

Reti di Calcolatori

Sommario

Stack Protocollare	9
Differenza Stack TCP-IP e ISO-OSI.....	9
Incapsulamento	9
Physical Layer	10
Nozioni base	10
Attenuazione e rumore	10
Modulazione	10
Teoremi fondamentali.....	11
Teorema di Nyquist	11
Teorema di Shannon	11
Datalink Layer	12
Time-Sequence Diagrams	12
Manchester encoding.....	12
Framing and Bit Stuffing.....	12
Bit Stuffing.....	12
Framing and Layers.....	13
Acknowledging Frames.....	14
Service Data Units (SDU)	14
Acknowledgment.....	14
Automa a stati finiti di Sender e Receiver.....	14
Affrontare gli errori	14
Bit di parità	14
IP Checksum	15
Trasporto affidabile	16
Alternating Bit Protocol (ABP)	16
ABP Performance.....	17
Go-back-n e Selective Repeat	17
Sliding Window	17
Go-Back-n.....	17
Limiti del Go-back-n	19
Selective Repeat.....	19
Considerazioni.....	19
Massima dimensione della finestra (Selective repeat e go-back-n)	20
Datalink Layer - Sharing Resources	21
Network Topologies	21
Bottleneck	22
Frequency division	22
Time-Division Multiple Access (TDMA)	22
Random Access	23

S-Aloha Protocol (slotted version)	24
Assunzioni matematiche	24
Dettagli del protocollo ALOHA.....	24
Evoluzione	24
Quando le reti con TDMA e ALOHA collassano?.....	24
Conclusioni.....	24
Network Layer	26
Datagram-oriented Organization	26
Forwarding Table.....	26
Indirizzamento ed eterogeneità	27
Flat Addressing.....	27
Hierarchical Addresses	27
Fragmentation	27
Routing Algorithms	28
Hot Potato.....	28
The Control Plane	28
Routing VS Forwarding Table	28
Distance Vector Routing	29
Generazione dei DV.....	29
Failure Recovery	31
Count-to-Infinity and Poison Reverse	33
Count-To-Infinity problem	33
Split-Horizon with Poison Reverse.....	33
Generazione dei DV (variante della precedente)	33
Timer Management	34
Conclusioni sul count-to-infinity.....	34
Link State Routing.....	35
Funzionamento	35
LSP Messages.....	35
LSP Flooding.....	35
LS Scalabilità	36
Overhead	36
Conclusioni.....	36
Transport Layer	38
Connectionless Transport Layer	38
Porte	38
Connection-Oriented Transport Layer	38
Transport Clock e ISN.....	39
Three Way Handshake.....	39
Reliable Data Transfer	40

Sliding Window size	40
MSL e Throughput	40
Application Layer	42
Header del livello applicazione	42
Sicurezza	43
HMAC	43
CRAM-MD5.....	44
Symmetric Key Encryption	44
Operazioni:.....	44
OTP	44
Feistel Scheme	45
Perché si chiama Symmetric Key Encryption?.....	45
Password Files	45
Public Key Encryption	45
Diffie-Hellman Protocol	45
RSA	46
Firma Digitale	47
Conclusioni	47
Distributing Public Keys QUI	47
Keyserver.....	47
Web Of Trust.....	47
Domain Name System (DNS)	48
Dominio	48
Accedere a un dominio	48
DNS Caching.....	48
DNS dettagli.....	49
Formato del messaggio	49
Header pacchetto	49
Reverse DNS	49
WHOIS protocol	50
Sicurezza	50
Electronic Mail (e-mail).....	51
Protocolli necessari.....	51
Formato delle email.....	51
Protocollo MIME	51
Protocollo SMTP	52
Operazioni.....	52
Autenticazione.....	53
Protocollo POP	53
Combattere lo SPAMs	53
IP blacklisting	53
FQDN check	54

RFC Compliance.....	54
Sender Policy Framework (SPF)	54
DKIM	55
Esempio SMTP.....	55
World Wide Web (WWW).....	56
Universal Resources Identifier (URI).....	56
HyperText Markup Language (HTML).....	56
Pagina HTML.....	56
Pagine dinamiche HTML.....	56
Tabelle di stile CSS.....	56
Protocollo HTTP.....	57
Metodi.....	57
HTTP header	57
HTTP Status Code	57
HTTP Connections	57
HTTP Cookies	58
Proxy	58
Proxy Server.....	58
HTTP 2.0.....	58
User Datagram Protocol (UDP)	59
UDP Header	59
Utilizzo di UDP.....	59
Transmission Control Protocol (TCP)	59
Header di un segmento TCP	59
Connessione TCP	60
Connessione rifiutata TCP	60
Macchina a stati per connessione TCP stabilita	60
Chiusura connessione TCP	61
Graceful Release	61
Release brutale.....	61
Nota sullo stato Time Wait.....	61
Trasferimenti affidabili TCP	61
TCB	61
Sending Data	62
Receiving Data.....	62
Miglioramenti dell'RFC originaria di TCP	63
TCP Scale Window Option	63
Miglioramento stima RTT: Algoritmo di Van Jacobson	64
Cosa fare quando RTO scade?	64
Delayed ACKs.....	64
Performance TCP	64
Congestion window	65
Algoritmo Slow Start.....	66

Socket	68
Transport Layer Security (TLS) qui	69
Possiamo cifrare l'header IP (in su)?	69
Modi di cifrare l'header IP	69
Possiamo cifrare l'header TCP (in su)?.....	69
Transport Layer Security (TLS).....	69
SSL / TLS	70
's' protocols	70
Servizi di TLS	70
TLS 1.2	70
Record protocol TLS.....	70
Handshake Protocol TLS	71
Protocollo IPv4	73
Indirizzo di Broadcast	73
Indirizzo di Multicast.....	73
Indirizzi speciali	73
Routers	74
Local Routes Vs Gateway	74
Header IPv4.....	74
Fragmentation.....	75
Indirizzi IP Privati	76
Network Address Translation	76
Forwarding Table	77
Esempio IP Address 1	77
Esempio IP Address 2	77
Internet Control Message Protocol (ICMP)	79
Header ICMP	79
Datagramma ICMP	79
Protocollo IPv6	80
Formato indirizzo IPv6	80
Disponibilità	80
Assegnamento indirizzo Ipv6.....	80
Indirizzi speciali	81
Header IPv6.....	81
IPsec.....	81
Fragmentation.....	81

ICMPv6	81
Intradomain Routing	82
Open Shortest Path First (OSPF)	82
OSPF Areas	82
Backbone Area	82
Comportamento Link State	82
Comportamento DV	82
Autonomous System	83
Border Gateway Protocol (BGP)	84
Esportare i prefissi	84
BGP Routers	84
Split Horizon	84
BGP è un Path Vector Protocol.....	84
Connessione BGP	84
BGP Path Prepending	85
BGP Anycast for Root Servers	86
IEEE 802.3 - Ethernet qui	87
MAC Address.....	87
Header frame ethernet	87
Ethernet Switches	87
Backward Learning	88
Switching loops.....	88
The Spanning Tree Protocol (STP).....	89
Port State	89
Operazioni semplificate	89
The Address Resolution Protocol (ARP).....	91
Local Destination IP.....	91
A remote IP	91
ARP.....	91
Header ARP	91
Dynamic Host Configuration Protocol (DHCP).....	92
DHCP Discover.....	92
DHCP Offer	92
DHCP Request	92
DHCP Acknowledgment	92
IEEE 802.11 - Wi-Fi	93
Livello Fisico	93
Livello MAC	93
Tipi di traffico	93
Definizioni	94
Entrare in una rete.....	94

Macchina di stato	94
MAC Header Wi-Fi.....	95
Controllo degli accessi	95
Efficienza del MAC	95
DCF	96
DCF: Carrier Sensing e Static Reservation	96
Esempio di MAC semplice: trasmettitore singolo	96
Esempio di MAC semplice: due trasmettitori	96
Esempio di MAC semplice: tre trasmettitori.....	97
Meccanismo RTS / CTS	99
Stima del Bitrate.....	99
Nyquist.....	99
Shannon.....	99
Capacità Massima	99
Modulating Coding Scheme (MCS)	99
Clear Channel Assignment (CCS)	100
Condivisione Canale	100

Stack Protocollare

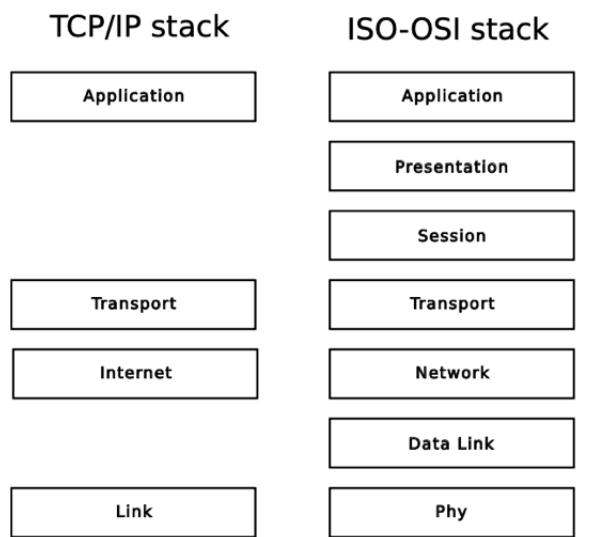
Differenza Stack TCP-IP e ISO-OSI

Nello stack TCP-IP:

- *Livello applicazione*: descrive il comportamento delle applicazioni.
- *Livello di trasporto*: rende le trasmissioni affidabili e multiplexa le connessioni.
- *Livello Internet*: rende interoperabili diverse reti fisiche, fornisce indirizzi e routing;
- *Livello link*: prescrive come dovrebbe funzionare la comunicazione fisica (wireless, cablato, accesso ai media...)

Nello stack ISO-OSI vengono specificati 7 livelli:

- *Collegamento dati*: come più terminali dovrebbero accedere allo stesso supporto fisico
- *Livello di sessione*: come mantenere le sessioni utente
- *Livello di presentazione*: come mostrare/visualizzare i dati



Incapsulamento

Ogni livello incapsula i dati ricevuti dal livello superiore:

- Il livello di trasporto riceve i dati grezzi da un'applicazione, li divide in *segmenti*, aggiunge l'intestazione TCP e li sposta verso il basso su IP.
- Il livello IP prende il segmento e aggiunge un'altra intestazione, l'intestazione IP. Questo diventa un *pacchetto* con due intestazioni.
- L'IP lo sposta verso il basso al livello di collegamento, che aggiunge un'altra intestazione, e poi lo trasmette. Noi la chiamiamo una cornice.

Quando il pacchetto viene ricevuto, viene seguito il processo opposto. Ogni livello legge la propria intestazione e passa il resto al livello superiore.

Physical Layer

Nozioni base

Bit-rate: velocità di comunicazione espressa in bit per secondo e i suoi multipli.

Sensibilità: la quantità minima di potenza che il sensore deve ricevere per rilevare la presenza della corrente.

Larghezza di banda: l'intervallo tra più bassa e la più alta frequenza che usi per le comunicazioni.

Attenuazione e rumore

Attenuazione: parte della potenza che il generatore introduce nel circuito viene persa, a causa della resistenza del circuito stesso. L'attenuazione varia in base al tipo di cavo alla sua lunghezza e alla frequenza di comunicazione. Più il cavo è lungo più effetti negativi ci possono essere

Ritardo (latenza del collegamento): simboleggia il tempo passato da quando il segnale è stato inviato a quando il ricevente lo ha ricevuto.

Rumore: non c'è mai corrente zero nel circuito, anche da spento ce ne sarà sempre un po', nel mondo reale ogni comunicazione è composta dal segnale sommato a un po' di rumore.

SNR (Signal Noise Ratio): rapporto segnale/rumore. La capacità di decodificare i dati dipende da tale rapporto, in quanto se la potenza ricevuta è inferiore alla sensibilità del ricevitore esso non riceverà nulla. Più è alto il valore di tale rapporto migliore sarà il segnale.

Sincronizzazione: Il destinatario e il mittente devono concordare quando un certo simbolo inizia e quando finisce, devono concordare la durata di ogni simbolo, che è generalmente una costante.

Modulazione

In un segnale modulato abbiamo:

- **Segnale Portante:** una sinusoida con una frequenza espressa in Hertz.
- **Segnale Modulante:** il segnale che vogliamo trasmettere.

L'idea chiave della modulazione è quella di influenzare **l'ampiezza**, **la frequenza** o **la fase** della sinusoida portante per codificare le informazioni che rappresentano il segnale modulante e ottenere un segnale modulato.

Modulazione di Frequenza

Possiamo modificare la frequenza della portante e incrementarla.

Questo permette di comunicare un segnale **binario** digitale tramite una portante (analogica).

Qual è il **bit-rate** di questa comunicazione?

- Assumiamo che il numero di oscillazioni in un secondo sia 4
- Nella figura la lunghezza di un bit è determinata da 4 oscillazioni, questo si chiama *simbolo*.
- Quindi, abbiamo un simbolo per secondo, ogni simbolo rappresenta un bit, dunque il bit-rate è 1 b/s.

Come **incrementare** il bit-rate?

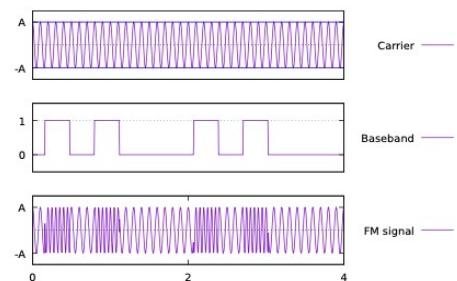
- incrementare la frequenza
- ridurre il Symbol Time
- codificare più di un bit per ogni simbolo

Tutte queste tecniche **riducono** il SNR (Signal Noise Ratio).

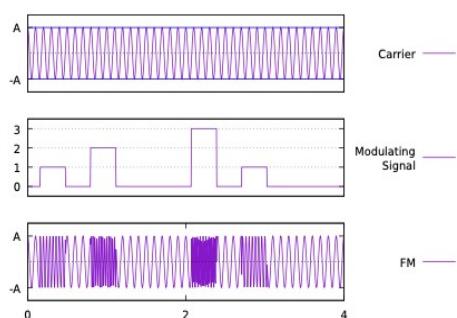
Idealmente possiamo codificare un numero di bit arbitrario, sempre se il ricevitore è abbastanza abile a riconoscere la differenza tra le sequenze che mandiamo.

Per incrementare il bit-rate dobbiamo incrementare la larghezza di banda.

FM



FM encoding with M=4 (2 bits per symbol)



Teoremi fondamentali

Teorema di Nyquist

Il teorema di Nyquist dice che se la tua comunicazione utilizza una larghezza di banda B , puoi trasmettere B simboli contemporaneamente, su frequenze ortogonali.

Se si usano M (generalmente potenza di 2) livelli discreti allora la **capacità massima teorica della trasmissione** C è:

$$C = 2B\log_2(M)$$

Il bit-rate reale è sempre minore a quello calcolato con tale teorema perché non tiene conto del rumore.

Esempio 1:

Un router wireless utilizza frequenze che vanno da 2401 MHz a 2441 MHz. Supponiamo che codifichi 2 bit per ogni simbolo. Qual è il bit-rate massimo?

$$\begin{aligned} B &= 2.441.000.000 - 2.401.000.000 = 40.000.000 \\ B &= C = 2B\log_2(4) = 2 * 40.000.000 * 2 = 160 \text{ Mb/s} \end{aligned}$$

Esempio 2:

Un collegamento utilizza frequenze che vanno da 2401 MHz a 2441 MHz. Il link deve supportare 300 Mb/s. Qual è il numero minimo richiesto di livelli M ?

$$\begin{aligned} B &= 2.441.000.000 - 2.401.000.000 = 40.000.000 \\ C = B &= C = 2B\log_2(M) = 2 * 40.000.000 * \log_2(M) > 300.000.000 \\ &\rightarrow M > \frac{300}{2^{80}} > 13.45 \end{aligned}$$

M deve essere una potenza del 2, quindi $M = 16$, Quindi per avere 300Mb/s, fornirai effettivamente 320 Mb/s.

Teorema di Shannon

Il teorema di Shannon introduce un limite rigido alla capacità (bit/s) di un segnale con alcune ipotesi sul tipo di rumore.

La capacità di un canale di larghezza di banda B influenzato da un rumore gaussiano bianco è data da:

$$C = B\log_2\left(1 + \frac{S}{N}\right)$$

Non ha senso trasmettere a una velocità maggiore del limite dettato da Shannon,

Infatti, il SNR definisce il limite per il quale non possiamo avere M come numero arbitrario, questo perché se trasmettiamo a un bit-rate superiore rispetto al limite di Shannon il traffico avrebbe degli errori *distribuiti randomicamente*.

Esempio:

Una comunicazione ADSL cablata può essere descritta da questi numeri:

- Potenza di trasmissione del router: 100mW ($mW = 10^{-3}W$)
- Rumore: 0,0001 mW
- Attenuazione: potenza divisa per 25 ad ogni km
- Larghezza di banda: 2,2MHz

Qual è il bitrate massimo raggiungibile a 2 km considerando il rumore?

- Dopo 2 km la potenza del segnale si riduce a $\frac{100}{25*25} = 0.16mW$
- $SNR = \frac{0.16}{0.0001} = 1600$
- Applicando Shannon: $C_S = 2.200.000\log_2(1 + 1600) \cong 23Mb/s$

Quale è la migliore modulazione che possiamo usare (massimo valore di M)?

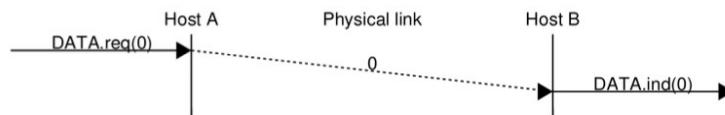
- $2B\log_2(M) \leq C_S$
- $\frac{C_S}{2B} = \log_2(M) \rightarrow 2^{\frac{C_S}{2B}} = M \rightarrow M = 2^{\frac{23.000.000}{4.400.000}} = 2^{5.227} \cong 32$
- $M = 2^5 \rightarrow C_N = 2B\log_2(M) = 2 * 2.200.000 * 5 \cong 22Mb/s \leq C_S$
- $M = 2^6 \rightarrow C_N = 2B\log_2(M) = 2 * 2.200.000 * 6 \cong \frac{26Mb}{s} > C_S$
 $\rightarrow 3Mb/s$ superflui che contengono errori

Datalink Layer

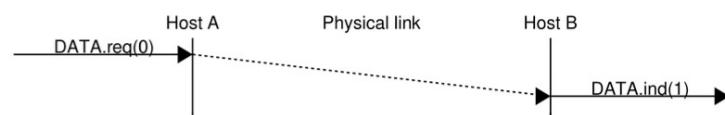
Time-Sequence Diagrams

Describe l'interazione della comunicazione tra i host.

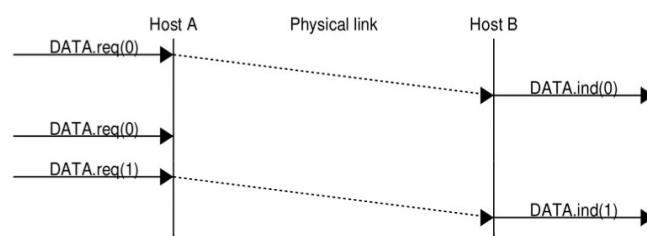
Il tempo passa dall'alto verso il basso. La trasmissione non è istantanea. Descriviamo l'invio di un bit come una chiamata di sistema di alto livello *Data.req(bit)* mentre descriviamo la ricezione di un bit come chiamata a sistema di alto livello *DATA.ind(bit)*.



I bit, a causa del rumore possono essere invertiti e di conseguenza saranno diversi rispetto a quelli inviati.



Il ricevitore può recepire un numero inferiore/superiore di bit rispetto al mittente a causa del disallineamento del clock.



Manchester encoding

Allineare i clock può essere difficile, invece di utilizzare un encoding intuitivo (basso \Rightarrow 0, alto \Rightarrow 1) è più conveniente flippare sempre da basso a alto a ogni simbolo. Se ad ogni periodo il livello del segnale cambia, il ricevitore può risincronizzare il clock con il mittente.

Framing and Bit Stuffing

Frame: sequenza di bit con la massima lunghezza ed una particolare sintassi o struttura.

Bit Stuffing

Idealmente, il supporto fisico smette semplicemente di inviare dati quando il frame è finito.

Tuttavia, abbiamo visto che modulare un segnale significa che il segnale può essere sempre attivo, quindi non possiamo semplicemente smettere di inviare segnali.

Dobbiamo codificare l'inizio e la fine di un frame.

Encoding:

- Inizialmente mittente trasmette il **marker**, cioè 01111110.
- Poi invia tutti i bit del frame
- Inserisce un ulteriore 0 bit dopo ogni sequenza di 11111 (5 uni) per assicurarsi che il frame di dati *non contenga mai il marker*.
- Infine, viene inviato il marker per contrassegnare la fine della comunicazione.

Decoding

Il ricevitore esegue l'opposto per decodificare un frame ricevuto:

- Rileva prima l'inizio del frame grazie al marker 01111110.
- Quindi, elabora i bit ricevuti e conta il numero di bit consecutivi impostati su 1. Se uno 0 segue cinque bit consecutivi impostati su 1, il bit successivo viene rimosso. Deve essere uno zero inserito dal mittente.
- Se il marcatore viene rilevato, il frame è finito.

Original frame	Transmitted frame
0001001001001001001000011	01111110000100100100100100001101111110
011011111111111111110010	0111111001101111101111101111101100100111110
0111110	0111111001111100111110
0111110	011111100111110100111110

Protocol overhead: Dati extra che devono essere trasmessi per rendere possibile la comunicazione. L'overhead riduce il bit-rate disponibile dato dalla formula di Shannon e Nyquist.

Bit stuffing: caso peggiore

Con il bit stuffing nel caso peggiore perdi $\frac{1}{6}$ del bit-rate, c'è un overhead costante di 2 bytes per i markers e poi è variabile in base al numero di 5 uni consecutivi.

Bit stuffing: caso medio

Per calcolare il caso medio dobbiamo fare qualche ipotesi sulla probabilità di avere zero e uno.

- Supponiamo che 0/1 sia una variabile di Bernoulli con $p = 0,5$.
- Si aggiunge uno zero quando si hanno 5 uno consecutivi, cioè $1/2^5 = 1/32$ dei casi. Ciò significa che in media il $\sim 3\%$ della capacità viene sprecato.
- Nel mondo reale, semplicemente non conosci il valore medio, perché non puoi prevedere quanti bit saranno zero o uno.

Più grande è il marker, più hai un overhead fisso, ma minore è la probabilità di fare bit stuffing.

Quindi una buona regola è: usa marcatori grandi, ma cerca di avere grandi frame per compensare il sovraccarico fisso dato dai marcatori.

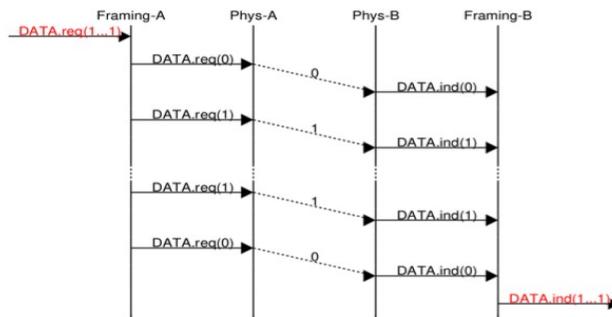
Esempio 1:

Assumi di trasmettere sempre 64 bit frame, il marker è 011110. Quanto è l'overhead medio del collegamento?

- Ogni 64 bits, si hanno $6 * 2 = 12$ bit per i marker
- Capita con una probabilità di $\frac{1}{8}$ di aggiungere uno 0 dopo una sequenza di 3 bit
- Abbiamo quindi un overhead di $\frac{62}{8} \cong 8 + 12 = 20$ bit per inviarne 64, $\frac{20}{84} \cong 23\%$ della capacità viene perso.

Nelle reti moderne non si usa il bit stuffing, in quanto si usa un lungo preambolo, il quale contiene l'inizio e la fine del frame ma anche la lunghezza dello stesso, evitando così l'uso di un marker alla fine del frame.

Framing and Layers



Esempio di layering delle reti:

- Possediamo già un livello fisico che supporta la primitiva **DATA.req()**.
- Aggiungiamo un livello che utilizza una primitiva simile per inviare frame (ci vuole più di un bit come parametro)

- Il **Framing Layer** si curerà anche di aggiungere i marcatori e i bit imbottiti, e il livello più alto non lo saprà mai. Questo potrebbe sembrare banale, ma è fondamentale. Se sei un programmatore del livello esterno, non hai bisogno di conoscere il livello interno, solo l'interfaccia con esso

Acknowledging Frames

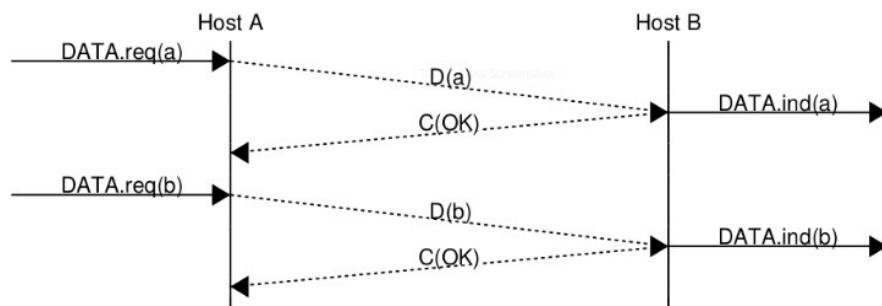
Service Data Units (SDU)

Il livello Datalink riceve quelle che vengono chiamate **Service Data Unit**, SDU. Li trasformerà quindi in frame come abbiamo visto precedentemente.



Acknowledgment

Un Acknowledgment è un pacchetto che non contiene alcun dato, segnala solo che i dati precedenti sono stati ricevuti correttamente. Per poter distinguere i due diversi pacchetti separiamo il frame in due parti, la prima parte (intestazione) non contiene dati ma un singolo bit (0 -> dati, 1 -> ACKS), la seconda parte invece contiene i dati effettivi da scambiare.

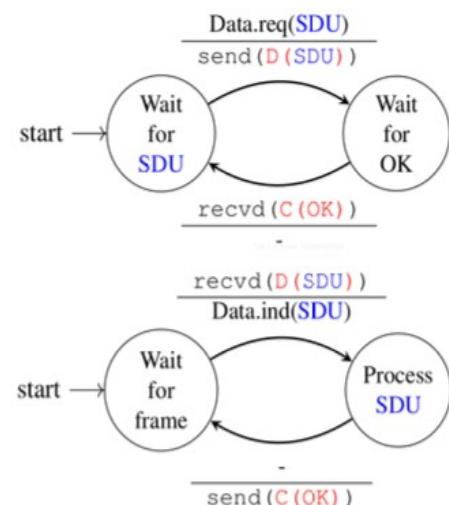


Automa a stati finiti di Sender e Receiver

Il **Sender** (metà superiore dell'immagine) e il **Receiver** (metà inferiore) hanno due stati

Nella parte superiore della label hai l'evento che attiva il cambiamento di stato. Nella parte inferiore della label hai l'evento che viene generato durante il cambio di stato.

D sta per Data, C per ACK. Se lo standard di comunicazione utilizza il bit stuffing, i bit verranno modificati secondo la necessità.



Affrontare gli errori

Un frame può contenere errori: bit capovolti, bit mancanti, bit aggiunti. Il primo problema da risolvere è come rilevare che un frame è influenzato da errori.

Bit di parità

La tecnica per il rilevamento degli errori più semplice è il bit di parità, che può essere pari o dispari. Il bit di parità viene aggiunto al frame per rendere il numero di 1 pari o dispari. Il ricevitore controlla quindi la parità e rimuove il bit.

Mittente:

- Supponiamo che l'informazione da trasmettere sia 0011011. Poiché il numero di uno è pari, la parità dispari aggiunge 1, la parità pari aggiunge 0
 - con la parità dispari i dati diventano 00110111
 - Con la parità pari i dati diventano 00110110
- Computazionalmente, un bit di parità pari è la somma nel modulo 2 di tutti i bit nei frame.

$$(0 + 0 + 1 + 1 + 0 + 1 + 1) \text{mod}_2 = 0$$

Mentre la parità dispari è la stessa, più uno.

$$(0 + 0 + 1 + 1 + 0 + 1 + 1 + 1) \text{mod}_2 = 1$$

Destinatario:

- Supponiamo che venga utilizzata la parità dispari, quindi il ricevitore sa che il frame deve essere fatto di un numero dispari di uno. Se questo non accade, c'è stato un errore di trasmissione:
 - 111011011 viene ricevuto: OK, 7 uni.
 - 001100110 viene ricevuto: NOK, 4 uni.
- Se non c'è alcun errore, il ricevitore rimuove semplicemente il bit di parità e passa la SDU al livello superiore.
- Nel caso in cui ci sia un errore, il ricevitore farà ciò che è appropriato, cioè chiedere al mittente di inviare di nuovo il frame, o semplicemente dropparlo, a seconda del tipo di design del livello Datalink.
- Se c'è un numero dispari di errori, il bit di parità *non può riconoscerli*.

IP Checksum

Il campo *checksum* è un campo lungo 16 bit che contiene il complemento a 1 (\sim) della somma in complemento a 1 ($+'$) di tutte le words a 16 bit.

