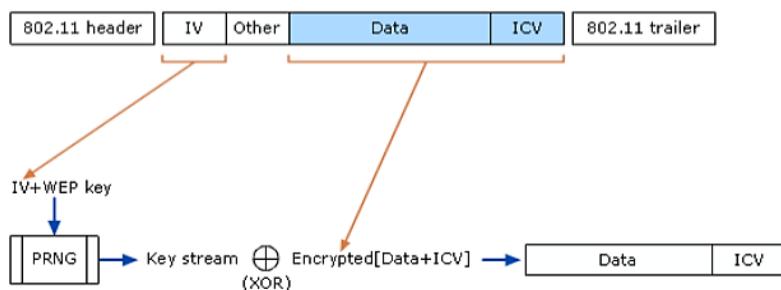


WEP

WEP è uno schema di codifica **a chiave simmetrica** tra host e access point definito dallo standard 802.11. Lo scopo è quello di offrire la stessa confidenzialità delle connessioni cablate e di limitare l'accesso ai soli host autorizzati. Solo i pacchetti autenticati possono accedere alla rete. La chiave è condivisa tra host e AP e può essere (e spesso lo è) la stessa per tutti gli host; lo schema non specifica come questa chiave venga scambiata e nemmeno per quanto tempo sia valida. Per quanto riguarda la **codifica** il payload 802.11 e il checksum (ICV) sono codificati usando una variante di RC4 con inizialization vector (IV) di 24 bit (3 byte). L'IV viene spedito insieme al testo cifrato.



In fase di decodifica il ricevitore inizializza il PRNG con l'IV appena ricevuto per ottenere il keystream.



La codifica e la decodifica possono essere svolte con l'hardware WiFi (quindi l'IV viene inviato prima del testo cifrato). 802.11 non specifica come l'IV debba essere generato e cambiato (ma è consigliato cambiarlo spesso).

WEP garantisce quindi: confidenzialità, integrità e controllo di accesso al mezzo.

Cifratura Asimmetrica

Esistono sistemi di sicurezza che non possono essere implementati tramite la crittografia simmetrica, per esempio il security goal di non ripudio.

Per questo sono stati proposti **sistemi di cifratura a chiave pubblica** che rendono possibile la firma digitale e forniscono un modo per implementare la distribuzione delle chiavi senza passare per il KDC (Key Distribution Channel).

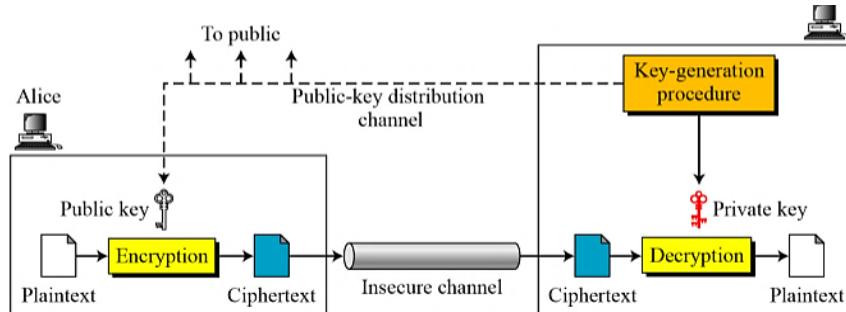
La chiave, in questi sistemi, è composta da due parti: una chiave pubblica e una chiave privata.

- La **chiave pubblica (PU_A)** può essere nota a chiunque e può essere usata per cifrare messaggi e per verificare firme.
- La **chiave privata (PR_A)** è nota solo al creatore della chiave (o meglio della coppia di chiavi) e viene usata per decifrare messaggi e per creare una firma (firmare)

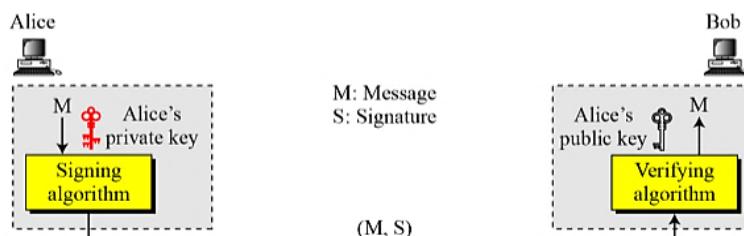
L'asimmetria è data dal fatto che chi decifra un messaggio o verifica una firma non può decifrare messaggi o creare firme.

Le applicazioni dei sistemi a chiave pubblica sono varie e vengono classificate in tre categorie:

1. **Cifratura** (e decifrazione): se A vuole mandare un messaggio segreto a B, la cifratura del messaggio prende come input la PU_B mentre la decifrazione avviene tramite la PR_B = il mittente cifra il messaggio con la chiave pubblica del destinatario. **Garanzia di segretezza**.



2. **Firma digitale**: il mittente A usa la **propria** chiave privata per firmare il messaggio. Il ricevente B userà la chiave pubblica del mittente A per verificare la firma = **tutti possono verificarne la firma**. Questo garantisce l'autenticità del messaggio e il non ripudio.



3. **Scambio di chiavi**.

Caratteristiche dei sistemi a chiave pubblica

Gli algoritmi a chiave pubblica fanno affidamento su due chiavi (o meglio, a due parti della stessa chiave) per cui deve essere computazionalmente impossibile risalire alla chiave privata partendo da quella pubblica e, d'altro canto, deve essere relativamente facile cifrare e decifrare il messaggio quando le chiavi sono note.

Gli schemi a chiave pubblica utilizzano **funzioni ONE-WAY** ovvero funzioni semplici da calcolare ma per cui è molto difficile calcolarne l'inversa. Data una funzione one-way f :

- f è facile da calcolare (P)
- f^{-1} è molto difficile da calcolare (NP completa)

Queste funzioni vengono chiamate **trap door** se, dato un parametro extra, calcolare f^{-1} diventa facile.

Euler Totient Function $\phi(n)$

In matematica, la funzione ϕ di Eulero o semplicemente funzione di Eulero o toziente, è una funzione definita, per ogni intero positivo n , come il numero degli interi compresi tra 1 e n che sono coprimi con n . Ad esempio $\phi(8) = 4$, poiché i numeri coprimi di 8 sono quattro: 1, 3, 5, 7.

In generale:

- per p (p primo): $\phi(p) = p - 1$
- per $p * q$ (p, q primi): $\phi(pq) = (p - 1) * (q - 1)$

Teorema di Eulero

Presi qualsiasi a, n coprimi, moltiplicando a per sé stesso tante volte quanto vale $\phi(n)$ si otterrà sempre 1 (modulo n).

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

RSA

È uno degli schemi a chiave pubblica più noti e utilizzati.

Ogni utente genera una coppia di chiavi pubblica/privata scegliendo due numeri primi grandi p, q a caso, poi calcola il modulo $n = p * q$. Successivamente sceglie a caso una chiave di codifica e dove:

$$1 < e < \phi(n), \text{ mcd}(e, \phi(n)) = 1$$

Per trovare la chiave di decodifica d risolve la seguente equazione:

$$e * d \equiv 1 \pmod{n}$$

Ogni utente, infine, pubblica la propria chiave pubblica $PU = \{e, n\}$ e tiene segreta la propria chiave privata $PU = \{d, n\}$.

Uso di RSA

Per cifrare un messaggio M , il mittente deve ottenere la chiave pubblica del destinatario $PU = \{e, n\}$ e calcolare:

$$C = M^e \pmod{n}, \text{ dove } 0 \leq M \leq n$$

Per decifrare il messaggio C , il destinatario deve usare la sua chiave privata $PU = \{d, n\}$ per calcolare:

$$M = C^d \pmod{n}$$

Si noti che il messaggio M deve essere più piccolo del modulo n .

Dimostrazione:

- $C^d = (M^e)^d = M^{e * d}$
- $e * d = 1 \text{ mod } \phi(n) = K * \phi(n) + 1$
- $M^{e * d} = M^{K * \phi(n) + 1} = (M^{\phi(n)})^K * M \rightarrow \text{per il teorema di Eulero} \rightarrow 1^K * M = M$

Esponenti molto grandi possono essere calcolati efficientemente con l'algoritmo Square and Multiply (algoritmo del contadino russo).

RSA Example - Key Setup

- Select primes: $p=17$ & $q=11$
- Compute $n = p*q = 17*11=187$
- Compute $\phi(n) = (p-1)(q-1)=16*10=160$
- Select $e: \gcd(e, 160)=1$; choose $e=7$
- Determine $d: d*e=1 \text{ mod } 160$ and $d < 160$ Value is $d=23$ since $23*7=161 = 1*160+1$
- Publish public key $PU=\{7, 187\}$
- Keep secret private key $PR=\{23, 187\}$

RSA Example - En/Decryption

- sample RSA encryption/decryption is:
- given message $M = 88$ (nb. $88 < 187$)
- encryption:
 $C = 88^7 \text{ mod } 187 = 11$
- decryption:
 $M = 11^{23} \text{ mod } 187 = 88$

Possibili attacchi su RSA

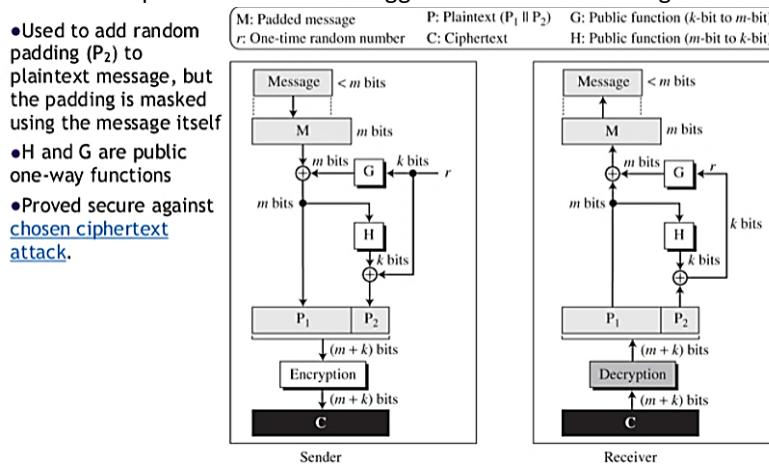
Gli attacchi brute-force sono sempre possibili: è possibile tentare di fattorizzare n tramite brute-force: al massimo \sqrt{n} ($p * p < p * q = n \rightarrow p < \sqrt{n}$). Chiavi da 2048 bit sembrano abbastanza sicure comunque.

Sono possibili attacchi chosen-ciphertext poiché cifrando M_1 ed M_2 separatamente si ha che:

$$M_1^e * M_2^e = (M_1 * M_2)^e$$

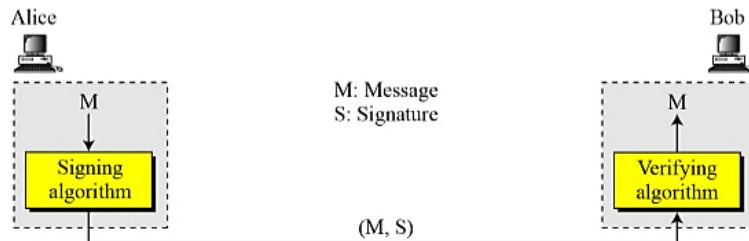
Per contrastare questa tipologia di attacchi è possibile usare OAEP.

Sono possibili anche attacchi di tipo plaintext su messaggi molto brevi; per contrastarli si "riempie" il messaggio con dei bit random per ottenere un messaggio sufficientemente lungo.



Firma digitale

La firma digitale fornisce la possibilità di **verificare l'autore, la data e l'ora della firma**. È possibile così garantire l'autenticità del messaggio il quale può essere controllato da una terza parte per risolvere dispute. La firma deve essere univoca per il mittente per prevenire falsificazione e rifiuto.

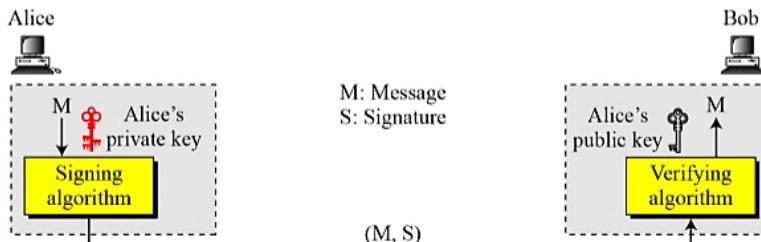


Deve essere anche relativamente facile da produrre e deve essere altrettanto facilmente riconoscibile e verificabile. Deve inoltre essere computazionalmente impossibile ricreare una firma digitale esistente a partire da un nuovo messaggio.

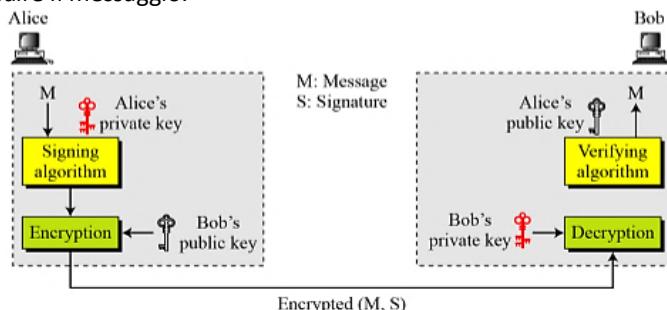
Firma digitale diretta

Coinvolge solo il mittente e il destinatario, presupponendo che il destinatario abbia la chiave pubblica del mittente; la firma digitale viene fatta dal mittente firmando l'intero messaggio o l'hash con la propria chiave privata. Può anche cifrare usando la chiave pubblica del destinatario; è importante però che per prima cosa firmi, solo dopo può cifrare il messaggi e la firma. La sicurezza dipende dalla chiave privata del mittente.

Firma con chiave asimmetrica



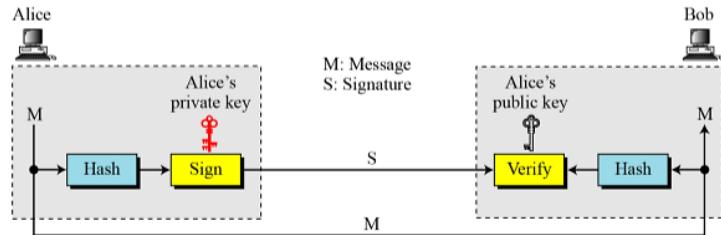
Una **firma digitale necessita di un sistema a chiave pubblica** dove il firmatario firma con la propria chiave privata e il verificatore verifica con la chiave pubblica del firmatario. Quando un messaggio viene firmato chiunque può verificarlo, perché tutti hanno accesso alla chiave pubblica del mittente A. A deve prestare attenzione a non usare la sua chiave pubblica per firmare il messaggio altrimenti chiunque potrebbe ricreare la sua firma. La sola firma digitale non garantisce la confidenzialità, per ottenerla è necessario aggiungere un passaggio prima di spedire il messaggio.



Da notare la differenza tra l'uso delle chiavi pubbliche e private per la firma digitale e per il sistema per ottenere la confidenzialità. In quest'ultimo vengono usate la chiave pubblica e la chiave privata del destinatario: il mittente usa la chiave pubblica del destinatario per cifrare il messaggio e il destinatario usa la propria chiave privata per decifrarlo. Nella firma digitale, invece, vengono usate le chiavi del mittente: il mittente usa la sua chiave privata per firmare e il destinatario usa la chiave pubblica del mittente per verificare.

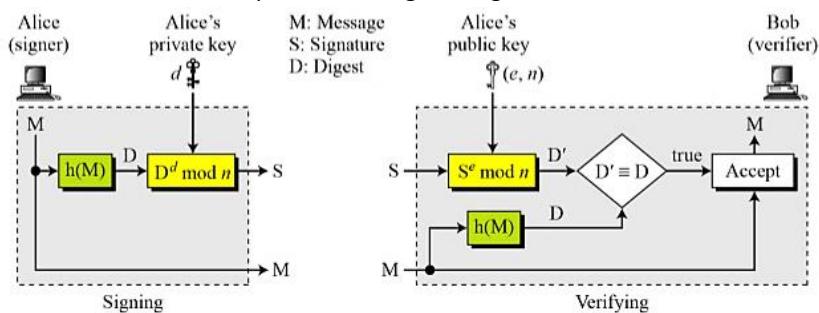
Firma del digest

La crittografia a chiave asimmetrica è piuttosto inefficiente quando affronta messaggi lunghi. La soluzione, per quel che riguarda la firma digitale, è quella di firmare il digest del messaggio, il quale è molto più corto del messaggio completo. Un digest ben scelto per un messaggio ha una relazione univoca uno-a-uno con il messaggio stesso; in questo modo il mittente può firmare il digest che può essere facilmente verificato dal ricevente.



Un digest viene creato dal messaggio di Alice tramite una funzione di hash pubblica e viene poi firmato con la chiave privata di quest'ultima. Successivamente A invia la firma e il messaggio a B. Nel lato di Bob viene creato un digest a partire dal messaggio ricevuto usando la stessa funzione pubblica di hash usata da Alice, il confronto viene fatto sul digest calcolato e sulla firma ricevuta verificata con la chiave pubblica di Alice.

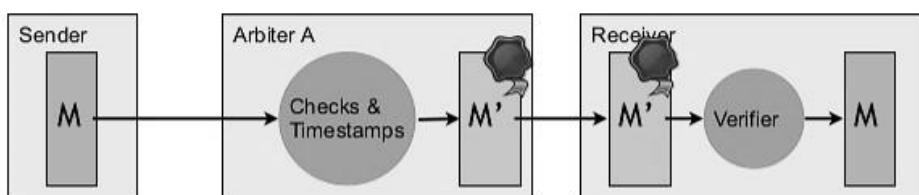
Esempio di firma digitale digest in RSA:



Firma digitale arbitrata

Prevede il coinvolgimento di un arbitro.

- Ogni messaggio firmato inviato dal mittente X al destinatario Y, viene prima recapitato a un arbitro A.
- Un soggetto del messaggio più la firma per verificare l'origine e il contenuto.
- Richiede un livello adeguato di fiducia nell'arbitro.
- Può essere implementato sia con algoritmi a chiave pubblica, sia con algoritmi a chiave privata.
- L'arbitro può leggere o meno il messaggio.
- L'arbitro non conserva una copia del messaggio ma può verificarlo in ogni momento.



Arbitrated Digital Signatures with symmetric encryption

- The arbiter sees the message
 1. $X \rightarrow A : M \parallel E(K_{XA}, ID_X \parallel H(M))$
 2. $A \rightarrow Y: E(K_{AY}, ID_X \parallel M \parallel E(K_{XA}, ID_X \parallel H(M))) \parallel T$
- The arbiter does not see the message
 1. $X \rightarrow A : ID_X \parallel E(K_{XY}, M) \parallel E(K_{XA}, ID_X \parallel H(E(K_{XY}, M)))$
 2. $A \rightarrow Y: E(K_{AY}, ID_X \parallel E(K_{XY}, M)) \parallel E(K_{XA}, ID_X \parallel H(E(K_{XY}, M))) \parallel T$
- In both cases, arbiter must be trusted by both sides
- T is used by Y against replay attacks

Arbitrated Digital Signatures with asymmetric encryption

- The arbiter does not see the message
 1. $X \rightarrow A : ID_X \parallel E(PR_X, ID_X \parallel E(PU_Y, E(PR_X, M)))$
 2. $A \rightarrow Y: E(PR_A, ID_X \parallel E(PU_Y, E(PR_X, M))) \parallel T$
- Does not need any previously shared information
- Timestamp depends on security of PR_A only
- Message M is secret to everybody but Y
- Slow - three public-key encryptions at each step

[Distribuzione delle chiavi pubbliche nella crittografia asimmetrica](#)

Bisogna avere chiara la paternità della chiave pubblica.

1) Annuncio Pubblico

La chiave pubblica viene esposta pubblicamente in chiaro (possibile anche in broadcast). Si tratta di un sistema non robusto. C'è il rischio di contraffazione: chiunque può creare una chiave sostenendo di essere qualcun altro e diffonderla in broadcast.

2) Directory pubblica

Uso di uno o più server che contengono le chiavi pubbliche.

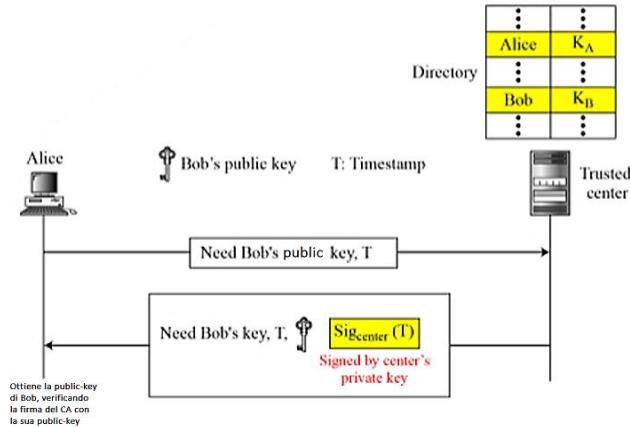
La directory è considerata attendibile se ha le seguenti proprietà:

- contiene la entry {name, public-key};
- i partecipanti si registrano in modo sicuro con la directory;
- i partecipanti possono sostituire la chiave in ogni momento;
- la directory viene periodicamente pubblicata;
- è possibile accedere elettronicamente alla directory.

Questo metodo di distribuzione è ancora vulnerabile a manomissione e a contraffazione: se si acquisisce la chiave privata della directory è possibile distribuire chiavi pubbliche arbitrarie.

3) Public-key Authority (Controlled Trusted Centre)

Migliora la sicurezza rafforzando il controllo sulla distribuzione delle chiavi da directory. Gli utenti interagiscono con la directory per ottenere in sicurezza una qualsiasi chiave pubblica desiderata. Prevede l'uso di timestamp.

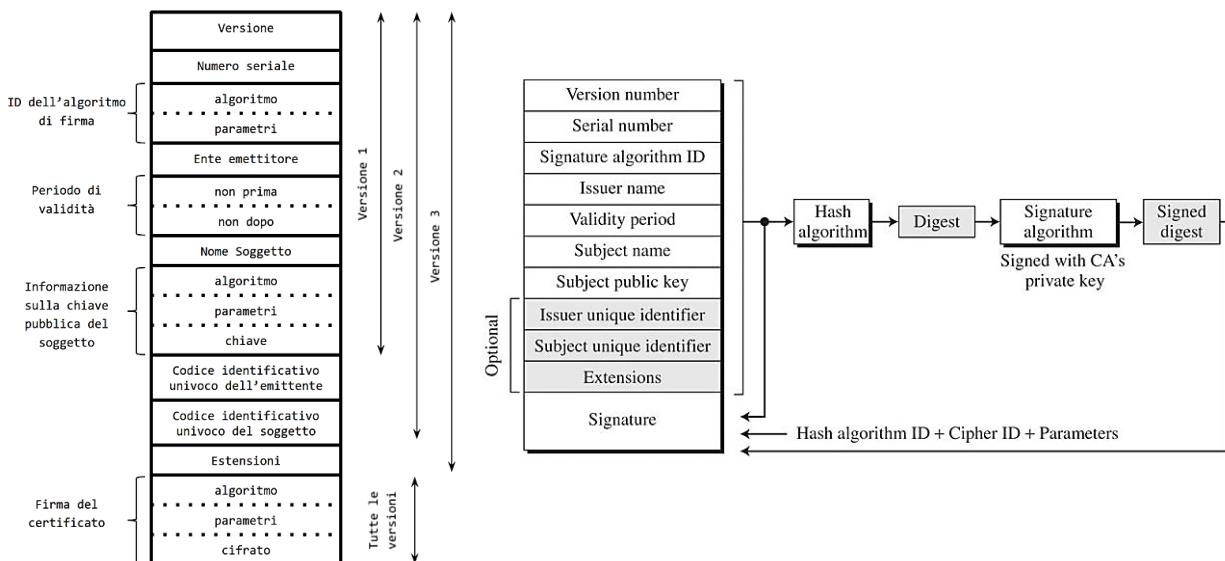


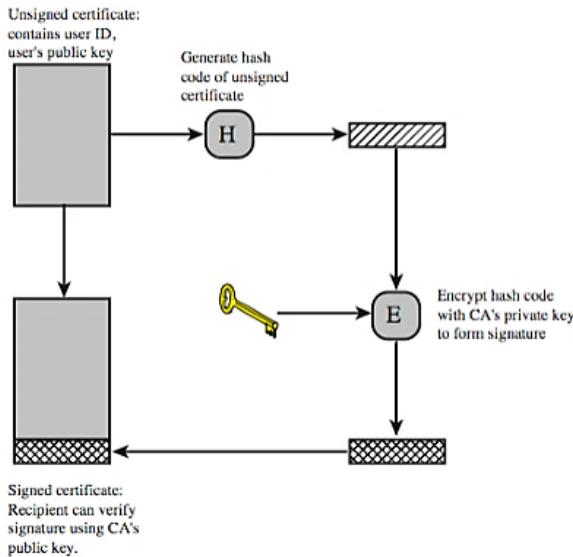
4) Certificato di chiave pubblica

Un certificato è una coppia $\langle ID, PU \rangle$ tale che sia facile accorgersi se e quando viene alterata: lega un identità a una chiave pubblica. Fa uso di firma digitale: il certificato viene rilasciato da un'entità terza (Certification Authority, CA) la quale **firma** il certificato con la sua chiave privata, e quindi verificabile da chi ha chiesto il certificato con la chiave pubblica del CA.

X.509 = CERTIFICATO: si tratta di un file contenente:

- versione
- numero seriale del certificato
- nome del soggetto (ID_A)
- informazioni sulla chiave del soggetto
- timestamp
- nome dell'issuer (chi rilascia il certificato)
- firma cifrata con la PR dell'issuer





Come si ottiene un certificato?

L'utente crea una K_{PR} e una K_{PU} . Poi, a partire dalla K_{PU} , crea un file di richiesta del certificato in cui si identifica. La CA crea il certificato, esegue la funzione di hash e la cifra con la propria chiave privata, ciò che viene ottenuto viene aggiunto in calce al certificato e inviato in risposta. L'utente che ha fatto richiesta decifra la risposta tramite la chiave pubblica della CA e ottiene così il certificato.

Revoca di un certificato

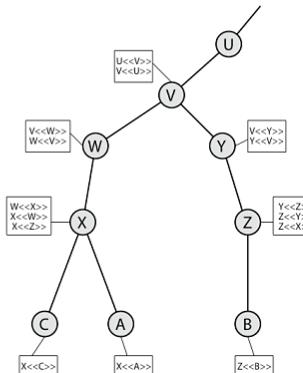
Venne generato un certificato di revoca, ovvero una lista di tutti i certificati revocati con annessa data di revoca (un certificato scaduto viene rimosso da questa lista).

Gerarchia di CA

La scalabilità si ottiene attraverso una gerarchia di autenticazione e CA. Le radici degli alberi delle gerarchie sono AUTO-CERTIFICATI. Se due utenti condividono una Certification Authority comune si presume che sappiano le rispettive chiavi pubbliche; altrimenti si rende necessaria una gerarchia (per scalabilità): ogni CA ha certificati per clients (forward – inoltro) e per partents (backward – arretrato). Ogni client “si fida” dei certificati dei parents. Ciò fornisce la possibilità di verificare qualsiasi certificato emesso da una CA da parte di utenti di tutte le altre CA nella gerarchia.

CA Hierarchy Use

- e.g. user C, knowing only the public key of V, can obtain a verified copy of B's public key:
- B sends to C a chain of certificates, $U<<V>>$, $V<<Y>>$, $Y<<Z>>$, $Z<>$.
- C validates $V<<Y>>$ using the public key of V. ($U<<V>>$ is not needed)
- C extracts the PK of Y from $V<<Y>>$.
- C validates $Y<<Z>>$ using the PK of Y.
- C extracts the PK of Z from $Y<<Z>>$.
- C validates $Z<>$ using the PK of Z.
- C extracts the PK of B from $Z<>$

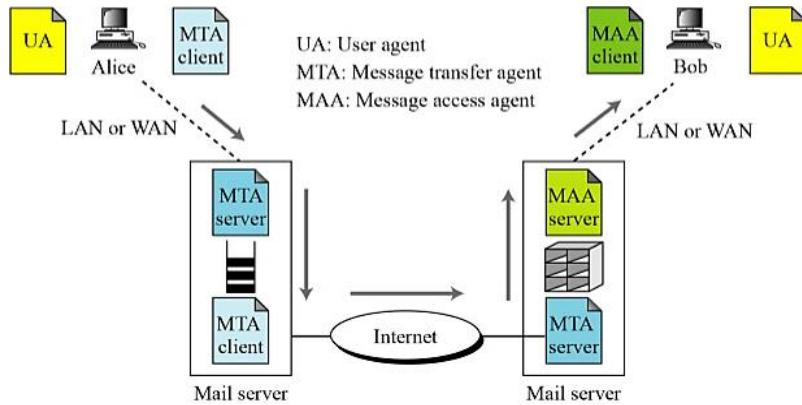


Infrastruttura di chiave pubblica (PKI)

Public Key Infrastructure – PKI: “l’insieme di hardware, software, persone, politiche e procedure necessarie per creare, gestire, immagazzinare, distribuire e revocare certificati pubblici basati su crittografia asimmetrica”.

E-mail Security

E-mail è uno dei servizi di rete più diffusi e considerati nell'ambito della sicurezza.



Nelle normali mail, il contenuto del messaggio non è sicuro:

- può essere visionato e modificato mentre in transito;
- è facile fingersi un qualsiasi utente.

Sono possibili dei miglioramenti su:

- **Confidenzialità** = protezione dalla divulgazione;
- **Autenticazione** = del mittente del messaggio;
- **Integrità del messaggio** = protezione dalle modifiche;
- **Non-ripudio dell'origine** = protezione dal rifiuto da parte del mittente;
- **Non-ripudio della destinazione** = protezione dal rifiuto da parte del destinatario.

In e-mail security il mittente deve includere l'identificatore dell'algoritmo che usa nel messaggio (possono essere usati alcuni algoritmi a chiave pubblica) e la codifica/decodifica può essere eseguita tramite algoritmi a chiave simmetrica ma la chiave segreta per decodificare il messaggio viene cifrata con la chiave pubblica del destinatario e inviata con il messaggio.

Pretty Good Privacy (PGP)

È lo standard de facto per le e-mail sicure.

Autenticazione:

- Il mittente crea un messaggio, usa SHA-1 per generare una hash del messaggio di 160 bit firmato con RSA usando la chiave privata del mittente e la attacca al messaggio.
- Il destinatario usa RSA con la chiave pubblica del mittente per decifrare e recuperare il codice hash e, infine, verifica il messaggio usando la hash ottenuta confrontandola con la hash decifrata.

Confidenzialità:

- Il mittente genera il messaggio e 128 bit di numeri casuali come chiave di sessione per esso; cifra il messaggio usando Triple-DES in modo CBC con chiave di sessione. La chiave di sessione viene cifrata usando RSA con la chiave pubblica del destinatario e attaccata al messaggio.
- Il destinatario usa RSA con la sua chiave privata per decifrare e recuperare la chiave di sessione. La chiave di sessione è poi usata per decifrare il messaggio.

Autenticazione + Confidenzialità:

1. È possibile implementare entrambi i servizi sullo stesso messaggio.

Tecniche di gestione delle chiavi: non usa X.509

Ogni utente possiede un portachiavi (keyring) privato e uno pubblico, database locali che contengono rispettivamente le proprie chiavi private e le chiavi pubbliche proprie e degli altri.
Il private keyring viene usato per le firme.

Scambio di chiavi pubbliche:

Web of trust: associa a ogni chiave pubblica un livello di affidabilità. Ogni volta che si riceve una nuova chiave pubblica si chiedono informazioni agli utenti di cui si possiedono le PU con livello di affidabilità più alto (ovvero: "ci si può fidare del proprietario di questa chiave pubblica?"): si forma così una rete di affidabilità in cui, solitamente, le chiavi di livello più alto sono state ottenute di persona (chiavetta USB, floppy...).

[S/MIME](#)

È lo standard per legge supportato da tutti i client di posta elettronica moderni.
Si appoggia a X.509 = usa certificati.

S/MIME Functions

- enveloped data
 - encrypted content and associated keys
- signed data
 - encoded message + signed digest
- clear-signed data
 - cleartext message + encoded signed digest
- signed & enveloped data
 - nesting of signed & encrypted entities

[Posta Elettronica Certificata \(PEC\)](#)

Migliora S/MIME aggiungendo non-ripudio del destinatario e garantisce che la mail sia stata consegnata (ma non letta). Quando il messaggio viene consegnato alla casella del destinatario, il mittente riceve una notifica di avvenuta consegna.

Autenticazione delle entità e distribuzione delle chiavi

L'autenticazione delle entità è una tecnica pensata per permettere a una delle parti di una comunicazione di dimostrare la propria identità all'altra parte. Un'entità può essere una persona, un processo, un client oppure un server.

- L'entità la cui identità deve essere verificata detta **claimant**.
- La parte che tenta di effettuare una verifica sull'identità del claimant è detta **verifier**.

È importante distinguere tra:

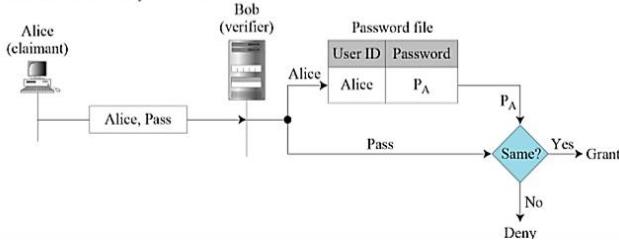
- **Autenticità**: riferita ai **messaggi**, può non avvenire in tempo reale. Il controllo va eseguito per ogni nuovo messaggio.
- **Autenticazione**: riferita alle **parti**, avviene in tempo reale. Autentica il claimant per l'intera sessione.

Esistono diverse tecniche, divise in tre categorie:

1. Tecniche basate su qualcosa che l'entità **sa** (password, pin, etc.);
2. Tecniche basate su qualcosa che l'entità **ha** (chiavi, smartcard, etc.);
3. Tecniche basate su qualcosa che l'entità **è** (voce, retina, etc.).

Password

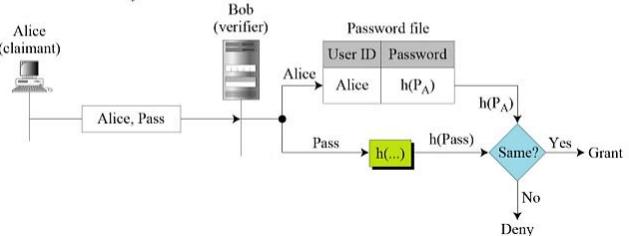
P_A : Alice's stored password
Pass: Password sent by claimant



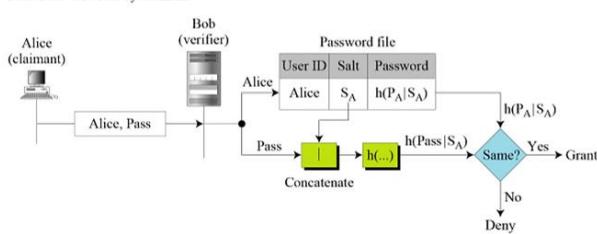
Le password non vanno salvate in chiaro.

Ciò che deve essere salvato sul database sono le hash delle password. Che tuttavia sono vulnerabili ad attacchi brute-force al dizionario.

P_A : Alice's stored password
Pass: Password sent by claimant



P_A : Alice's password
 S_A : Alice's salt
Pass: Password sent by claimant



Sarebbe meglio quindi usare password "salate", dove il sale in questo caso è una nonce esterna alla password usata per introdurre rumore.

Protocolli di autenticazione

Sono usati per convincere le parti di una comunicazione sulle rispettive identità e anche per scambiare le chiavi di sessione: può trattarsi di autenticazione one-way o mutua.