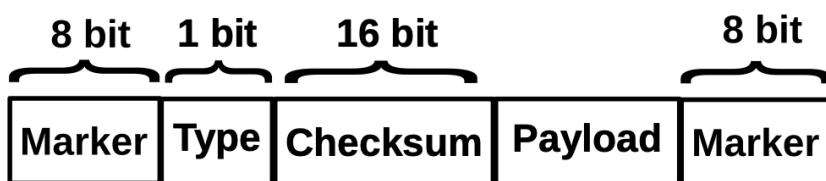
$$\begin{aligned} \text{complemento a 1: } & \overline{1100} = 0011 \\ \text{somma complemento a 1: } & 1100 + 1010 = \textcolor{red}{10110} \rightarrow \textcolor{red}{1} + 0110 = 0111 \end{aligned}$$

Il checksum viene aggiunto all'header del frame, così il receiver riceve il dato e il suo checksum. Successivamente il ricevitore dovrà solo fare la somma $+$ ' su tutto il dato compreso il checksum. Se il risultato di $+$ ' contiene tutti 1, allora non ci sono errori.

La probabilità di avere alcuni errori ma di avere ancora un checksum corretto è limitata da 2^{-C} dove C è la dimensione del checksum.

La lunghezza dei pacchetti ora deve essere un multiplo di 16, se ce ne fossero di meno è necessario aggiungere dei *bogus bits*.

Il pacchetto dopo il livello collegamento contiene checksum, tipo e marker oltre ai dati.



Esempio:

Assumiamo di usare 4 bits invece di 16. I dati da inviare sono 1100 0011 1010 1110 1001

$$\begin{aligned} 1100 + 0011 + 1010 + 1110 + 1001 &= \textcolor{red}{110000} \\ \textcolor{red}{11} + 0000 &= 0011 \rightarrow \textit{checksum} = \overline{0011} = 1100 \end{aligned}$$

Il dataframe da inviare sarà **1100** 1100 0011 1010 1110 1001.

Il ricevente per verificare che i dati arrivati siano corretti:

$$\begin{aligned} \textcolor{red}{1100} + 1100 + 0011 + 1010 + 1110 + 1001 &= \textcolor{red}{111100} \\ \textcolor{red}{11} + 1100 &= 1111 \end{aligned}$$

Il pacchetto è corretto.

Trasporto affidabile

Come principio generale, si presume che la rete funzioni ragionevolmente bene: si presume che la probabilità di errore sia abbastanza bassa, quindi gli errori sono rari. Detto questo, non è conveniente sprecare troppe risorse per correggere/evitare gli errori, è più conveniente buttare via i frame con errori e chiedere la ritrasmissione.

Alternating Bit Protocol (ABP)

Per rilevare un errore si usava uno schema molto semplice:

- Si invia il frame
- Si avvia un timer
- Se ACK non arriva si ripete la procedura

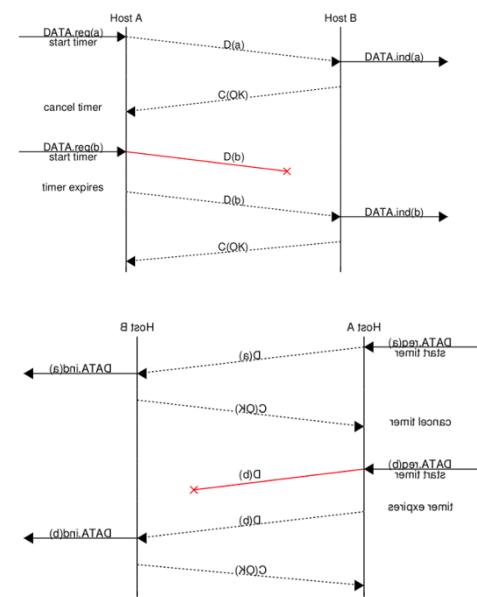
Questa tecnica non distingue il caso di errore o perdita del pacchetto.

Il receiver deve essere in grado di distinguere le copie dei frame.

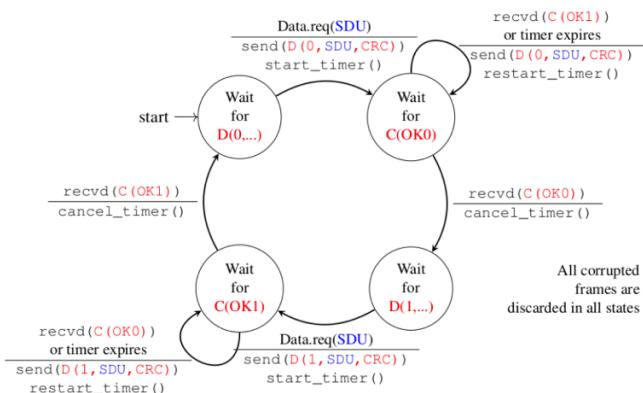
Potrebbe però succedere che il timer scada e prima che il destinatario invii ACK e di conseguenza venga inviato di nuovo il pacchetto. Il destinatario ora ha due copie dello stesso pacchetto senza averlo richiesto e senza riconoscerle.

Aggiungiamo all'intestazione un altro campo: il **numero di sequenza**. Nel caso più semplice, il numero di sequenza può essere un singolo bit, che viene capovolto ad ogni frame, il frame ACK riporterà a quale numero di sequenza si riferisce.

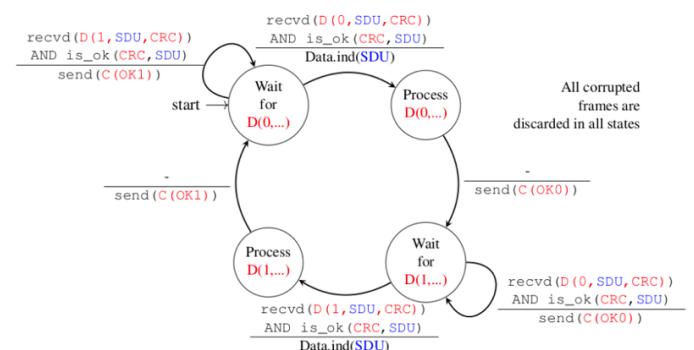
Questo semplice schema è chiamato **Alternating Bit Protocol**.



ABP: macchina di stato mittente

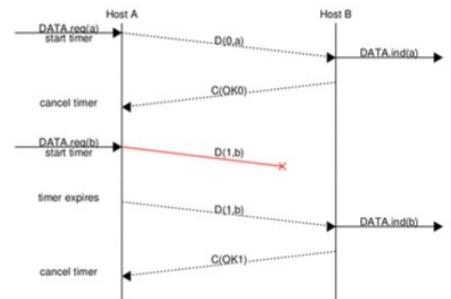


ABP: macchina di stato mittente



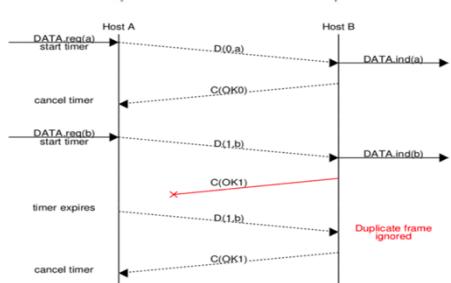
Caso di SDU drop:

- Host B non invia ACK
- Il timer di host A scade e quindi reinvia il frame
- Una volta ricevuto ACK da host B, host A cambia stato



Caso di ACK drop:

- Host B aspetta il frame con numero di sequenza 1
- Il timer di host A scade e reinvia il frame con numero sequenza 1
- Host B si accorge di una copia ed inoltra comunque ACK



Round Trip Time (RTT) : Supponiamo che ci vogliono 10 ms perché un frame vada dal mittente al destinatario, il timer che il mittente usa deve essere quindi di almeno 20 ms, questo si chiama **RTT**.

Protocol Overhead: Se facciamo l'ACK di ogni singolo frame, non importa quanto sia veloce il collegamento, il Round Trip Time rallenta la trasmissione. Questa è un'altra forma di overhead di protocollo, che viene introdotta per offrire un servizio migliore a livello Datalink: **reliable delivery**. Possiamo aumentare la capacità di un collegamento, ma ci sono limiti fisici dovuti al RTT.

Alternating Bit Protocol è semplice da implementare e affidabile, tuttavia, introduce un **delay intollerabile**.

ABP Performance

Supponiamo che ci vogliono 10 ms perché un fotogramma vada dal mittente al ricevitore.

Quindi il timer utilizzato dal mittente deve essere di almeno 20 ms, questo è chiamato Round Trip Time (RTT).

Supponiamo che il fotogramma sia 1500B, che è la dimensione standard di un pacchetto Internet.

Supponiamo che il collegamento sia un collegamento gigabit: 10^9 b/s .

Ci vogliono $1500 * \frac{8}{10^9} = 0,012 \text{ ms}$ per inviare un frame di dati.

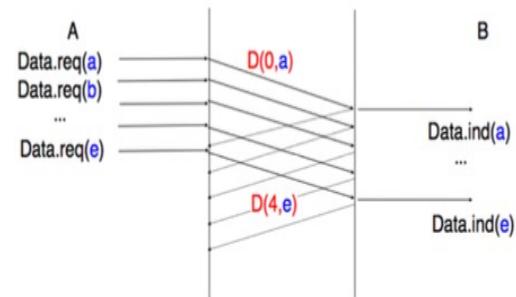
Un frame ACK è composto da due bit (uno per l'intestazione, uno per il seq_n): $\frac{2}{10^9} \text{ ms}$.

Per inviare quindi il frame di 1500B ci vogliono 20.012 ms , corrispondente a un bit-rate pari a $\frac{1500 * 8}{0.020012} \cong 0.6 \text{ Mb/s}$

Go-back-n e Selective Repeat

Il modo migliore per sfruttare un canale veloce è fare **pipelining**, significa inviare un lotto di frame, e mentre A li invia, B può iniziare a fare ACK.

Tuttavia, dobbiamo ancora evitare l'overflow e garantire l'affidabilità, che si ottiene con un meccanismo **sliding window**.



Sliding Window

I numeri di sequenza ora sono numeri **intei** finiti, in quanto la dimensione dell'header è finita.

A e B concordano su una dimensione della finestra scorrevole.

- Il mittente impone una dimensione 5, quindi A invierà 5 frame e si fermerà.
- Man mano che i frame di dati arrivano B invierà ACK per numeri di sequenza specifici.
- Quando il frame con il numero di sequenza più basso è ACKed, A sposta la finestra a destra e ne invia un altro, e così via.

I numeri di sequenza sono finiti, se ci sono n bit dedicati al numero di sequenza nell'header del frame, allora il range sarà ovviamente tra 0 e $2^n - 1$.

maxseq = numero di sequenza più grande rappresentabile.

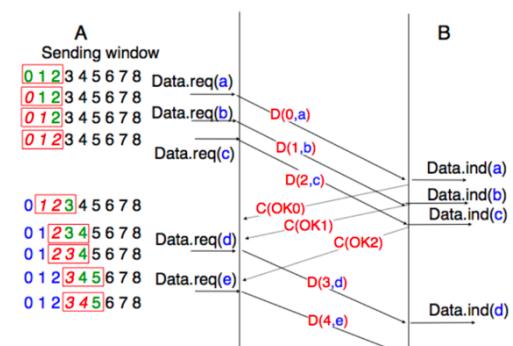
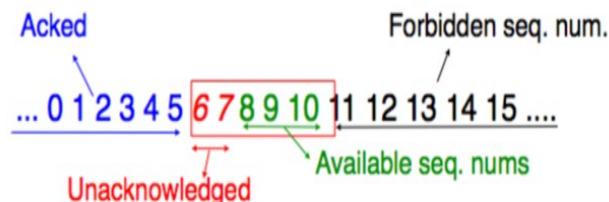
La Sliding Window implementa il Flow Control e riduce l'impatto del RTT. Ora i frame arrivano in lotti, anche gli ACKS sono generati in batch; quindi, le prestazioni del livello Data Link migliorano.

Tuttavia, abbiamo bisogno di una policy per decidere come trattare la perdita di alcuni frame: **Go-back-n**.

Go-Back-n

Destinatario B:

- B accetta solo i frame che arrivano in sequenza scartando tutti i frame fuori sequenza che riceve.

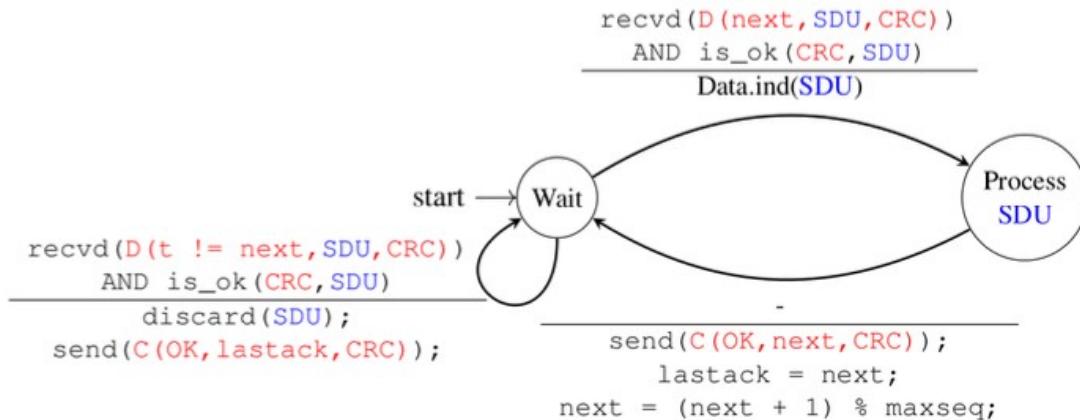


- Quando B riceve un frame di dati, restituisce sempre un **ACK** contenente il numero di sequenza dell'ultimo frame in sequenza che ha ricevuto.

Questo riconoscimento è detto **cumulativo**, il che significa che riconosce l'ultimo frame e tutti i precedenti con un numero di seq inferiore. I frame vengono elaborati uno per uno, non c'è buffering al ricevitore.

Il ricevitore mantiene, inoltre, due variabili:

- *Lastback*: ultimo numero di sequenza ACKed
- *Next*: prossimo numero di sequenza che si aspetta

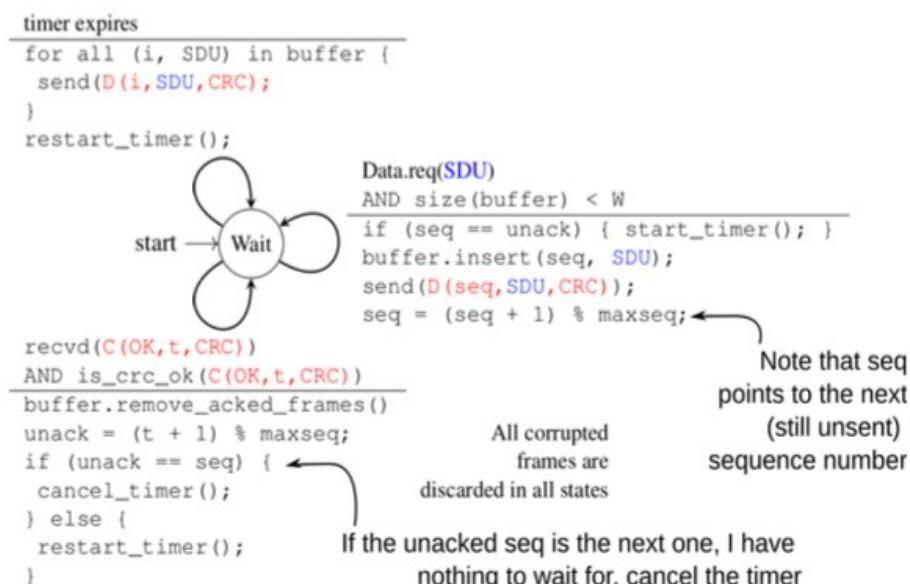


Mittente A:

- A mantiene un buffer di invio che può memorizzare un'intera finestra scorrevole di frame.
- A utilizza un **singolo timer** di ritrasmissione che viene avviato quando viene inviato il primo frame.
- I frame vengono inviati con numeri di sequenza crescenti, fino a quando il buffer di invio è pieno. Poi si ferma e aspetta un ACK.
- Quando A riceve un ACK:
 - Rimuove dal buffer di invio **tutti i frame che hanno ricevuto un ACK**, con *seq_num* inferiore rispetto a quello ricevuto
 - riavvia il timer di ritrasmissione solo se ci sono ancora frame unacknowledged nel suo buffer di invio
- Se il timer di ritrasmissione scade, A presuppone che tutti i frame unacknowledged attualmente memorizzati nel suo buffer di invio siano andati persi, quindi ritrasmette tutti i frame nel buffer e riavvia il suo timer di ritrasmissione.

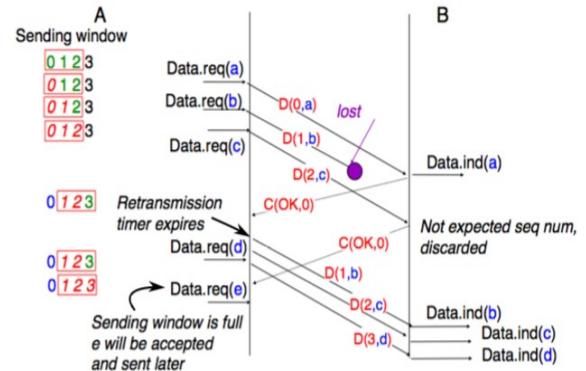
Il Mittente mantiene, inoltre, tre variabili:

- *Size(buffer)*: quantità di frame inviati nel buffer
- *Seq*: ultimo *seq_num* inviato
- *Unack*: *seq_num* del primo frame unacknowledged



Esempio:

- Quando A riceve C(OK, 0), seq = viene rimossa, window shifts, e il timer riparte.
- Quando il timer scade i frame b - c vengono rinviati.
- Quando il frame è arrivato ed il buffer è pieno esso viene rifiutato e A deve reinviararlo.



Limiti del Go-back-n

Go-back-n è facile da implementare, le macchine di stato sono molto semplici, ci sono solo due variabili di stato e un timer. Funziona bene quando vengono persi solo pochi frame e riduce la dipendenza dall'RTT.

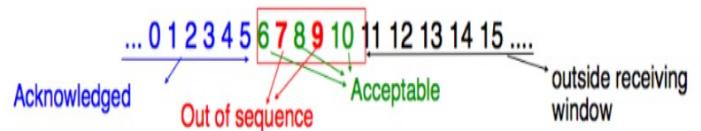
Tuttavia, quando ci sono molte perdite, le prestazioni di go-back-n diminuiscono rapidamente per due motivi:

- il ricevitore non accetta frame fuori sequenza
- il mittente ritrasmette tutti i frame non riconosciuti una volta che ha rilevato una perdita

Selective Repeat

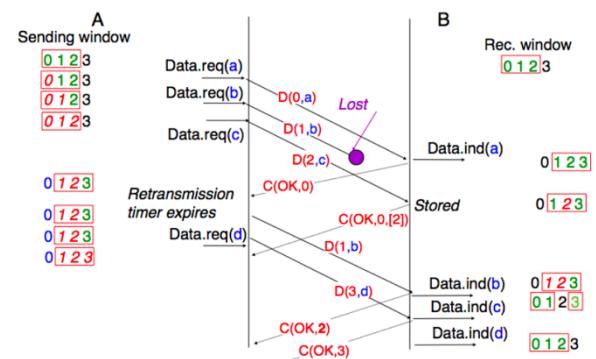
Go-back-n può essere migliorato con uno schema **selective repeat**:

- B ora ha un buffer e accetta i fotogrammi purché siano nella finestra.
- In ogni ACK, B riporta l'ultimo numero di sequenza prima dell'inizio della finestra e un elenco di fotogrammi arrivati correttamente fuori ordine.
- A ritrasmetterà solo quelli senza ACK, il che richiede un timer per numero di sequenza.



Nota:

- Ogni volta che A invia avvia un timer diverso
- Quando A riceve un ACK, rimuove tutti i timer corrispondenti a quel frame e se possibile window shifts.
- Quando un timer scade, invia solo il frame corrispondente.



Nel mondo reale, gli scambi di dati non avvengono mai in un'unica direzione. I protocolli ad alto livello includono lo scambio di dati in due direzioni.

È conveniente includere AC_number nell'header del protocollo così ACK può essere **piggybacked** (aggiunto) al frame dati e non richiede un frame dedicato. Il Destinatario B non invierà subito ACK ma aspetterà un po' di tempo per inviare un frame dati. Se il tempo di attesa è troppo lungo viene generato un frame ACK ed inviato singolarmente.

Considerazioni

La dimensione della finestra non deve essere la stessa per il destinatario e il mittente. Le finestre dovrebbero essere negoziabili, in modo che la loro dimensione cambi a seconda delle condizioni di rete.

Il più delle volte, i protocolli di comunicazione sono bidirezionali, quindi i dati fluiscono in entrambe le direzioni. Se questo accade i messaggi ACK vengono scambiati come **piggybacked** in un data frame.

Massima dimensione della finestra (Selective repeat e go-back-n)

Un protocollo reliable che utilizza n bit per codificare il proprio numero di sequenza può inviare fino a 2^n frame successivi. Tuttavia, per garantire una consegna affidabile dei frame, go-back-n e selective repeat non possono utilizzare una finestra di invio di 2^n frame.

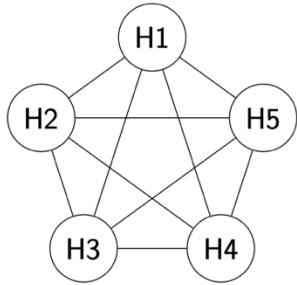
Consideriamo innanzitutto go-back-n e assumiamo che un mittente invii 2^n frame. Questi frame vengono ricevuti in sequenza dalla destinazione, ma tutte le conferme restituite vengono perse. Il mittente ritrasmetterà tutti i frame. Questi frame verranno tutti accettati dal destinatario e consegnati una seconda volta all'utente.

È facile vedere che questo problema può essere evitato se la dimensione massima della finestra di invio è pari a $2^n - 1$ frame.

Un problema simile si verifica con la selective repeat. Tuttavia, poiché il ricevitore accetta frame fuori sequenza, una finestra di invio di $2^n - 1$ frame non è sufficiente per garantire una consegna affidabile. Si può facilmente dimostrare che per evitare questo problema, un mittente selective repeat non può utilizzare una finestra più grande di $\frac{2^n}{2}$ frame.

Datalink Layer - Sharing Resources

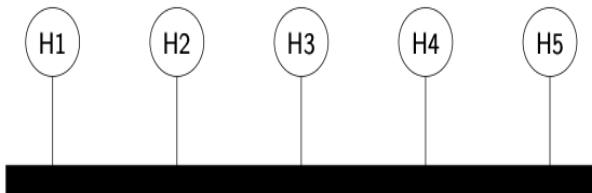
Network Topologies



Full Mesh Network

- Massima capacità
- Massima robustezza

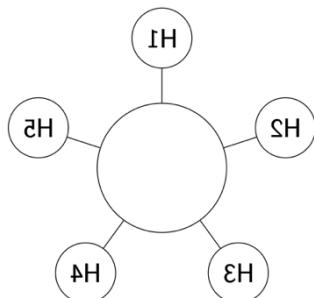
Raramente utilizzata.



Bus Network

- Molto economico
- Non scalabile
- Non robusto

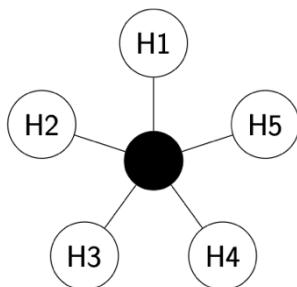
Utilizzato nei primi giorni delle reti o in alcuni scenari specifici (ad esempio, il cosiddetto CAN-bus della tua auto).



Ring Network

- Massima robustezza
- Non scalabile

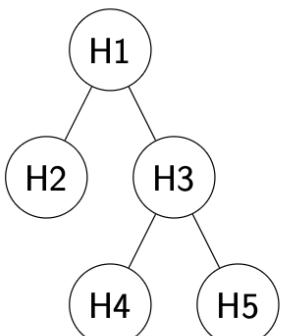
Utilizzato nelle reti metropolitane per aggiungere ridondanza quando il cablaggio è molto costoso.



Star Network

- Minori performance
- Miglior controllo
- Resistente ai danni
- Se il centro muore, la rete si ferma

Utilizzato nelle LAN.



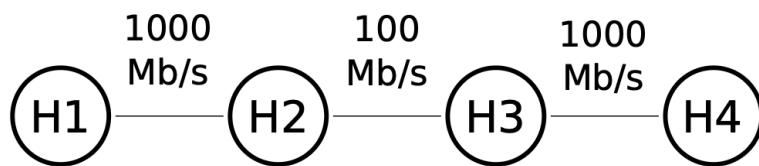
Tree Network

- Minor rapporto qualità/prezzo
- Se muore una connessione, la rete si ferma

Per estendere le LAN con interruttori a cascata e in molte altre circostanze.

Bottleneck

LA comunicazione tra H1 e H4 è limitata dalla connessione 100Mb/s tra H2 e H3, è impossibile quindi comunicare alla velocità di 1000Mb/s tra i due host.



Dobbiamo programmare l'accesso ai media fisici. Questo è uno dei compiti del layer Datalink. Quindi, con qualche organizzazione di rete, i terminali condivideranno necessariamente i media fisici. Quando questo accade devono trovare un modo per non interferire tra loro. Ci sono diversi modi per ottenere questo

Frequency division

Un modo per evitare le collisioni è usare una frequenza diversa per ogni comunicazione. Abbiamo visto che al livello fisico il segnale è modulato su una certa portante (carrier).

Radio

Le radio sono in grado di filtrare le comunicazioni che non utilizzano una specifica frequenza. Quindi, se una usa una frequenza X e un'altra utilizza la frequenza Y, le comunicazioni possono avvenire contemporaneamente, anche se i due ricevitori sono uno accanto all'altro. Questo non produrrà una collisione.

Wi-Fi

Il Wi-Fi può funzionare su diverse frequenze, chiamate canali. Quando accendi il tuo Access Point Wi-Fi (AP) per la prima volta, cercherà un canale che non viene utilizzato da altre reti Wi-Fi. Ciò consente a due reti di condividere lo stesso spazio fisico. Tuttavia, gli host collegati a un AP devono utilizzare la stessa frequenza e la priorità per inviare messaggi va organizzata.

Time-Division Multiple Access (TDMA)

Il modo più intuitivo per risolvere il problema di dare il diritto di inviare messaggi a multipli host connessi ad un unico terminale è il Time-Division-Multiplexing (TDM).

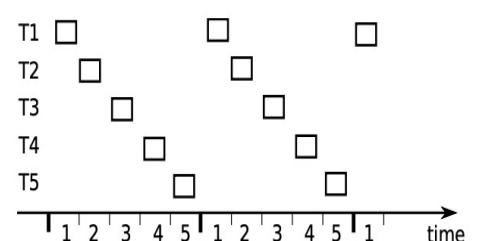
In un TDM, se la rete ha m terminali, il tempo è diviso in m slot di dimensione fissa, con uno slot assegnato a un solo terminale.

Gli slot sono ripetuti in modo ciclico, quindi il terminale i può trasmettere solo al frame i .

TDMA è efficiente quando la rete è satura

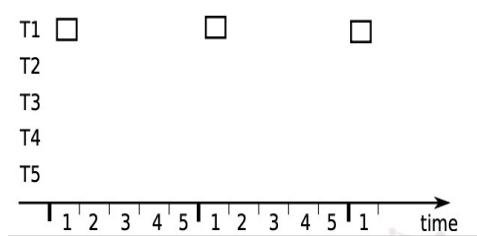
Quando tutte le stazioni hanno sempre qualcosa da trasmettere, la rete è in quella che chiamiamo saturazione. In questa condizione, il TDMA è alla massima efficienza: ogni slot è occupato e non ci sono collisioni.

Datalink Layer Efficiency: la frazione di tempo in cui il collegamento dati viene utilizzato per comunicare dati utili.



TDMA è poco efficiente quando la rete è libera

La limitazione del TDM è che ogni terminale avrà l'opportunità di trasmettere tutti gli $1/m$ slot, indipendentemente da ciò che fanno gli altri. Se la stazione i ha un frame da inviare al tempo $i+1$, dovrà attendere un ciclo completo di m slot prima di poter trasmettere, anche se gli altri nodi non hanno nulla da trasmettere. TDM introduce quindi un **ritardo medio di $m/2$ slot**.



Quindi il TDMA è efficiente quando la rete è congestionata, ed è inefficiente quando la rete è scarica.

Si noti che quando la rete è congestionata, generalmente significa che hai più traffico da inviare di quanto che puoi inviare; quindi, un po' di traffico verrà droppato. In pratica, se hai un'aspettativa del carico, devi dimensionare la capacità della tua rete in modo che sia raramente congestionata.

Pertanto, avere un sistema che funziona bene nel peggior dei casi e funziona male nelle condizioni normali (come TDMA) non è conveniente.

Esempio:

Considera un sistema TDMA con bit-rate 100Mb/s con n=5 terminali, ad ognuno viene assegnato uno slot di s=1ms.

Qual è la quantità massima di dati che un terminale può inviare in uno slot? Qual è la quantità massima di dati che il terminale può inviare in un secondo?

$$b = 100Mb/s = 10^8 b/s = 10^{-3}s$$

La capacità massima di dati per slot è $s * b = 10^5 = 100kb$

$n = 5$, quindi in 1 secondo ci sono $\frac{100}{5}$ slot per terminale, bit-rate per terminale $10^5 * 200 = 20.000.000 = 20Mb/s$, un quinto della capacità disponibile.