Fondamentalmente si presentano due tipi di problemi:

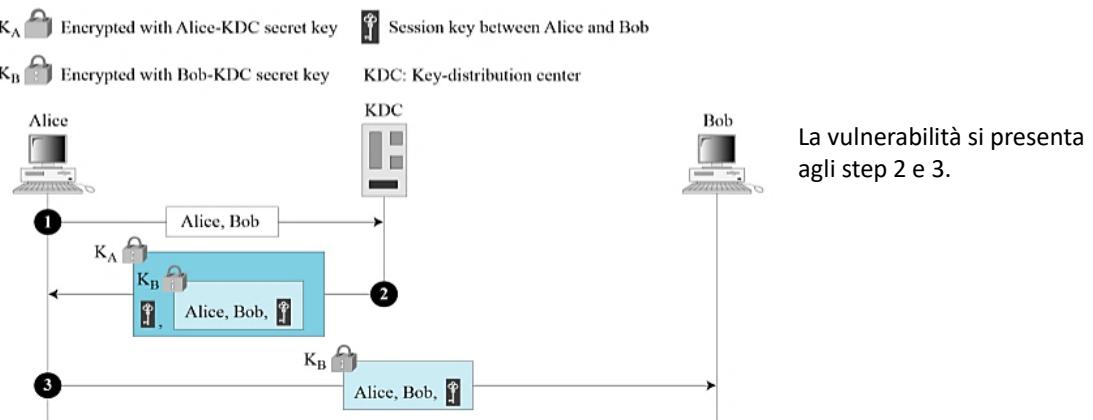
1. **Confidenzialità** = proteggere le chiavi di sessione.
2. **Timeliness** (tempestività) = prevenire attacchi reply.

Le chiavi di sessione servono per autenticarsi al KDC, non per la comunicazione vera e propria; di solito il KDC crea una chiave di sessione tra due utenti. **Una chiave di sessione simmetrica va usata una volta sola.**

Attacchi reply

(Un tipo di attacco attivo = cambia il corretto comportamento del protocollo.)

Un messaggio valido può venire copiato e inviato di nuovo in un secondo momento.



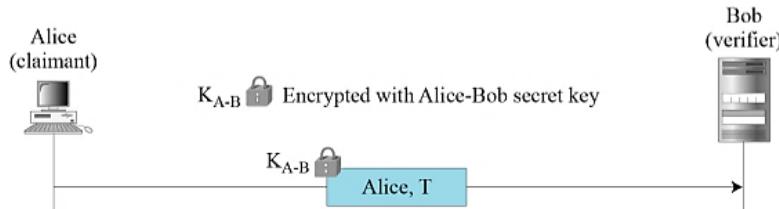
Per affrontare questo tipo di attacchi è possibile usare un **un contatore (numero di sequenza)**, il quale viene incrementato ogni volta che viene creata una nuova sessione. In questo modo, qualora si ricevesse un messaggio con il contatore ha un valore minore di quello noto, tale messaggio verrebbe identificato come "riciclato".

Problema: tanti contatori richiedono tanta memoria.

Un altro tipo di contromisura prevede l'uso di **timestamp**. Un timestamp T è valido se:

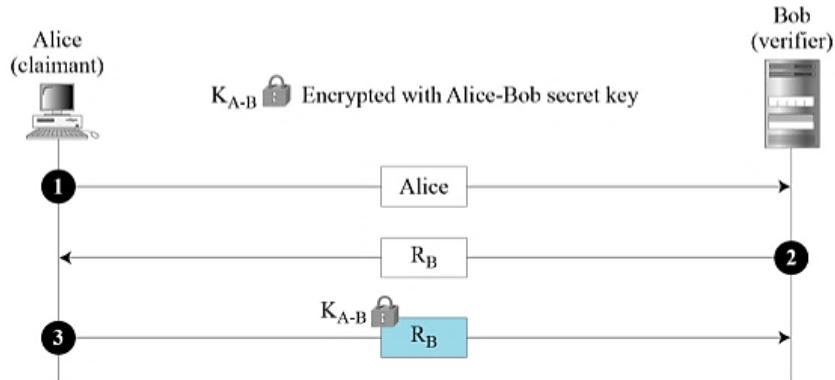
$$|\text{Clock} - T| \leq \Delta t_{\text{local}} + \Delta t_{\text{network}}$$

Questo metodo richiede sincronizzazione con l'orologio macchina (non facile) e un'attenta scelta degli intervalli. Non è adatto ai protocolli orientati alla connessione.

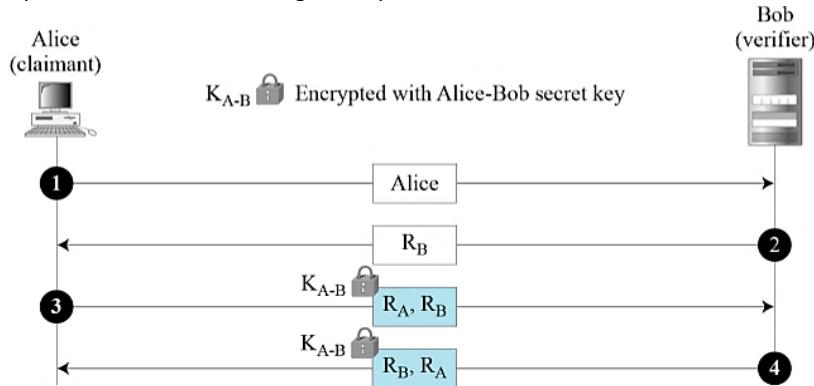


Una terza contromisura è quella **challenge-response**. Questo metodo richiede più traffico di rete rispetto alle altre soluzioni e, fa uso di **nonce** (durante l'handshake).

“B per verificare che A sia realmente A, le invia una nonce e le chiede di cifrarla con la chiave K_{A-B} nota solo ad A”. In questo modo B è sicuro dell’identità di A ma NON viceversa.

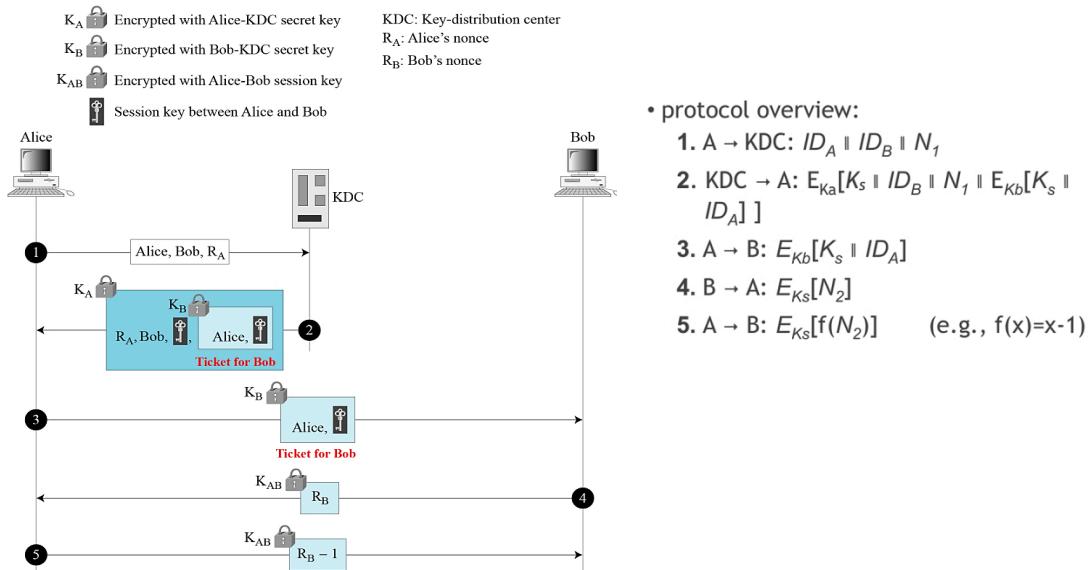


Per il viceversa, A può chiedere una challenge in risposta a B.



Protocollo Needham-Schroeder

Si tratta di un protocollo di distribuzione third-party per una sessione tra due entità A e B mediata da un Key Distribution Centre (KDC) attendibile.



È usato per distribuire in sicurezza una nuova chiave di sessione per le comunicazioni tra A e B. È però vulnerabile ad attacchi reply se una vecchia chiave di sessione è stata compromessa:

- Suppose X has broken an old session key K'_s . Then:
 1. A → KDC: $ID_A \parallel ID_B \parallel R_A$
 2. KDC → A: $E_{Ka}(K_s \parallel ID_B \parallel R_A \parallel E_{Kb}(K_s \parallel ID_A))$
 3. A → X: $E_{Kb}(K_s \parallel ID_A)$
 - 3'. X → B: $E_{Kb}(K'_s \parallel ID_A)$
 4. B → X: $E_{K's}(R_B)$
 5. X → B: $E_{K's}(f(R_B))$
- il messaggio del punto 3 può essere inviato di nuovo,
- il messaggio del punto 4 può quindi essere intercettato e decifrato con la vecchia chiave e dunque l'attaccante può rispondere correttamente al messaggio facendo credere a B di star comunicando con A.

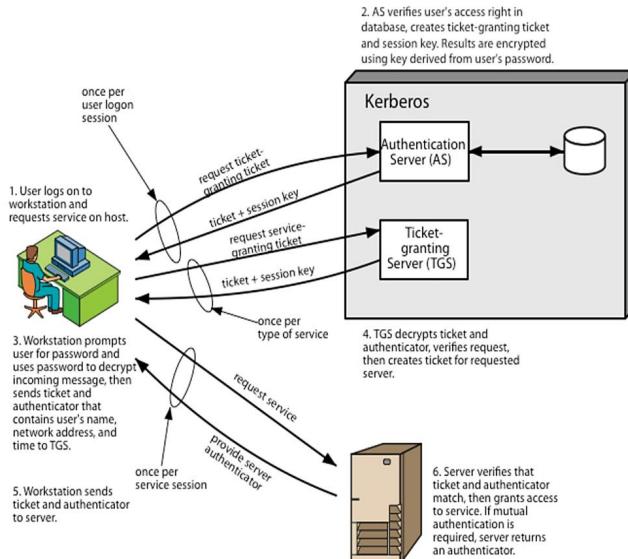
Autenticazione tramite cifratura simmetrica: KERBEROS

Kerberos è il Trusted Key System del MIT e fornisce un'autenticazione third-party centralizzata della chiave privata in una rete distribuita. Permette agli utenti di accedere ai servizi distribuiti attraverso la rete senza doversi "fidare" di tutte le workstation ma, piuttosto, tutti fanno affidamento a un server centrale di autenticazione. È basato sul protocollo Needham-Shroeder.

- Prevede l'uso di un Authentication server (AS),
- gli utenti inizialmente "negoziato" con l'AS per identificarsi,
- l'AS fornisce all'utente una credenziale di autenticazione non corruttibile (ticket granting ticket TGT),
- prevede l'uso di un Ticket Granting Server (TGS) che distribuisce ticket specifici per servizi specifici,
- gli utenti, in sequenza, richiedono l'accesso agli altri servizi al TGS sulla base del proprio TGT.

Nel processo si possono individuare tre fasi:

Kerberos 4 Overview



- l'ottenimento del **ticket granting ticket** dall'AS una volta per sessione, quando l'utente fa login. Ogni ticket ha un tempo di scadenza e una chiave di sessione che va usata dal client per il server specifico;
- l'ottenimento del **service granting ticket** dal TGS, uno per ogni servizio richiesto;
- uno scambio client/server (per ogni servizio richiesto) per ottenere il servizio.

Ogni comunicazione in Kerberos è protetta da crittografia a chiave simmetrica. Ogni client (utente) e ogni server ha una chiave segreta nota solo al possessore e al Kerberos Authentication Server. Per facilitare il processo, gli utenti hanno una password che viene trasformata nella chiave segreta quando serve. Per qualsiasi altro tipo di comunicazione, sono usate chiavi di sessione generate dall'AS o dal Ticket Granting Server (TGS).

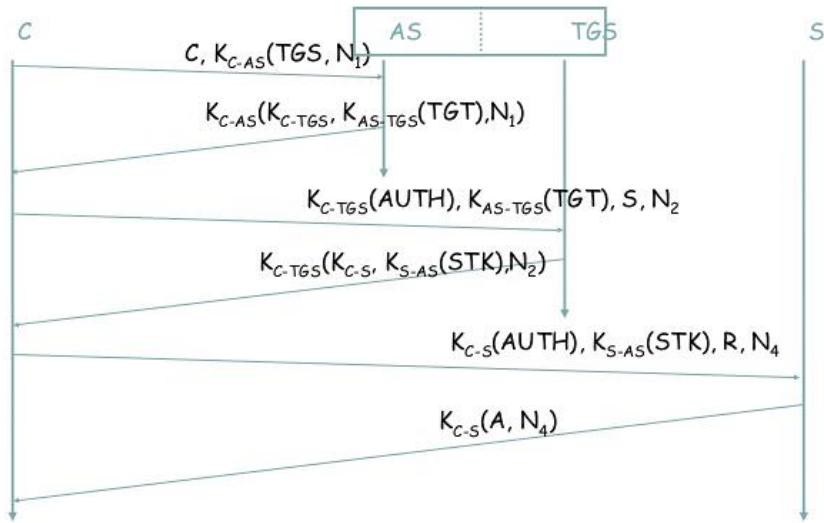
Reami di Kerberos

L'ambiente Kerberos consiste in:

- un server Kerberos,
- un certo numero di client, tutti registrati al server
- degli application server che condividono le chiavi con il server.

Questo è definito un **realm**, in genere un singolo dominio amministrativo. In caso di presenza di più realm, i loro server Kerberos devono condividere le chiavi.

Il protocollo di Kerberos (6-STEP)



STEP 1

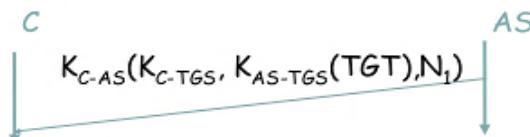
Il primo step consiste nell'autenticazione da parte del client C con il Kerberos Authentication Server e nella richiesta di un TGT.



Ciò richiede un ticket per il client C per il Ticket Granting Server TGS, con N_1 come timestamp nonce.

STEP 2

Quando l'Authentication Server riceve la richiesta, la decifra e verifica l'identità del client. Quindi genera una chiave di sessione per il client e il Ticket Granting Server e lo invia in risposta.



Dove:

$$TGT = (C, TGS, T_1, L_1, K_{C-TGS})$$

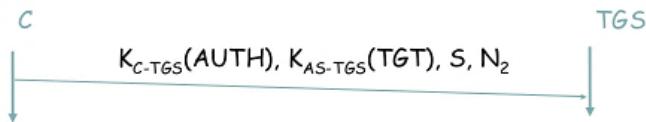
T_1, L_1 sono il timestamp e la durata del ticket.

Finché questo è cifrato con la chiave segreta di C, solo C potrà usarlo e solo l'AS potrà averlo inviato.

STEP 3

Il client, ormai autenticato, decifra la risposta ottenuta ottenendo una chiave di sessione per il TGS e un ticket granting ticket. Quando il client ha bisogno di contattare il server, crea un nuovo (fresh) authenticator (AUTH) e richiede un ticket al Ticket Granting Server.

Dove: $AUTH = (C, N_3)$

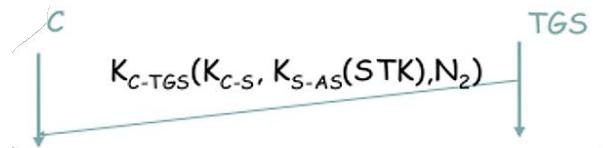
**STEP 4**

Il Ticket Granting Server decifra il ticket granting ticket e ottiene la chiave di sessione in esso contenuta. Decifra poi l'authenticator e compara l'id in esso contenuto con il ticket. Genera poi una chiave di sessione sotto forma di service ticket e lo invia a C.

Dove:

$STK = (C, S, T_2, L_2, K_{C-S})$

T_2, L_2 sono il timestamp e la durata del ticket.

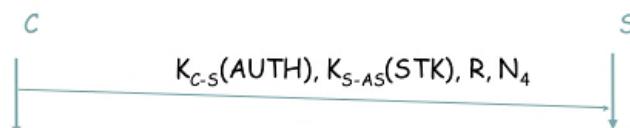


Finché questo è cifrato con la chiave di sessione, solo il client potrà usarlo e solo il TGS potrà averlo inviato.

STEP 5

Il client decifra la risposta del TGS ottenendo la chiave di sessione per il server e un service ticket da usare con lo stesso server. Quando il client ha bisogno di contattare il server, crea un nuovo authenticator (AUTH) e lo invia insieme al service ticket e alla richiesta R.

Dove: $AUTH = (C, N_5)$

**STEP 6**

Il server decifra il service ticket e ottiene la chiave di sessione in esso contenuta. Decifra poi l'authenticator e compara l'id del client in esso contenuta con il ticket. Il server esegue la richiesta R e invia una risposta A e la nonce della richiesta del client.



Finché questo è cifrato con la chiave di sessione, solo il client potrà usarlo e solo il server potrà averlo inviato; la chiave di sessione può essere usata per una richiesta addizionale in questa sessione e poi scartata.

Kerberos Wrap-Up

Attraverso la capacità di concessione dei ticket, si può ottenere scalabilità con un KDC che concede un ticket che permette a un client di contattare un KDC in un altro reame. Vi sono, in primo luogo, problemi di impostazione delle chiavi e il pericolo di attacchi di interruzione sul KDC stesso. Kerberos dipende da un buon network time service per sincronizzare i tempi di sistema per accettare i timestamp. Ciò rende il time service un buon bersaglio per gli attacchi di interruzione.

Gestione delle chiavi con codifica a chiave pubblica

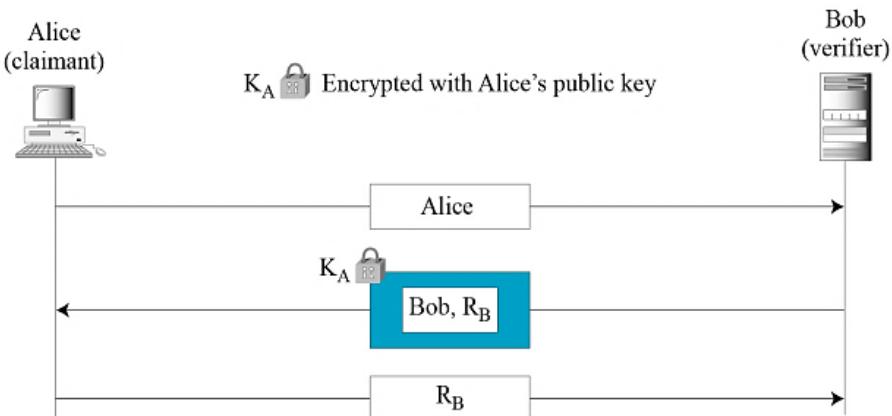
“B invia ad A un messaggio R_B cifrato con la chiave pubblica di A. Solo A con la propria chiave privata può decifrarlo (solo A è autenticato). A fa lo stesso con un messaggio R_A e la chiave pubblica di B” (entrambi autenticati). Si vedano gli schemi sottostanti.

Questo tipo di gestione ha bisogno di avere la garanzia che una parte abbia la corretta chiave pubblica dell'altra parte: una soluzione potrebbe essere quella di utilizzare un Authentication Server (AS) centrale che distribuisca i certificati di chiave pubblica.

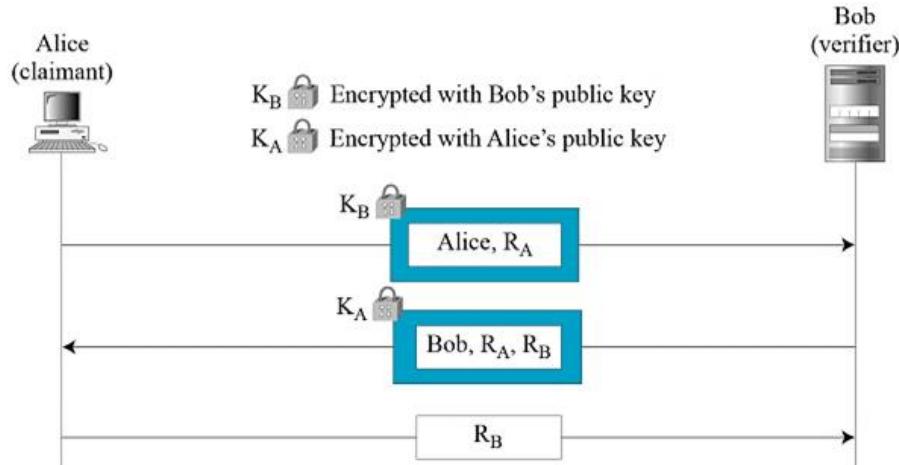
Autenticazione con codifica a chiave asimmetrica

Il segreto deve essere la chiave segreta del claimant.

Protocollo a una direzione:



Protocollo bidirezionale:



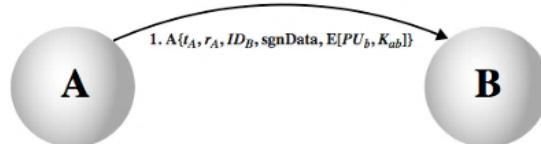
Procedure di autenticazione in X.509

Lo standard X.509 definisce tre procedure di autenticazione:

1. autenticazione one-way, per messaggi unidirezionali;
2. autenticazione two-way, per sessioni interattive basate su timestamp;
3. autenticazione three-way, per sessioni interattive basate su nonces senza timestamp.

Tutte usano firma a chiave pubblica.

Autenticazione one-way



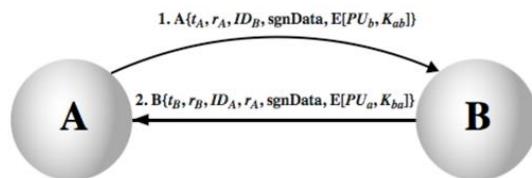
Viene usato un solo messaggio (A → B) per stabilire:

- l'identità di A e che il messaggio proviene da A,
- che il messaggio è destinato a B,
- l'integrità e l'originalità del messaggio.

Il messaggio DEVE includere **timestamp**, **nonce**, **l'identità di B** e deve essere firmato da A.

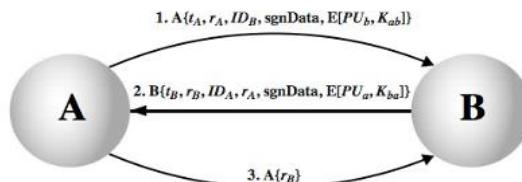
Autenticazione two-way

La risposta di A a B contiene una copia firmata della nonce arrivata da B, ciò vuol dire che il timestamp non ha bisogno di essere controllato.



Autenticazione three-way

- 3 messaggi (A→B, B→A, A→B) che consentono l'autenticazione senza orologi sincronizzati
- ha una risposta da A a B contenente una copia firmata di nonce da B
- significa che non è necessario controllare o fare affidamento sui timestamp



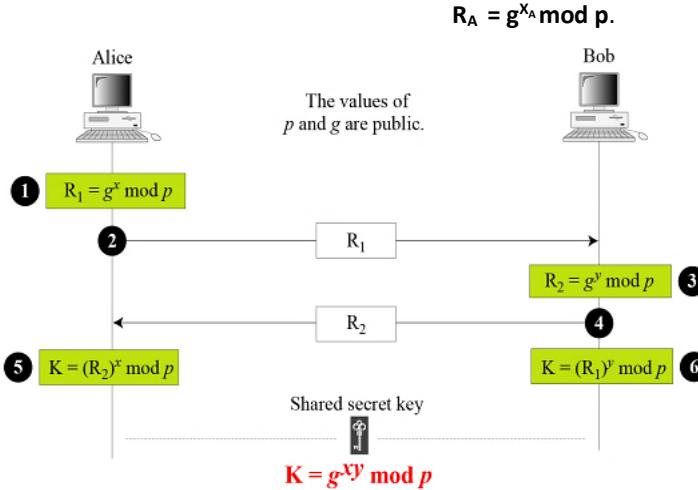
Diffie-Hellman Key Agreement

Basato sull'aritmetica a campi finiti. Tutti gli utenti devono decidere dei parametri da settare: un numero primo grande p e una radice primitiva g di p .

(Si chiama "radice primitiva" di un intero n ogni numero r primo rispetto a n , tale che $r^k \bmod n$ assume $\phi(n)$ valori distinti se k varia da 1 a $n - 1$):

$$\{g^i \bmod p \mid 0 \leq i \leq p-1\} = \{0 \dots p-1\}$$

Ogni utente (per esempio A) genera la propria chiave scegliendo un numero di chiave segreto $X_A < p$ e calcola la propria chiave pubblica:



• Example: users Alice & Bob who wish to swap keys:

1. agree on prime $p=353$ and $a=3$

2. select random secret keys:

 ? A chooses $x_A=97$, B chooses $x_B=233$

3. compute respective public keys:

 ? $R_A = 3^{97} \bmod 353 = 40$ (Alice)

 ? $R_B = 3^{233} \bmod 353 = 248$ (Bob)

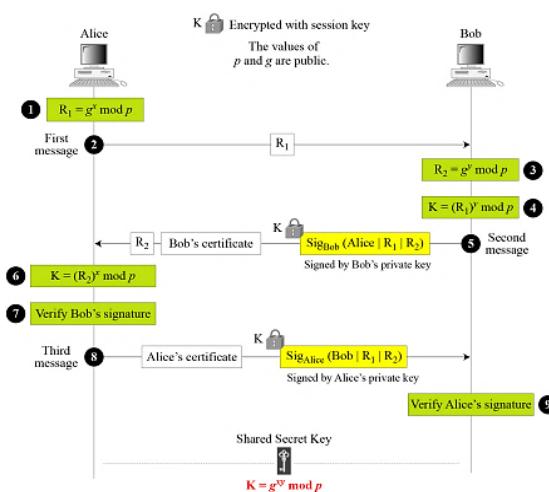
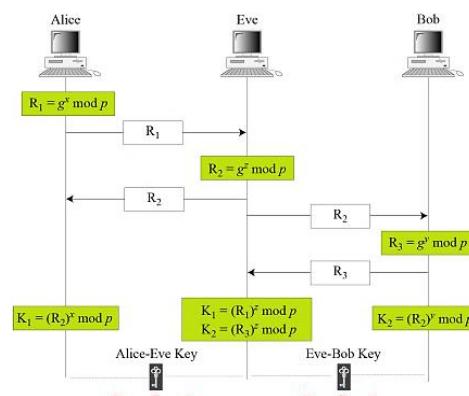
4. compute shared session key as:

 ? $K_{AB} = R_B^{x_A} \bmod 353 = 248^{97} = 160$ (Alice)

 ? $K_{AB} = R_A^{x_B} \bmod 353 = 40^{233} = 160$ (Bob)

Protocollo di scambio di chiave in Diffie-Hellman

Ciascun utente potrebbe creare chiavi private/pubbliche Diffie-Hellman casuali ogni volta che comunica; oppure potrebbe creare una chiave privata/pubblica D-H e pubblicarla in una directory e poi consultarla e usarla per comunicare in modo sicuro. Entrambe queste possibilità sono vulnerabili ad attacchi **man-in-the-middle**, perché le parti non sono autenticate: l'attaccante può spacciarsi per una delle due parti e ottenere due chiavi di sessione, una per ogni parte della comunicazione.



Safe station-to-station key agreement

È necessaria l'autenticazione delle chiavi; solitamente le chiavi sono firmate usando certificati a chiavi pubbliche.

Sicurezza al livello di rete e al livello di trasporto

Security services & protocols vs ISO OSI architecture

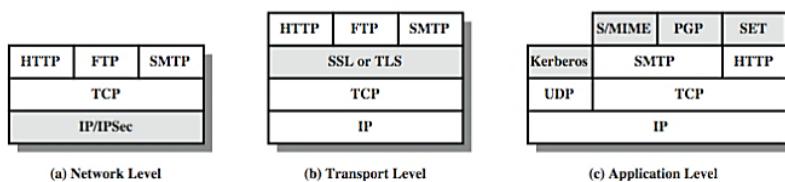
Level	Services	Protocols/applications
7 Application	End-to-end services	PGP, S/MIME, SAML, SSH, EMV, SET, ...
6 Presentation		
5 Session	Authentication encryption traffic analysis...	SSL/TLS, Secure UDP
4 Transport		
3 Network	Authentication, encryption	IPSec
2 Datalink	Authentication for link access, confidentiality	WEP, WPA, 802.1X
1 Physical		

Approcci alla web security:

- a) Il livello di rete è a livello di sistema ma è pesante da implementare.
- b) Il livello di trasporto è abbastanza generale e la sicurezza può essere implementata nello spazio dell'utente.
- c) Al livello di applicazione si trovano sistemi di sicurezza specifici.

Implementare i servizi di sicurezza a livello di rete (livello 3) vuol dire andare a toccare il sistema operativo e l'amministratore di sistema. Una sicurezza di questo tipo garantisce che ogni cosa che esce dalla macchina è sicura e il programmatore non si accorge nemmeno della cifratura.

Non è detto però che il SO supporti tutti i sistemi (come ad esempio IP-Sec), quindi è più facile implementare i sistemi di sicurezza al livello di trasporto, creando delle socket sicure che il programmatore può usare. Il buon funzionamento di questo sistema prevede che il programmatore possieda il codice sorgente di queste socket.



Sicurezza implementata a livello di trasporto: Secure Socket Layer

Si tratta di una libreria, per cui non influisce sul sistema operativo, adatta a tutti i protocolli a flusso. (TLS = SSL versione 3, NON sono compatibili).

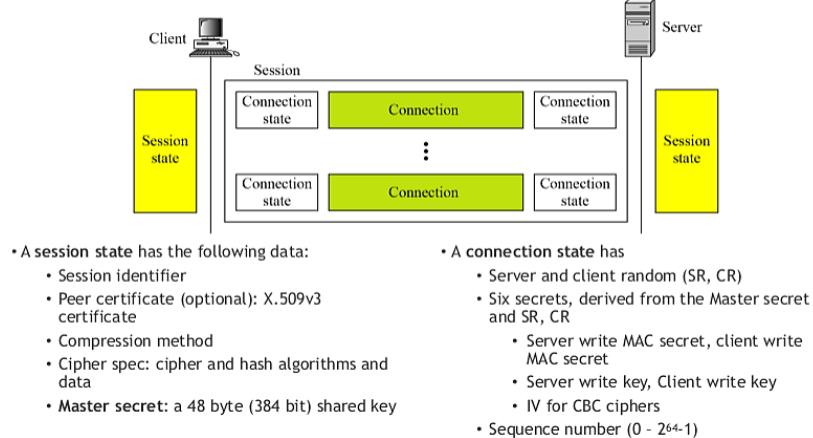
Architettura SSL

SSL ha due strati di protocolli:

- **protocollo SSL record:** prende i payload, li cifra e li spedisce, ovvero implementa la socket sicura. Per funzionare ha bisogno di alcune chiavi segrete gestite da protocolli ausiliari;
- **SSL Handshake, Change Cipher Spec e Alert Protocols:** usati per l'autenticazione delle entità e per settare le informazioni necessarie per i servizi di sicurezza.

Concetto di sessione SSL: la sessione è l'associazione tra client e server, creata da un protocollo di handshake, tramite cui si definiscono le chiavi. Una sessione SSL può contenere tante connessioni e prevede l'uso di un **master secret**, ovvero una chiave master di 4 byte.

SSL Session and Connection states



SSL Session state

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.

La combinazione del metodo scambio chiave, hash e algoritmo di cifratura usati definiscono la **cipher suite** per ogni sessione SSL. La cipher suit, quindi, stabilisce:

- il modo con cui vengono scambiate le chiavi;
- il modo con cui vengono cifrati i dati;
- il modo con cui si applica la funzione di hash.

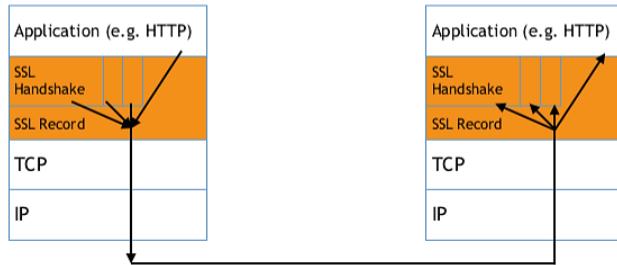
SSL cipher suit list

Cipher suite	Key Exchange	Encryption	Hash
SSL_NULL_WITH_NULL_NULL	NULL	NULL	NULL
SSL_RSA_WITH_NULL_MD5	RSA	NULL	MD5
SSL_RSA_WITH_NULL_SHA	RSA	NULL	SHA-I
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4	MD5
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4	SHA-I
SSL_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA	SHA-I
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES	SHA-I
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES	SHA-I
SSL_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4	MD5
SSL_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES	SHA-I
SSL_DHE_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES	SHA-I
SSL_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES	SHA-I
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES	SHA-I
SSL_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES	SHA-I
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES	SHA-I
SSL_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES	SHA-I
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES	SHA-I
SSL_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES	SHA-I
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES	SHA-I
SSL_FORTEZZA_DMS_WITH_NULL_SHA	Fortezza	NULL	SHA-I
SSL_FORTEZZA_DMS_WITH_FORTEZZA_CBC_SHA	Fortezza	Fortezza	SHA-I
SSL_FORTEZZA_DMS_WITH_RC4_128_SHA	Fortezza	RC4	SHA-I

SSL Record protocol

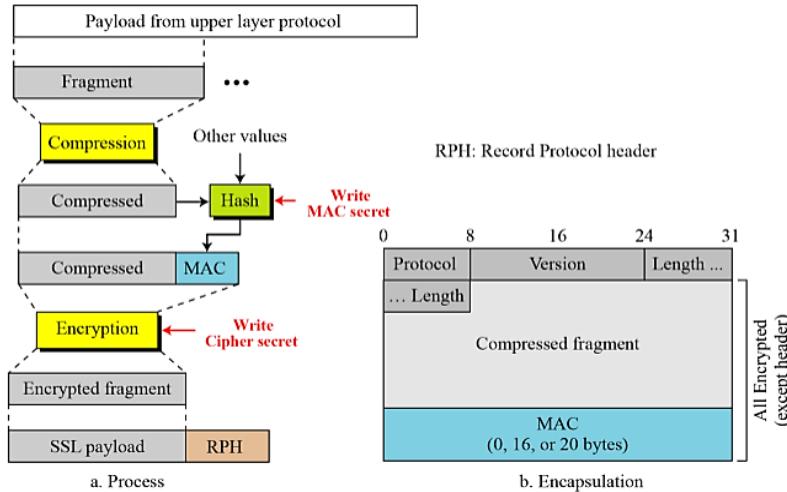
Protocollo che, a sessione creata, esegue la trasformazione in entrambi i sensi e invia i messaggi su un canale TCP normale.

- SSL Record implements the real security transformations, using connection-specific information set up from session keys



SSL garantisce quindi:

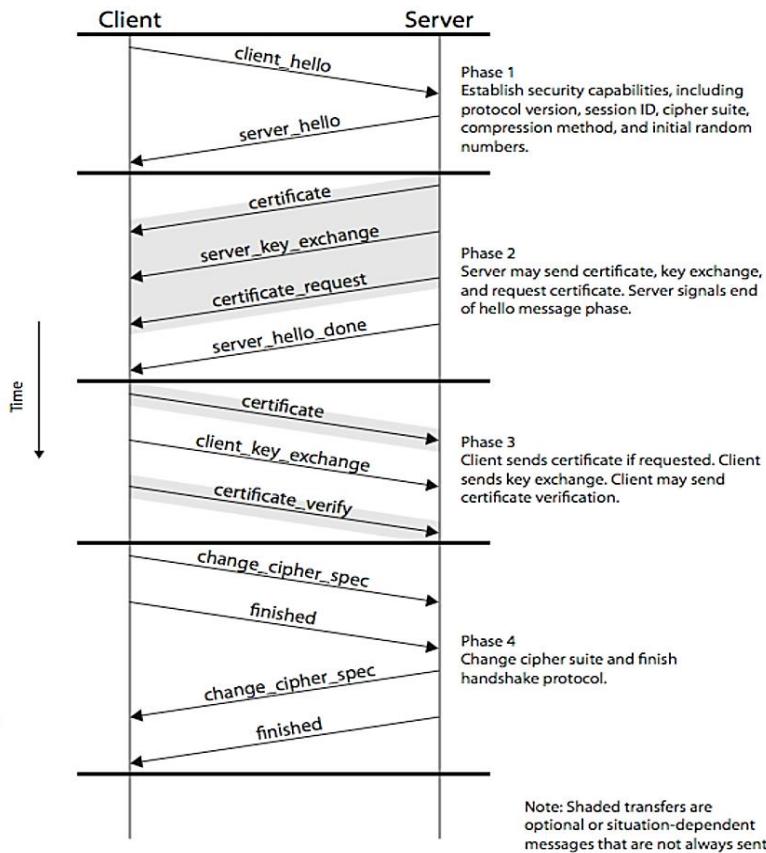
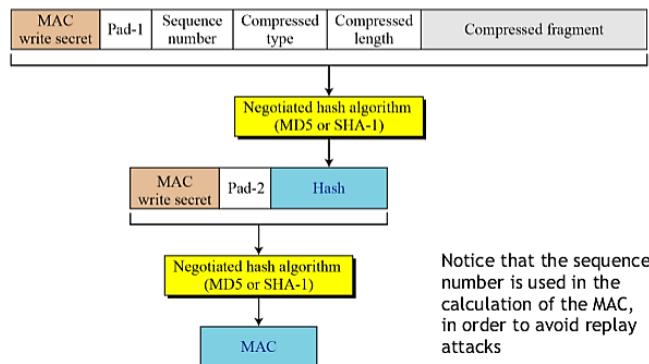
- **integrità dei messaggi** utilizzando MAC con chiave condivisa e meccanismi anti-reply;
- **confidenzialità** utilizzando cifratura simmetrica con una chiave segreta condivisa definita tramite il protocollo di Handshake.