Supponiamo che il terminale 1 debba inviare 250kb, qual è il tempo necessario per inviare completamente il fotogramma, a partire dal momento in cui i dati vengono ricevuti dal livello superiore?

- caso migliore: $t = 10.5 ms$
- caso medio: $t = 10.5 + (5 + 0) = 15.5 ms$
- caso peggiore: $t = 10.5 + \frac{5+0}{2} = 13ms$

Random Access

Un altro modo di accedere ai media è attraverso l'**accesso casuale**.

Nell'accesso casuale, i terminali tentano di trasmettere e, se si verifica una collisione, la collisione viene rilevata e il fotogramma viene inviato di nuovo.

Il padre di tutti gli schemi di accesso casuale è **ALOHA**.

S-Aloha Protocol (slotted version)

Con S-ALOHA quando un nodo ha bisogno di inviare un frame, aspetta l'inizio dello slot successivo e lo trasmette. S-ALOHA ammette una probabilità di collisione, le collisioni fanno fallire la trasmissione.

La differenza con TDMA è che ogni terminale può trasmettere in ogni slot.

Assunzioni matematiche

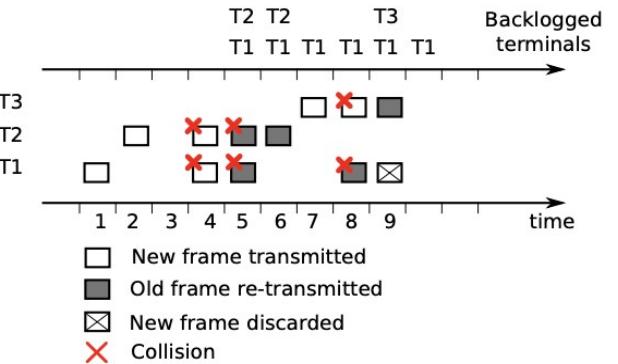
1. Ci sono **m** terminali sincronizzati che trasmettono a un singolo ricevitore.
2. Essendo sincronizzati, i terminali **conoscono l'inizio di ogni slot**.
3. Le trasmissioni sono sempre **più brevi** di un tempo di slot.
4. Due trasmissioni contemporanee fanno fallire la ricezione, il terminale sa immediatamente se è successo.
5. Il **ricevitore ha uno slot dedicato** in cui nessun altro può trasmettere. Lo usa per inviare gli **ACK** dei frame.
6. Se si verifica una collisione, il mittente non riceve un ACK e può rilevare il guasto.

Dettagli del protocollo ALOHA

- Quando un nodo riceve un frame SDU aspetta fino allo slot successivo e trasmette. Chiamiamo q_a la probabilità che un nodo abbia qualcosa da trasmettere.
- Se un altro nodo trasmette nello stesso frame, entrambi i nodi entrano in uno stato **backlogged**. Ciò significa che trasmetteranno nuovamente negli slot successivi con probabilità $q_b > q_a$.
- Un terminale **backlogged** torna allo stato normale quando è in grado di trasmettere con successo.
- Se un terminale **backlogged** riceve un altro (nuovo) frame da inviare, lo scarta.
- q_a dipende dal traffico in entrata mentre q_b è un parametro di rete.

Evoluzione

- Al tempo 1 e 2, T1 e T2 inviano un nuovo frame, con successo.
- Al tempo 3, nessuno sta trasmettendo
- Al tempo 4, si verifica una collisione. T1 e T2 entrano nello stato **backlogged**.
- Al tempo 5, T1 e T2 cercano di ritrasmettere il vecchio frame, ma si scontrano di nuovo.
- Al tempo 6, T2 riesce a ritrasmettere il vecchio frame, esce dallo stato arretrato
- Al tempo 7, T3 trasmette un nuovo frame con successo, T1 è arretrato e non trasmette.
- Al tempo 8, T1 e T3 cercano di trasmettere un nuovo frame e si scontrano, entrano nello stato backlogged.
- Al tempo 9, T3 riesce a ritrasmettere il frame ed esce dallo stato backlogged. Allo stesso tempo T1 riceve un nuovo frame ma lo scarta senza inviarlo perché non ha la coda.



Quando le reti con TDMA e ALOHA collassano?

Congestion collapse: il collasso della congestione si verifica quando le prestazioni relative del sistema si degradano bruscamente con l'aumento del carico.

TDMA: mai, anche se la rete è over congestionata ogni terminale ha sempre uno spazietto.

ALOHA: collassa se le risorse per ogni terminale vanno praticamente a 0. Questo è il motivo per cui quando sei in un luogo affollato il Wi-Fi smette semplicemente di funzionare, o se lo fa, produce un comportamento esplosivo: per un breve intervallo di tempo sei fortunato e puoi trasmettere/ricevere, e poi improvvisamente sei sfortunato e non puoi trasmettere per un bel po'.

Conclusioni

ALOHA ha altre proprietà indesiderate, per esempio, ha dimostrato di essere instabile.

Quando studieremo le reti wireless 802.11 vedremo come hanno migliorato ALOHA per stabilizzare la rete e ridurre la probabilità di un collasso.

Da un lato hai sistemi deterministici come TDMA, che garantiscono prestazioni e sono facili da implementare, ma sono inflessibili e sprecano risorse a basso carico.

Dall'altro hai schemi di accesso casuali, che sono più veloci a basso carico, ma possono crollare totalmente ad alto carico.

Sono entrambi utilizzati, in applicazioni diverse.

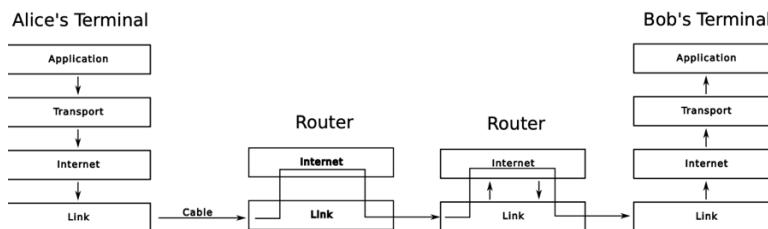
Il livello di collegamento dati è intrinsecamente inedificabile, può far cadere frame. Gli strati superiori devono essere consapevoli di questo e cercare di compensare.

Network Layer

Il livello Datalink si occupa di fornire connettività da ogni host agli altri host fisicamente connessi.

Il livello di rete consente invece la trasmissione di informazioni tra host che non fanno parte della stessa rete fisica attraverso router intermedi

Al livello di rete i frame sono chiamati **pacchetti**. Un pacchetto contiene i dati, più un'intestazione del livello di rete. Sarà incapsulato nel payload di un frame di livello Datalink.



Ci sono due tipi di livelli rete utilizzati:

- **Datagram-oriented organization**, usato in internet
- **Control-oriented organization**

Datagram-oriented Organization

Ogni host è identificato da un indirizzo di livello di rete.

Per inviare informazioni a un host remoto, un host crea un pacchetto che contiene:

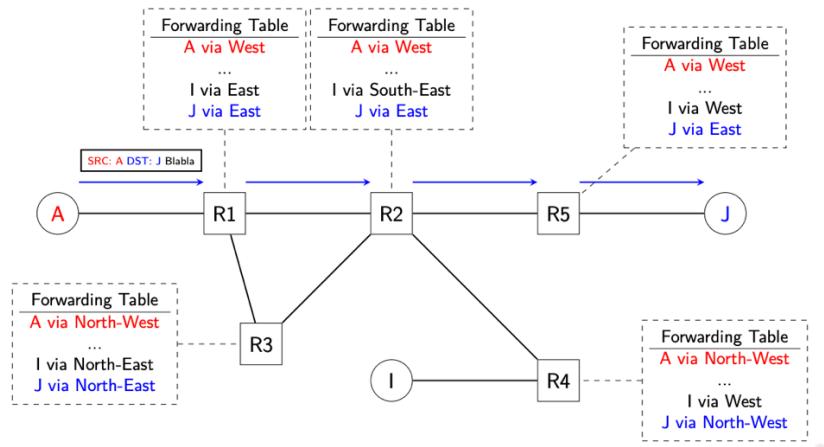
- l'indirizzo del livello di rete dell'host di destinazione
- il proprio indirizzo del livello di rete
- le informazioni da inviare

I router utilizzano l'inoltro **hop-by-hop**: ciò significa che quando un router riceve un pacchetto che non è destinato a sé stesso, prende una decisione su chi inviare il pacchetto in seguito, questa decisione è una scelta del router stesso.

Per effettuare queste decisioni sono necessarie le **Forwarding Table**.

Forwarding Table

Una **Forwarding Table** è una struttura dati che mappa ogni indirizzo di destinazione (o insieme di indirizzi di destinazione) all'interfaccia in uscita attraverso la quale un pacchetto destinato a questo indirizzo deve essere inoltrato per raggiungere la sua destinazione finale. Il router consulta la sua Forwarding Table per inoltrare ogni pacchetto che gestisce.



Possono accadere due errori:

1. **Black Holes**: un buco nero è la situazione in cui un router dovrebbe inoltrare un pacchetto ma non ha una voce nella Forwarding Table verso la destinazione. Drop del pacchetto.
2. **Routing Loops**: per raggiungere l'host J, il router R1 invia il pacchetto a R2, ma R2 lo rimanda a R1, che lo rimanda a R2.... I loop possono essere più lunghi di un semplice salto e sprecare la capacità dei collegamenti coinvolti.

Il software del router è logicamente suddiviso in due funzioni:

- **Data Plane**: funzione usata per instradare i pacchetti in accordo con la forwarding table.
- **Control Plane**: funzione usata per la creazione della forwarding table.

Indirizzamento ed eterogeneità

Flat Addressing

Una Forwarding Table ha bisogno contenere tutte le informazioni per raggiungere ogni destinazione. Sappiamo che ogni host e router ha un indirizzo diverso.

Nello schema di indirizzamento piatto, questi indirizzi sono totalmente estranei l'uno all'altro. Ad esempio, dipendono da un componente hardware, quindi il loro indirizzo è deciso dal produttore.

PRO: non hai bisogno di *configurare la rete* perché gli indirizzi sono già pronti.

CONTRO: non è *scalabile* ogni Forwarding Table dovrebbe avere una voce per ogni indirizzo; *problemi di performance*, ad ogni hop il router deve fermarsi a cercare l'indirizzo su una tabella con miliardi di elementi.

Hierarchical Addresses

Gli indirizzi sono raggruppati in blocchi. Ad ogni router è assegnato un blocco. La Forwarding Table così memorizza solo l'indirizzo di un blocco. Questo riduce le Forwarding Table in ordini di grandezze migliorando le performance.

Fragmentation

Il livello datalink ha una dimensione massima fissa dei frame, si chiama **Maximum Transfer Unit (MTU)**.



Il router R1 non può mandare un frame di grandezza 1000B al router R2 perché è più grande della grandezza massima consentita.

R1 può fare 3 cose:

- **Drop and Notify**

R1 rifiuta il pacchetto e invia un pacchetto di controllo indietro all'host A per indicare che non può inoltrare pacchetti più lunghi di 500 byte (compresa l'intestazione del pacchetto).

L'host A riceve il pacchetto di controllo e reagisce ritrasmettendo le stesse informazioni in pacchetti più piccoli.

Non è una soluzione ottimale. La stessa condizione può verificarsi più di una volta sul percorso verso la destinazione: l'host A deve bufferire i pacchetti per molto tempo.

- **Fragment and Reassemble at next hop**

R1 è in grado di frammentare un pacchetto in due parti. La prima parte contiene l'inizio del carico utile e la seconda la fine.

Entrambi i pacchetti vengono trasmessi al router R2.

Il router R2 riassembra i due frammenti di pacchetto in un pacchetto più grande prima di trasmetterli sul link verso l'host B.

Meglio, ma costringe i router a riassemblare sempre i pacchetti because it involves introduce ritardo e complessità difficile da implementare alla velocità di linea.

- **Fragment and Reassemble at Destination**

Come nell'opzione 2, ma ogni frammento è un pacchetto indipendente. Le intestazioni del livello di rete vengono copiate in ogni frammento, in modo che ogni frammento possa raggiungere la destinazione.

Quando arrivano a B, B li raccoglierà tutti e costruirà il pacchetto originale.

Miglior compromesso: introduce un piccolo ritardo dovuto alla frammentazione su un router, ma delega il riassemblaggio a destinazione.

Routing Algorithms

Algoritmo: finito numero di istruzioni che risolvono un certo problema.

Routing protocols: insieme di specifiche usate per creare algoritmi nei router. Se il protocollo è ben formalizzato, qualsiasi fornitore può implementarlo con il proprio software e i router saranno interoperabili.

Demone: software attivo nel router che implementa un routing protocol. È il pezzo di logica che creerà i pacchetti di controllo, li invierà sull'interfaccia di rete, elaborerà i pacchetti che arrivano, manterrà la tabella di routing nella memoria del router e la tradurrà in una tabella di inoltro quando si verificano alcune modifiche.

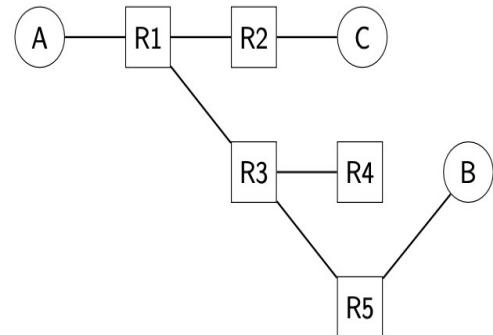
Hot Potato

Assumiamo che la rete è una Tree Network

La rete si avvia, ogni host ha un indirizzo, tutte le tabelle di inoltro sono vuote.

Assumiamo che host A voglia mandare un pacchetto ad host B

- Host A invia il pacchetto via l'unico collegamento disponibile con l'intestazione del destinatario B
- Quando R1 riceve il pacchetto da A, inserisce la forwarding table $A \rightarrow \text{west}$
- Non conoscendo il collegamento con B, inoltra il pacchetto a tutte le interfacce disponibili
- Quando R2 riceve il pacchetto da R1, inserisce la forwarding table $A \rightarrow \text{west}$ ed inoltra ancora il pacchetto a tutte le interfacce disponibili
- Il pacchetto arriva a C, esso lo scarta in quanto non è lui il destinatario
- Quando R3 riceve il pacchetto da R1, inserisce la forwarding table $A \rightarrow \text{Nwest}$ ed inoltra ancora il pacchetto a tutte le interfacce disponibili
- Il pacchetto arriva a R4, esso lo scarta in quanto non è lui il destinatario
- Quando R5 riceve il pacchetto da R3, inserisce la forwarding table $A \rightarrow \text{Nwest}$ ed inoltra ancora il pacchetto a tutte le interfacce disponibili
- Il pacchetto arriva a B, esso lo accetta in quanto è lui il destinatario



Step riassuntivi

- Se non c'è un link nella forwarding table ogni router manda a tutti (broadcast).
- Quando un router riceve un pacchetto aggiorna la forwarding table con le indicazioni per il mittente.
- Non funziona se la rete non è un albero (loop).

The Control Plane

Nei protocolli del mondo reale ci servono algoritmi di routing più complessi.

L'obiettivo del control plane o di un protocollo di routing è quello di costruire una tabella di routing (RT).

Una **tabella di routing** R può essere modellata come una struttura dati che memorizza, per ogni indirizzo di destinazione noto d , i seguenti attributi:

- $R[d].link$: è il collegamento in uscita che il router utilizza per inoltrare i pacchetti verso la destinazione d .
- $R[d].cost$: è la somma delle metriche dei collegamenti che compongono il percorso più breve per raggiungere la destinazione d .
- $R[d].time$: è il timestamp dell'ultimo pacchetto di controllo che trasportava informazioni su d .

Routing VS Forwarding Table

Una tabella di routing è un concetto più astratto di una Forwarding Table. Contiene più informazioni e può esserci più di una tabella di routing nello stesso router.

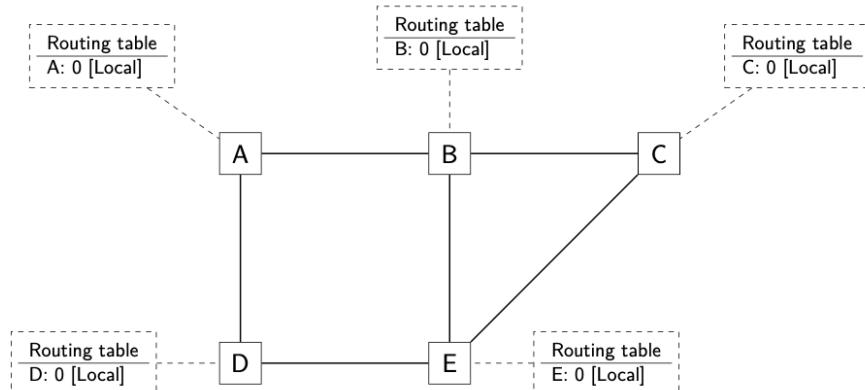
Esempio: una Routing Table può essere gestita dal demone di routing, un'altra può essere configurata manualmente dall'amministratore.

L'obiettivo di un protocollo di routing è quello di riempire le tabelle di routing in uno qualsiasi dei router della rete. In un router tutte le Routing Table vengono fuse in una Forwarding Table, questo è ciò che il router usa per prendere decisioni pacchetto per pacchetto.

Distance Vector Routing

Supponiamo che ogni collegamento abbia un costo. Questa deve essere una penalità per passare attraverso quel collegamento. Nel caso più semplice tutti i collegamenti hanno lo stesso costo unitario, ma è possibile utilizzare metriche più complicate.

Ogni DV router inizializza la tabella di routing R con una voce per il proprio indirizzo, con distanza zero.



Generazione dei DV

Periodicamente un router invia la sua tabella di routing (il **Distance Vector**) ai suoi nodi vicini. L'ordine in cui i vicini ricevono il DV non è definito, può essere qualsiasi ordine.

Ogni router eseguirà lo stesso codice quando genera e riceve un DV da un vicino

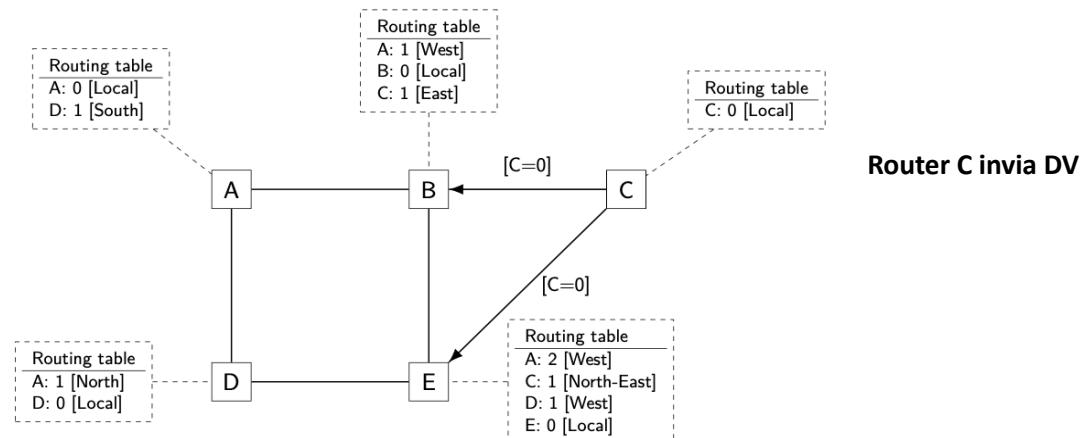
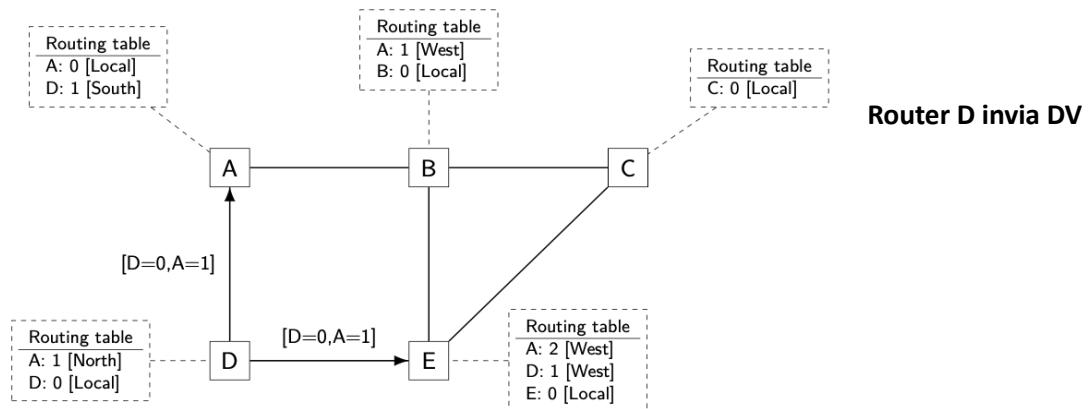
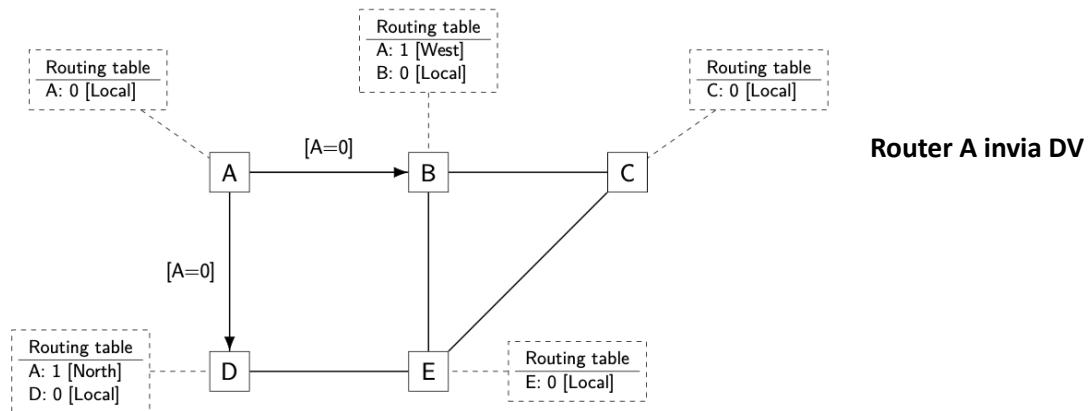
Sending DV (pseudo code)

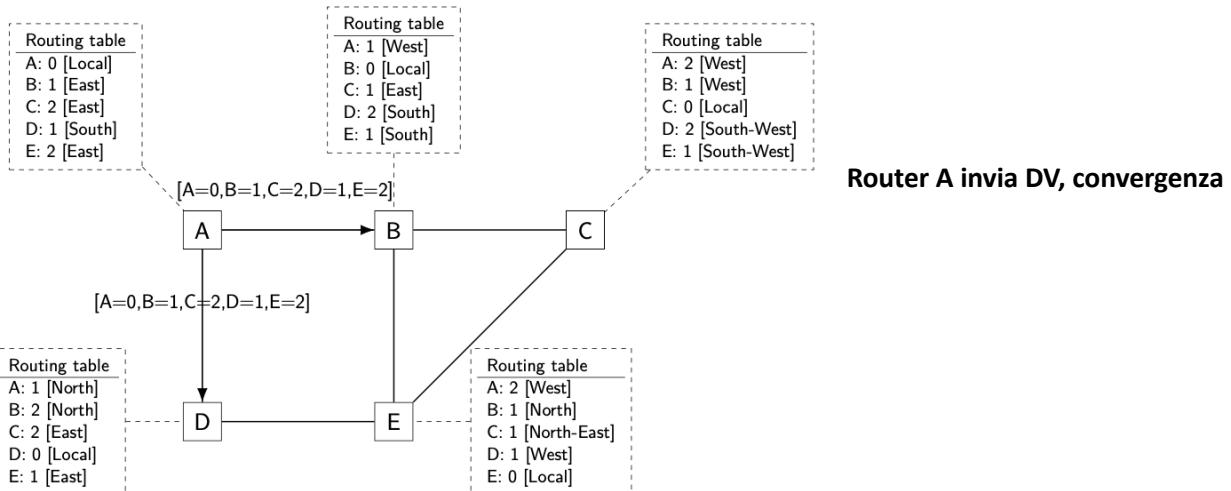
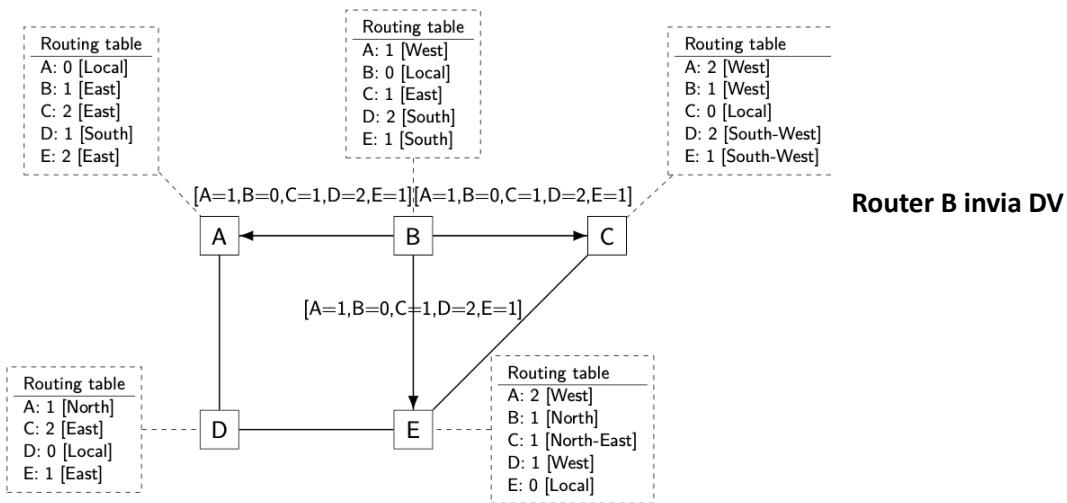
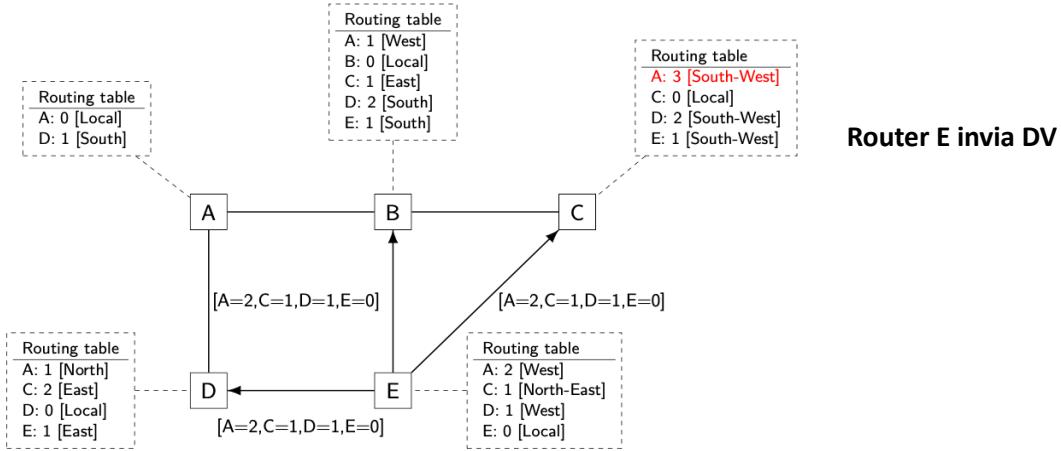
```
1 Every N seconds:
2     v = Vector()
3     for d in R[]:
4         # add destination d to vector
5         v.add(Pair(d, R[d].cost))
6     for i in interfaces
7         # send vector v on this interface
8         send(v, i)
9
```

Receiving DV (pseudo code)

```
1 # V : received Vector
2 # l : link over which vector is received
3 def received(V, l):
4     # received vector from link l
5     for d in V[]
6         if not (d in R[]):
7             # new route
8             R[d].link = l
9             R[d].cost = V[d].cost + l.cost
10            R[d].time = now()
11        else:
12            # existing route, should I update ?
13            if ((V[d].cost + l.cost) < R[d].cost) or (R[d].link == l):
14                R[d].link = l
15                R[d].cost = V[d].cost + l.cost
16                R[d].time = now()
17
```

Esempio:





Failure Recovery

Poiché tutti i router inviano il loro vettore di distanza ogni N secondi, il timestamp di ogni percorso dovrebbe essere regolarmente aggiornato. Pertanto, **nessun percorso dovrebbe avere un timestamp più vecchio di N secondi**. I router hanno un processo che, ogni N secondi controlla il timestamp di tutte le rotte memorizzate nella loro tabella di routing: se viene trovata una rota più vecchia di $3 \times N$ secondi, il suo costo è impostato su ∞ .

Dopo altri $3 \times N$ secondi, *il percorso viene rimosso dalla tabella di routing*, in modo tale che prima di rimuovere il percorso, i router vicini avrebbero dovuto ricevere la cattiva notizia di un **link failure**.