NB: SSL può cifrare al massimo 16 kB. Il numero di sequenza non è incluso nell'intestazione ma è tenuto aggiornato da chi invia e da chi riceve.

Calcolo del Mac

Pad-1: Byte 0x36 (00110110) repeated 48 times for MD5 and 40 times for SHA-1
 Pad-2: Byte 0x5C (01011100) repeated 48 times for MD5 and 40 times for SHA-1



Segue a pagina successiva la descrizione di ogni fase.

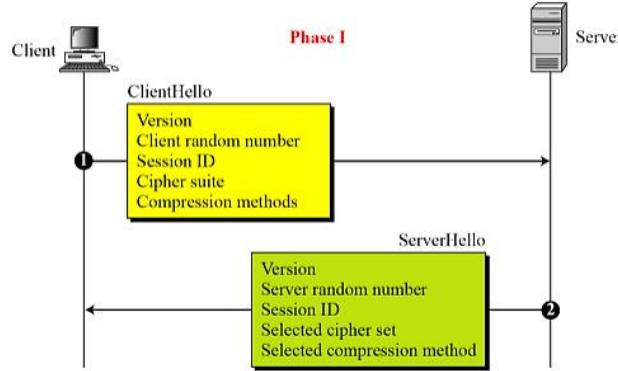
Protocollo di Handshake SSL

Il protocollo di Handshake SSL si sviluppa in quattro fasi:

1. presentazione tra client e server e decisione dei parametri di sicurezza;
2. autenticazione server-client e scambio di chiavi;
3. autenticazione client-server e scambio di chiavi;
4. chiusura = viene attivato il cifrario.

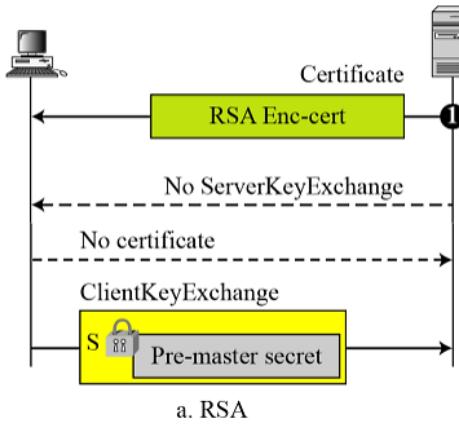
Handshake: fase 1

Il client si presenta e propone “un’offerta” con tutte le cipher suite che può implementare, ordinate per preferenza. Il server scandisce la lista (partendo dall’alto, quindi dalle cipher suite che il client più preferisce) e sceglie la prima suite che è in grado di implementare. Viene così deciso l’algoritmo di cifratura e il metodo di scambi della chiave master.



Handshake: fase 2-3

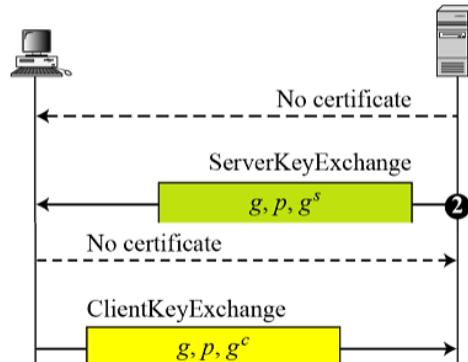
Ciò che avviene nelle fasi 2 e 3 dipende da quanto è stato stabilito nella fase 1 = sono possibili 4 modi di scambio delle chiavi:



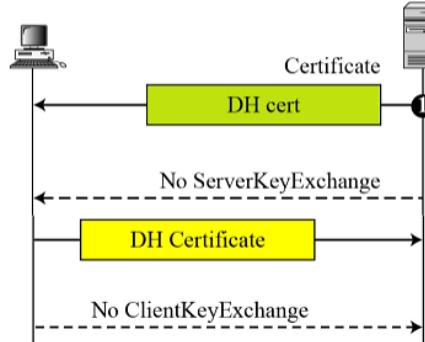
- **RSA:** quando il client non ha un certificato X.509 ma il server sì. Il server invia il certificato al client il quale lo verifica e, se il certificato risulta valido, genera un numero random detto **pre-master secret** e lo spedisce al server cifrandolo con la chiave pubblica del server stesso.

a. RSA

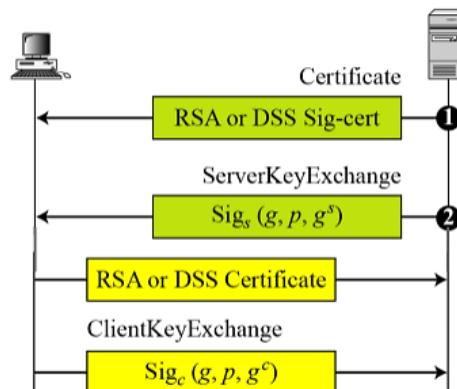
- **Diffie-Hellman anonimo:** il server fissa g e p e calcola la sua mezza chiave; stessa cosa viene fatta dal client (vedi Diffie-Hellman). Vulnerabile ad attacchi attivi man-in-the-middle ma non necessita di alcun certificato.



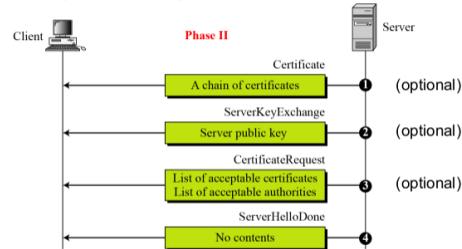
- **Fixed Diffie-Hellman:** quando sia il client sia il server possiedono un certificato che contiene una mezza chiave D-H, possono scambiarcela e calcolare la mezza chiave corrispondente. È robusto perché le mezze chiavi sono autenticate dalla CA.



- **Diffie-Hellman effimero:** sia il client sia il server possiedono un certificato X.509. Il server manda valori random cifrati con la chiave RSA e il client può verificare la chiave, stessa cosa succede a parti invertite. È IL SISTEMA PIÙ ROBUSTO; il contributo delle parti è simmetrico ma necessita che anche il client abbia un certificato.

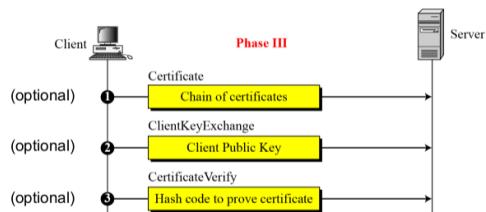


SSL handshake protocol: phase 2



- After Phase 2
 - The server is authenticated to the client.
 - The client knows the public key of the server if required

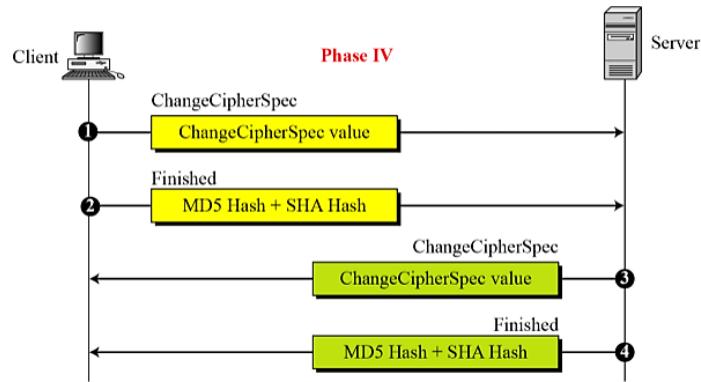
SSL handshake protocol: phase 3



- After Phase 3
 - The client is authenticated for the server.
 - Both the client and the server know the pre-master secret.

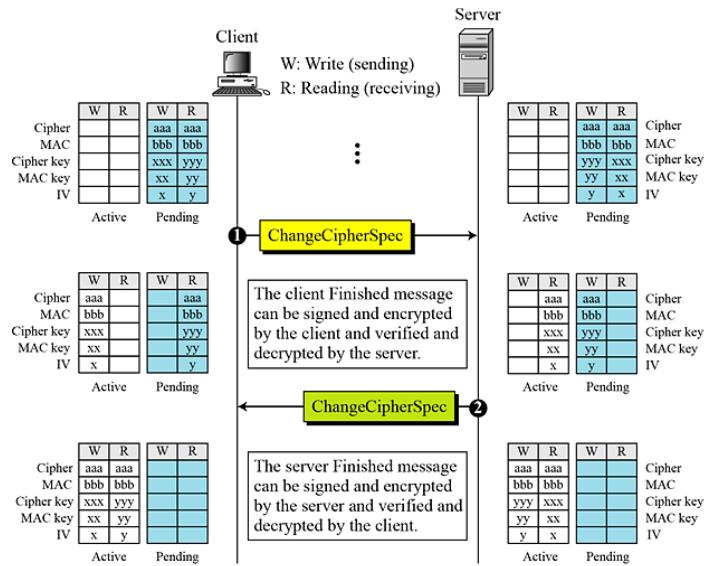
Handshake: fase 4 (Change Cipher Spec)

Si sceglie una nuova Cipher Suite con i parametri selezionati. Alla fine di questa fase client e server sono pronti a scambiare dati.



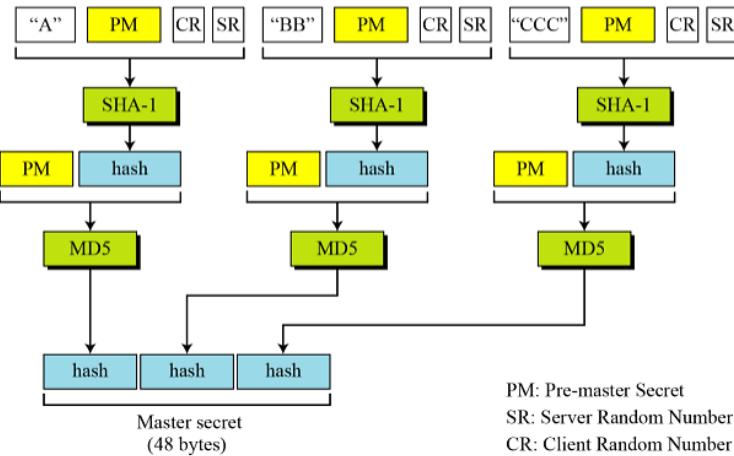
SSL Change Cipher Spec

Un singolo messaggio fa sì che lo stato in sospeso diventi corrente e quindi provoca l'aggiornamento della cipher suite in uso. Il numero di sequenza viene resettato.



Generazione del master secret

- SR e CR sono i numeri scambiati nella fase 1;
- “A”, “BB”, “CCC” sono stringhe ASCII;
- può essere generato sia dal client sia dal server.



TLS (Transport Layer Security)

- IETF standard RFC 2246 similar to SSLv3
- with minor differences
 - in record format version number
 - uses HMAC for MAC
 - a pseudo-random function expands secrets
 - has additional alert codes
 - some changes in supported ciphers
 - changes in certificate types & negotiations
 - changes in crypto computations & padding
 - does not support Fortezza

Come fare un porting da un'applicazione che usa socket normali a SSL?

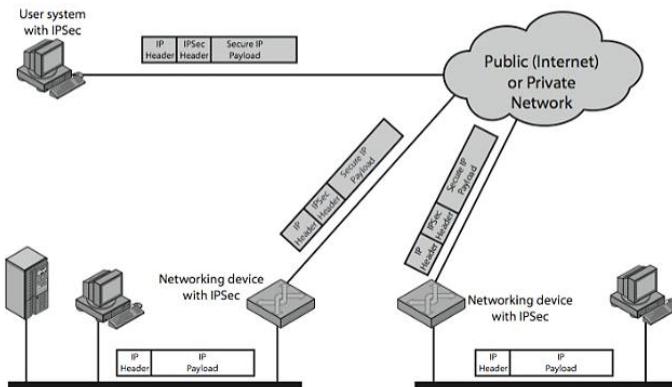
Bisogna mappare le chiamate SSL su quelle originali con l'aggiunta, all'inizio, della chiamata `SSL_new` per creare un sessione SSL.

Sicurezza implementata a livello di rete

IPSec (RFC 2401-2412)

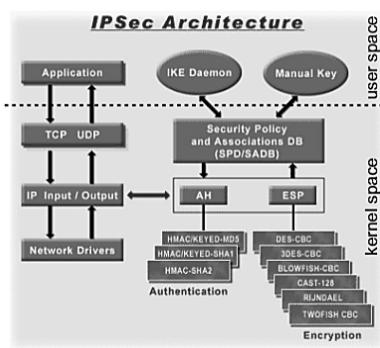
RFC 2401-2412: insieme di protocolli che mirano a garantire autenticazione, confidenzialità e gestione delle chiavi

Architettura di IPSec



I pacchetti IP, prima di essere inviati (ricevuti), vengono trattati da protocolli ausiliari di autenticazione (AH) e di cifratura (ESP). Questi protocolli hanno bisogno di segreti che possono essere condivisi manualmente o automaticamente (IKE DAEMON) per popolare il Security Policy and Associations Data Base (SPDAA/DB).

- specification is quite complex
- defined in numerous RFC's
 - incl. RFC 2401-2408 (1998), 4301-4306 (2005)
 - many others, grouped by category
- mandatory in IPv6, optional in IPv4
- Two operating modes: **transport/tunnel**
- have two security header extensions:
 - Authentication Header (AH)
 - Encapsulating Security Payload (ESP)
- Further protocol for exchanging keys (IKE)



IPSec può aggiungere ai pacchetti IP:

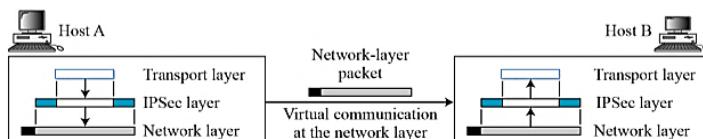
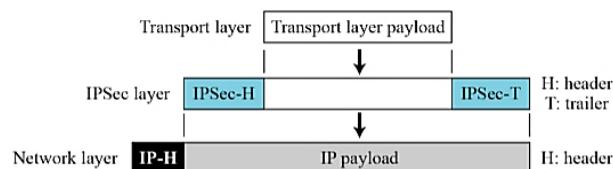
- AH
- ESP

IPSec presenta due modalità:

- Trasporto
- Tunnel

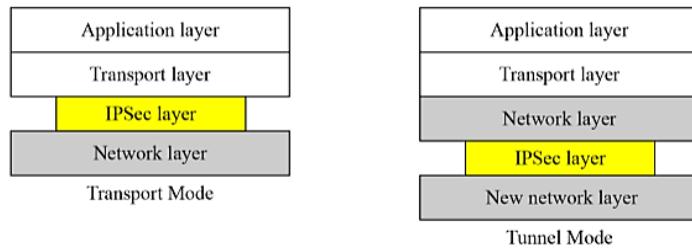
IPSec in Transport mode

L'header di sicurezza viene inserito dopo l'originale header IP e prima dell'originale payload IP. I router intermedi vedono quindi dei semplici pacchetti IP.



IPSec in Tunnel mode

Fornisce protezione all'intero pacchetto IP. Dopo che l'header AH/ESP è stato aggiunto, l'intero pacchetto più i campi di sicurezza viene trattato come il payload di un uovo pacchetto IP. Il risultante pacchetto IPSec viene considerato come un pacchetto IP standard dalle reti intermedie.



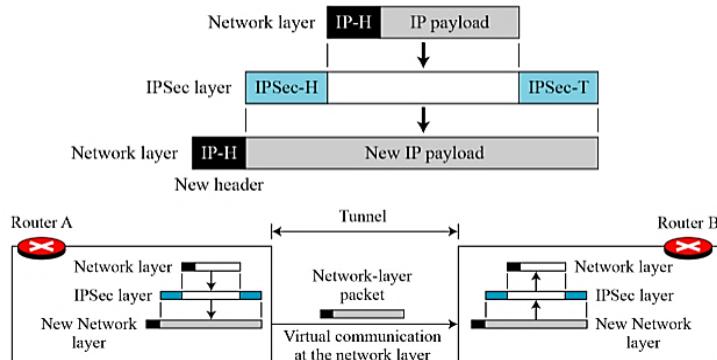
Protocolli di sicurezza IPSec

IPSec definisce due protocolli per garantire autenticazione e/o cifratura per i pacchetti IP:

- Authentication Header (AH) Protocol
- Encapsulating Security Payload (ESP) Protocol

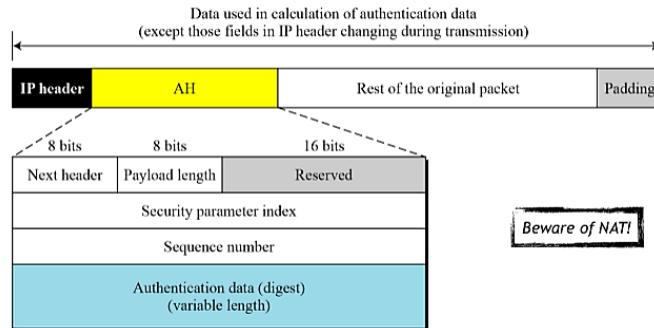
ESP è stato progettato quando AH era già in uso, quindi ESP fa le stesse cose che fa AH ma con l'aggiunta della funzionalità della privacy (confidenzialità).

Services	AH	ESP
Access control	yes	yes
Message authentication (message integrity)	yes	yes
Entity authentication (data source authentication)	yes	yes
Confidentiality	no	yes
Replay attack protection	yes	yes



Authentication Header (AH)

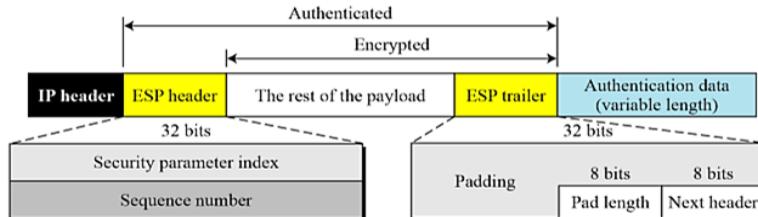
L'Authentication Header (AH) protocol è progettato per autenticare l'host sorgente e per garantire l'integrità del payload trasportato dal pacchetto IP. Il protocollo usa una funzione di hash e una chiave simmetrica per creare un digest, il quale viene inserito all'interno dell'authentication header. L'AH viene quindi posizionato nella posizione appropriata a seconda della modalità (trasporto o tunnel). Quando un datagramma IP trasporta un authentication header, il valore originale del campo protocol viene sostituito con il valore 51. Un campo all'interno dell'AH (Next header) conserva il valore originale del campo protocol. (Digest = tutti i dati messi in hash con il segreto, anche l'intestazione viene autenticata.) Non è compatibile con NAT il quale cambia l'indirizzo del mittente.



Encapsulating Security Payload (ESP)

Il protocollo AH non fornisce privacy ma solo autenticazione della fonte e integrità dei dati. IPSec ha successivamente definito un protocollo alternativo Encapsulating Security Payload (ESP) che fornisce autenticazione, integrità e privacy. Aggiunge sia un header sia un trailer; inoltre i dati di autenticazione di ESP sono aggiunti alla fine del pacchetto, il che li rende più facilmente calcolabili. L'intestazione IP è esclusa dal calcolo per l'autenticazione, quindi ESP è compatibile con NAT.

Quando un datagramma IP trasporta un header e un trailer ESP, il valore nel campo protocol viene sostituito con il valore 50; un campo dentro il trailer (Next header) ne conserva il valore originale.



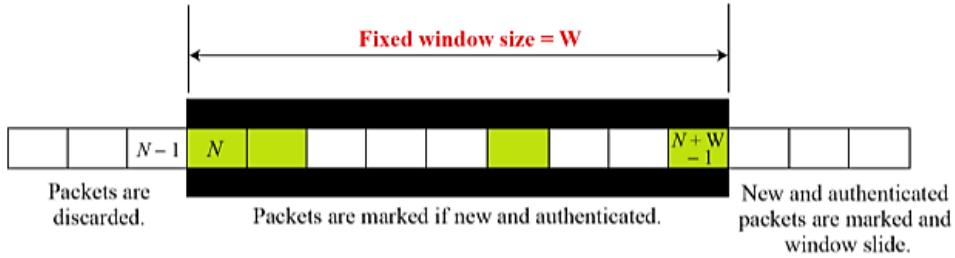
Transport vs Tunnel mode ESP

- **Transport mode:**
 - usata per cifrare e, optionalmente, autenticare dati IP;
 - i dati sono protetti ma l'header rimane in chiaro;
- **Tunnel mode:**
 - cifra l'intero pacchetto IP;
 - aggiunge un nuovo header per il next hop;

Protezione anti-reply

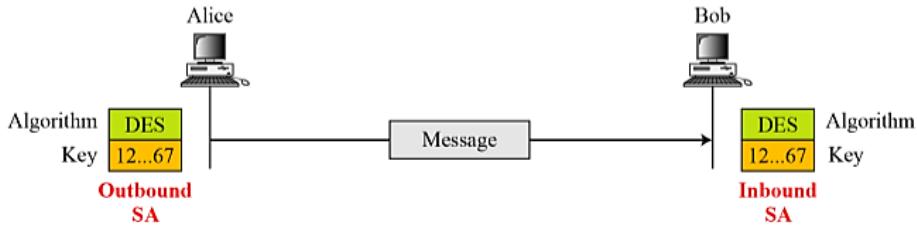
IPSec dispone di una tecnica anti-reply: utilizza un numero di sequenza (strano per IP), quindi i pacchetti sono numerati. Chi riceve i pacchetti usa una tecnica simile a sliding-window con finestra di dimensione fissata per decidere se il pacchetto è "buono": considerato il numero del più recente pacchetto ricevuto, se è compreso tra gli estremi della finestra si verifica se si tratta di un pacchetto "fresco" (ovvero non ancora ricevuto). Se arriva un pacchetto con un numero più vecchio, fuori dall'intervallo indicato dalla finestra (per la precisione a sinistra della finestra), viene scartato senza eseguire ulteriori controlli (BEST-EFFORT). Se arriva un pacchetto

con un numero più grande, viene accettato e la finestra viene spostata in avanti fino a che l'estremo superiore non coincide con il numero del pacchetto appena ricevuto.



Security Associations (SA)

Per far funzionare IPSec servono delle chiavi (segreti). Le **security associations** sono relazioni logiche unidirezionali tra mittente e destinatario che offrono sicurezza per il traffico trasmesso. Definiscono il dominio di interpretazione necessario per i meccanismi sia di autenticazione sia di confidenzialità.



Security Association parameters

Parameters	Description
Sequence Number Counter	This is a 32-bit value that is used to generate sequence numbers for the AH or ESP header.
Sequence Number Overflow	This is a flag that defines a station's options in the event of a sequence number overflow.
Anti-Replay Window	This detects an inbound replayed AH or ESP packet.
AH Information	This section contains information for the AH protocol: <ul style="list-style-type: none"> 1. Authentication algorithm 2. Keys 3. Key lifetime 4. Other related parameters
ESP Information	This section contains information for the ESP protocol: <ul style="list-style-type: none"> 1. Encryption algorithm 2. Authentication algorithm 3. Keys 4. Key lifetime 5. Initiator vectors 6. Other related parameters
SA Lifetime	This defines the lifetime for the SA.
IPSec Mode	This defines the mode, transport or tunnel.
Path MTU	This defines the path MTU (fragmentation).

Security Association Database (SAD)

Ogni implementazione di IPSec ha un database delle Security Association (SED) attive in qualsiasi momento.

Index	SN	OF	ARW	AH/ESP	LT	Mode	MTU
< SPI, DA, P >							
< SPI, DA, P >							
< SPI, DA, P >							
< SPI, DA, P >							

Security Association Database

Legend:

SPI: Security Parameter Index	SN: Sequence Number
DA: Destination Address	OF: Overflow Flag
AH/ESP: Information for either one	ARW: Anti-Replay Window
P: Protocol	LT: Lifetime
Mode: IPSec Mode Flag	MTU: Path MTU (Maximum Transfer Unit)

Security policy

Un altro importante aspetto di IPSec è rappresentato dalle **security policy (SA)** che definiscono il tipo di sicurezza applicata a un pacchetto quando viene inviato o quando viene ricevuto. Prima di usare il SAD, un host deve definire una policy predefinita per il pacchetto, scegliendo se scartare, applicare o bypassare il processo di sicurezza. Le security polici sono tenute in un database indicizzato attraverso gli indirizzi sorgente e destinazione, le porte e i protocolli.

Index	Policy
< SA, DA, Name, P, SPort, DPort >	
< SA, DA, Name, P, SPort, DPort >	
< SA, DA, Name, P, SPort, DPort >	
< SA, DA, Name, P, SPort, DPort >	

Legend:

SA: Source Address	SPort: Source Port
DA: Destination Address	DPort: Destination Port
P: Protocol	

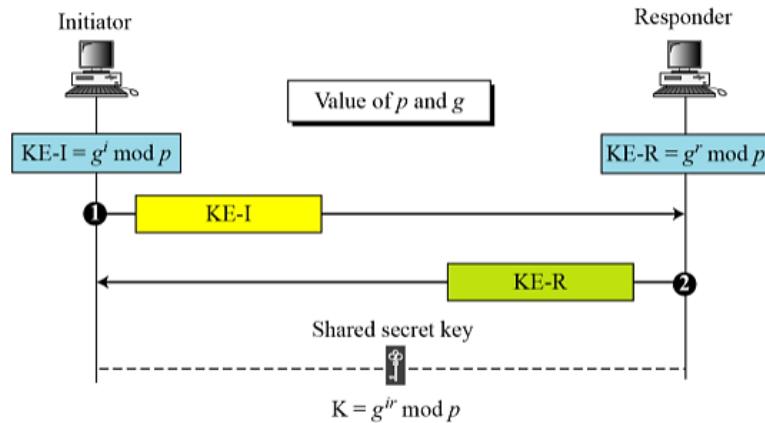
IPSec: gestione delle chiavi

Tipicamente sono necessari due paia di chiavi, due per ogni direzione per AH e ESP.

- Gestione manuale
- Gestione automatica: **IKE (Internet Key Exchange)**

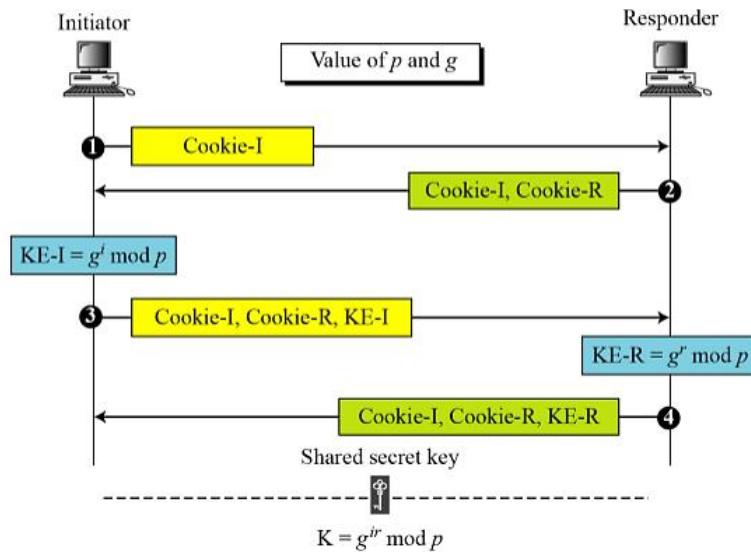
Quando viene inviato il primissimo pacchetto, ovviamente non esistono policy, quindi viene chiamato IKE per popolare le tabelle.

Diffie-Hellman basico



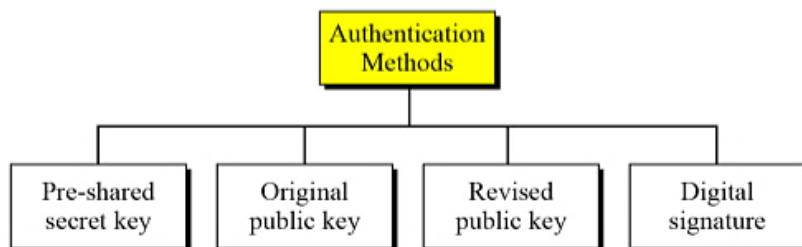
Vulnerabile a **clogging**: un attaccante può inviare numeri a caso a un responder forzandolo a calcolare chiavi inutile (costoso).

Diffie-Hellman con cookies



Diffie-Hellman migliorato in IKE

- Per proteggersi da attacchi clogging, IKE utilizza i cookies.
- Per proteggersi da attacchi reply, IKE usa nonces nel terzo e nel quarto messaggio.
- Per proteggersi da attacchi MITM, IKE richiede che ogni parte dimostri di possedere un segreto.



Fasi di IKE

IKE è diviso in due fasi:

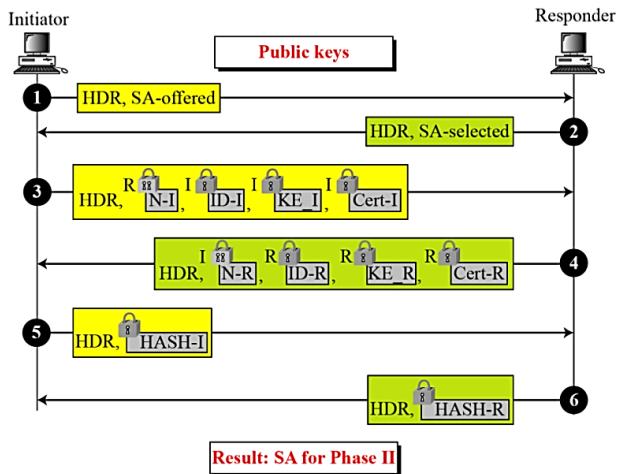
1. fase 1 (generale): creazione di SA per la fase 2.
2. fase 2 creazione di SA per uno specifico protocollo di scambio dati (es. IPSec).

FASE 1

Basata sull'esistenza di un pre-secret tra le due parti; prevede quattro metodi di autenticazione differenti.
(Le modalità aggressive sono più brevi).

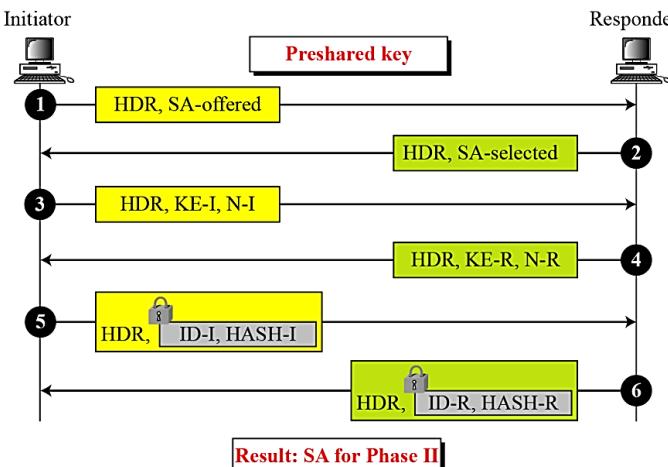
IKE Phase 1, main mode, revised public key method

HDR: General header including cookies	I Encrypted with initiator's public key
KE-I (KE-R): Initiator's (responder's) half-key	R Encrypted with responder's public key
Cert-I (Cert-R): Initiator's (responder's) certificate	R Encrypted with responder's secret key
N-I (N-R): Initiator's (responder's) nonce	I Encrypted with initiator's secret key
ID-I (ID-R): Initiator's (responder's) ID	Encrypted with SKEYID_e
HASH-I (HASH-R): Initiator's (responder's) hash	



IKE Phase 1, main mode, pre-shared key method

KE-I (KE-R): Initiator's (responder's) half-key	HDR: General header including cookies
N-I (N-R): Initiator's (responder's) nonce	Encrypted with SKEYID_e
ID-I (ID-R): Initiator's (responder's) ID	
HASH-I (HASH-R): Initiator's (responder's) hash	= hash(SKEYID, KE-I, KE-R, Cookie-I, Cookie-R, SA-I, ID-I)



IKE fase 1, aggressive mode, revised public key method

1. L'initiator propone la policy, i dati per lo scambio delle chiavi, la nounce id.
2. Il responder si autentica e fissa la policy e lo scambio delle chiavi.
3. L'ultimo messaggio è usato per autenticare l'initiator e per fornire una prova di partecipazione allo scambio.

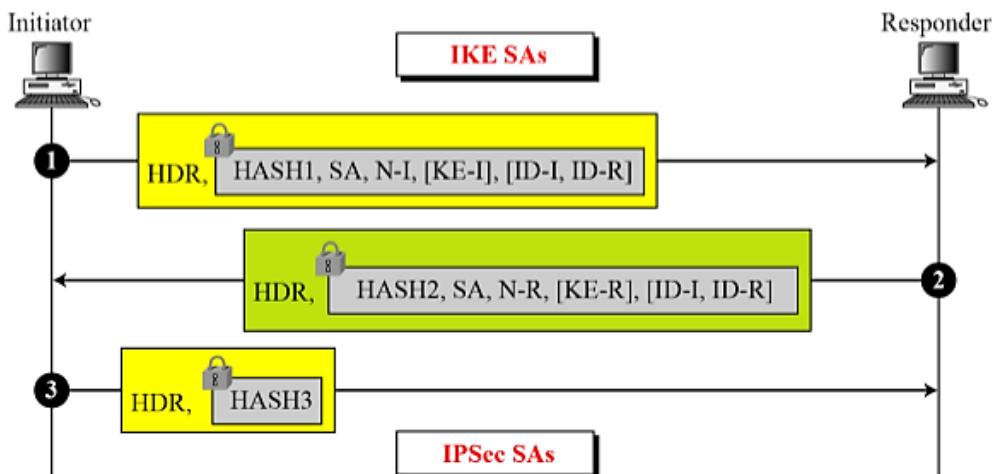
IKE fase 2, quick mode

La fase 2 crea un SA IPSec a partire dal SA IKE contenente nuove nonces e una nuova mezza chiave D-H. Da queste nonces e mezza chiave viene creato il materiale chiave K usando una funzione pseudo-casuale. Da questo materiale viene costruito il SA IPSec.

KE-I (KE-R): Initiator's (responder's) half-key HDR: General header including cookies

N-I (N-R): Initiator's (responder's) nonce  Encrypted with SKEYID_e

ID-I (ID-R): Initiator's (responder's) ID SA: Security association

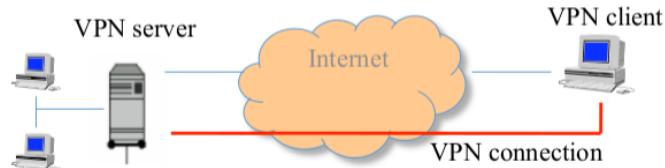


Virtual Private Networks (VPN)

Una Virtual Private Network (VPN) è una rete di comunicazione privata usata per connettere due o più reti private in modo sicuro e confidenziale entro una rete pubblica (per esempio internet).

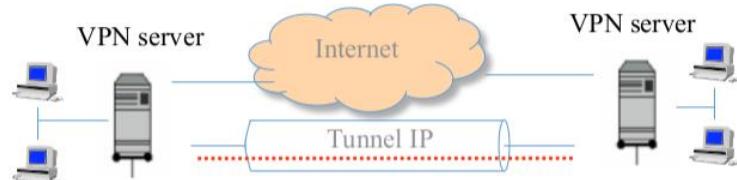
Host-to-LAN VPN

Nelle VPN Host-to-LAN la connessione è tra un singolo host e un gateway VPN della LAN. I meccanismi di autenticazione e sicurezza sono implementati su ogni client remoto.