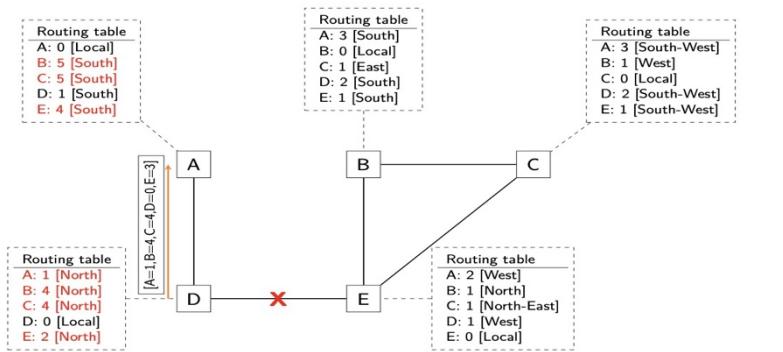
Il valore di 3 usato come moltiplicatore è arbitrario, può differire da protocollo a protocollo.

Il punto è che ci dovrebbe essere abbastanza tempo per garantire che anche se alcuni DV vengono persi a causa di qualche errore di trasmissione temporaneo, la rete converge. Poiché supponiamo che il tasso di errore sui collegamenti sia piccolo, dovrebbe essere sufficiente un piccolo moltiplicatore.

Count-to-Infinity and Poison Reverse

Count-To-Infinity problem

Quando un collegamento si rompe in modo tale da **partizionare la rete**, i router di una partizione si inviano a vicenda i DV all'infinito, questo perché un router aggiorna la sua tabella sempre quando riceve un DV da un router **adiacente** per capire se ci sono cambiamenti nella rete.



In questo esempio si può vedere che entrambi pensano che il modo per arrivare a B,C,E sia tramite l'altro router e viceversa: così facendo le routing table vengono incrementate all'infinito fino a raggiungere l'overflow.

Nota: D non può sapere che per raggiungere E, B, C deve passare per forza da sé stesso.

Split-Horizon with Poison Reverse

Generazione dei DV (variante della precedente)

```

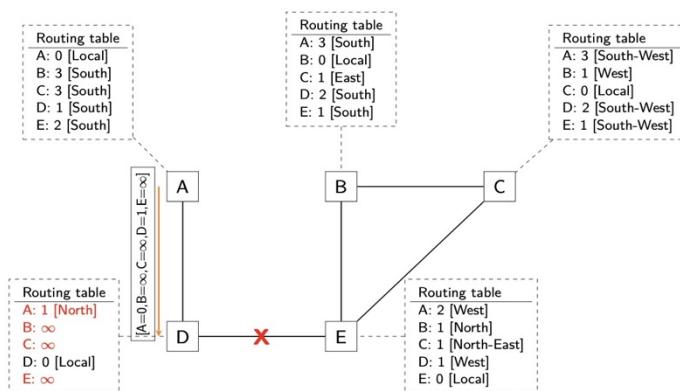
1 Every N seconds:
2   for l in interfaces:
3     # one vector for each interface
4     v = Vector()
5     for d in R []:
6       if (R[d].link != 1):
7         v = v + Pair(d, R[d].cost)
8       else:
9         v = v + Pair(d, infinity)
10    send(v, 1)
11  # end for d in R []
12 # end for l in interfaces
13

```

Split horizon dice che per un router l'insieme dei router con cui comunica non è tutto uguale, è diviso tra quei router che usa come next-hop, e tutti gli altri.

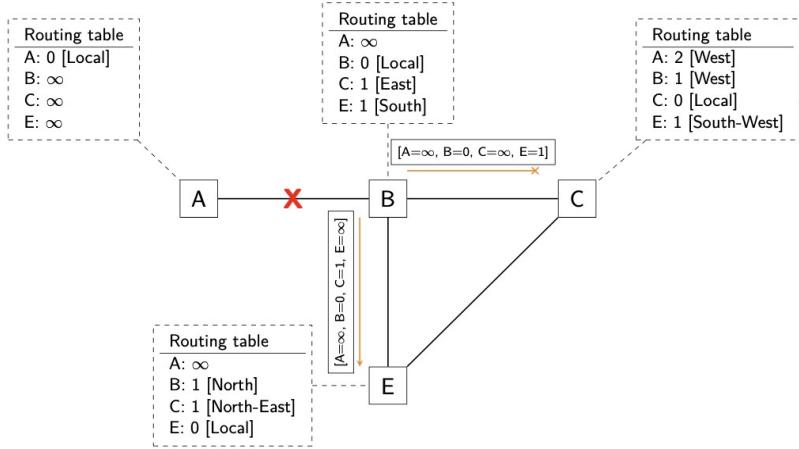
Poison Reverse: il router sta *avvelenando* la tabella di routing del suo vicino, dicendogli che non ha un percorso verso una destinazione, se il percorso passa attraverso il vicino stesso.

In questo modo il problema count-to-infinity è risolto



A manda il DV “avvelenato” a D

Se viene utilizzato Poison Reverse, il router D non aggiornerà la sua tabella di routing. Il router A ha ancora tabelle rotte. Nel prossimo intervallo di tempo, il router D invierà di nuovo il DV. Non appena un DV raggiunge il router A, il router A aggiornerà correttamente la sua tabella di routing.



C non riceve il DV, è la causa del count-to-infinity

Collegamento A-B rotto, il router B invia DV a C e E. tutti i router usano poison reverse. Il router E riceve correttamente DV, mentre router non lo riceve.

Ora se router C invia DV a B e E, senza poison reverse, si verrà a creare un loop in quanto E aggiornerà la sua tabella.

Timer Management

Un modo per evitare questo è quello di prevenire gli aggiornamenti del percorso per un certo tempo dopo. Ciò significa che quando il costo di un percorso è impostato su ∞ , il router non accetta un aggiornamento per un certo periodo di tempo; quindi, aspetta che le cattive notizie vengano propagate a tutti, con probabilità alta che non ci siano perdite di pacchetti.

Questo rende possibile evitare singoli eventi di perdita di pacchetti e fare affidamento sul fatto che le informazioni si propagheranno ai vicini.

Conclusioni sul count-to-infinity

Count-to-infinity può sempre accadere quando ci sono loop e perdita di pacchetti.

Split Horizon e Poison Reverse risolvono il problema solo quando il ciclo è composto da 2 router, ma non quando il ciclo è più grande di 2 router.

Rallentare la convergenza può risolvere la maggior parte dei casi pratici, ma un costo elevato.

Per risolvere il count-to-infinity i router non dovrebbero esportare solo la distanza, ma l'intero percorso verso la destinazione. Questo viene fatto nei protocolli di routing path-vector, come BGP, che vedremo più avanti.

Link State Routing

Il **Link State Routing** è la seconda famiglia di protocolli di routing.

Mentre i Distance Vector router utilizzano un algoritmo distribuito per calcolare le loro tabelle di routing, i Link State Router si scambiano messaggi per consentire a ciascun router di **apprendere l'intera topologia di rete**.

Sulla base di questa topologia appresa, ogni router è quindi in grado di calcolare la sua tabella di routing utilizzando un calcolo di percorso più breve come l'algoritmo di Dijkstra. La rete viene rappresentata come un grafo, i router sono i vertici e i collegamenti sono gli archi. I collegamenti possono essere pesati sui tre fattori:

- Peso unitario, uguale per tutti i collegamenti.
- Peso proporzionale al ritardo di propagazione sul collegamento. Il percorso più breve utilizza i percorsi con il più piccolo ritardo di propagazione.
- $\frac{C}{link_capacity}$ dove C è una costante. Maggiore è la capacità di un collegamento, minore è il peso.

Funzionamento

Ogni router ha un indirizzo univoco.

Un router invia un messaggio HELLO ogni N secondi su tutte le sue interfacce.

I **messaggi HELLO** contengono l'indirizzo del router, servono per capire a chi sono vicini e vengono utilizzati anche per rilevare guasti del collegamento e del router.

Un collegamento è considerato fallito se non è stato ricevuto alcun messaggio HELLO da un router vicino per un periodo di $k \times N$ secondi per qualche costante k.

LSP Messages

Una volta che un router ha scoperto i suoi vicini, deve distribuire in modo affidabile i suoi link locali a tutti i router.

Ogni router costruisce un **Link-State Packet** (LSP) contenente le seguenti informazioni:

- **LSP.Router**: identificazione (indirizzo) del mittente del LSP
- **LSP.page**: età o durata residua del LSP
- **LSP.seq**: numero di sequenza del LSP (incrementato ad ogni LSP)
- **LSP.links[]**: collegamenti comunicati nel LSP, con due campi:
 - **LSP.links[i].id**: id del vicino
 - **LSP.links[i].cost**: costo del collegamento

LSP Flooding

Questi LSP devono essere distribuiti in modo affidabile prima che i router abbiano una tabella di routing funzionante.

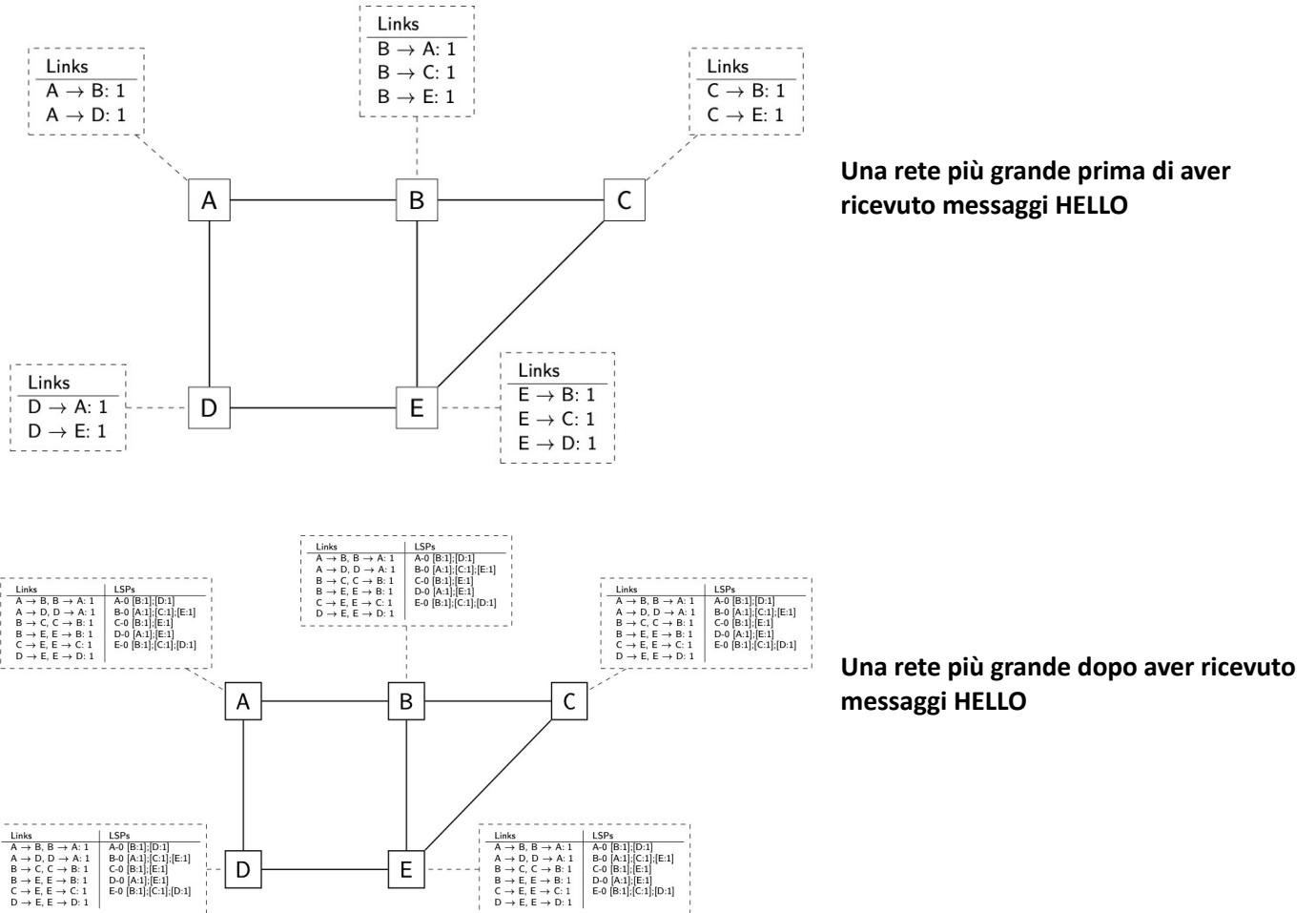
Un **algoritmo Flooding** viene utilizzato per distribuire in modo efficiente gli LSP di tutti i router.

Ogni router che implementa il Flooding mantiene un Link State Database (**LSDB**) contenente l'LSP più recente ricevuto da tutti gli altri router.

Quando un router riceve un LSP:

- verifica prima se questo LSP è già memorizzato all'interno del suo LSDB utilizzando il numero di sequenza:
 - Se è così, il router ha già distribuito l'LSP in precedenza e non ha bisogno di inoltrarlo.
 - Altrimenti, il router inoltra l'LSP su tutti i link tranne il link su cui è stato ricevuto l'LSP.

```
1 # links is the set of all links on the router
2 # Router R LSP arrival on link l
3 if newer(LSP, LSDB(LSP.Router)) : # get last LSP from the DB, compare with current
4     LSDB.add(LSP) # implicitly removes older LSP from same router
5     for i in links:
6         if i!=l:
7             send(LSP,i)
8 # else, LSP has already been flooded
```



Quando un collegamento tra due router si rompe, essi scoprono del danno quando non ricevono più HELLO. Successivamente siccome è cambiato qualcosa nella topologia della rete manda un LSP a tutti i suoi vicini così da informarli, e loro invieranno nuovi LSP di conseguenza.

Nota: dal codice si può vedere che gli LSP vengono mandati quando ne viene ricevuto uno nuovo, questo vuol dire che gli LSP vengono ridistribuiti quando c'è un aggiornamento, anche se non sempre è così: è possibile fare in modo che gli LSP vengano mandati solo quando c'è un aggiornamento oppure che vengano mandati periodicamente.

LS Scalabilità

Nel routing LS anche una singola modifica che non ha alcun effetto pratico oltre un salto verrà propagata a tutta la rete. Tutti i router riapplicheranno l'algoritmo di Dijkstras per niente

Overhead

Il Link State Routing crea molto overhead, immaginiamo di calcolare dijkstra sull'intera rete internet, non è fattibile computazionalmente, su internet si usa BGP. Link State Routing viene utilizzato su reti locali con relativamente pochi router.

Conclusioni

I protocolli LS forniscono maggiori informazioni ai router. Gli algoritmi complessi possono essere utilizzati per costruire le tabelle di routing, non solo quelle del percorso più breve (ad esempio: le metriche moltiplicative possono essere utilizzate per la scelta del percorso o il flusso può essere distribuito in base alla capacità di collegamento).

Quando c'è un cambiamento nella topologia di rete, l'LSP viene inondato rapidamente e la convergenza è veloce.

I messaggi H non vengono propagati in modo che possano essere generati ad alta frequenza, il rilevamento dei guasti del collegamento è più veloce

Il routing LS è computazionalmente più impegnativo del routing DV.

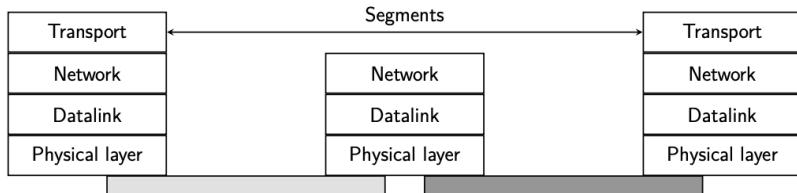
Su Internet ci sono circa centinaia di migliaia di reti di destinazione. L'esecuzione dell'algoritmo di Djikstra su una rete così grande richiede un hardware potente.

Inoltre, una grande rete implica un'alta probabilità che qualcosa si rompa da un momento all'altro. Queste informazioni devono essere propagate a tutto Internet, il che ha due conseguenze:

- Un sacco di inondazioni LSP rispetto a DV. DV può riparare un guasto localmente.
- Ricomputazione costante dei percorsi più brevi

Indipendentemente dal Routing Protocol che viene usato possono sempre avvenire black hole temporanei, questo vuol dire che **il livello rete non può garantire sempre l'arrivo dei pacchetti a destinazione (unreliable)**.

Transport Layer



Un router di internet non ha bisogno di nient'altro che stia sopra al Network Layer. Su internet il network layer fornisce un servizio di consegna degli SDU che è **connectionless** e **unreliable**.

Il **livello di Trasporto**, che sta sopra il network layer, dovrà tenere in considerazione questi due fattori.

Il livello applicazione vuole che il livello di trasporto soddisfi due concetti:

- Garantire reliability.
- Garantire multiplexing: più applicazioni devono poter comunicare senza che i loro dati vengano mischiati.

L'unica cosa che il **livello applicazione** vuole fare per comunicare con il livello di trasporto (che sta sotto) è condividere con esso un **buffer** che contiene uno stream di dati, che poi il livello di trasporto andrà a **segmentare**.

Il livello di trasporto lavora con **segmenti**, mentre al livello di rete si parla di pacchetti e al livello datalink di frame.

Connectionless Transport Layer

Il connectionless transport layer service fornisce:

- Consegna di dati inaffidabile, con error detection.
- Multiplexing.

Normalmente è stateless: ogni segmento viaggia indipendentemente dal precedente. Gli endpoint del livello di trasporto non mantengono alcuna variabile di stato.

È ideale per applicazioni real-time (streaming video, ecc.) perché è inutile farsi rimandare un segmento che contiene un frame che è già passato. Il livello di trasporto **non fa controllo di flusso**, questo vuol dire che se per esempio un video lagga o perde tanti frame perché la connessione fa schifo sarà il livello applicazione a richiedere al server di inviare i dati in una qualità minore.

Pie garantire error detection, viene aggiunto un header a livello trasporto che contiene un **checksum** dei dati.

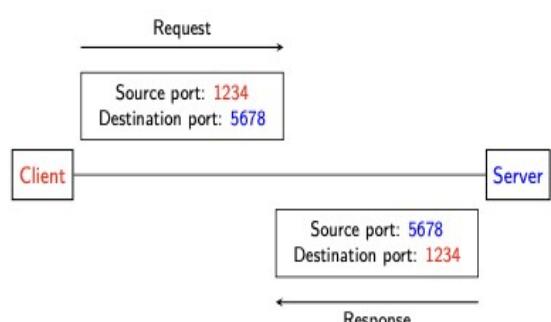
Quando il pacchetto viene ricevuto, il destinatario dovrà calcolare il checksum e verificare la correttezza del segmento.

Porte

L'header del livello di trasporto contiene due porte:

- una identifica l'applicazione sul client
- una identifica l'applicazione sul server

Quando il livello di trasporto riceve il segmento, lo consegna all'applicazione corretta in base al numero di porta

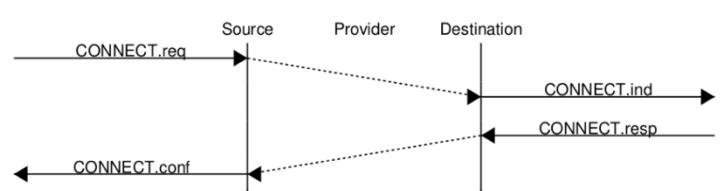


Connection-Oriented Transport Layer

Il servizio crea una connessione tra due entità, normalmente client-server. Tale connessione ha uno stato: alcune variabili interne che vengono utilizzate per tracciare l'evoluzione della connessione.

Prima di inviare i dati, è quindi necessario configurare la connessione. Quando la connessione non serve più viene eliminata. La connessione è associata alle porte come nei servizi connectionless.

Fare il setup di una connessione potrebbe essere facile come lo scambio di due pacchetti, ma il livello di rete è inaffidabile, i pacchetti possono avere dei problemi, ma soprattutto possono essere **duplicati**.

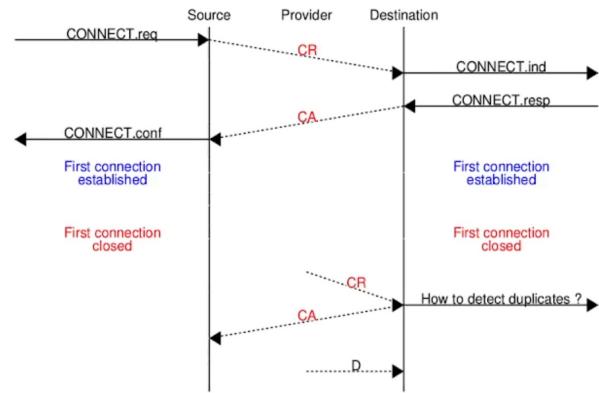


Ci serve un modo per tenere a mente al livello trasporto gli identificativi dei pacchetti, non è possibile tenerli tutti in memoria.

Anche se il livello di rete non è affidabile, dobbiamo fare ipotesi: la duplicazione dei pacchetti può dipendere dai cicli. I loop sono temporanei, quindi definiamo:

Maximum Segment Life (MSL)

Massimo tempo per il quale un segmento (o un ack relativo a quel segmento) può sopravvivere su Internet.
Su internet MSL < 120s (molto minore in realtà)



Transport Clock e ISN

L'idea iniziale era quella di avere un **Transport Clock**, ovvero un clock che ogni livello di trasporto mantiene e viene incrementato con il tempo. Il clock aumenta con il tempo e non deve essere sincronizzato.

Il clock si utilizza per inizializzare un **Initial Sequence Number (ISN)** associato all'inizio di ogni connessione.

In questo modo quando ricevo un ACK posso verificare che sia riferito a una connessione recente e non appartenga a un pacchetto inviato precedentemente (presumibilmente dunque duplicato).

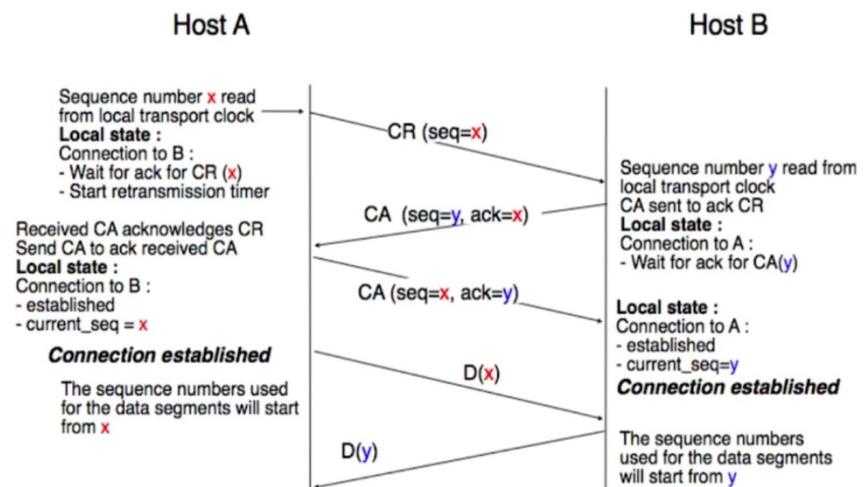
Si consideri il seguente scenario:

- t0 Il client apre una connessione molto breve con ISN=X,
- t1 La connessione termina
- t2 Il client ne apre una nuova con ISN=Y>X verso lo stesso server
- t3 Il client riceve la conferma CA (Connection Accepted) X, cosa farà il client?
- Caso 1: Possiamo supporre che il tempo di wrap up del clock sia molto grande rispetto a MSL, quindi, la CA si riferisce ad una nuova connessione.
- Caso 2: Il tempo di wrap up del clock è paragonabile a MSL; quindi, potrebbe essere che la CA sia una copia della vecchia connessione. questo non deve accadere in quanto abbiamo bisogno che il tempo di wrap up sia molto più grande di MSL.

La soluzione potrebbe essere richiedere al server di fare la stessa cosa che fa il client quindi di usare anch'esso ISN, questo ci porta al **Three Way Handshake**.

Three Way Handshake

- Il CR contiene un numero di porta e un numero di sequenza (seq=x nella figura) il cui valore viene estratto dall'orologio di trasporto.
- Il server elabora il segmento CR e crea lo stato per il tentativo di connessione. Non sa ancora se si tratta di un nuovo tentativo di connessione o di un pacchetto duplicato.
- Restituisce un segmento CA che contiene un numero di riconoscimento per confermare la ricezione del segmento CR (ack=x) e un numero di sequenza (seq=y) il cui valore viene estratto dal suo orologio di trasporto. In questa fase, la connessione non è ancora stabilita.
- Il cliente riceve il segmento CA. Il numero di riconoscimento di questo segmento conferma che l'entità remota ha ricevuto correttamente il segmento CR.
- Si ritiene che la connessione di trasporto sia stabilita dal cliente e la numerazione dei segmenti di dati inizia al numero di sequenza x.
- Prima di inviare segmenti di dati, l'entità di avvio deve confermare i segmenti CA ricevuti inviando un altro segmento CA.



- Il server considera che la connessione di trasporto deve essere stabilita dopo aver ricevuto il segmento che riconosce il suo segmento CA (ora è sicuro che il CR iniziale non fosse un duplicato). La numerazione dei segmenti di dati inviati dall'entità remota inizia al numero di sequenza y.

Reliable Data Transfer

Dal momento che abbiamo i numeri di sequenza ora si possono usare checksum e selective-repeat/go-back-n per rafforzare la corretta consegna dei dati.

È importante sapere che il Livello Trasporto riceve uno stream di byte, e **i numeri di sequenza riferiscono la posizione del byte nella stream**. Questo aiuta quando i dati sono ricevuti in disordine, che normalmente non accade al livello datalink.

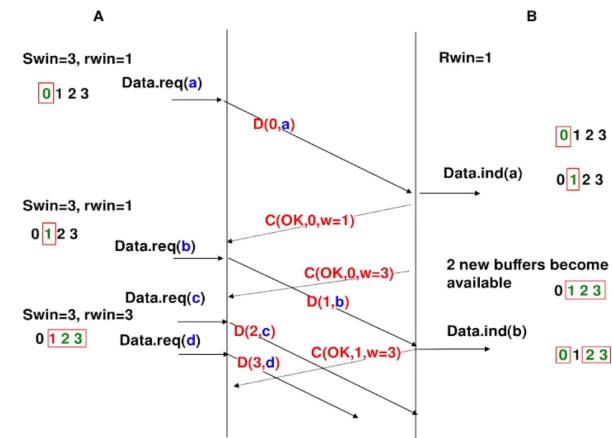
Sliding Window size

Le applicazioni potrebbero non essere abbastanza veloci per ricevere e analizzare tutti i dati inviati, le dimensioni delle finestre di ricezione devono essere negoziabili, chi riceve deve poter comunicare la grandezza della sua finestra e questo avviene comunicandolo negli ACK .

Chi invia i dati deve sempre considerare come limite massimo **$\min(swin, rwin)$** , dove $swin$ è la grandezza della finestra di chi invia e $rwin$ è la grandezza della finestra di chi riceve.

Window size può essere zero se il destinatario è in overload, mettendo così in pausa il mittente e garantendo l'invio di ACK dal destinatario.

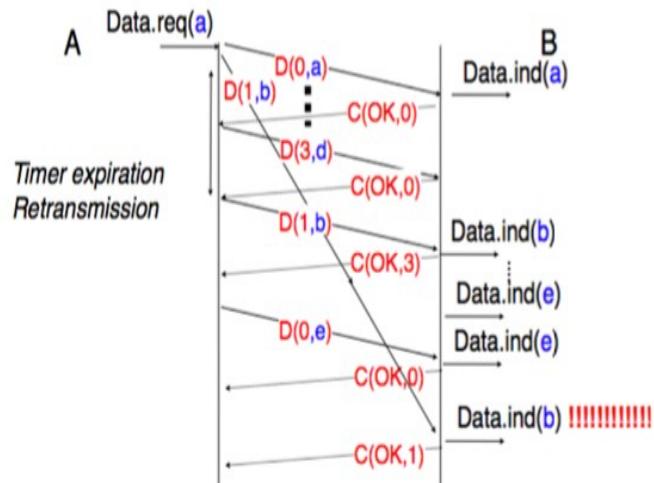
Periodicamente, il mittente rinvia vecchi dati per dare la possibilità al destinatario di inviare un altro ACK che probabilmente si era perso.



MSL e Throughput

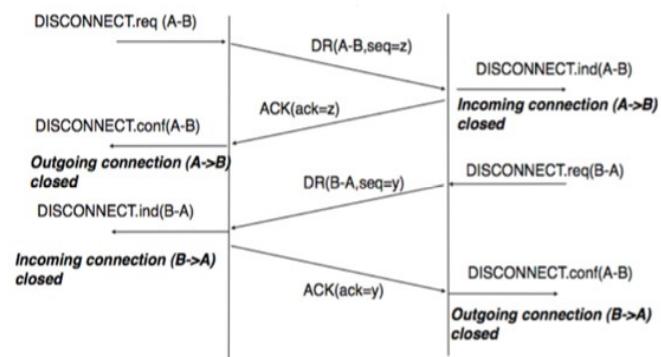
Dobbiamo evitare che durante la connessione i numeri di sequenza facciano wrap-up entro l'MSL, immaginiamo di avere numeri di sequenza da 0 a 3, può verificarsi l'ACK di un pacchetto vecchio.

Dobbiamo fare in modo che prima di 120s (esempio di MSL) non si ripeti un numero di sequenza. Questo ci impone un **limite sul Throughput**: se ho una connessione molto veloce posso fare wrap up prima dei 120s. Nella realtà il MSL è più piccolo, e il problema si pone raramente.