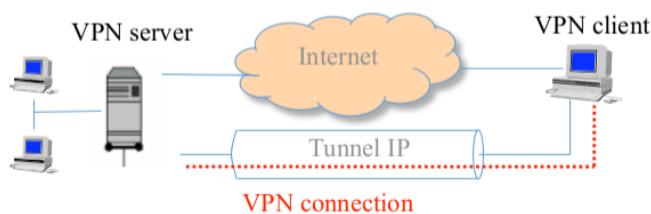
LAN-to-LAN VPN

In una VPN LAN-to-LAN il tunnel sicuro è implementato tra due gateway di sicurezza sulle due reti. È utile per connettere due intranet della stessa organizzazione senza usare linee dedicate molto costose. I meccanismi di sicurezza vanno implementati solo sulle gateway.



Servizi di VPN

- Cifratura del payload, dei dati IP...; può essere implementata tramite ESP;
- Autenticazione dei dati; disponibile sia con AH sia con ESP;
- Tunnelling: i pacchetti vengono inoltrati in modo sicuro alla destinazione finale in modo trasparente per gli utenti/applicazioni; può essere implementato usando ESP/AH in tunnel mode.



Esercizi vari

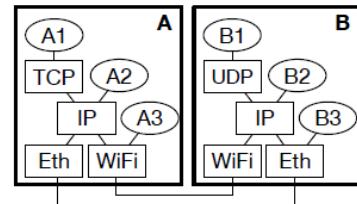
Segue una lista di vari esercizi con relativa risposta e spiegazione estese quando necessario presi da compiti degli anni precedenti:

ES: Lo schema a lato rappresenta due host A e B connessi da due mezzi fisici e su cui sono in esecuzione le applicazioni A1, A2, A3 e B1, B2, B3 rispettivamente. Le applicazioni accedono ai vari protocolli come indicato. Si dica quali delle seguenti comunicazioni può avere luogo:

- a) A1-B1 b) A2-B2 c) A3-B3

Risposta:

- a) no: B1 non può ricevere pacchetti TCP
 b) sì: I pacchetti IP possono essere scambiati via eth. o wifi.
 c) no: B3 non ha una connessione wifi.



ES di fisica: Si vuole realizzare una linea di trasmissione della capacità di 1 Mb/s di bitrate di dati senza errori, utilizzando un fascio di microonde in una banda di frequenza centrata a 10 GHz e di larghezza 500 kHz. Determinare il rapporto segnale/rumore (in dB) della linea in tali condizioni.

RIPASSO Teoria necessaria:

Rapporto segnale rumore (**SNR**): P_{segna}/P_{rumore} con P =Potenza

$$\text{Teorema S-H (Shannon Hartley): } C = B \log_2 \left(1 + \frac{S}{N} \right)$$

con: C = capacità del canale (bit/sec); B = larghezza di banda (Hz); S = Psegna; N = Prumore quindi $S/N = SNR$

1Mbit/s = 1000000 bit/sec

500Khz = 500000Hz

Risposta: Allora abbiamo che: **1000000 = 500000 * log₂(x)** e dobbiamo trovare SNR.

Svolgendo i calcoli si trova che $x=3$ soddisfa l'equazione. quindi x in db è pari a $10 * \log_{10}(3) = 4.7$ db.

Risposta: $SNR \geq 4.7$ dB

Risposta prof:

R: Dal teorema di S-H è $1Mb/s \leq 500kHz \log_2(1+SNR)$ quindi $SNR \geq 3$ ossia $SNR \geq 4.7$ dB

ES di fisica: Sulla linea di trasmissione dell'esercizio precedente, viene utilizzato un Forward Error Coding che codifica 12 bit in pacchetti di 24 bit. Determinare il numero minimo di bit che deve venir codificato da un simbolo per baud, per avere un bitrate netto di 1 Mb/s.

Risposta: Con un bitrate netto di 1 Mb/s e un code rate di $12/24 = 0.5$, il bitrate grezzo deve essere di 2 Mb/s. Dal **teorema di Nyquist** il baud rate è al massimo $500 \text{ kHz} * 2 = 1\text{MBaud}$, e quindi il numero di bit codificati per baud deve essere almeno $2\text{Mb/s} / 1 \text{ MBaud} = 2$.

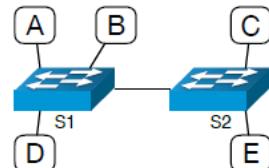
ES: Una certa linea ha una probabilità di errore $p = 10^{-3}$ per ogni bit. Prima della trasmissione ogni 3 bit viene aggiunto un bit di parità. Qual è la probabilità che in un pacchetto di 4 bit così formato avvengano degli errori non rilevati?

Risposta: affinché ci siano errori non rilevati dal bit di parità, gli errori devono essere pari. Possono verificarsi quindi due casi:

1. 2 errori
2. 4 errori

La probabilità che questo accada è: $P = \binom{4}{2}p^2(1-p)^2 + \binom{4}{4}p^4 = 6 * p^2(1-p)^2 + p^4 = 5.99 * 10^{-6}$

ES: Nella rete a lato, gli switch S1 e S2 sono ad autoapprendimento. S2 ha le tabelle completamente popolate, mentre S1 è appena stato resettato. L'host A invia un frame indirizzato a C. (a) A quali host viene recapitato tale frame? (b) Se C risponde ad A, il suo frame a chi viene recapitato?



Risposta:

- (a) Agli host B, C, D perché S1 non sa dove si trova C, di conseguenza il frame viene recapitato a tutti gli host connessi in questa rete e inondato su tutte le reti collegate (in questo caso S2 che sa dov'è C e di conseguenza non invia il frame anche ad E).
(b) Solo ad A (perché S1 ha imparato dove è A).

ES: Un'azienda ha tre reti A, B, C con 100, 300 e 200 postazioni rispettivamente. Per ognuna di queste reti si dia una sottorete (minima) all'interno della rete 192.168.0.0/16.

Risposta: si ha che:

$$2^6 < 100 < 2^7 \text{ quindi: } 32-7 = 25 \text{ quindi serve una sottorete /25, ad esempio: } 192.168.0.0/25$$

$$2^8 < 300 < 2^9 \text{ quindi: } 32-9 = 23 \text{ quindi serve una sottorete /23, ad esempio: } 192.168.1.0/23$$

$$2^7 < 200 < 2^8 \text{ quindi: } 32-8 = 24 \text{ quindi serve una sottorete /24, ad esempio: } 192.168.2.0/24$$

ES: Si dia la tabella di inoltro del router R3 della rete a lato, sapendo che le interfacce dei router hanno i seguenti indirizzi:

$$R1:\text{if1} = 192.168.1.1$$

$$R1:\text{if2} = 192.168.2.1$$

$$R1:\text{if3} = 10.1.1.1$$

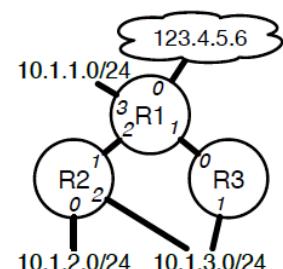
$$R2:\text{if0} = 10.1.2.1$$

$$R2:\text{if1} = 192.168.2.2$$

$$R2:\text{if2} = 10.1.3.1$$

R:	dest	if	next hop
	10.1.2.0/24	if1	10.1.3.1
	10.1.3.0/24	if1	-
	/	if0	192.168.1.1

$$R3:\text{if0} = 192.168.1.2$$



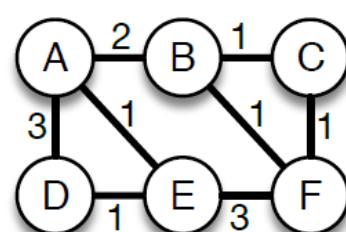
ES: I router in figura impiegano un algoritmo basato sul vettore delle distanze.

(a) Si dia la tabella di instradamento del router A.

(b) Cosa cambia se si interrompe il collegamento A-E?

Dest.	dist	next hop
A	0	-
B	2	B
C	3	B
D	2	E
E	1	E
F	3	B

Dest.	dist	next hop
A	0	-
B	2	B
C	3	B
D	3	D
E	4	D
F	3	B



ES: Diversi servizi utilizzano gruppi multicast per la comunicazione tra processi vicini, ossia in esecuzione su host della stessa sottorete ma che non si conoscono a priori; ad esempio mDNS usa il gruppo 224.0.0.251 per risolvere gli hostname del dominio .local.

(a) Come si può limitare la diffusione dei pacchetti multicast alla sola sottorete locale?

(b) È necessario usare il protocollo IGMP? Perché?

Risposta:

(a) Impostando il TTL=1

(b) no, perché IGMP serve per comunicare il router gateway di quali gruppi siamo interessati, ma in questa situazione i gateway non hanno alcun ruolo.

ES: Syslog è uno standard di logging remoto ampiamente utilizzato, in cui gli host possono inviare messaggi diagnostici ad un server (syslogd) in ascolto su una porta ben nota. Il server memorizza tali messaggi in un file (o li processa in altri modi); non è necessario che dia conferma di ricezione al mittente. Quale protocollo di trasporto può essere usato per implementare tale servizio su rete locale? Perchè?

Risposta: Va bene UDP, se la rete sottostante è abbastanza libera e affidabile. Così è scalabile e non richiede la creazione e mantenimento di una connessione. Se avessimo avuto necessità di acknowledgement, controllo di flusso, controllo di congestione, TCP sarebbe risultato adeguato.

ES: Una socket TCP è appena entrata nello stato ESTABLISHED e, ancora prima che spedisca il primo segmento, riceve un segmento con ACK=1, Length=0. È corretto? Cosa deve fare TCP a fronte di tale segmento?

Risposta: Si è legittimo: probabilmente è un duplicato dell'ultimo segmento che il client ha inviato al server alla fine della fase di handshake. Per essere certo è sufficiente vedere se l'Acknowledge corrisponde al valore fissato durante la fase di handshake. In tal caso il segmento può essere tranquillamente ignorato, e la comunicazione può continuare normalmente.

Teoria a riguardo: la fase di three-way hadshake viene utilizzata per instaurare la connessione tra i due host che desiderano comunicare tramite TCP.

ES: Una socket TCP ha inviato tre segmenti di 1400 byte ciascuno e con SeqNum pari a 1000, 2400 e 3800 rispettivamente. Riceve un segmento con Acknowledgment = 2400, AdvertisedWindow = 6000, e poi uno con Acknowledgment=1000, AdvertisedWindow=5000. A questo punto quanti byte può ancora inviare?

Risposta: Il secondo segmento di ACK è stato consegnato in ritardo rispetto al primo (lo si deduce dal fatto che arriva l'ACK del primo segmento dopo quello del secondo), evidentemente, quindi bisogna fare riferimento al primo (quindi l'advertised window più "nuova" è quella del segmento con ack=2400, in quanto è stato generato dopo, anche se è arrivato prima). In quel momento la finestra era di 6000 byte, ma 2800 sono già stati spediti e non ancora riconosciuti, quindi si possono inviare ancora $6000 - 2800 = 3200$ byte.

ES: Una connessione TCP, che utilizza la ritrasmissione veloce e il recupero veloce, ha attualmente CongestionWindow = 10000. Dopo aver inviato 7 segmenti da 1 MSS (pari a 1000 byte) a partire dal numero di sequenza 0, ha ricevuto i seguenti Acknowledgment: 1000, 3000, 3000, 3000. (a) Quale numero di sequenza ha il prossimo pacchetto da inviare? (b) Quanto diventa CongestionWindow?

Risposta: (a) 3000, per la ritrasmissione veloce. (b) L'incremento per ack è di:

$$\text{MSS} * \text{MSS} / \text{CongestionWindow} = 100 \text{ byte}$$

quindi dopo quattro ack ricevuti CongestionWindow è pari a 10400. A questo punto viene dimezzato per il recupero veloce, e quindi diventa 5200 byte.

ES di sicurezza: In che modo si può rendere confidenziale il traffico tra un client e un server che comunicano via TCP attraverso una rete insicura, nel caso in cui si disponga del codice sorgente del client ma non del server?

Risposta: Purtroppo non si può usare SSL/TLS, perché richiederebbe di intervenire anche sul codice del server che non abbiamo. Quindi non rimane che usare IPsec, o in trasport da host a host o in tunnel attraverso i router di frontiera.

Teoria relativa:

- **SSL:** secure socket layer (sicurezza a livello di trasporto)
 - TLS è una alternativa a SSL
 - IPsec: sicurezza implementata a livello di rete.
-

ES di fisica: V.92 è uno standard per i modem analogici (quelli che si usavano sulle linee telefoniche, prima dell'ADSL) introdotto nel 1999. Determinare il bit rate grezzo di un modem V.92 che opera nella banda tra 0 e 4kHz, con una modulazione in ampiezza di 128 livelli di segnale.

Dati:

- B = ampiezza di banda = 4Hz
- n = ampiezza (numero di segnali distinti) = 128

Risposta:

- sappiamo che con 128 simboli, abbiamo $\log_2(n)$ bit codificati per simbolo: $\log_2(128) = 7$ bit per simbolo
 - il baud rate (fpulse) è pari a $2B = 2 \cdot 4 = 8$ baud (perché fpulse < 2B)
 - da cui si ottiene un bit rate grezzo di $7 \cdot 8 = 56$ kb/sec
-

ES di fisica: Nelle condizioni della domanda precedente, determinare il minimo rapporto segnale/rumore necessario per una trasmissione senza errori con un Forward Error Coding con code rate pari a 0.5.

Risposta:

- con un code rate pari a 0.5 si ha che il numero di bit trasmessi per simbolo (netti) sono $0.5 \cdot 7 = 3.5$
- dal teorema di Shannon-Hartley ($R \leq C = \frac{1}{2} \log_2(1+SNR)$) si ha che:

$$3.5 \leq \frac{1}{2} \log_2(1+SNR) \text{ da cui } SNR \geq 127 \text{ (circa 21dB)}$$

ES: In una certa cella Bluetooth (versione 2), l'accesso al mezzo è a divisione di tempo, con slot di 625 μ s e un bitrate grezzo di 3Mbit/s; il master trasmette negli slot pari, mentre gli slave negli slot dispari. Supponendo che un frame occupi un solo slot, e ricordando che ogni frame ha un'intestazione di 54 bit e un preambolo ("access code") di 72 bit, a quale bitrate massimo netto può trasmettere il nodo master?

Risposta: In uno slot ci stanno $625 * 10^{-6} * 3 * 106 = 1875$ bit, a cui bisogna togliere 54+72 bit di overhead, per cui i bit netti trasmessi sono 1749 bit, in ogni slot. Dato che il master può trasmettere solo negli slot pari, il bitrate complessivo con cui può trasmettere è $1749 = (2 * 625 * 106) = 1,4$ Mbps.

NB: $1\text{Mb} = 10^6$ bit e $1\text{sec} = 10^6 \mu\text{s}$ (microsecondi)

ES: Per ognuna delle seguenti reti, si dica se l'indirizzo 192.168.7.14 vi appartiene o no:

- (a) 128.0.0.0/1; --> SI
- (b) 192.0.0.0/16; --> NO
- (c) 192.160.0.0/12. --> SI

192.168.7.14 in binario è = 11000000. 10101000.00000111.00001110

(a) \1 --> la rete è indicata dal primo bit, i restanti 31 indicano la/le sotto rete/i. Perché l'indirizzo appartenga a questa rete, deve avere il primo bit uguale. Essendo 192 maggiore di 128, avrà sicuramente il primo bit = 1 (128+x). L'indirizzo quindi appartiene alla rete a.

(b) \16 --> i primi 16 bit (i primi 2 byte su 4) indicano la rete, il 3 e 4 byte indicano la/le sotto rete/i. Perché l'indirizzo faccia parte di questa rete, i primi 2 byte devono combaciare: il primo combacia 192=192; il secondo invece no; di conseguenza l'indirizzo non appartiene alla rete b.

(c) \12 --> i primi 12 bit (il primo byte più i primi 4 su 8 bit del secondo) indicano la rete. Partiamo dal primo byte, questo combacia quindi per ora appartiene. Per controllare che appartenga effettivamente alla rete, dobbiamo controllare che i primi 6 bit del secondo byte combacino, convertiamo quindi i due indirizzi in binario (basterebbe convertire solo il secondo byte ai fini dell'esercizio):

indirizzo: 11000000.1010 | 1000.00000111.00001110

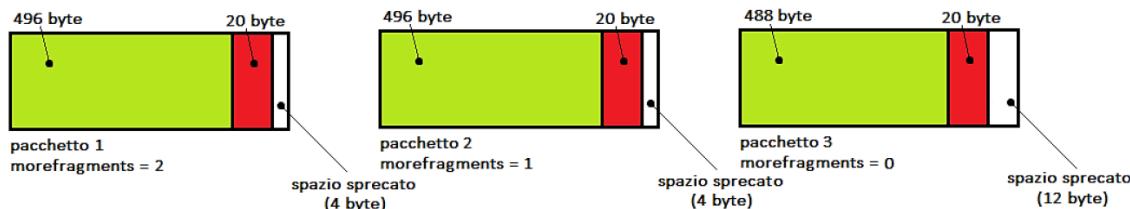
rete c: 11000000.1010 | 0000.00000000.00000000 (in rosso i primi 12 bit indicanti la rete)

devono combaciare i primi 12 bit (quelli a sinistra della |): l'indirizzo appartiene alla rete c.

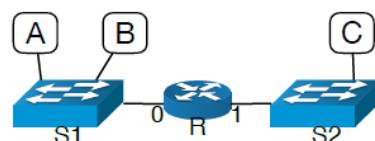
ES: Un router riceve un pacchetto IP di 1500 byte, compresa l'intestazione di 20 byte, e deve inoltrarlo attraverso una interfaccia che ha una MTU di 520 byte. Quanto è lungo il payload del frammento che ha MoreFragments = 0?

- dati netti: 1500-20=1480
- dati trasportabili a 520 byte di MTU = 520-20 (500 di dati netti, ovvero 520 – l'intestazione di ciascun pacchetto)

Risposta: i primi due frammenti portano dei payload di 496 byte, perché 496 è il più grande numero inferiore a 520-20 = 500 e divisibile per 8 (perché l'offset va diviso per 8). Quindi il terzo ed ultimo frammento deve avere $1480 - (496 \times 2) = 488$

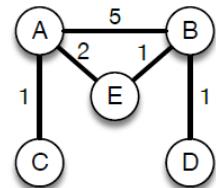


ES: Nella rete a destra, gli indirizzi assegnati alle varie interfacce sono i seguenti: R0: 192.168.1.1; R1: 192.168.2.1; A: 192.168.1.50; B: 192.168.1.200; C: 192.168.2.42. (a) Quale può essere il CIDR della rete di A e B? (b) Se A deve inviare un pacchetto IP a C, per quale indirizzo IP deve trovare il corrispondente MAC address, con ARP?



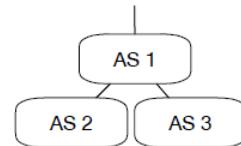
Risposta: (a) \24 (o inferiori) perché gli IP di A e B combaciano per il primi 24 (b) Quello dell'interfaccia del router sulla sua rete, ossia 192.168.1.1.

ES: I router a lato impiegano OSPF, un algoritmo di instradamento basato sullo stato dei collegamenti. Ad un certo punto il collegamento tra A e C viene interrotto, e poco dopo ripristinato. (a) È possibile che i pacchetti LSP generati da A vengano consegnati a D fuori ordine? (b) In tal caso, come fa B a capire quale è il pacchetto da utilizzare?



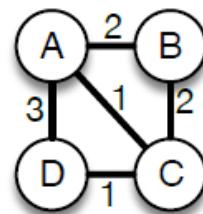
Risposta: (a) Sì, perché OSPF diffonde i pacchetti in flooding appoggiandosi direttamente a IP, che si sa che non è affidabile; (b) Ogni pacchetto contiene l'identificatore di chi l'ha generato (A, in questo caso) e un numero progressivo. Il router B tiene traccia del più recente pacchetto LSP generato da A, cosicché se arriva un pacchetto (originato da A) con un numero più basso, viene scartato.

ES: Gli autonomous system AS 2 e AS 3, le cui reti sono rispettivamente 130.7.1.0/24 e 130.7.2.0/24, sono connessi ad Internet attraverso un provider AS 1, il quale al suo interno ha anche le reti 130.7.0.0/24. (a) Che tipo di autonomous system è AS 1? (b) Quali reti vengono pubblicizzate dallo speaker di AS 1, al resto dell'Internet?



Risposta: (a) AS1 è di tipo "di transito" in quanto ha connessioni con più AS (2 e 3) facenti parte della sua rete, di conseguenza trasporta traffico indirizzato ad essi. (b): AS1 pubblicizza 130.7.0.0/23 che ingloba la rete AS1 e 130.7.0.0; pubblicizza anche 130.7.2.0/24 (non appartenente a 130.7.0.0/23).

ES: I router della rete a lato usano DVMRP per instradare il traffico multicast. Sia A il core router di un certo gruppo G. (a) Quando viene inviato un pacchetto IP a tale gruppo, quanti pacchetti arrivano a C, e da quali router adiacenti? (b) Quale di questi viene inoltrato in flooding?



Analizziamo il susseguirsi degli eventi:

- la source S (ovvero A) invia il pacchetto a B, C e D.
 - B cosa fa? inoltra il pacchetto (verso tutti escluso la source, quindi a C) in quanto il pacchetto è arrivato dal next-hop verso S (S = A stesso).
 - D cosa fa? diversamente da B, il pacchetto non è arrivato dal next-hop verso S (in quanto la strada più breve sarebbe passare per C che è appunto il next-hop); quindi non lo inoltra.
 - C cosa fa? C riceve un pacchetto da B e da A; ma inoltra solo quello di A che arriva da un percorso più breve verso la source S (A stessa).

Risposta: (a) 2, da A e B. (b) quello che arriva da A.

ES: Una applicazione sta producendo un flusso di dati su una socket TCP, alla velocità di 100 byte ogni 10 ms. La connessione TCP ha un MSS di 1460 byte e un RTT di 80ms. Si supponga che CongestionWindow e AdvertisedWindow siano abbondantemente grandi. (a) Quanto è grande il payload di ogni segmento inviato dall'host? (b) In percentuale, quanto è l'overhead introdotto dalle intestazioni IP e TCP?

Teoria relativa:

- header di IPv4 = 20byte in assenza di campi opzionali;
- header di TCP = 20 byte in assenza di campi opzionali;
- RTT: round trip time;

Risposta: (a) In un RTT il buffer accumula $100 * 80 = 800$ byte, inferiore al MSS. Quindi, per l'algoritmo di Nagle, ogni volta che arriva un ACK (nel nostro caso appunto ogni 80ms) si invia un segmento di 800 byte. (b) L'overhead delle due intestazioni è di 40 byte, quindi $40 = (40 + 800) = 4,76\%$.

ES: Un'applicazione A ha scritto 8KB su una socket TCP il cui buffer di output è di 4KB. L'applicazione controparte B ha consumato 3KB, e il buffer di input della sua socket è di 2KB. Cosa succede all'applicazione A se prova ad eseguire tre scritture da 1KB l'una? (Si supponga che nel frattempo B non consumi altri dati).

Risposta: La differenza tra i dati scritti sulla socket e dati consumati non può mai superare la somma dei buffer in gioco, perché altrimenti i dati andrebbero persi. Quindi, lo spazio ancora disponibile per le scritture di A è uguale a: $(4 + 2) - (8 - 3) = 1\text{KB}$. Per cui, la prima scrittura da 1KB ha successo, ma la seconda si blocca in attesa che si liberi spazio nel buffer.

ES: Un router applica la strategia RED alla coda di una linea che ha una velocità di 3 MB/s, mentre i pacchetti che devono essere accodati arrivano con un datarate medio di 10 MB/s. Sapendo che MinThreshold = 50kB, MaxThreshold = 100kB, MaxP = 1, quant'è la lunghezza media della coda?

Teoria: RED --> randomly early detection: ogni router viene programmato perché tenga sotto controllo la lunghezza delle proprie code e, quando si accorge che una congestione è imminente, segnali alla sorgente di modificare la propria finestra di congestione.

Risposta: La lunghezza della coda si stabilizza quando i dati che vengono accodati equivalgono a quelli che escono dalla coda. Quindi la probabilità p di accodamento deve essere tale che $10p = 3$, e quindi $p = 0,3$. Pertanto, la probabilità di eliminazione per ogni pacchetto che arriva è $q = 1 - p = 0,7$. Ricordando che

$$q = (\text{AvgLen} - \text{MinThreshold}) / (\text{MaxThreshold} - \text{MinThreshold})$$

dobbiamo risolvere l'equazione:

$$0,7 = (\text{AvgLen} - 50) / (100 - 50)$$

che ci porta a $\text{AvgLen} = 85\text{kB}$.

ES sicurezza: Dal punto di vista della quantità di dati messi in sicurezza end-to-end, è meglio implementare i servizi di cifratura e/o autenticità dei dati a livello trasporto, a livello di rete, o a livello datalink?

Risposta: A livello trasporto o meglio ancora rete. A livello datalink i dati rimangono scoperti nei router.

ES fisica: Intel HEX è un modo per rappresentare dati binari in un formato testuale, in cui ogni byte è rappresentato da un coppia di caratteri esadecimali ASCII. La struttura dati è organizzata in linee delimitate da un carattere ASCII ":" all'inizio e da LF (linefeed – vale 1 ascii) alla fine; ogni linea contiene un header di 4 byte, un blocco dati (payload) di lunghezza specificata nell'header, e un byte di checksum del blocco dati. Ad esempio, questa è una linea con un blocco dati di 16 byte:

header	blocco dati	cs	(LF)
:10010000	214601360121470136007EFE09D2190140		

Si determini il bit rate netto del payload se si trasmettono in questo formato, con blocchi dati di 32 byte, su un canale fisico di baud rate 1kBaud/s, nel quale ogni simbolo codifica 1 carattere ASCII utile.

Risposta: Ogni linea consta di $1+4*2+32*2+1*2+1 = 76$ caratteri e codifica $32*8 = 256$ bit. Poiché il numero di caratteri trasmessi per secondo è 1000, vengono trasmesse $1000/76 = 13,2$ linee al secondo e quindi con un bit rate netto di $256*1000/76 * 1/1000 = 3.4$ kbit/s.

ES: Se nella domanda precedente un carattere ASCII trasmesso è codificato con 7 bit, determinare il rapporto S/N minimo necessario per una trasmissione senza errore.

Risposta: Dal teorema SH si ha: $\log_2(M) = 1/2 \log_2(1 + SNR)$ ove $M = 27$ e quindi si ha in condizioni ideali $SNR = 42$ dB o $SNR = 16383$.

ES: Nella codifica 4B/5B, i bit 0100 vengono rappresentati come 01010. La codifica della sequenza 01000100 viene trasmessa su un canale che, a causa di un disturbo, inverte uno dei bit (ma non si sa quale). Qual è la probabilità che tale errore venga rilevato dal decodificatore?

Risposta: La sequenza codificata è 0101001010. La regola di buona formazione della codifica 4B/5B richiede che non ci siano mai più di tre 0 consecutivi; quindi un errore per essere rilevato deve portare almeno 4 zeri consecutivi. Questo succede solo per i bit 4 e 7, quindi 2 possibilità su 10, quindi la probabilità è 20%.

ES: Due stazioni 802.11n hanno negoziato un bitrate grezzo di 240 Mbps. Sapendo che SIFS = 9 μ s e DIFS = 34 μ s, il frame è di 20 byte e ACK e CTS sono di 14 byte, il frame dati ha una intestazione di 30 byte e un CRC-32 di 4 byte, si dica quale bitrate netto massimo si può ottenere con payload di 1500 byte (trascurando i tempi per la contesa del canale).

Risposta: Per trasmettere un frame di dati servono 1 DIFS e 3 SIFS, più un RTS, un CTS e un ACK alla fine. I byte trasmessi complessivamente sono $20 + 14 + 30 + 1500 + 4 + 14 = 1582$ byte, che impiegano $1582 * 8/240 = 52.73\mu$ s. A questo bisogna aggiungere $34 + 9 * 3 = 61\mu$ s, per un totale di 113.73μ s. Quindi bitrate netto è quindi $1500 * 8 / 113.73 = 105.51$ Mbps.

ES: Tre host, collegati alla stessa LAN, hanno indirizzi IP A = 12.10.0.7, B = 12.70.7.7, e C = 12.110.7.23.

- (a) Se il CIDR è /10, a quali reti appartengono?
- (b) Quali host possono comunicare direttamente senza passare per un router (a meno di configurazioni ad hoc)?

Risposta:

(a) bisogna guardare i primi 10 bit dell'indirizzo IP per capire quali si riferiscono alla rete: ogni blocco dell'ip vale 8bit, di conseguenza sappiamo che per tutti e 3 gli indirizzi IP i blocchi formanti il 12 sono la prima parte dell'indirizzo di rete; i restanti 2 bit vanno presi dai primi 2 su 8 del secondo blocco; per fare ciò dobbiamo convertire in binario i relativi secondi blocchi (secondi byte): si ha che per:

a --> 10 --> .00001010
 b --> 70 --> .01000110
 c --> 110 --> .01101110

quindi bisogna prendere i primi due bit, e i restanti 6 pari a 0; otteniamo:

a --> .00000000 che in decimale è pari a 0; b,c --> .01000000 che in decimale è pari a 64. Il terzo e quarto byte, sono pari a 0; quindi gli indirizzi delle reti sono: 12.0.0.0/10 per A e 12.64.0.0/10 per B e C.

- (b) B e C si trovano sulla stessa rete e possono comunicare direttamente.
-

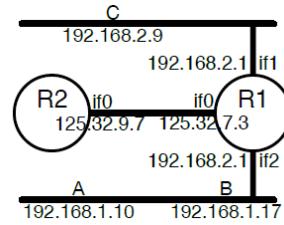
ES: Un'azienda ha bisogno di 300 indirizzi IP. Se gli viene assegnata una singola rete, qual è l'utilizzo percentuale di tale spazio di indirizzamento? E se invece le vengono assegnate tre reti più piccole?

Risposta: La più piccola rete che contiene 300 indirizzi è una /23, ossia 9 bit di host, per totali $512 - 2 = 510$ indirizzi utili. La percentuale di utilizzo è $300/510 = 58,8\%$. Se invece possiamo usare tre reti /25 da $128 - 2 = 126$ indirizzi l'una, abbiamo $300/(3 * 126) = 79,4\%$. Si può fare anche di meglio, con due reti /25 e una /26, per totali 320 indirizzi e un'efficienza del 95%.

Teoria su quel -2: un indirizzo è quello di rete e non può essere usato, l'altro è quello broadcast.

ES: Nella rete a lato, il router R2 è quello di frontiera, ed è connesso al resto dell'Internet. L'host A deve inviare un pacchetto IP all'indirizzo 123.45.67.89. (a) Quale indirizzo deve risolvere con il protocollo ARP? (b) Chi gli risponderà, e con quale informazione?

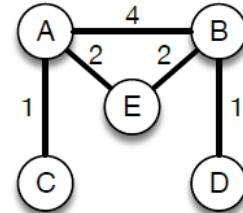
Risposta: (a): L'indirizzo dell'if2 di R1 (192.168.2.1) in quanto suo next hop verso 125.32.9.7 (b): Gli risponderà R1 con il MAC address della sua interfaccia if2.



ES: I router a lato utilizzano un algoritmo di instradamento basato sullo stato delle linee. Si dia la tabella di instradamento di B (a) a regime, e (b) dopo che viene aggiunto un collegamento tra C e D di peso 1.

Dest.	dist	next hop	Dest.	dist	next hop
A	4	A	A	3	D
B	0	-	B	0	-
C	5	A	C	2	D
D	1	D	D	1	D
E	2	E	E	2	E

Risposta: (a)



ES: (a) Perché nel routing interdominio non si utilizza un algoritmo basato sul vettore delle distanze? (b) Supponiamo che AS1 sia connesso ad AS2 stub e ad AS3 che è di transito. Che tipo di AS è AS1?

Risposta: (a): È impossibile definire delle metriche per il calcolo delle distanze congruo con tutti i metodi di instradamento e tecnologie utilizzati dalle varie reti, inoltre è costoso ed è difficile raggiungere la stabilità. (b): di transito.

ES: IPv6 offre varie funzionalità avanzate, come il supporto per il real-time e il Quality of Service. (a) A questo scopo, quale informazione è presente nell'intestazione del pacchetto? (b) Cosa succede a tale informazione se IPv6 viene incapsulato in un tunnel IPv4?

(a) Il TrafficClass per la priorità e la FlowLabel per identificare i flussi all'interno dei router. (b) non è più utilizzabile dai router IPv4, che quindi non possono implementare il servizi real-time richiesti.

ES: Lo strato TCP di un host A ha appena inviato un segmento con SYN=1, ACK=0, SeqNum = 31541 ad un altro host B. Poi riceve da B un segmento con SYN=1, ACK=1, SeqNum = 53628, Acknowledgement = 23143. (a) È corretto? Perchè? (b) Cosa deve fare A?

Risposta: (a) no, non è corretto, perché Acknowledgement dovrebbe essere 31542 (SeqNum+1). Probabilmente è un vecchio segmento appartenente alla fase di handshake di una connessione precedente. (b) Scartare il segmento appena ricevuto, e aspettare (fino al timeout) quello con l'Acknowledgement corretto.

ES di sicurezza: Una rete è realizzata con degli switch Ethernet e router a 1Gbps. (a) Supponendo che una connessione TCP tra due host riesca a sfruttare il 50% del bitrate grezzo, qual è il tempo di wraparound? (b) Cosa potrebbe succedere se un pacchetto IP viene consegnato (o reinviato da un router o un attaccante) con un ritardo maggiore di tale tempo?

Risposta: (a) Il bitrate netto è 500 Mbps, ossia 62,5 MB/s. A questa velocità, il wraparound avviene dopo $232 = (62.5 * 10^6) = 68.72$ secondi. (b) Potrebbe essere scambiato per buono durante un nuovo ciclo del contatore di sequenza. Ossia, un segmento portante dati vecchi verrebbe preso al posto di quelli nuovi, se arriva quando la finestra copre proprio il suo numero di sequenza.

ES: Un router inizia a servire due flussi secondo la politica di accodamento equo (Fair Queueing); il flusso A ha in coda 2 pacchetti di lunghezza A1=100, A2=300; il flusso B ha in coda 3 pacchetti di lunghezza B1=200, B2=100; B3=300. Inoltre, all'istante 800 arriva un pacchetto A3=100. In che ordine vengono trasmessi i sei pacchetti?

Risposta: per ciascun flusso, la trasmissione sequenziale porterebbe i seguenti pacchetti a venir trasmessi agli istanti:

$$A1:100, A2: (100+300) = 4$$

$$B1:200, B2: (100+200) = 300, B3:(300+300) = 600$$

Il susseguirsi degli eventi sarà il seguente: viene trasmesso A1, B1, A2, B2 e siamo all'istante 700. Inizia la trasmissione di B3 (che dura 300) ma dopo 100 arriva A3 (da 100); non essendoci prelazione, finisce la trasmissione di B3 e poi inizia quella di A3: la sequenza è quindi A1, B1, A2, B2, B3, A3.

ES: Un server TCP/IP, prima di mettersi in ascolto passivo, deve associare una socket ad un certo indirizzo. Quali informazioni sono contenute in tale indirizzo? Quale invece non è necessario specificare?

Risposta: Ci sono l'indirizzo di livello 4 (porta) e l'indirizzo di livello 3 (indirizzo IP). Non è necessario mettere l'indirizzo di livello 2 (MAC address) perché è gestito dal livello 3.

ES: Un server TCP/IP, prima di mettersi in ascolto passivo, deve associare una socket ad un certo indirizzo. Quali informazioni sono contenute in tale indirizzo? Quale invece non è necessario specificare?

Risposta: ci sono l'indirizzo di 4 livello (porta) e l'indirizzo di 3 livello (indirizzo IP). Non è necessario l'indirizzo di livello 3 perché è gestito dal livello più basso (MAC address).