Chiusura della connessione

La connessione in questo modo si chiude normalmente, ma è possibile che un terminale venga spento, il livello di trasporto garantisce **reliability** solo quando la chiusura della connessione avviene correttamente.



Esempio:

Assumiamo MSL = 120s e numeri di sequenza di 32 bit.

Qual è il massimo throughput S per assicurare che non ci siano errori?

$$S < \frac{2^{32} * 8}{120}$$

È possibile avere due o più connessioni aperte tra alcune paia di host?

Sì, gli header hanno ip sorgente e ip destinazione

Cosa avrebbe più impatto su internet tra cambiare il transport layer e cambiare il network layer?

Il transport layer, perchè bisognerebbe cambiare sia client che server.

Application Layer

Le applicazioni sono diverse l'una dall'altra, quindi non è facile standardizzare il loro comportamento. In generale, tendono a utilizzare un modello di richiesta/risposta, diverso per ogni applicazione specifica. Un protocollo a livello applicativo deve specificare la sintassi e la semantica del modo in cui le informazioni vengono richieste e vengono fornite

Il testo è generalmente codificato con **ASCII** o **UTF-8**.

I dati binario potrebbero essere anche scambiati come documenti. Ci sono due principali architetture:

- **Big-endian**: byte meno significativo precede il più significativo (usato in Internet)
- **Little-endian**: byte più significativo precede il byte meno significativo.

Esempio:

256 in esadecimale è 0x0100:

- big-endian: 01 00
- little-endian: 00 01

Header del livello applicazione

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1			
+-----+	First field (16 bits) A B C D E F G H Second		
+-----+	field (24 bits) First Byte Second Byte		
+-----+			

Sicurezza

Cosa dobbiamo garantire in un servizio per quanto riguarda la sicurezza?

- **Availability:** possibilità di accedere sempre al servizio, gli attacchi di Denial Of Service (Ddos, saturazione delle risorse) danneggiano questo aspetto
- **Autenticazione dei dati:** garantire che chi manda i dati sia effettivamente lui e che il ricevitore a cui ci stiamo interfacciando sia quello che crediamo che sia.
- **Integrità dei dati:** i dati ricevuti devono essere gli stessi che vengono mandati.
- **Segretezza**
- **Controllo degli accessi:** l'accesso ai servizi offerti deve essere ristretto a chi è autorizzato
- **Non repudiation:** Impedisce al mittente o al destinatario di negare un messaggio trasmesso. Così, quando viene inviato un messaggio, il destinatario può dimostrare che il presunto mittente ha effettivamente inviato il messaggio. Allo stesso modo, quando viene ricevuto un messaggio, il mittente può dimostrare che il presunto destinatario ha effettivamente ricevuto il messaggio" (PEC).
- **Anonymity,** il sistema non identifica chi invia.

Security Mechanisms: Funzioni che aiutano a garantire in un servizio la sicurezza:

- Crittografia
- Firma digitale
- Protocollo di autenticazione
- ...

Cryptographic algorithm: sequenza di operazioni matematiche applicate ad un messaggio

Cryptographic function: blocco di codice che implementa cryptographic algorithm

Secure protocols: convenzione per scambio di messaggi criptati

Attack source: generato dal ritrovamento delle vulnerabilità di un applicativo

Computationalmente impossibile: si dice che un problema sia computazionale impossibile se non esiste un algoritmo noto di complessità polinomiale per risolvere il problema.

IP Spoofing: Il protocollo IP ti consente di fare lo spoofing dei pacchetti, hacker puoi inviare pacchetti utilizzando gli indirizzi IP del mittente di un host che non è tuo.

Funzioni Hash testo → digest di dimensione fissa (**one way**)

HMAC

L'utilizzo di un hash per garantire l'integrità ha senso se il digest viene inviato su un canale diverso dal canale utilizzato per il messaggio. Gli hash sono efficaci se l'attaccante non può intercettare entrambi i canali.

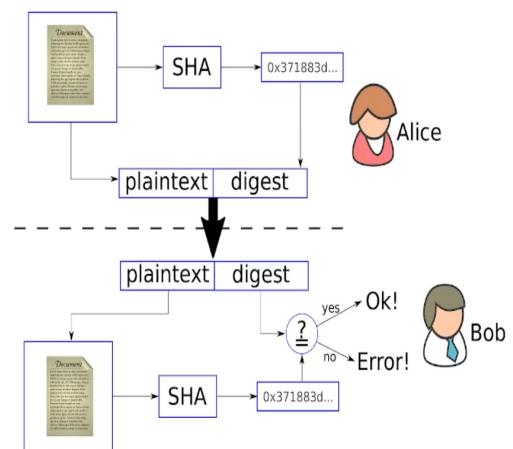
L'esempio di Ubuntu che utilizza il sito web (per fornire l'hash) e Torrent (per fornire l'immagine) è esplicativo: se il sito web è sicuro, l'integrità viene preservata. Ma se l'attaccante può modificare sia il messaggio che il digest (l'immagine di Ubuntu e il sito web), ha poco senso.

Prima delle comunicazioni le due parti devono stabilire una chiave segreta condivisa e una funzione hash.

Garantisce **authentication**, si sa il mittente in quanto solo loro due conoscono la chiave, e **integrity**, HMAC usa un canale di comunicazione per messaggio e digest.

L'assenza di un canale sicuro per condividere la chiave è un problema in quanto potrebbe essere letta da estranei per poi usarla per decodificare.

Una volta che Alice e Bob hanno una chiave condivisa possono utilizzare la stessa chiave per fare Encryption e HMAC: **Encryption Then MAC**.



CRAM-MD5

il protocollo è stato un miglioramento rispetto alla trasmissione in chiaro della chiave segreta. Tuttavia ora è deprecato. Dividiamo le ragioni in tre categorie:

- *Crypto insicure e mancanza di estensibilità*: MD5 non è più considerato sicuro. In generale, non è saggio fare affidamento su una singola funzione di crittografia, ma se il protocollo si basa su di più, c'è la necessità di scambiare alcuni messaggi iniziali per concordare su quale usare. Questo introduce complessità, rallenta il protocollo e può essere ingannato da attacchi di declassamento in cui Eve cambia i messaggi e costringe Alice e Bob a concordare sulla funzione più debole.
- *L'autenticazione non è mutua*: Alice non sa se il server è davvero Bob. Potremmo ripetere il protocollo due volte, invertendo il ruolo del server e del client; quindi, alla fine Alice sa che Bob conosce anche la password. Questo è simile a quello che fa WPA2 nella tua rete wi-fi, ma CRAM-MD5 non lo supporta.
- *Attacco di brute force off-line*: Eve può osservare C e D in chiaro, il che significa che può produrre un attacco di brute force off-line. Dato un elenco di password L.

Per risolvere questi problemi abbiamo bisogno di più criptografia.

Symmetric Key Encryption

Ci sono due elementi protagonisti di questo sistema, algoritmo e chiave. L'algoritmo è pubblico ed è la chiave ad essere segreta. Conoscendo la chiave possiamo cifrare e decifrare documenti ed in alcuni casi può essere anche usata come verifica dell'identità.

funzione unidirezionale applicata ai dati che genera una stringa di dimensione fissa.

Operazioni:

- Per quanto riceve per un HMAC, Alice e Bob usano un canale sicuro per concordare una chiave condivisa K e un codice da utilizzare. Ad esempio, oggi la funzione di crittografia AES è considerata sicura.
 - Quando Alice vuole inviare un messaggio M a Bob, utilizza l'algoritmo AES per creare un messaggio crittografato C. L'input di AES è la chiave K e il messaggio M : C = AES(K,M).
 - Quando Bob riceve il messaggio crittografato C, utilizza la stessa funzione per recuperare M : M = AES(K,C)
- Generalmente l'algoritmo è lo stesso sia per la crittografia che per la decrittazione

Poiché le funzioni hash si basano su XOR e shift, la crittografia si basa anche su due operazioni, sostituzione e trasposizione:

- Gli *algoritmi di sostituzione* sostituiscono un simbolo con un altro.
- Gli *algoritmi di trasposizione* (o permutazione) scambiano l'ordine dei simboli.

Cosa si usa?

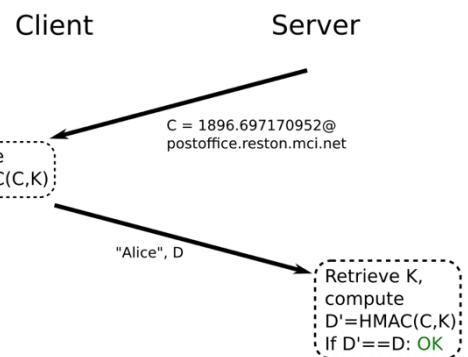
- **MD5** : deprecato, si può rompere
- **SHA256**

Le funzioni hash **ammettono collisioni**, la soluzione per renderle praticamente impossibili è allungare il digest (256 bit in SHA-256).

OTP

L'algoritmo perfetto in crittografia esiste, ma non può essere utilizzato perché richiede una chiave lunga quanto il messaggio.

- Alice genera una stringa casuale di n bit. Questa stringa viene trasmessa a Bob tramite un canale sicuro (un canale che Eve non può intercettare).
- Per crittografare un messaggio M di m < n bit Alice prende una porzione K' di K di lunghezza m e calcola C = M \oplus K'.
- La porzione K' del tasto K originale non deve essere riutilizzata.



- Si può facilmente dimostrare che non esiste una correlazione tra C e M: l'analisi statistica non può essere eseguita.

L'OTP è teoricamente impossibile da rompere.

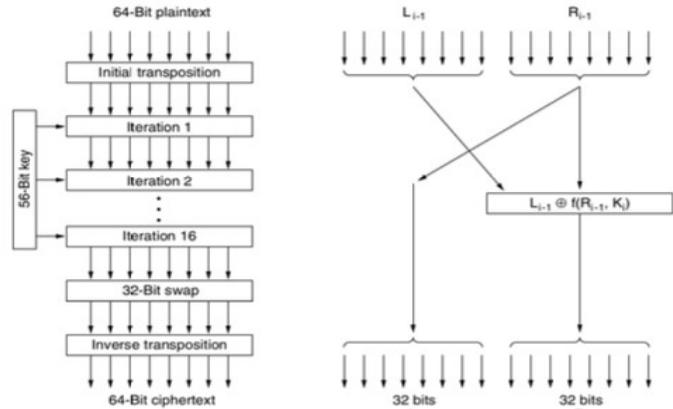
Esempio:

$$K = 1100100111010110, M = 00101101 \\ C = 11001001 \oplus 00101101 = 11100101$$

M non può essere recuperata da C senza conoscere K. Il prossimo messaggio dovrà essere cifrato con la prossima parte di K.

Feistel Scheme

I moderni cifrari a blocchi dividono l'input in blocchi e su ogni blocco applicano una sequenza di trasposizione e sostituzione. Questo si chiama schema Feistel.



Perché si chiama Symmetric Key Encryption?

Abbiamo analizzato la crittografia a chiave simmetrica e ne abbiamo compreso i principi:

- Alice e Bob condividono una chiave segreta utilizzando un canale sicuro.
- La stessa chiave viene utilizzata sia per la crittografia che per la decrittografia (ecco perché si chiama simmetrica)

La crittografia a chiave simmetrica richiede un canale sicuro che garantisca **segretezza e autenticazione**.

Ma è ancora in un miglioramento rispetto alle condizioni normali: hanno bisogno di un canale sicuro solo di tanto in tanto per scambiare e aggiornare una piccola chiave condivisa.

Per tutto il resto del tempo possono scambiare una quantità arbitraria di dati utilizzando un canale insicuro.

Password Files

Per utilizzare il controllo degli accessi è necessario salvare in un database la combinazione nome/password.

La password deve essere salvata in un formato che ti permetta di compararla con l'input dell'utente. Allo stesso tempo non deve essere salvata in chiaro.

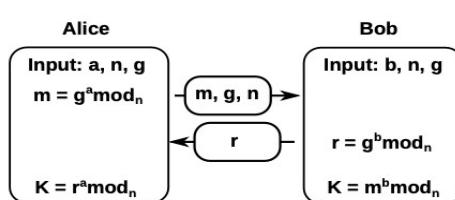
Usare una funzione hash prima di salvare la password è una buona cosa, in quanto può testare i vari input utente ma l'hash non può essere mai invertito.

Public Key Encryption

Con la crittografia a chiave pubblica Alice e Bob possono comunicare in modo sicuro senza scambiare alcuna informazione segreta.

Diffie-Hellman Protocol

DH è un protocollo che consente ad Alice e Bob di negoziare una chiave segreta simmetrica senza scambiare alcuna informazione segreta. Si basa sulla difficoltà di calcolare il logaritmo discreto.



- Alice picks n prime, a random, g is a prime root modulo n .
- Bob generates a random value b
- a and b are secret numbers, they never leave Alice and Bob
- Alice sends n, g, m to Bob
- Bob sends r to Alice
- the generated key is $K = g^{ab} \text{mod } p$

Il problema non è risolto perché Eve può modificare il traffico, e far credere ad Alice di parlare con Bob ma sta parlando con Eve e viceversa. Per questo motivo DH può garantire la **riservatezza** solo se viene utilizzato con qualche altro strumento che garantisce **l'autenticazione**.

RSA

I principi della crittografia a chiave pubblica sono:

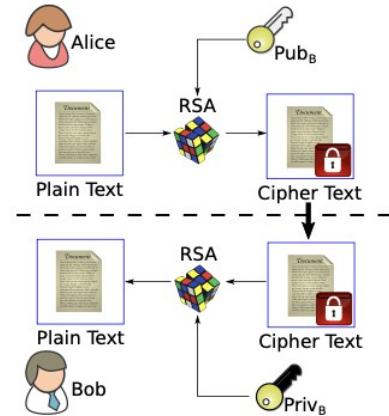
- Alice e Bob hanno **due chiavi ciascuno**.
- Alice possiede una chiave pubblica Pub_A e una chiave privata Priv_A
- Bob possiede una chiave pubblica Pub_B e una chiave privata Priv_B

La chiave privata deve rimanere segreta ed è unica per ogni proprietario.

La chiave pubblica può essere pubblicata ovunque su internet.

Ciò che è crittografato con una chiave pubblica può essere decrittografato solo con la chiave privata corrispondente.

La crittografia a chiave pubblica fornisce segretezza, come crittografia a chiave simmetrica. Il grande vantaggio rispetto agli algoritmi a chiave simmetrica è che non c'è bisogno di un canale segreto per scambiare la chiave. Tutto ciò di cui Alice ha bisogno per crittografare qualcosa a Bob è Pub_B, che è pubblico per definizione.



Operazioni:

La generazione della chiave pubblica e privata è il risultato di un singolo processo matematico, c'è una connessione matematica tra loro. Non sono due numeri casuali come due chiavi simmetriche, non possono essere generati da alcune informazioni note e non puoi generarli indipendentemente l'uno dall'altro.

- Se Bob invia il messaggio M cifrato in modo $C = M^e \text{mod}_n$
- Alice, dopo aver ricevuto il messaggio cifrato C lo decodifica $M = C^d \text{mod}_n$

Esempio:

- Scegliamo $p = 7, q = 11, n = 77, \phi = 60$
- Scegliamo $e = 13$, numero primo e non divisore di 60
- Troviamo ora $d \rightarrow (d * 13) \text{mod}_{60} = 1 \rightarrow d = 37$
- Assumiamo che il messaggio sia $M = 00010100$
- Convertiamo in intero $M = 20$
- Messaggio cifrato $C = 2^{13} \text{mod}_{77} = 69$
- Messaggio decifrato $M = 60^{37} \text{mod}_{77} = 20$

Proprietà:

- È computazionalmente efficiente generare Pub_A, Priv_A.
- Dato Pub_A è computazionalmente efficiente calcolare $C = E(\text{PubA}, M)$.
- Dato Priv_A è computazionalmente efficiente calcolare $M = D(\text{PrivA}, C)$.
- Dato solo Pub_A è computazionalmente impossibile derivare Priv_A
- Dati Pub_A e C è computazionalmente impossibile derivare M.

Possibili attacchi:

Supponiamo che Alice e Bob vogliano scambiarsi due messaggi M0 e M1. La prima cosa che Alice e Bob devono fare è scambiare chiavi pubbliche. Ora considera il caso in cui Eve controlla il canale. Questo può accadere:

- Alice chiede a Bob Pub_B
- Eve intercetta il messaggio, scambia Pub_B con Pub_E
- Alice riceve il messaggio ed è convinta di aver ricevuto la chiave pubblica di Bob
- Ora, quando Alice vuole crittografare qualcosa per Bob, userà la chiave di Eve.
- Eve lo decifrerà, lo crittograferà con Pub_B e lo invierà a Bob.
- Eve può leggere e modificare il messaggio, crittografarlo con Pub_B e inviarlo a Bob. Bob non riesce a rilevare l'attacco riuscito.
- Le stesse cose accadono nella direzione opposta.

L'attacco produce una **perdita totale di riservatezza e integrità**.

La crittografia a chiave pubblica ha dunque bisogno di un **canale autenticato**, la differenza importante è che con la crittografia a chiave simmetrica avevamo bisogno di un canale autenticato ma anche **sicuro**.

Firma Digitale

Se utilizzata con **chiavi invertite**, la crittografia delle chiavi pubbliche e private non è una garanzia di segretezza, ma fornisce l'**autenticazione del mittente**.

Questo uso della crittografia a chiave pubblica è chiamato firma digitale e fornisce l'**autenticazione** dei dati.

Alice critta il messaggio con la sua chiave privata, così tutti possono decifrarlo con la sua chiave pubblica, ma **lei e solo lei può essere il mittente del messaggio**.

Conclusioni

La principale differenza tra la crittografia a chiave pubblica e la crittografia a chiave simmetrica è che la seconda ha bisogno di un canale autenticato, intero e segreto, mentre la prima ha bisogno solo di un canale autenticato e intero.

Shared Key Encryption

Provides:

- Secrecy, Integrity, Authentication

Needs:

- A secret and authenticated (and thus integer) channel to set-up the communication

Public Key Encryption

Provides:

- Secrecy, Integrity, Authentication, **Non-repudiation**

Needs:

- An authenticated (and thus integer) channel to set-up the communication

RSA ha però **problemi di performance**. Cifrare una grossa mole di dati è molto pesante, quello che si fa nella realtà è cifrare le chiavi con RSA comunicando con uno schema a chiave simmetrica o firmare il digest di un contenuto.

Distributing Public Keys QUI

Keyserver

Raccoglitrice di chiavi pubbliche, non è una prova che una chiave appartenga effettivamente ad una persona, è solo un comodo raccoglitrice. [Ubuntu Keyserver](#)

Web Of Trust

Se Alice si fida di Carl e viceversa.

Bob si fida di Carl e viceversa.

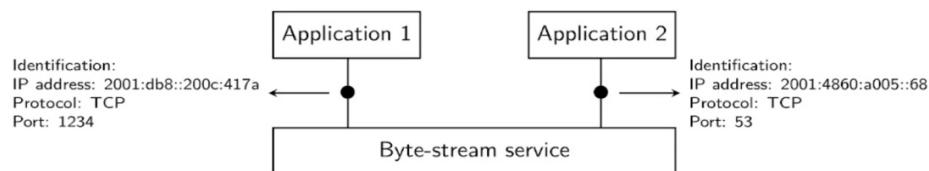
Allora Carl può fornire a Bob

$$S = E(\text{priv}_c, (\text{Pub}_A, \text{"Alice"}))$$

In questo modo Carl certifica a Bob che la chiave pubblica `Pub_A` appartiene effettivamente ad Alice grazie al token `S`.

Domain Name System (DNS)

I protocolli del livello applicazione fanno affidamento sul livello di trasporto, varie applicazioni possono girare sullo stesso server, quello che le differenzia è il numero di porta fornito dal livello di trasporto.



I server sono identificati **univocamente** da un indirizzo IP, ma su internet sono identificati dai domini, questo vuol dire che è necessaria una traduzione.

Dominio

Un nome di dominio è una stringa separata da punti divise da punti, organizzata in maniera gerarchica, da destra verso sinistra. Il livello più alto è chiamato Top Level Domain (TLD), e per molti anni ce ne sono stati solo pochi ammessi. C'è un'organizzazione, ICANN, che è responsabile dell'assegnazione dei domini.

I TLD sono delegati da ICANN ad altre organizzazioni, come gli stati nazionali. Queste organizzazioni poi delegano ulteriormente ad altre organizzazioni, che sono chiamate Domain Name Registrars. I registrar sono normalmente grandi fornitori o società specializzate.

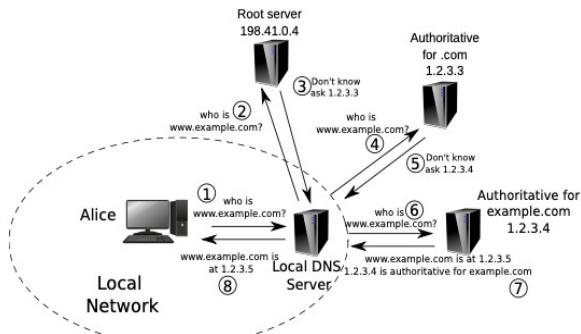
Quando a qualcuno viene assegnato un determinato dominio, è responsabile di tutti i sottodomini. Quindi, se il proprietario di example.com delega my.example.com a qualche altra identità, allora quest'altra identità può estenderlo autonomamente e alla fine delegare di nuovo. Questo rende possibile ridimensionare l'albero dei nomi di dominio senza un controllo centrale.

Un sottoalbero dell'intero albero di dominio è chiamato zona, e parliamo di delega di zona: www.example.com, main.example.com, ns.example.com sono tutti domini nella stessa zona.

Un server autorevole per una zona può anche essere al di fuori della zona stessa.

Accedere a un dominio

- Un browser web vuole navigare su example.com
- Il browser non sa chi è **authoritative** per questo. Il browser conosce invece un elenco dei cosiddetti root server.
- I **root server** sono 13 server sparsi in tutto il mondo i cui indirizzi sono codificati nei browser. I server root forniscono al browser l'indirizzo del server DNS che è authoritative per tutti i Top Level Domain.
- Il browser contatterà il server DNS autorevole per .com e chiederà l'IP di example.com
- Il dominio autorevole risponderà "Non lo so, ma so che 1.2.3.4 è autorevole per questo". Quindi il browser interrogherà 1.2.3.4, che finalmente risponderà che example.com è a 1.2.3.5



Questo processo è molto inefficiente, ci sono molte query da fare e spesso i DNS Server rispondono con un ulteriore dominio dove andare a chiedere informazioni, così facendo sarà necessaria un ulteriore query per tradurre quel dominio in un indirizzo IP.

I server DNS sono ridondanti, se non si ha risposta da uno la richiesta viene inoltrata ad un altro.

L'autoritative, in presenza di più DNNS server, risponderà con uno degli ip. Più richieste potrebbero avere come risposte ip diversi.

DNS Caching

Il DNS Server può mantenere un dominio tradotto in precedenza in memoria, così facendo si ha una ottimizzazione. Quando il server authoritative fornisce una risposta a una richiesta DNS allega un **Time To Live (TTL)**, così da dire al DNS Server per quanto tempo mantenere un indirizzo in cache.

DNS dettagli

DNS usa un livello trasporto connectionless di tipo datagram in quanto le richieste sono tutte atomiche e non ha senso utilizzare un livello trasporto connection-oriented.

Formato del messaggio

Header	
Question	the question for the name server
Answer	RRs answering the question
Authority	RRs pointing toward an authority
Additional	RRs holding additional information

Header pacchetto

- **Transaction ID:** id univoco che fornisce il client che poi il server scriverà nella risposta, serve per filtrare le varie response.
- **Header Flag:**
 - **QR:** Tipo di query, richiesta (0), risposta (1)
 - **AA:** Se il server che risponde è authoritative (1) o meno (0)
 - **RD:** Il client richiede una richiesta ricorsiva (1)
 - **RA:** Se il server supporta la ricorsione.
 - **Z:** inutilizzato, riservato per uso futuro.
 - **RCODE:** codice di risposta, OK (0), errori (1-5).
- I seguenti 4 campi esprimono il numero di domande, risposte, autorità e informazioni aggiuntive nel resto del messaggio. Ognuno di questi elementi è chiamato **Resource Records (RR)** e ha un proprio formato, ho omesso la discussione di alcuni campi per brevità.
 - **Name:** il nome che era stato richiesto.
 - **Type:**
 - A: IPV4 indirizzo di host
 - AAAA: IPV6 indirizzo di host
 - NS: Authoritative Name Server
 - MX: Mail Server
 - CNAME: alias per la stessa risorsa
 - SOA: informazioni generali del proprietario
 - TXT: testo generico
 - **TTL:** Il tempo in cui queste informazioni possono essere memorizzate nella cache, prima che debbano essere interrogate di nuovo. Ci sono due estremi:
 - Se TTL è molto lungo, le cache sono valide per molto tempo e vengono ricevute meno richieste DNS.
 - Se TTL è molto breve, qualsiasi modifica per qualsiasi motivo viene propagata rapidamente.
 - **RDLENGTH:** Lunghezza del campo RDATA
 - **RDATA:** attuale informazione

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5
ID															
QR	Opcode	AA	TC	RD	RA	Z									
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
QDCOUNT															
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
ANCOUNT															
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
NSCOUNT															
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
ARCOUNT															
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5
NAME															
/															
/	TYPE														
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	
/	CLASS														
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	
/	TTL														
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	
/	RDLENGTH														
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	
/	RDATA														
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	

Reverse DNS

Quando registri un dominio, puoi aggiungere un **PTR Resource Record**, in cui specifichi un dominio in una forma molto speciale, come: 5.3.2.1.in-addr.arpa

Il dominio in-addr.arpa viene utilizzato per effettuare le cosiddette ricerche inverse, che consentono alle persone di tradurre gli indirizzi IP in nome di dominio (l'opposto del DNS).

Quando Alice vuole sapere se c'è un dominio associato a 1.2.3.5, farà una query DNS per cercare 5.3.2.1.in-addr.arpa, e questo fornirà l'elenco dei domini ad esso associati (nota l'inversione dei numeri).

PTR RR non sono obbligatori, è facoltativo aggiungerli e rendere il tuo indirizzo IP ricercabile in query inverse.

WHOIS protocol

Il protocollo WHOIS (ora aggiornato dal protocollo RDAP, ma ci riferiamo ancora ad esso come WHOIS) consente di accedere alle informazioni associate a un determinato dominio.

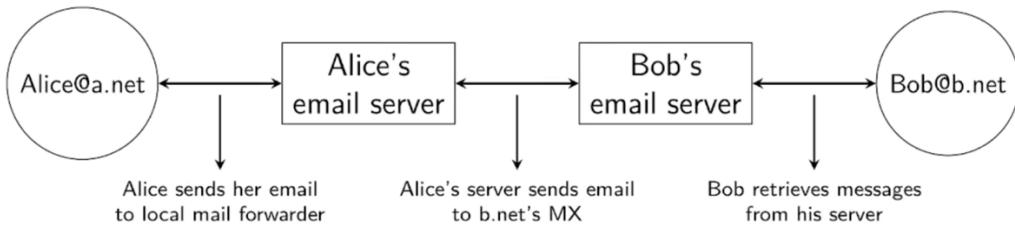
Una query WHOIS chiede al registro regionale (RIR) di fornire l'indirizzo del registrar responsabile di un dominio, che viene quindi interrogato per fornire tutti i dati utilizzati per la registrazione (proprietario, e-mail di contatto, data di scadenza...).

\$ whois unive.it	Name: Stefano Claut Organization: Universita di Venezia Address: Universita Ca Foscari Dorsoduro 3861 Venezia 30123 VE IT Created: 2022-10-05 11:12:53 Last Update: 2022-10-05 11:12:53
Domain: unive.it Status: ok Signed: no Created: 1996-01-29 00:00:00 Last Update: 2023-02-14 01:04:59 Expire Date: 2024-01-29	
Registrant Organization: Universita degli Studi di Venezia Address: Area Servizi Informatici e Telecomunicazioni Venezia 30123 VE IT Created: 2007-03-01 10:27:43 Last Update: 2022-01-28 11:20:26	
Admin Contact Name: hidden Organization: hidden	Registrar Organization: Consortium GARR Name: GARR-REG Web: http://www.garr.it DNSSEC: no
Technical Contacts Name: Alvise Rabitti Organization: Universita degli Studi di Venezia Address: Area Servizi Informatici e Telecomunicazioni Venezia 30123 VE IT Created: 2022-01-28 11:14:40 Last Update: 2022-01-28 11:14:40	Nameservers algol.unive.it vega.unive.it dns.cineca.it ns1.garr.net

Sicurezza

Il DNS è un servizio critico dal punto di vista della sicurezza. Se un utente malintenzionato può modificare le risposte di una query DNS, può avere conseguenze nefaste sulla privacy degli utenti.

Electronic Mail (e-mail)



L'e-mail ovviamente non va direttamente da un computer all'altro, ci sono diversi componenti coinvolti:

- Un client di posta elettronica, che formatta il messaggio nel modo corretto.
- Il server di posta elettronica di Alice a cui il client di posta elettronica di Alice invia l'e-mail.
- Il server di posta elettronica di Bob che riceve l'e-mail dal server di e-mail di Alice.
- Il client di posta elettronica di Bob che riceve l'e-mail dal suo server.
- Quando si utilizza un servizio di posta elettronica web, la pagina web in esecuzione su un server web sostituisce il client di posta elettronica.
- La catena di server può essere più lunga di soli due.

Protocolli necessari

Abbiamo bisogno di tanti protocolli per definire le mail:

- RFC5322: **Internet Message Format**, specifica come il messaggio dell'e-mail deve essere formattato. Questi formati sono estremamente ricchi e complicati, molto più dei messaggi DNS.
- RFC2045: Multipurpose Internet Mail Extensions (**MIME**), specifica come il messaggio dell'e-mail deve essere formattato.
- RFC5321: Simple Mail Transfer Protocol (**SMTP**), specifica il protocollo necessario per consegnare il messaggio dal client di posta elettronica di Alice al server di Bob.
- RFC1939,9051: Post Office Protocol, Internet Message Access Protocol (**POP,IMAP**), specificano come Bob può recuperare le e-mail dal suo server.

Formato delle email

Le e-mail hanno alcune righe iniziali che sono **header** e poi più righe che compongono il corpo del messaggio. Le intestazioni valide sono **From, To, Date, CC, BCC, Subject**....

```
1 From: Bob Smith <Bob@machine.example>
2 To: Alice Doe <alice@example.net>, Alice Smith <Alice@machine.example>
3 Subject: Hello
4 Date: Mon, 8 Mar 2010 19:55:06 -0600
5 This is the "Hello world" of email messages.
6 This is the second line of the body
```

Altri header rilevanti sono:

- **Message-Id**: questo è un ID univoco che viene aggiunto all'e-mail e viene utilizzato da altre e-mail che si riferiscono a questo messaggio
- **In-reply-to**: viene utilizzato quando si risponde a un messaggio precedente. Questo file contiene l'ID del messaggio di quello a cui sta rispondendo
- **Received**: poiché il messaggio passerà attraverso una sequenza di server prima di essere consegnato, ciascuno di essi può aggiungere una riga Received. Viene utilizzato per scopi di debug.
- **X-**: il client o il server possono aggiungere qualsiasi campo personalizzato che inizia con X-.

Protocollo MIME

Inizialmente le mail prevedevano solo testo nel body del messaggio, dall'introduzione di Multipurpose Internet Mail Extensions (**MIME**) rende possibile suddividere il corpo del messaggio in diverse sezioni per inviare file binari.

- **MIME Version**: il numero di versione del protocollo (1.0 è il più recente)
- Tipo di contenuto:
- **text/plain**: messaggi di testo semplici
- **multipart/mixed**: messaggi di testo semplici + altri contenuti non testuali
- **multipart/alternative**: stesso messaggio nel testo e in qualche altro formato

- **image, audio, video...** : formati binari.

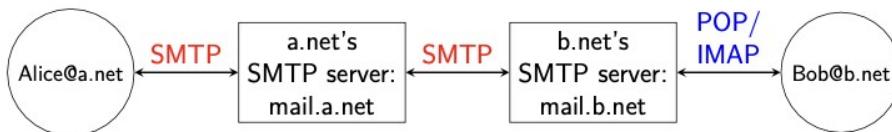
Quando il tipo di contenuto è multipart, l'header specifica anche un confine per dividere le sezioni.

- **Content-Transfer-Encoding:** ASCII, UTF-8, Base64... Questo può essere ripetuto in ogni sezioni.

Protocollo SMTP

Il formato di un'e-mail è principalmente rilevante per gli endpoint del trasferimento di e-mail, chiamati **Mail User Agent (MUA)**.

I **Mail Transfer Agent (MTA)** sono invece i server che si occupano della consegna. Dobbiamo ancora sapere come avviene la comunicazione tra di loro. Per questo abbiamo bisogno di due protocolli, SMTP e POP/IMAP



SMTP è un protocollo basato sul testo, che fa affidamento su un byte stream service (connection-oriented TCP). I server ascoltano sulla porta 25, ricevono comandi ASCII terminati da CRLF e rispondono con linee ASCII contenenti 3 cifre di errore/ successo.

Per inviare un'e-mail è necessario utilizzare i comandi. I più usati sono **EHLO:**, **MAIL FROM:**, **RCPT TO:**

I **codici di risposta** sono raggruppati in base al loro numero iniziale:

- 2XX: risultato positivo.
- 3XX: risultato positivo ma più input richiesto
- 4XX: risultato negativo, ma il problema è transitorio, riprova con lo stesso comando
- 5XX: risultato negativo, il problema è permanente, non riprovare

Operazioni

Alcune note pratiche su SMTP. Queste non sono regole imposte dal protocollo, sono configurazioni comunemente usate.

- Il MUA deve essere configurato con un nome di dominio di un server SMTP, nell'immagine, Alice configurerà il suo MUA per utilizzare mail.a.net come server SMTP.
- Il MUA si conserverà e utilizzerà alcuni comandi speciali per eseguire un'autenticazione.
- In questo caso, il client può inviare e-mail a qualsiasi dominio. Diciamo che il server SMTP consente il **relaying** di e-mail ad altri domini, ad esempio bob@b.net
- Quando il server SMTP di a.net ha ricevuto l'e-mail dal MUA, deve consegnarla al server SMTP del dominio di destinazione: b.net.
- Tuttavia, il server non sa qual è l'IP del server di posta che gestisce le e-mail con il dominio di destinazione b.com.
- Questo potrebbe essere qualsiasi dominio del mondo. Quindi farà una **risoluzione DNS** utilizzando il tipo MX al suo server DNS e riceverà l'indirizzo IP di mail.b.com.
- Il server di a.com si conserverà ora (come client) al server SMTP mail.b.net. Questo è lo stesso processo che il MUA ha fatto nei confronti di mail.a.net
- Tuttavia, l'MTA non esegue alcuna autenticazione, quindi il server di b.net non consentirà l'inoltro: sarà consentita solo la posta elettronica destinata a b.com.
- Alla fine, l'e-mail è stata consegnata al server su b.com. L'e-mail verrà salvata nella mailbox dell'utente, che sarà in grado di recuperarla con un altro protocollo: POP (o IMAP).

Autenticazione

```
1 S: 220-smtp.example.com ESMTP Server
2 C: EHLO client.example.com
3 S: 250-smtp.example.com Hello client.example.com
4 S: 250-AUTH DIGEST-MD5 CRAM-MD5
5 S: 250-ENHANCEDSTATUSCODES
6 S: 250 STARTTLS
7 C: AUTH CRAM-MD5
8 S: 334 PDQxOTI5NDIzNDEuMTI4Mjg0NzJAc291cmNlZm91ci5hbmcRyZXcuY211LmVkdT4=
9 C: cmpzMyB1YzNhNTlmZWQzOTVhYmExZWM2MzY3YzRmNGIOMWFjMA==
10 S: 235 2.7.0 Authentication successful
```

Inizialmente, non c'era l'autenticazione per SMTP, quindi sono state aggiunte estensioni al protocollo per supportare l'autenticazione.

Un server che supporta l'autenticazione aggiungerà più 250- righe come risposta EHLO, con un elenco di metodi di autenticazione supportati (linea 4-6). Il client può sceglierne uno (linea 7).

CRAM-MD5 sta per Challenge-Response Authentication Mechanism, basato su MD5. È uno dei tanti metodi di autenticazione che abbiamo, in questo caso la riga 8 contiene un timestamp (codificato in base64), la riga 9 contiene il nome utente, concatenato con l'hash MD5 della password dell'utente e il timestamp. L'utilizzo di MD5 consente di mostrare al server che il client conosce la password, senza inviarla in chiaro. Questo è ovviamente un po' rudimentale, al giorno d'oggi usiamo il protocollo TLS che aggiunge un livello di crittografia in cima a questo.

Protocollo POP

POP è un altro protocollo testuale, con i suoi comandi. I comandi pertinenti sono USER (seguito dal nome utente), PASS (seguito dalla password), STAT (chiede lo stato della cassetta postale), LIST (elenca nuovi messaggi), RETR (seguito da un numero, recupera il n-esimo messaggio).

Il server risponde con +OK o codici di errore simili a SMTP.

Si noti che nel protocollo originale l'autenticazione è stata eseguita nel chiaro

```
1 S: +OK POP3 server ready
2 C: USER alice
3 S: +OK
4 C: PASS 12345pass
5 S: +OK alice maildrop has 2 messages (620 octets)
6 C: STAT
7 S: +OK 2 620
8 C: LIST
9 S: +OK 2 messages (620 octets)
10 S: 1 120
11 S: 2 500
12 S: .
13 C: RETR 1
14 S: +OK 120 octets
15 S: <the POP3 server sends message 1>
16 S: .
17 C: DELE 1
18 S: +OK message 1 deleted
19 C: QUIT
20 S: +OK POP3 server signing off (1 message left)
```

Combattere lo SPAMs

Normalmente lo spam non avviene da MUA, quindi da computer zombie, perché sono facili da combattere, se avviene uno spam avviene da un server SMTP controllato da qualcuno che bombarda un altro server SMTP come quello di gmail per spammare a tutti i suoi account.

Sono state proposte diverse tecniche per ridurre lo spam, ma nessuna di esse risolve completamente il problema, ma i server possono usarne molte per dare un punteggio a un'e-mail e decidere se è spam o meno, e possibilmente rifiutare l'e-mail.

IP blacklisting

Il server può fare la risoluzione di un nome di dominio e controllare se l'IP appartiene a una blacklist, e non accetterà mail da quell'IP.

Se finisci in una blacklist è un macello uscirne, può capitare che se ti bucano il sito inizi a spammare mail a manego e ti blacklistano.

Diverse organizzazioni mantengono elenchi aggiornati di indirizzi IP che sono stati trovati a generare messaggi di spam; quindi, i server testano l'IP rispetto agli elenchi. Esiste un protocollo convenzionale basato su DNS, che si chiama DNSBL (DNS Block List, storicamente questo era chiamato Realtime Block List, RBL).

Prima di consentire l'invio di un'e-mail, l'MTA interrogherà un server DNSBL per verificare se l'IP è presente nel database. Se l'IP è presente, necherà l'e-mail.

Spamhaus è uno dei fornitori di servizi più noti per DNSBL.

Il processo di ingresso (e uscita) da una lista nera non è davvero trasparente, e gli spammer possono affittare temporaneamente e utilizzare grandi insiemi di IP e poi cambiarli del tutto. Quindi questa tecnica non funziona da sola.

FQDN check

È possibile che qualche MTA faccia la risoluzione inversa, quindi da dominio a ip (vedi dns).

In questo modo viene controllato che l'ip con cui ti presenti sia lo stesso associato al dominio, se così non fosse è probabile che tu venga blacklisted e il messaggio droppato.

È praticamente impossibile mettere su un MTA senza un IP pubblico associato ad un nome di dominio.

RFC Compliance

Tutte le volte che un server SMTP riceve un errore 4xx (riprova più tardi) riproverà, se quel server appartiene ad uno spammer molto probabilmente non riproverà perché ha bisogno di velocità e di spammare a più non posso.

Questo si chiama **greylisting**, il server mantiene una lista dei server con cui ha già comunicato, se un server è nuovo gli farà rimandare la mail, così da sgamarlo se è un sudicio spammer.

È assolutamente non costoso perché il costo è a carico del mittente, ma sicuramente rallenta, è particolarmente rilevante per i messaggi di reset della password.

MTA di Google è whitelisted perché manda costantemente e-mail. MTA che mandano raramente e-mail devono reinviarle.

Sender Policy Framework (SPF)

- Il comando EHLO è seguito da un nome di dominio.
- Il MAIL FROM è seguito da un'e-mail
- l'intestazione From nel corpo contiene anche un'e-mail

Qual è il loro vero significato?

- **EHLO**: identifica il nome di dominio dell'MTA, cioè il server SMTP che funge da client, in pratica è come ti presenti.
- **MAIL FROM**: Questo non è necessariamente il mittente dell'e-mail. Questo è dove inviare gli errori, che potrebbe essere un indirizzo e-mail diverso da quello dell'utente che ha generato l'e-mail.
- **FROM**: Questo è ciò che viene mostrato all'utente ricevente come mittente dell'e-mail e l'indirizzo utilizzato quando si utilizza Reply-to.

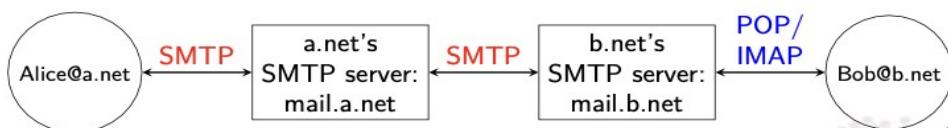
```
1 S: 220 smtp.example.com ESMTP MTA information
2 C: HELO mta.example.org
3 S: 250 Hello mta.example.org, glad to meet you
4 C: MAIL FROM:<alice@example.org>
5 S: 250 Ok
6 C: RCPT TO:<bob@example.com>
7 S: 250 Ok
8 C: DATA
9 S: 354 End data with <CR><LF>.<CR><LF>
10 C: From: "Alice Doe" <alice@example.org>
11 C: To: Bob Smith <bob@example.com>
12 C: Date: Mon, 9 Mar 2010 18:22:32 +0100
13 C: Subject: Hello
14 C:
15 C: Hello Bob
16 C: This is a small message containing 4 lines of
text.
17 C: Best regards,
18 C: Alice
19 C:
20 S: 250 Ok: queued as 12345
21 C: QUIT
22 S: 221 Bye
```

Qual è il pericolo in ottica di spam?

L'MTA del server su b.net riceve un'e-mail dall'MTA client su a.net. Questo può utilizzare un determinato dominio nel campo EHLO, MAIL FROM o From come alice@a.net o alice@c.net.

Un singolo client SMTP può inviare legittimamente e-mail da più domini.

Ad esempio, quando registri un nome di dominio, spesso il provider ti dà anche una serie di indirizzi e-mail che puoi usare. L'MTA del provider invia e-mail utilizzando molti domini diversi. Quindi, l'MTA di b.net non può rilasciare un'e-mail perché il campo FROM: non corrisponde al dominio utilizzato nell'EHLO



Questo è normalmente usato dagli spamer/truffatori per ingannare le loro vittime, perché un'e-mail è più credibile se proviene da un dominio noto. Spamer/e-mail truffa, infatti, normalmente non chiedono di rispondere all'e-mail, ma di fare clic su qualche link contenuto nel corpo dell'e-mail.

Consideriamo un sottodominio (non esistente) di unive.it, ad esempio computer-networks.dais.unive.it. Questo dominio può essere delegato a una sotto parte di un'organizzazione, ad esempio il professore del corso di reti. Quel professore lo usa per fornire materiale agli studenti, tuttavia, è pigro e non aggiorna Wordpress. Qualche utente malintenzionato prende il controllo del server, può impostare un MTA e ora può inviare e-mail da un nome di

dominio valido all'interno di unive.it, un'organizzazione rispettabile che non fa parte delle blacklist. **Sender Policy Framework (SPF)** aiuta a prevenire questo scenario.

SPF protegge **EHLO** e **MAIL FROM** funziona in combinazione con il protocollo DNS. Non protegge il campo From. Una entry SPF è una entry TXT associata al nome di dominio di un'organizzazione che specifica quali sono gli indirizzi IP autorizzati a inviare e-mail.

Quando usi SPF la entry ha un formato simile a questo: *unive.it. 86400 IN TXT "v=spf1 ip4:17.18.7.120 -all"*

Ciò significa:

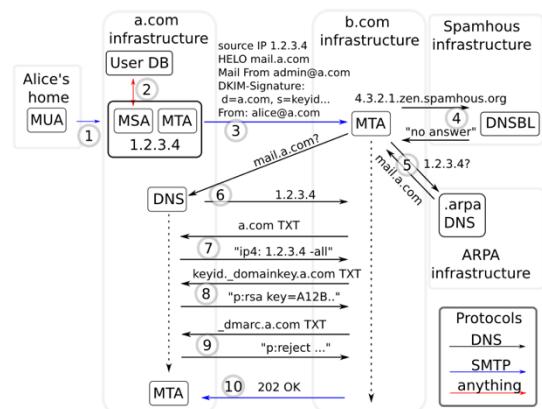
- la versione del protocollo è la versione 1
- un indirizzo IP è esplicitamente autorizzato a inviare e-mail - tutti gli altri sono vietati

DKIM

Protocollo che si concentra tra alcuni headers/body di e-mail e un MTA valido. Opera come SPF solo che è criptato. Richiede MSA, autenticazione in più passaggi, per possedere una coppia di chiavi pubbliche/private. Pone la firma digitale sull'e-mail, la quale potrà essere verificata tramite chiave pubblica.

Esempio SMTP

- Alice usa SMTP con il suo MSA locale, vuole inviare un'e-mail da alice@a.com a bob@b.com.
- L'MSA autentica Alice con alcune credenziali.
- L'MTA in a.com utilizza SMTP per consegnare un'e-mail, utilizzando i campi HELO, Mail From, From appropriati e DKIM-Signature. L'IP sorgente è 1.2.3.4.
- 1.2.3.4 viene testato contro Spamhaus DNSBL con una query DNS per 4.3.2.1.zen.spamhous.com (o servizio simile).
- Il server MTA in b.com effettua una query DNS inversa per l'indirizzo IP del client MTA.
- L'MTA del server risolve mail.a.com, controlla che entrambe le risoluzioni siano coerenti.
- L'MTA del server interroga il DNS di a.com per una voce TXT, controlla che la politica SPF consenta alla 1.2.3.4 di inviare e-mail per conto di a.com.
- L'MTA del server interroga il DNS di a.com per un sottodominio _domainkey, riceve la chiave DKIM, controlla la firma e-mail.
- Il server MTA interroga il DNS di a.com per un sottodominio _dmarc, controlla quali sono le politiche che devono essere rispettate.
- Se tutte le politiche sono rispettate, l'e-mail è accettata



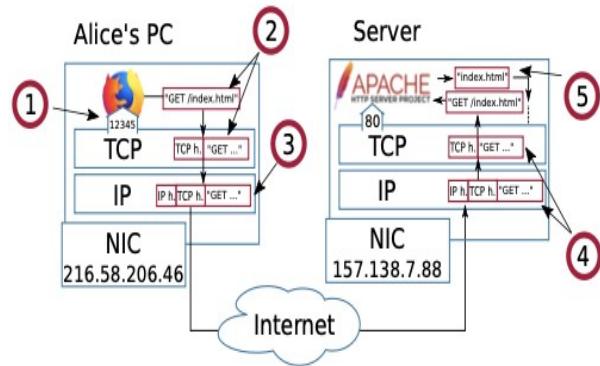
World Wide Web (WWW)

PC di Alice:

- Il browser assegna una porta dal livello TCP, il server web ottiene la porta 80
- Il browser emette un comando HTTP (ne parleremo più avanti), è incapsulato in un segmento TCP
- Questo viene incapsulato in un segmento IP

Server:

- I livelli IP e TCP lo decapsulano
- Il comando viene ricevuto dal server web, che elabora e risponde



Universal Resources Identifier (URI)

Un **URI** è una stringa di caratteri che identifica in modo inequivocabile una risorsa sul world wide web.

Un **URL (Universal Resources Locator)** è un URI che fornisce la posizione per recuperare una risorsa, contrariamente a un URN che è un URI che identifica semplicemente il suo nome e non la sua posizione.

Un formato (semplificato) di un URI/URL è il seguente:

$$URI = \text{scheme} ":" "// \text{authority path} ["?" \text{query}] ["#" \text{fragment}]$$

- Scheme:** è il protocollo che viene usato, alcuni scheme validi sono http, https, ftp, mailto.
- Authority:** comprende il nome di dominio o l'indirizzo ip, può essere preceduto da alcune informazioni sulle credenziali dell'utente prima del carattere @. Il nome dell'host può essere seguito dai due punti (:) e da un numero di porta. La porta dovrebbe essere inclusa solo se lo schema non ne impone implicitamente una.
- Path:** è strutturato come nomi di file su un host Unix (ma non implica che i file siano effettivamente memorizzati in questo modo sul server). Se il percorso non è specificato, il server restituirà un documento predefinito.
- Query e Fragment:** vengono utilizzati per fornire un parametro di query, che verrà analizzato dal codice sul server ricevente (? Parameter=value) e per indicare un frammento specifico nel documento #SubsectionTitle a cui puntare.

ftp://cnn.example.com&story=breaking_news@10.0.0.1/top_story.html

In questo esempio quello che potrebbe sembrare un authority path valido è il nome utente.

HyperText Markup Language (HTML)

Inizialmente gestito da IETF, ora dal consorzio W3C (IETF + altre aziende big).

Pagina HTML

Una pagina HTML è solo un file, formato da header, body e include i rispettivi tag. All'interno della pagina il testo può essere formattato utilizzando i tag, inclusa la presenza di oggetti esterni. Questi oggetti esterni saranno referenziati dai loro URL e saranno caricati separatamente dal browser, quando rende la pagina.

Pagine dinamiche HTML

Una pagina HTML è solo un file, quindi se vuoi avere una pagina con contenuti dinamici, ci sono due modi:

- Server Side Code**, aggiungere il codice lato server, che fa cambiare la pagina quando la visiti, in base a ciò che hai fatto prima o in base alla tua query. Ciò significa che l'URL non punta a un file HTML, ma a un file in qualche lingua che viene eseguito sul server e restituirà un file HTML al browser.
- Client Side Code**, programmare client side significa mettere dentro le pagine HTML del codice Javascript che viene eseguito dal browser.

Tabelle di stile CSS

Il Cascade Style Sheet CSS è un altro standard che consente di regolare graficamente i contenuti della tua pagina web. Fornisce il disaccoppiamento tra dati e stile.

Protocollo HTTP

Il terzo componente del world wide web è l'HyperText Transfer Protocol (HTTP), un protocollo basato su testo come SMTP. Il client invia una richiesta e il server restituisce una risposta. HTTP viene eseguito sopra il servizio bytestream (TCP) e i server HTTP ascoltano per impostazione predefinita sulla porta 80.

Ogni richiesta HTTP contiene tre parti:

- un metodo, che indica il tipo di richiesta, un URI e la versione del protocollo HTTP utilizzato dal client.
- un'intestazione, che viene utilizzata dal client per specificare parametri opzionali per la richiesta. Una linea vuota viene utilizzata per contrassegnare la fine dell'intestazione.
- un documento MIME opzionale allegato alla richiesta

La risposta inviata dal server contiene anche tre parti:

- una riga di stato, che indica se la richiesta è andata a buon fine o meno.
- un'intestazione, che contiene ulteriori informazioni sulla risposta. L'intestazione della risposta termina con una riga vuota.
- un documento MIME

Metodi

- **GET**: utilizzato per recuperare un documento da un server. È seguito dal percorso dell'URI del documento richiesto e dalla versione di HTTP utilizzata dal client.
- **HEAD**: consente il recupero delle righe di intestazione per un determinato URI senza recuperare l'intero documento. Può essere utilizzato da un client per verificare se esiste un documento per esempio.
- **POST**: utilizzato da un client per inviare un documento a un server. Il documento è allegato alla richiesta HTTP come documento MIME.

HTTP header

- **user-agent**: info sul software in esecuzione nel client. Usato per fornire diverse versioni del sito.
- **referrer**: sito precedentemente visionato dal client.
- **host**: dominio richiesto dal client, in caso ci siano più domini sul server con lo stesso percorso interno.
- **server**: info software su server (non usato per motivi di sicurezza)

HTTP Status Code

La riga di stato della risposta HTTP include la versione HTTP seguita da uno status code a tre cifre e informazioni aggiuntive in inglese.

I codici di stato HTTP hanno una struttura simile ai codici di risposta utilizzati da SMTP:

- **2XX Risposta valida** 200 Ok indica che la richiesta HTTP era OK.
- **3XX** Il documento non è più disponibile sul server.
- **4XX** il server ha rilevato un errore nella richiesta HTTP inviata dal client.
- **400 Bad Request** indica un errore di sintassi nella richiesta HTTP.
- **404 Non trovato** indica che il documento richiesto non esiste sul server.
- **5XX** Tutti i codici di stato che iniziano con la cifra 5 indicano un errore sul server.

HTTP Connections

Inizialmente, una connessione TCP era disponibile per una sola richiesta HTTP, il che andava bene per pagine semplici. Oggi abbiamo pagine HTML fatte di centinaia di parti (immagini, icone, CSS...) e ognuna richiede una richiesta HTTP diversa. Quindi, per scaricare una pagina web con 10 immagini, abbiamo bisogno di una per la pagina e una per ogni immagine.

La **connessione**: l'intestazione Keep Alive è stata aggiunta per consentire di inviare più di una richiesta in una sola connessione. Il client invia l'intestazione e il server risponde con la stessa intestazione specificando quante richieste possono essere utilizzate nel resto della connessione

È importante sapere che **ogni richiesta HTTP**, essendo o meno nella stessa connessione, **non è correlata alle precedenti**. Il protocollo HTTP non mantiene uno stato, o qualche collegamento da una richiesta all'altra. Questo è diverso dal dire che il backend (il software in esecuzione sul server che crea e fornisce le pagine al server HTTP) non mantiene uno stato. In effetti lo fa.

HTTP Cookies

I cookie introducono uno stato, per HTTP siamo sempre una persona diversa ad ogni connessione, HTTP non sa niente della sessione, ma il server ora ci riconosce.

Ci sono diversi tipi di cookie:

- **session management**: user log-in
- **personalization**: lingua del client o posizione geografica
- **tracking**: profiling user activities

Gli unici cookie che possiamo disabilitare sono quelli di tracking. Questi cookie sono l'unico modo legale per tracciare le attività di un utente, ma ci sono molti modi sporchi per farlo lo stesso.

Un cookie è utilizzato come token crittografico, se ci rubano il cookie ci rubano l'identità su una sessione http, questo è fonte di attacchi.

Proxy

Con il passare del tempo, la progettazione iniziale di HTTP si è rivelata troppo lenta per i requisiti moderni.

In particolare, una sequenza di chiamate iterative al server dipende troppo dal Round-Trip-Time della connessione e quindi caricare una pagina può richiedere molti secondi.

Inoltre, il server HTTP ora utilizza pagine dinamiche e crea pagine da contenuti inclusi nei database, e il tempo di elaborazione necessario per recuperare tutti i componenti è un'altra fonte di ritardo.

Abbiamo affrontato il secondo problema con i proxy e il primo con una versione totalmente nuova di HTTP

Proxy Server

Un server proxy è una cache che salva le pagine sfogliate per un certo periodo di tempo.

Deve essere configurato all'interno del browser, e quindi il browser, invece di connettersi al server HTTP di destinazione, si conserverà al proxy.

Il proxy si collegherà quindi al server HTTP originale, recupererà il contenuto e lo salverà per un po'.

Se il browser naviga di nuovo nel sito web, il proxy non avvierà una nuova connessione ma servirà semplicemente di nuovo la stessa pagina

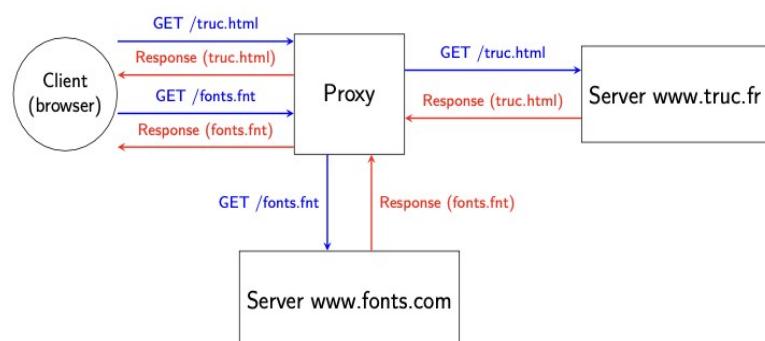
Il proxy viene impostato dall'amministratore di rete e può avere accesso da tutti i terminali della rete.

Il proxy **vede tutto** è a tutti gli effetti in the middle .

Come si comporta il server quando restituiamo una cache?

Alcune header HTTP vengono utilizzati per sapere se la pagina è cambiata e deve essere ricaricata.

- **Cache-Control**: viene utilizzato dal server per notificare se:
 - la pagina non può essere memorizzata nella cache (**no-store**),
 - può essere memorizzata nella cache fino a un certo numero di secondi (**max-age=10**) o può essere memorizzata nella cache, ma tutte le volte che me la richiedi devi chiedere al server se si è verificata qualche modifica utilizzando l'intestazione del client **If-Modified-Since: (no-cache)**
- **If-Modified-Since: (no-cache)**: questo è un header che il client utilizza per richiedere una pagina solo se è stata modificata da un certo momento. Altrimenti non viene restituito dal server, la cache è ancora valida.



HTTP 2.0

Standard ragionevolmente nuovo ma lo utilizzano praticamente tutti, cosa cambia?

- è **binario** e non più di testo, molto più veloce
- **parallel streams**: le pagine vengono suddivise in frame così da scaricare i contenuti in parallelo, il server può dire al client in che ordine scaricare le cose
- **push mechanism**: il server può pushare contenuti che non vengono richiesti (es. notifiche)

User Datagram Protocol (UDP)

UDP fornisce un servizio **inaffidabile** ma garantisce **error detection** con il checksum, si basa sul livello di trasporto connectionless. Con UDP si fa multiplexing e demultiplexing.

Le caratteristiche principali del servizio UDP sono:

- la dimensione massima delle SDU è di 65467 byte.
 - non garantisce la consegna delle SDU (possono verificarsi perdite e le SDU possono arrivare fuori sequenza).
 - rilevamento degli errori

Il vantaggio principale dell'utilizzo del servizio UDP rispetto all'utilizzo del servizio di rete senza connessione è che consente a diverse applicazioni in esecuzione su un host di scambiare SDU con diverse altre applicazioni in esecuzione su host remoti. Le applicazioni su uno degli host comunicanti sono identificate da un numero di porta.

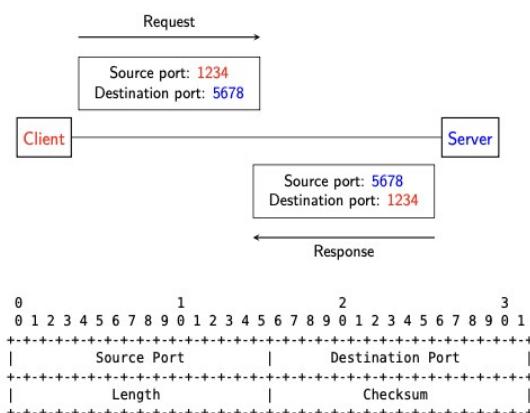
UDP Header

L'header UDP è molto semplice e contiene quattro campi:

- una porta di source di 16 bit (max. 65535 processi di un server allo stesso tempo)
 - una porta di destinazione 16 bit
 - un campo di lunghezza 16 bit
 - un **IP checksum** (vedi livello di trasporto) a 16 bit: Il checksum viene calcolato su campi di header e payload.

Utilizzo di UDP

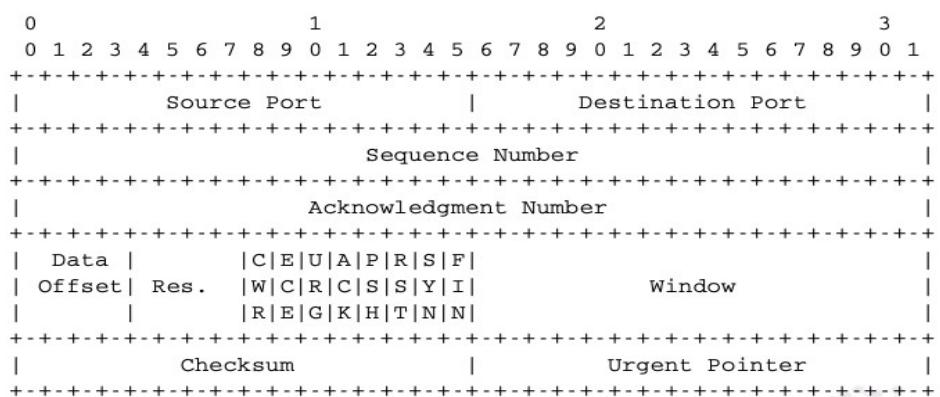
Come già detto viene utilizzato per applicazioni che non hanno bisogno di reliability. (es. DNS, QUIC (nuovo livello di trasporto che si basa su UDP), RTP, VOIP).



Transmission Control Protocol (TCP)

TCP è un macello, se leggi l'RFC sei in grado di parlare con qualcuno, ma ci sono un sacco di modifiche che non possono essere implementate contemporaneamente. TCP fornisce **reliable** byte stream.

Header di un segmento TCP



- Source Port e Destination Port sono di 16 bit
 - ACK number e Sequence number sono di 32 bit
 - Data offset è di 4 bit e indica la grandezza dell'header TCP di massimo 64 byte

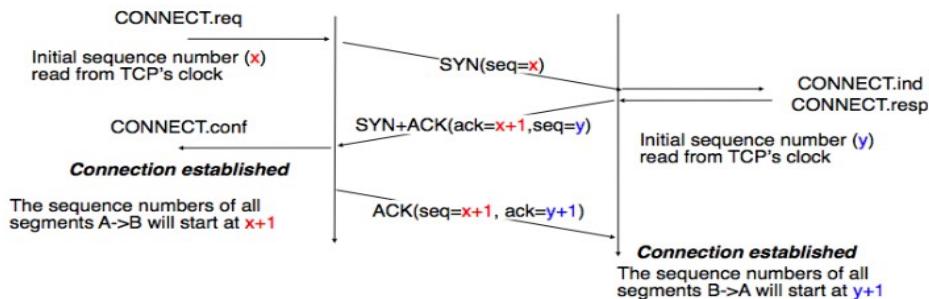
Flag (alcuni di essi):

- SYN: utilizzato durante la creazione della connessione
 - FIN: utilizzato durante il rilascio della connessione
 - RST: chiude la connessione in caso di problemi
 - ACK: campo acknowledgment è valido
 - CWR, ECE: per la congestione esplicita della rete (non li consideriamo)
 - PSH: serve a far sì che il ricevente mandi immediatamente all'applicazione i dati che gli sto mandando.

- URG fa il contrario, il ricevitore deve passare questo dato all'applicazione a prescindere out-of-sequence.
- Window: dimensione della receiving window del sender, allego al segmento la dimensione del mio buffer.
- Checksum: IP-style checksum calcolato su tutto il segmento TCP e qualche campo dell'header (indirizzi IP)

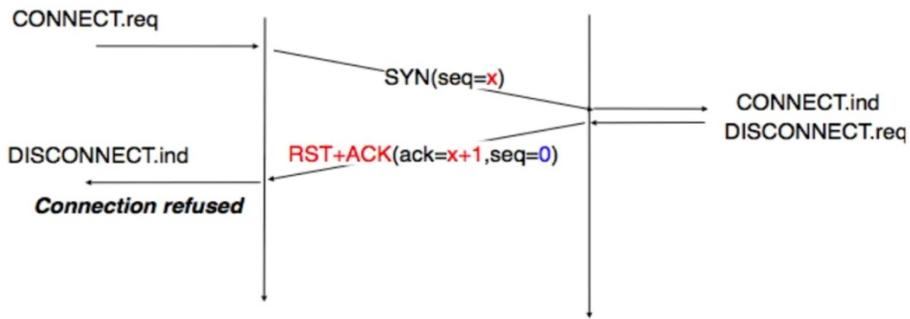
Connessione TCP

Abbiamo già visto come viene stabilita una connessione a livello di trasporto, con TCP cambia l'interpretazione degli ACK.



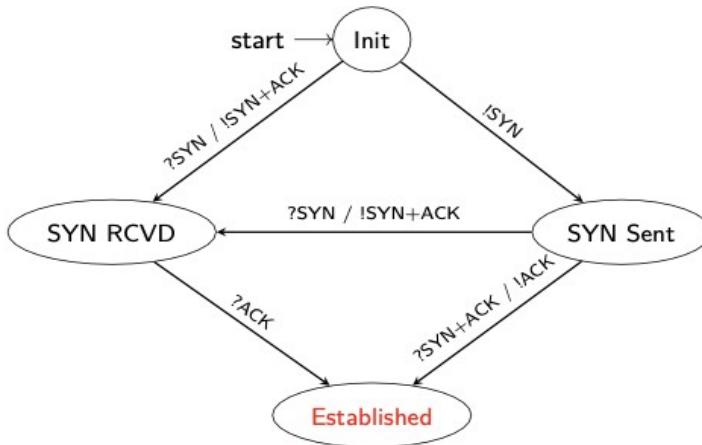
Un TCP ACK uguale a $X+1$ significa: ho ricevuto tutti i dati fino a X, mi aspetto $X+1$

Connessione rifiutata TCP



I frame che portano RST hanno sempre un ACK con il numero di sequenza valido a cui si riferisce.

Macchina a stati per connessione TCP stabilita



L'unica cosa nuova qui è la possibilità che due host siano sia client che server. Se aprissero una connessione allo stesso tempo, utilizzando le stesse porte esatte, sarebbero stagnanti in assenza del bordo tra SYN Sent e SYN RCVD. Invece, inviano un SYN+ACK, e questo ovviamente conta come un ACK, e fanno spostare lo stato su Established. Naturalmente ci sono timer di ritrasmissione per tenere conto della perdita di segmenti (non mostrati in figura)

MSS (Maximum Segment Size) viene trasmesso in SYN o SYN + ACK (quindi nei primi due messaggi) ed è fondamentale. È normalmente settato sul **MTU** della rete, la dimensione più grande di un frame nella rete del receiver, per evitare fragmentation.

TCP deve essere stateful

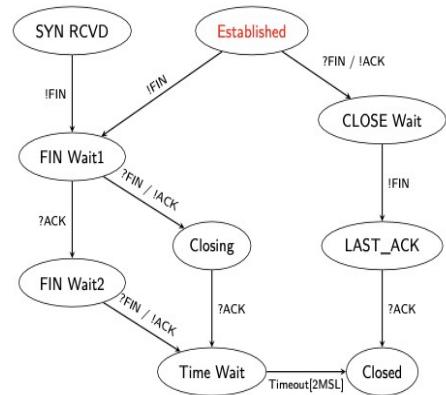
Quando abbiamo una connessione TCP dobbiamo mantenere uno stato. Per ogni connessione in corso TCP mantiene un **Transmission Control Block (TCB)**, un insieme di informazioni necessarie per mantenere lo stato della connessione, vedi dopo, possiamo già dire che il TCB mantiene i buffer di sender e receiver di TCP.

Chiusura connessione TCP

Graceful Release

Avviene con un segmento FIN con numero di sequenza x, questo significa "ho finito di mandarti tutti i dati, l'ultimo byte valido è x". Il peer sarà comunque abilitato a rimandare tutti i dati persi prima di x, e aspetterà per un FIN. I segmenti FIN devono essere acknowledged.

MSL è Maximum Segment Life, massimo tempo per il quale il segmento può rimanere nella rete



Release brutale

Avviene con un segmento RST, appena viene ricevuto il TCB viene cancellato.

Un attaccante potrebbe provare a resettare una connessione tra due peer, come evitarlo?

Ci si può difendere se l'attaccante non è nel cammino, deve indovinare la porta sorgente (1 su 16 mila) e il numero di sequenza (1 su 2^{16} della finestra). Infatti, è importante che i segmenti RST facciano l'ACK dell'ultimo numero di sequenza valido.

Nota sullo stato Time Wait

L'RFC impone uno stato Time Wait, che deve essere mantenuto per $2 \times MSL$ (4 minuti) → vedi macchina a stati Questo perché l'ACK all'ultimo FIN potrebbe essere perso e deve essere ritrasmesso, quindi prima di rimuovere il TCB, dobbiamo essere sicuri che nulla debba essere ritrasmesso.

Tuttavia, questo può portare a una grande occupazione della memoria

Esempio:

Supponendo che tu abbia 32 GB di memoria e che tu possa allocare 16 GB per il TCB.

Supponiamo che un server riceva X connessioni al secondo, tutte durano per z secondi che è molto breve, trascurabile rispetto a MSL, e poi terminano. Quante connessioni/secondo può accettare il server?

Quanto è grande un TCB?

Su linux questo è facile da dire

```
cat /proc/sys/net/ipv4/tcp_rmem 4096 131072 6291456
```

Questa è la dimensione minima / predefinita / massima del buffer di ricezione, che è la principale fonte di occupazione della memoria nel TCB. Il valore può cambiare durante la durata di una connessione. Supponendo il valore predefinito 131072:

$$16GB = 2^{34}B > 131072 * X * 4 * 60$$
$$\rightarrow X < \frac{2^{34}}{131072 * 4 * 60} = 546$$

nota: $4 * 60$ è il tempo per il quale dobbiamo mantenere le connessioni. (4 minuti)

Trasferimenti affidabili TCP

TCP usa una trasmissione Go-Back-n con selective repeat.

TCB

I contenuti del TCB che sono statici sono:

- l'indirizzo IP locale e remoto e le porte. Questi non cambiano mai durante la durata della connessione,
- **buffer di invio**: un buffer utilizzato per memorizzare tutti i dati non riconosciuti,

- **buffer di ricezione:** un buffer per memorizzare tutti i dati ricevuti dall'host remoto che non sono ancora stati consegnati all'utente. I dati possono essere memorizzati nel buffer di ricezione perché non sono stati ricevuti in sequenza o perché l'utente è troppo lento per elaborarli.
- I buffer vengono allocati all'inizio della connessione e riempiti/svuotati quando i dati arrivano/partono.

I contenuti del TCB che cambiano durante l'evoluzione della connessione sono:

- lo stato corrente del **FSM** (SYN-Send, SYN-Received, Established. . .)
- **MSS** (Maximum Segment Size)
- **snd.nxt** : il numero di sequenza del prossimo byte in uscita nel byte stream (il primo byte di un nuovo segmento di dati in uscita utilizza questo numero di sequenza)
- **snd.una** : il più recente numero di sequenza che è stato inviato ma non è ancora stato riconosciuto
- **snd.wnd** : la dimensione corrente della mia finestra di invio (in byte)
- **rcv.nxt** : il numero di sequenza del prossimo byte che ci aspettiamo sarà ricevuto dall'host remoto
- **rcv.wnd** : la dimensione corrente della finestra di ricezione comunicata dall'host remoto

Sending Data

Quando l'applicazione inserisce alcuni nuovi dati nel buffer di invio, il livello TCP. . .

1. controlla che il buffer di invio non contenga più dati della finestra di ricezione pubblicizzata dall'host remoto (rcv.wnd).
2. fino a MSS byte di dati vengono inseriti nel payload di un segmento TCP.
3. Il numero di sequenza di questo segmento è impostato su snd.nxt e la variabile snd.nxt nel TCB viene incrementata della lunghezza del payload del segmento TCP.
4. Il numero di ACK è impostato su rcv.nxt e il campo della finestra del segmento TCP viene calcolato in base allo spazio libero lasciato nel buffer di ricezione (non al buffer di invio!).
5. I dati vengono conservati nel buffer di invio nel caso in cui debbano essere ritrasmessi.

Receiving Data

Quando arriva un nuovo segmento, il livello TCP controlla se il bit ACK è impostato, in quel caso.

1. imposta rcv.wnd sul valore del campo finestra del segmento ricevuto.
2. confronta il numero di conferma con snd.una.
3. I dati appena riconosciuti vengono rimossi dal buffer di invio e snd.una viene aggiornato.
4. Il buffer di invio può essere vuoto o meno.

Idealmente, tutto ciò che devi sapere per implementare TCP è quello che ho descritto finora. Tuttavia, ci sono molte scelte di progettazione che sono fondamentali per far funzionare TCP (e le reti che lo usano). Alcuni di questi sono:

1. Quando devo inviare i dati?
2. Quanto deve essere grande la finestra?
3. Come è necessario impostare i timer di ritrasmissione?

Quando mandare i dati

L'applicazione sta riempiendo il buffer di dati, dovrei semplicemente inviarli? • Questa domanda sembra semplice, tuttavia, ci sono scelte non banali da fare. Ci sono **2 approcci estremi opposti** con i loro pro e contro

1. Inviare **As Soon As Possible**: Idealmente, non appena il buffer di invio non è vuoto, potresti inviare un segmento TCP. In alcuni casi (come nella shell remota menzionata prima) questo è un comportamento desiderato. Tuttavia, un pacchetto TCP ha almeno 40B di header (20 IP + 20 TCP). Se si invia un byte per segmento, 40 della capacità viene sprecato, grosso overhead.
2. Aspettare di avere **MSS byte** da mandare: In questo caso si ottiene la massima efficienza. Se l'applicazione produce dati con un throughput molto elevato, questa è probabilmente una buona scelta. Se invece il tuo buffer richiede molto tempo per essere riempito abbastanza, i dati al suo interno diventano vecchi prima di inviarli. se facciamo gaming online non possiamo aspettare di avere MSS byte prima di sparare.
3. Algoritmo di Nagle

```

1 if rcv.wnd >= MSS and len(data) >= MSS:
2     send one MSS-sized segment
3 else:
4     if there are unacknowledged data:
5         place data in buffer until acknowledgment has been received
6     else:
7         send one TCP segment containing at most rcv.wnd data

```

- **Righe 1-2:** controlla che la finestra del ricevitore consenta i dati MSS, e se li ho da inviare, fallo e basta → Questo consente trasferimenti in blocco: se ho segmenti molti grandi da inviare, li invierò e basta. E se il ricevitore vuole che rallenti?
→ il ricevitore può rallentare questo in un pacchetto ACK inviando una finestra più piccola
- **Riga 3:** Altrimenti, i byte che devo inviare sono inferiori a MSS, o la finestra del ricevitore è piccola,
→ Questo è quando devo prendere una decisione saggia, i dati sono abbastanza grandi da valere la pena di essere inviati?
- **Righe 4-5:** Se sto aspettando un ACK sui vecchi dati, aspetta ancora un po', in modo che il buffer possa riempirsi. Invia i dati quando viene ricevuto l'ACK
- **Righe 6-7:** Altrimenti, non ho inviato dati finora, o sono stati tutti nascosti. Mi è permesso inviare un piccolo segmento, ma solo una volta per RTT.

È misurato che su internet la maggioranza dei pacchetti sono 1460B o 0B. 1500B è l'MTU dell'Ethernet link Layer, il più usato; quindi, la maggioranza delle connessioni TCP usa 1500B come MSS.

Questo rende però TCP non adatto al traffico real time, che ha bisogno di disattivare Nagle (sempre pacchetti piccoli da inviare subito).

Quanto dovrebbe essere lunga una finestra?

- Supponiamo che una connessione sia aperta e che il mittente invii molti segmenti, riempiendo la finestra del mittente
- Si fermerà e aspetterà un ACK
- Supponiamo che l'ACK arrivi e acceda tutti i dati inviati finora.
- Quindi il mittente ricomincerà a inviare.

In pratica, non è possibile avere più di una finestra di dati inviata per RTT, ma la dimensione massima della finestra è impostata su $2^{16} = 65536$.

calcolo da fare: $2^{16} * 8 / \text{msec} * 1000 \Rightarrow \text{b/s}$

RTT	Maximum Throughput
1 msec	524 Mbps
10 msec	52.4 Mbps
100 msec	5.24 Mbps
500 msec	1.05 Mbps

Miglioramenti dell'RFC originaria di TCP

TCP Scale Window Option

Poiché l'header TCP non può essere esteso, la dimensione della finestra non può essere ingrandita.

Una soluzione che è stata adottata è quella di aggiungere un'altra estensione opzionale che introduce un moltiplicatore per la finestra.

L'opzione TCP Scale Window viene scambiata solo nei pacchetti SYN e SYN-ACK e contiene un numero $0 \leq S \leq 14$. Il numero di dimensione della finestra è spostato di S bit, cioè la dimensione della finestra ricevente è $\text{rcv.wnd} * 2^S$

RTT	Maximum Throughput
1 msec	8590 Gbps
10 msec	859 Gbps
100 msec	86 Gbps
500 msec	17 Gbps

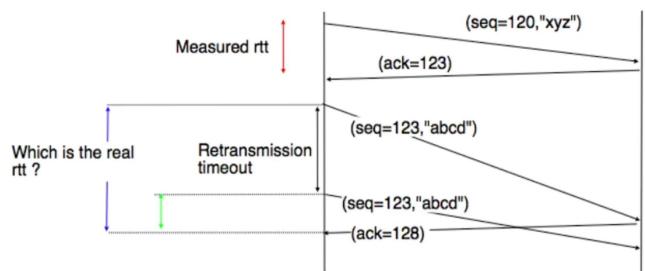
Come dovrei settare il retransmission timer?

Idealmente il retransmission timer dovrebbe essere leggermente più grande dell'RTT, se alla fine del RTT l'ack non arrivasse il pacchetto dovrebbe essere ritrasmesso. Ma come faccio a conoscere l'RTT? Devo misurarla.

N.B.:

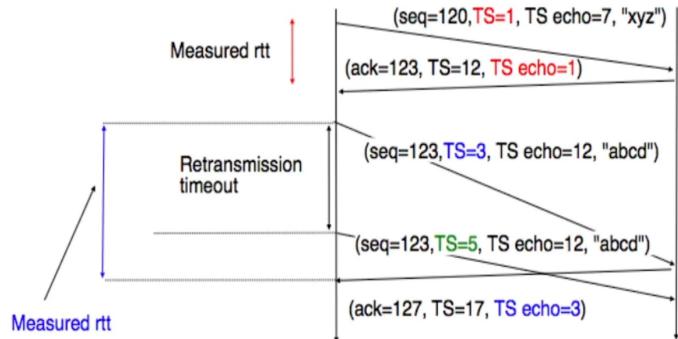
- la misurazione dell'RTT in una connessione con perdita di pacchetti è soggetta a errori
- RTT cambia durante la durata di una connessione

La prima contromisura potrebbe essere ignorare l'RTT misurato con una retransmission. Tuttavia, cosa succede se l'ACK è quello che si è perso o è stato duplicato?



La seconda soluzione è un'altra estensione opzionale TCP, **Timestamp TCP**.

In ogni segmento TCP, vengono aggiunti due timestamp, quello corrente (**TS**) e l'ultimo ricevuto dall'altro endpoint (**TS_echo**). Ciò consente di calcolare l'RTT dall'ultimo segmento ricevuto, indipendentemente dal fatto che sia uno ritrasmesso o meno.



Miglioramento stima RTT: Algoritmo di Van Jacobson

Introduce tre variabili:

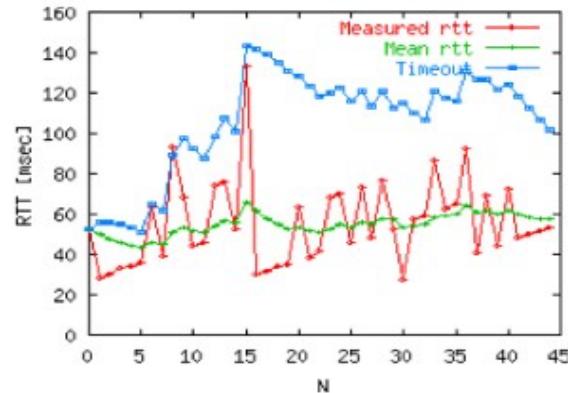
- **rtt**: l'ultimo valore RTT misurato
- **srtt**: l'RTT smoothed, inizializzato al primo rtt
- **rttvar**: la varianza stimata dell'RTT, inizializzata a $rtt/2$
- **rto**: il timer di ritrasmissione

$$srtt = (1 - \alpha) \times srtt + \alpha \times rtt$$

$$rttvar = (1 - \beta) \times rttvar + \beta \times |srtt - rtt|$$

$$rto = srtt + 4 \times rttvar$$

Where $\alpha = 1/8$, $\beta = 1/4$.



Cosa fare quando RTO scade?

RTO Exponential Backoff (RFC 6298): incluso nella RFC TCP originale, questo impone che l'RTO sia raddoppiato dopo ogni tentativo fallito. Dopo un tentativo riuscito l'RTO viene reimpostato secondo il normale algoritmo. La logica è: se il ricevitore è congestionato, non può gestire troppi pacchetti/i. Quindi il mittente dovrebbe rallentare.

Delayed ACKs

La maggior parte delle volte, la comunicazione è in gran parte monodirezionale.

Il ricevitore invierà quindi spesso molti ACK che sono molto inefficienti.(non vogliamo mandare tanti pacchetti piccoli).

La strategia Delayed ACKs invia un ACK ogni secondo frame di dati ricevuto, o, dopo un certo timeout. Ciò consente di riconoscere più segmenti di dati che rispondere ad ogni singolo segmento ricevuto. Questo non funziona bene con l'algoritmo di Nagle. Perché? Abbiamo tanti pacchetti piccoli e con Nagle vengono bufferizzati.

Selective Ack go-back-n sottoperfoma sempre quando ci sono perdite elevate, perché la finestra non si muove.

In TCP il ricevitore è autorizzato a salvare i dati fuori sequenza, ma dovrebbe comunque riconoscere solo quelli in sequenza. Con l'opzione SACK viene aggiunta una nuova estensione di header opzionale, in cui il ricevitore riconosce i blocchi di dati contigui, fuori sequenza che ha ricevuto. A causa dello spazio, non sono inclusi più di 3 blocchi.

Ad esempio, l'ACK può contenere qualcosa come: [1100 – 1200], [1250 – 1500], [1800 – 1900]. L'ACK conterrà anche il numero di riconoscimento nell'header TCP fisso, ad esempio 850.

Questo è interpretato dal mittente come: il ricevitore ha ricevuto correttamente tutti i byte prima di 850, e gli intervalli 1100-1200, 1250-1500, 1800-1900.

È la scelta del mittente decidere cosa fare dopo.

Performance TCP

Non abbiamo ancora considerato il caso in cui all'interno della rete avviene una **congestione**.

Sappiamo che nel protocollo go-back-n, ci sono due dimensioni della finestra, la dimensione della finestra del mittente (**swnd**) e la dimensione della finestra del ricevitore (**rwnd**). Sappiamo che il mittente invierà i dati fino al minimo tra le due finestre = $\min(\text{swnd}, \text{cwnd})$, quindi aspetterà gli ACK.

swnd viene ridotto ogni volta che viene inviato un segmento e un ACK non è stato ancora ricevuto, questo perché il mittente deve mantenere nel buffer di invio i dati che non sono stati riconosciuti.

rwnd viene ridotto ogni volta che i dati vengono ricevuti correttamente da TCP ma non recuperati dall'applicazione del ricevitore.

Il throughput massimo è dato da **window / RTT**.

A causa dell'algoritmo di Nagle, il mittente invierà tutti i segmenti di dimensioni MSS che **swnd** consente (praticamente tutti insieme), questo accade al tempo t_0

Supponiamo che la capacità di collegamento di entrambe le estremità sia molto alta e che i segmenti siano tutti ricevuti. I segmenti vengono ricevuti al tempo $t_0 + \text{RTT} / 2$.

Quindi il destinatario invierà un ACK cumulativo. Al momento $t_0 + \text{RTT}$ il mittente riceve l'ACK, il buffer di invio viene cancellato e quindi il mittente ricomincerà.

Quindi per ogni RTT, al massimo i dati della finestra vengono inviati e il throughput massimo è **window / RTT**. Questo throughput viene raggiunto fin dall'inizio della connessione TCP.

problema: Se invece il ricevitore non è abbastanza veloce, invierà un ACK, ma nel messaggio ACK, ci sarà una nuova dimensione (più piccola) per la finestra del ricevitore, la chiamiamo **rwnd' < rwnd**

In tal caso quando viene ricevuto l'ACK, al tempo $t_0 + \text{RTT}$ il mittente può inviare solo tanti dati quanto **window' = $\min(rwnd', swnd)$** .

Il throughput è **window' / RTT**, quindi in generale, la formula è ancora valida, ma la finestra RTT è ridotta.

Tuttavia, **cosa succede se c'è congestione di rete?**

Questa è una situazione in cui i router sul percorso dal client al server droppano pacchetti. Ciò è dovuto al fatto che hanno bisogno di instradare troppi pacchetti e le loro code si riempiono, quindi iniziano a far droppare i pacchetti.

Tuttavia, **non ha nulla a che fare con la congestione del ricevitore**: potrebbe esserci congestione della rete, ma l'applicazione al ricevitore è in grado di leggere tutti i dati non appena arrivano.

Supponiamo che non ci siano problemi nell'applicazione del ricevitore, ma abbiamo congestione di rete. Questo è rilevato dal mittente come una perdita di pacchetti

Ad esempio, al tempo $t_0 + \text{RTT}$ il messaggio ACK cumulativo non consente di svuotare l'intero buffer di invio, ma solo, ad esempio, la metà di esso.

Tuttavia, la window del ricevitore non sarà diminuita. Quindi, se il mittente ha sempre dati da inviare (come nel trasferimento dei file) l'effetto è:

- il buffer di invio è riempito di nuovi dati
- il mittente può semplicemente inviare nuovamente l'intero buffer, compresi i vecchi dati da inviare nuovamente e i nuovi dati

Di conseguenza, la quantità media di dati inviati è ancora **WINDOW / RTT**, anche se i nuovi dati RTT consegnati sono al massimo **WINDOW** (perché metà del buffer contiene vecchi dati $2 \times \text{RTT}$ da inviare di nuovo).

Cioè, il mittente non rallenterà l'invio dei dati e il router sarà ancora congestionato e farà ancora droppare i pacchetti.

Se vogliamo risolvere il problema della congestione della rete, l'unica possibilità è ridurre **WINDOW**, quindi il mittente invia meno dati e il router diventa meno congestionato e si ferma a far cadere i pacchetti.

Quindi abbiamo bisogno di un altro modo per rilevare e mitigare la congestione e ridurre la finestra del mittente

Congestion window

Il problema della congestione inizia anche prima della congestione stessa.

Il mittente, all'inizio della connessione non dovrebbe semplicemente inondare il ricevitore con un throughput troppo alto, o questo potrebbe effettivamente creare la congestione.

La finestra iniziale deve essere **piccola e crescere con il tempo**. Quindi il controllo della congestione riguarda principalmente la risposta a queste due domande:

- Qual è la funzione che viene utilizzata per aumentare la finestra da un piccolo valore iniziale?
- Come si riduce la finestra quando si verifica una perdita?

Congestion Window

Introduciamo un'altra finestra, la cosiddetta finestra di congestione (**cwnd**)

cwnd è inizializzata a qualche valore fisso, inizialmente era $cwnd_0 = MSS$, ora è comune essere $cwnd_0 = 10 \times MSS$.

Poiché MSS è normalmente 1460B (1500 Ethernet MTU - 40B di intestazioni IP e TCP) $\Rightarrow cwnd_0 = 14600B$

In ogni momento abbiamo **window = min(cwnd, swnd, rwnd)**.

Introduciamo anche un altro parametro: la threshold(soglia) di slow start **sstrash**, che è inizializzato a qualche piccolo valore.(come MSS).

Algoritmo Slow Start

L'algoritmo Slow Start è stato proposto da Van Jacobson ed è stato il primo algoritmo di controllo della congestione adottato. Ora è nella RFC 5681. Dopo di che, seguirono molti altri metodi.

Ad ogni RTT, se tutti i segmenti sono ACKed, cwnd è raddoppiato, quindi c'è una crescita esponenziale di cwnd.

Questo non è così lento, ma è meglio che iniziare con la finestra massima. . .

Tuttavia, dopo un evento di congestione, se $cwnd > sstrash$ allora cwnd non viene raddoppiato ad ogni MSS, ma viene aumentato di un solo MSS per RTT.

Ci sono due eventi che vengono interpretati come un evento di congestione

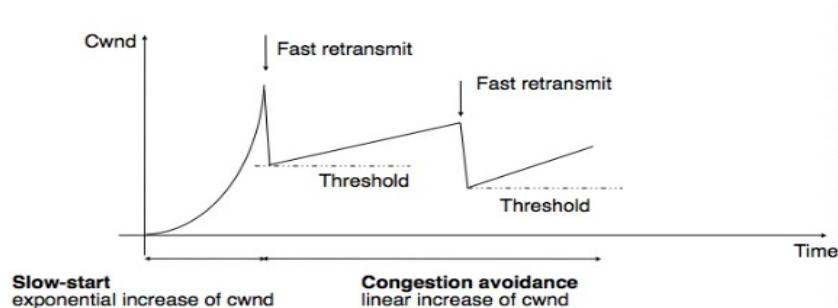
- **mild congestion:** tre ACK vengono ricevuti con lo stesso numero di sequenza, viene tentata una ritrasmissione e ha successo (l'ACK seguente conferma la ricezione)
- **severe congestion:** il timeout della ritrasmissione si accende

Mild Congestion

Quando si verifica un evento di congestione lieve allora:

$$cwnd' = cwnd, \quad cwnd = sstrash, \quad sstrash = cwnd' / 2$$

Cioè: cwnd viene ripristinato al valore di sstrash e sstrash viene ripristinato a metà del valore di cwnd prima dell'evento di congestione

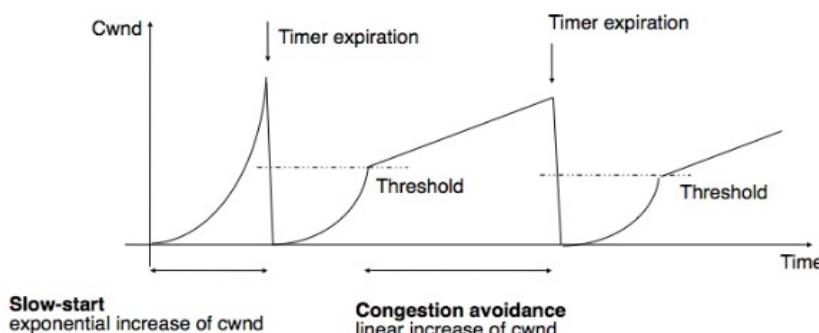


Severe Congestion

Quando si verifica un grave evento di congestione:

$$cwnd' = cwnd, \quad cwnd = cwnd_0, \quad sstrash = cwnd' / 2$$

Cioè: fare slow-start dall'inizio, e (come nella congestione lieve) resettare sstrash a metà del valore di cwnd prima dell'evento di congestione



Performance

Si può dimostrare che il throughput massimo raggiungibile tra due host può essere stimato come:

$$throughput = \sqrt{\frac{3}{2}} \times \frac{MSS}{RTT \times \sqrt{p}}$$

Dove p è la probabilità di un evento di mild congestion, supponendo che una congestione grave sia molto improbabile.

Poiché MSS è generalmente impostato su 1460, il throughput è fissato da RTT e p

Socket

Un socket è una interfaccia standard definita tra applicazioni e livello 4 (livello app).

Quando un'app ha bisogno di inviare/ricevere dati dalla rete utilizza API che crea, gestisce ed utilizza socket.

Transport Layer Security (TLS) qui

TLS serve ad implementare meccanismi di sicurezza tra i livelli di cui abbiamo fatto fino ad'ora.

Abbiamo studiato modi efficienti per ottenere una comunicazione sicura bidirezionale. Ciò comporta l'uso di:

- **funzioni hash** per garantire l'integrità
- **Crittografia a chiave simmetrica** per garantire una crittografia / decripttografia rapida
- **Crittografia a chiave pubblica** per garantire l'autenticazione e la non ripudiabilità
- **Certificati** per ottenere un Web of Trust

A quale livello inseriamo la sicurezza?

A seconda del livello che usi, ci sono pro e contro.

Più basso è il livello, più informazioni vengono crittografate; tuttavia, le cose potrebbero non funzionare affatto.

Più alto è il livello, meno supporto di cui hai bisogno dal sistema operativo e più è facile supportare nuove applicazioni, ma più informazioni sono trapelate.

Inoltre, sappiamo che per utilizzare la crittografia tra due entità Alice e Bob abbiamo bisogno di quella che viene chiamata **security association**, il che significa che Alice e Bob devono scambiare qualche chiave prima di utilizzare un determinato servizio di sicurezza.

Concentriamoci sulla segretezza.

Per rispondere alle prossime domande, chiediti per ognuna di esse:

- Quali informazioni sono effettivamente crittografate?
- Quali informazioni non sono crittografate?
- Le entità coinvolte possono istituire un'associazione di sicurezza?
- Cosa mette di funzionare se non possono farlo?

Possiamo cifrare l'header IP (in su)?

Non proprio. L'intestazione IP è necessaria per il routing e i router sono sconosciuti ad Alice e Bob. Alice non può negoziare un'associazione di sicurezza con i router. Quindi non c'è modo di inviare un pacchetto IP con un'intestazione crittografata e farlo instradare. Ci sono alcune eccezioni. . .

Modi di cifrare l'header IP

Crittografare al livello MAC. Questo può essere fatto, ma applica la crittografia solo nella LAN. Ad esempio, un AP wireless riceve dataframe crittografati, ma poi li decifra e li invia su Internet in chiaro.

In alternativa, utilizzare una VPN, che incapsula un pacchetto IP (crittografato) all'interno di un altro pacchetto IP (con intestazioni in chiaro). Per questo devi essere il manager dei computer (o delle reti) di Alice e Bob, non è una soluzione generale.

Possiamo cifrare l'header TCP (in su)?

Se voglio implementare la cifratura a livello TCP non devo cifrare tutto internet, solo client e server. Ma se cifriamo TCP il NAT non può più vedere le porte.

Se forniamo un servizio di crittografia oltre a TCP (quindi TCP non è crittografato), quali informazioni sono disponibili per un utente malintenzionato?

TCP fornisce informazioni non banali, grazie alle porte si può capire a quale servizio facciamo accesso.

Transport Layer Security (TLS)

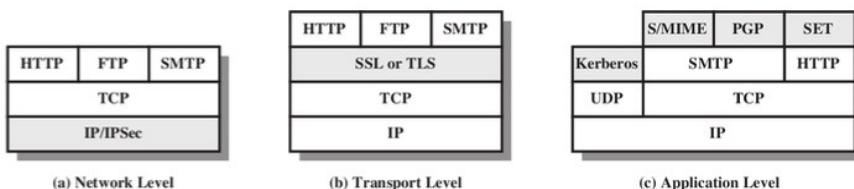
Poiché i protocolli Internet originali sono stati progettati senza alcun principio di sicurezza in mente, abbiamo dovuto aggiungere la sicurezza oltre ai protocolli esistenti e al codice in esecuzione

La soluzione più gestibile per fornire un uso diffuso della crittografia è stata quella di introdurla in cima a TCP.

Ciò ha fornito un ragionevole compromesso tra la facilità di applicazione (e in effetti, al giorno d'oggi quasi tutto il traffico TCP è crittografato) e la perdita di informazioni dai livelli inferiori.

Tuttavia, la sicurezza può essere introdotta in qualsiasi livello dello stack di rete.

1. Al livello di rete: **VPN**
2. Al livello di trasporto: **TLS**
3. Al livello applicazione: **PGP** (es. email cifrate)



SSL / TLS

SSL (secure socket layer) introdotto da Netscape nel 1995 aggiunge servizi di sicurezza "in cima a" TCP

Nel corso degli anni è stato standardizzato e rinominato TLS (Transport Layer Security) • Ci sono diverse versioni di SSL, tutte deprecate, e tre versioni di TLS, con una penetrazione ancora non marginale di quelle vecchie (grafico di SSL Labs).

SSL e TLS sono nomi che usiamo ancora in modo intercambiabile.

's' protocols

Mentre molti protocolli Internet a livello di applicazione sono stati progettati prima dell'avvento di questi strumenti ed erano effettivamente insicuri, li abbiamo sostituiti con versioni sicure.

I protocolli sicuri generalmente usano lo stesso nome del protocollo non sicuro, con una "S" finale, che viene eseguita su una porta diversa:

- HTTP diventa HTTPS: dalla porta 80 alla porta 443
- POP3 diventa POP3S: dalla porta 110 alla porta 995
- IMAP diventa IMAPS: dalla porta 143 alla porta 993

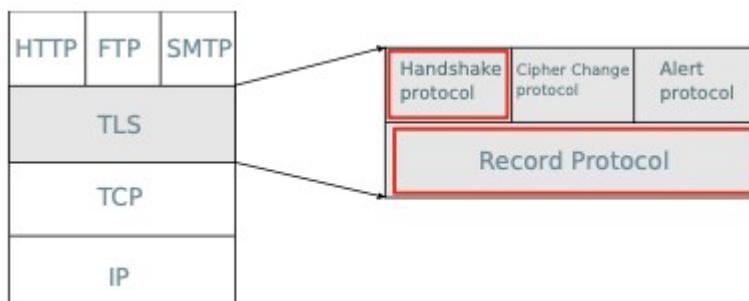
Servizi di TLS

TLS fornisce i seguenti servizi di sicurezza ai livelli superiori:

- **Origin Authentication**: il server e il client possono identificarsi con l'uso di certificati X.509 (o anche con una chiave pre-condivisa, estremamente insolita)
- **Segretezza**: dopo l'autenticazione segue una fase di accordo chiave in cui viene generata una chiave condivisa per crittografare il traffico utilizzando la crittografia a chiave simmetrica.
- **Integrità**: viene generata un'altra chiave simmetrica che garantisce l'integrità utilizzando un HMAC

Fino a TLS 1.2 lo standard è stato arricchito, ma i principi sono stati più o meno mantenuti. A partire dalla versione 1.3 molte cose sono state cambiate.

TLS 1.2



TLS contiene diversi protocolli, ne esamineremo brevemente due:

- **Handshake Protocol TLS**: viene utilizzato per negoziare chiavi e parametri di sicurezza
- **Record Protocol TLS**: la trasformazione della sicurezza applicata ai dati

Record protocol TLS

- Il flusso è frammentato.
- Alla fine viene compresso (rimosso in 1.3)
- Viene aggiunto un MAC
- È crittografato
- viene aggiunto un header

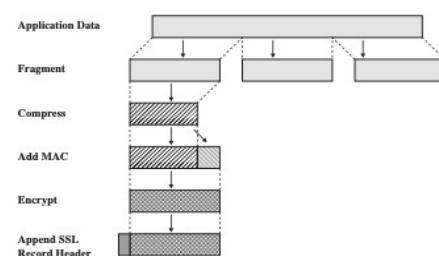


Figure 17.3 SSL Record Protocol Operation

Nota TLS 1.2 usava **MAC-then-Encrypt**, che ora è deprecato.

Nel capitolo di security abbiamo parlato del contrario, ovvero Encrypt-then-MAC, il motivo è che richiede un'operazione in meno, se facciamo E-t-M cifriamo e applichiamo il MAC, così chi riceve verifica se il MAC è sbagliato e se il MAC è sbagliato non fa la decrypt, questa è l'operazione in meno.

Il **TLS record protocol** fornisce la sicurezza della connessione che ha due proprietà di base:

- **La connessione è privata.** La crittografia simmetrica viene utilizzata per la crittografia dei dati (ad esempio, AES). Le chiavi per questa crittografia simmetrica sono generate in modo univoco per ogni connessione e si basano su un segreto negoziato da un altro protocollo (come il TLS Handshake Protocol).
- **La connessione è reliable.** Il trasporto dei messaggi include un controllo dell'integrità del messaggio utilizzando un MAC con chiave. Le funzioni di hash sicure (ad esempio, SHA-256, ecc.) vengono utilizzate per i calcoli MAC.

Handshake Protocol TLS

Il protocollo TLS Handshake consente al server e al client di autenticarsi a vicenda e di negoziare un algoritmo di crittografia e chiavi crittografiche prima che il protocollo dell'applicazione trasmetta o riceva il suo primo byte di dati. Alcune delle cose che devono essere negoziate:

- Quali funzioni crittografiche verranno utilizzate: RSA? Crittografia a curva ellittica?
- Quali funzioni hash verranno utilizzate: SHA-256? SHA-384?
- Come negoziamo la chiave simmetrica: DH? Scambio RSA?

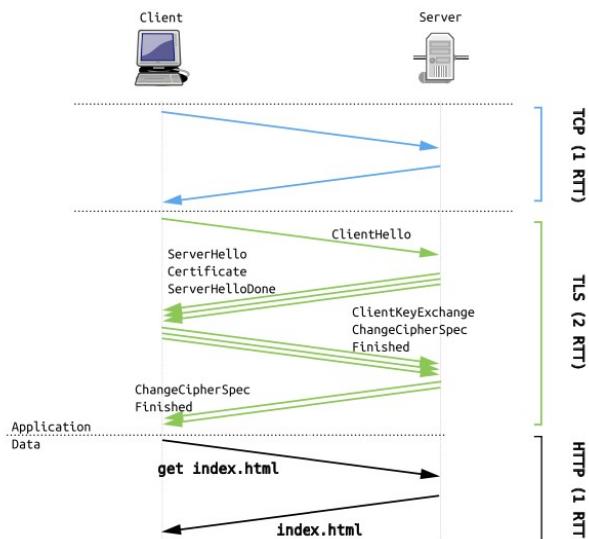
La stretta di mano è **critica** perché, per definizione, **accade prima che qualsiasi chiave sia negoziata**, quindi un MiTM può modificarla.

Il protocollo TLS Handshake fornisce la sicurezza della connessione che ha tre proprietà di base:

1. L'identità del peer può essere autenticata utilizzando crittografia asimmetrica o a chiave pubblica (ad esempio, RSA, DSA, ecc.). Questa autenticazione può essere resa facoltativa, ma è **generalmente richiesta per almeno uno dei peer**.
2. La negoziazione di un segreto condiviso è sicura: il segreto negoziato non è disponibile per gli intercettatori, e per qualsiasi connessione autenticata il segreto non può essere ottenuto, **nemmeno da un utente malintenzionato che può mettersi nel mezzo della connessione**.
3. La negoziazione è reliable: nessun utente malintenzionato può modificare la comunicazione di negoziazione senza essere rilevato dalle parti della comunicazione.

Operazioni

- I primi tre messaggi sono la 3-way Handshake di TCP
- Ciò include il messaggio **ClientHello**, che riproduce anche la funzione ACK di TCP handshake
- I certificati vengono scambiati nei messaggi **Certificate**.
- In HTTPS il messaggio **Certificate** contiene l'intero albero dei certificati, dalla CA principale fino al certificato del sito web !!!!
- Il messaggio **ChangeCipherSpec** significa che la negoziazione delle chiavi è finita. I prossimi messaggi saranno crittografati e autenticati.
- Il messaggio **Finished** è il primo crittografato e autenticato. Contiene un hash di tutti i messaggi precedenti per verificare (a posteriori) che non sono stati modificati da un MiTM.
- Per 3 RTT non viene ricevuto un singolo byte utile dal client
- **Il primo byte utile arriva dopo 4 RTT** (il compresso contrassegnato come index.html)
- La configurazione della connessione è molto più lunga del normale TCP. **Normalmente l'autenticazione è unidirezionale**



Si noti che normalmente, come utente Internet, non è necessario un certificato valido. **Con HTTPS, il più delle volte solo il server web fornisce un certificato valido al browser**. Il browser è sicuro dell'identità del server, ma non il

contrario, che generalmente non è richiesto. Tuttavia, il client e il server possono ancora negoziare una chiave condivisa per rendere sicura la connessione TCP in entrambe le direzioni con la crittografia a chiave simmetrica.

TLS 1.2 era molto buggato

TLS 1.2 ha sofferto di un gran numero di bug critici, nascosti nelle caratteristiche più complesse del protocollo. Questi bug sono stati patchati dai fornitori e con nuove revisioni dello standard.

TLS 1.3 è una riscrittura completa del protocollo, che semplifica TLS, rimuovendo molte delle funzionalità raramente utilizzate. La rimozione delle funzionalità non sembra un miglioramento, ma le funzionalità aumentano la superficie di attacco e se vengono utilizzate raramente, non ha senso mantenerle

Modifiche di TLS 1.3

Solo 5 algoritmi di crittografia/autenticazione, con un solo schema di blocchi

- Una stretta di mano iniziale semplificata che può funzionare in un solo RTT
- La sessione di sicurezza può essere ripresa su una connessione diversa con 0 RTT
- Forward Security è obbligatoria.

Forward Security TLS 1.3

Forward Secrecy è una caratteristica di specifici protocolli di key agreement che garantisce che le chiavi di sessione non siano compromesse anche se la chiave privata del server è compromessa.

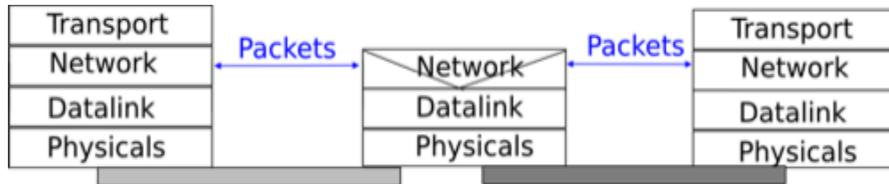
In pratica, diciamo che :

1. Eve intercetta una comunicazione TLS tra Alice e Bob e ne salva il contenuto, anche senza essere in grado di decifrarlo
2. Più tardi, Eve irrompe nel server di Bob e ruba la chiave privata.
3. Eve non dovrebbe essere in grado di decifrare il traffico che ha salvato in anticipo

TLS 1.2 non impone Forward Secrecy, mentre TLS 1.3 lo fa.

Forward Secrecy normalmente implica l'uso di **chiavi pubbliche effimere**: chiavi che vengono generate durante la stretta di mano e cancellate subito dopo essere state utilizzate. Questo è possibile con Diffie Hellman, perché con DH non è necessario utilizzare dati stabili di client o server (es. Chiave privata di uno dei due), non vengono mai comunicate (vedi capitolo di Security).

Protocollo IPv4



Ci sono due tipi di datalink:

- **PPP**, Point-To-Point, due dispositivi che comunicano. I frame non devono essere reindirizzati.
- **LAN**, local area network, rete di dispositivi collegati tra di loro, ognuno con un indirizzo univoco (MAC). I frame contengono MAC di destinazione e Mac di partenza.

Ogni host connesso a una rete ha un numero di schede di interfaccia di rete (NIC). A una NIC può essere assegnato un indirizzo IP. Il più delle volte, assumiamo che un computer che non è un router abbia una sola NIC, quindi diciamo, per semplicità, l'indirizzo IP del computer.

Gli indirizzi IPv4 sono composti da 32 bit, normalmente rappresentati in "notazione decimale punteggiata" (cioè quattro numeri decimali da 0 a 255 separati da punti)

Esempio:

1.2.3.4 → 00000001000000100000001100000100
127.0.0.1 → 01111110000000000000000000000000
255.255.255.255 → 111111111111111111111111111111

Una rete è organizzata in sottoreti (sottoreti) interconnesse da router. Una sottorete è identificata da un gruppo di indirizzi nella forma x.y.z.w/N, il numero N dopo la "barra" viene utilizzato per calcolare il numero totale di indirizzi nella sottorete: 2^{32-N} .

Un indirizzo IP è quindi diviso in un prefisso che identifica una sottorete (netid) e un suffisso che identifica la NIC all'interno della rete (hostid). Tutte le NIC nella stessa sottorete condividono lo stesso prefisso.

Mentre in passato la dimensione del prefisso era costante e lo spazio degli indirizzi era diviso in 3 classi:

A → /8, B → /16, C → /24

Indirizzo di Broadcast

L'indirizzo Broadcast viene utilizzato per inviare un pacchetto a ogni IP nella sottorete.

Quindi, se una NIC ha l'indirizzo 1.2.3.4 in una sottorete /24 e vuole inviare un pacchetto a tutti gli altri host, lo invierà a 1.2.3.255.

C'è un caso speciale in cui a una NIC non è ancora stato assegnato un indirizzo IP, ma vuole inviare un pacchetto di trasmissione. Poiché non conosce ancora il netid, non può inviarlo. Quindi utilizzerà l'"Indirizzo di Broadcast limitato": 255.255.255.255. Ogni NIC che riceve un pacchetto di trasmissione (entrambi i tipi) dovrebbe passarlo nello stack, come se fosse diretto all'IP della NIC stessa.

Indirizzo di Multicast

Gli indirizzi che iniziano con 1110, ovvero 224.0.0.0/4 sono indirizzi multicast. Questi sono utilizzati da gruppi di host che vogliono comunicare, quelli ben noti sono:

- 224.0.0.1 (tutti gli host nella sottorete, esclusi i router)
- 224.0.0.2 (tutti i router)

Indirizzi speciali

- 0.0.0.0/8: questa macchina, in questa rete. Può essere utilizzato solo come indirizzo sorgente, ad esempio quando si utilizza DHCP.

- 127.0.0.0/8: interfaccia loopback. Ogni host IP deve avere un'interfaccia di loopback virtuale, non collegata a nessuna interfaccia fisica, con indirizzo 127.0.0.1. È utilizzato per gestire le connessioni TCP/IP tra i processi all'interno della macchina.
- 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16: classi private non instradabili: significa che un router connesso a Internet non deve instradare i pacchetti da/per queste reti (in seguito spiegheremo cosa significa privato).
- 169.254.0.0/16: link-local. Non instradabile, viene utilizzato in modo che le interfacce di rete a cui non è stato ancora assegnato un indirizzo IP possano comunicare. Alcuni sistemi operativi assegnano un indirizzo casuale in questa classe alle interfacce all'avvio.

Routers

I router sono dispositivi che appartengono a più di una rete. Ciò significa che i router hanno più di una NIC e ogni NIC ha un indirizzo in una sottorete diversa. Pertanto, possono ricevere un pacchetto da una rete e consegnarlo a un'altra.

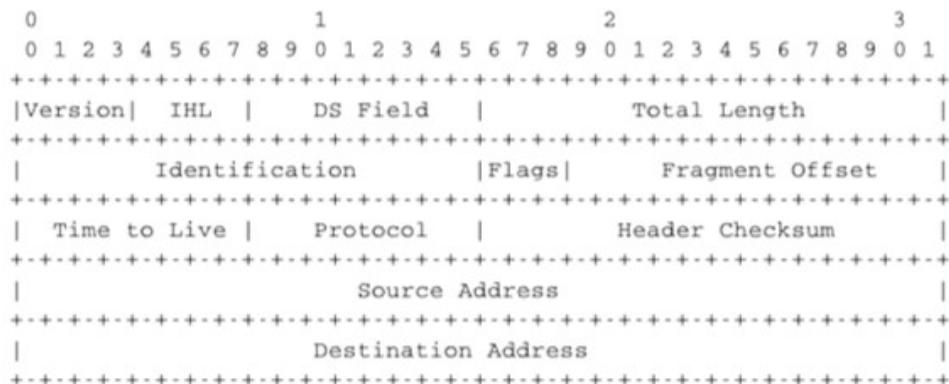
Ogni sottorete, a meno che non sia isolata, ha almeno un router. Ogni host ha un router predefinito che utilizza per inviare pacchetti quando non sono indirizzati a qualche indirizzo IP nella stessa sottorete.

Local Routes Vs Gateway

Quando configuri un indirizzo IP sul tuo PC, nella tabella di inoltro viene aggiunto automaticamente un percorso per la rete locale. I pacchetti inviati a qualche indirizzo IP locale verranno inviati al dispositivo e si presume che la destinazione sia raggiungibile direttamente.

Invece, una voce di routing che utilizzerà un gateway per raggiungere la destinazione, dovrà specificare un gateway, che è generalmente specificato con il suo IP.

Header IPv4



- **Version:** 4/6
- **IP Header Length:** numero di 32 bit words (min 5 max 15)
- **DS:** QoS related, per esempio quando viene ha inizio una Explicit Congestion Notification
- **Total Length:** lunghezza in byte di tutto il pacchetto, max 65535B
- **Identification:** per abbinare i fragments
- **Flags:** Evil Bit, non frammentare, più frammenti
- **Fragment Offset:** offset di un fragment nel pacchetto originale (in multipli di 8 byte. Solo dati, esclude l'intestazione)
- **TTL:** Decrementato ad ogni hop
- **Protocol:** 1/ICMP, 6/TCP, 17/UDP
- **Header Chk:** checksum per i dati dell'header
- **Source and Destination addresses**

Fragmentation

Sappiamo che diversi datalink hanno differenti MTUs. Un pacchetto di dimensioni maggiori della MTU della rete in cui sta entrando deve essere frammentato. L'IP Fragmentation è fatta da qualunque router sul cammino e assemblata alla destinazione.

Ogni fragment dello stesso pacchetto IP originale ha:

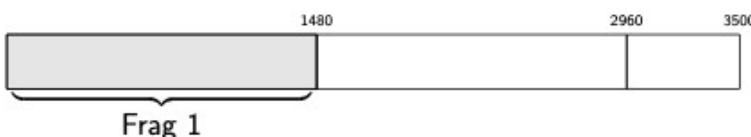
- lo stesso campo di identificazione
- campo di lunghezza impostato sulla lunghezza del frammento
- un offset diverso, espresso in multipli di 8 byte
- il flag More Fragments impostato su 1, tranne l'ultimo

Ogni frammento viaggia verso la destinazione come pacchetto indipendente. Potrebbero anche prendere percorsi diversi, solo quando arrivano a destinazione, vengono riassemblati utilizzando i valori di identificazione e offset, in un unico pacchetto IP.

La destinazione non sa chi ha frammentato i pacchetti, può accadere in qualsiasi salto dall'origine alla destinazione.

Si noti che la frammentazione introduce diversi problemi, ma non può essere completamente deprecata per la retrocompatibilità

Esempio con size del pacchetto 3500 e MTU = 1500

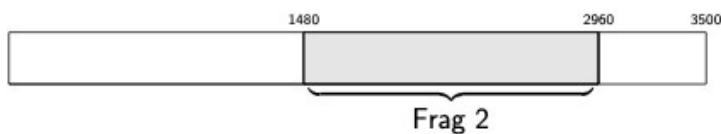


- First Fragment (size: 20 Byte IP header + 1480 Byte payload <= MTU):



Frag 1 contains the transport layer header, the IP header contains:

- more fragment is set to True, fragment offset is set to 0, length is 1500

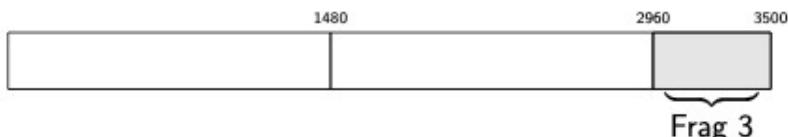


- Second Fragment (size: 20 Byte IP header + 1480 Byte payload <= MTU):



The IP header contains:

- more fragment is set to True, fragment offset is $1480/8 = 185$, length is 1500



- Third Fragment (size: 20 Byte IP header + 540 Byte payload <= MTU):



The IP header contains:

- more fragment is set to False, fragment offset is 370, length is 560

Gli indirizzi ip sono 2^{32} , inizialmente ok ma ora sono troppo pochi, per questo esiste il NAT (e IPv6).

Indirizzi IP Privati

La quantità totale di indirizzi IPv4 è $2^{32} = 4.294.967.296$.

Mentre questo era un numero considerato sufficiente negli anni '80, oggi abbiamo bisogno di più indirizzi. Una soluzione a lungo termine è stata quella di ridisegnare l'IP, con una nuova versione IPv6 che utilizza indirizzi a 128 bit. Una soluzione a breve termine è stata quella di riutilizzare alcuni indirizzi, introducendo l'indirizzo di traduzione di rete.

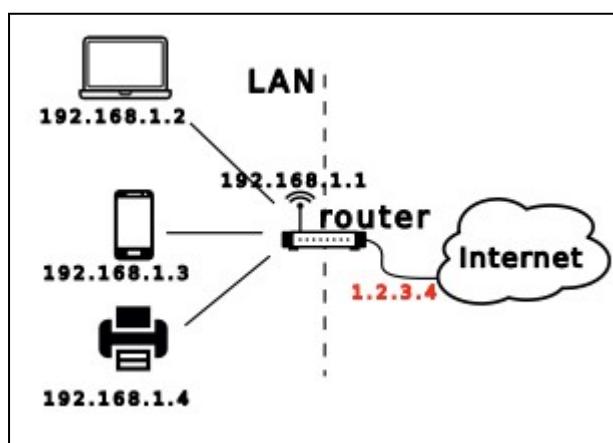
Tre classi di indirizzi sono state riservate per l'uso gratuito:

- 10.0.0.0 - 10.255.255.255 (/8)
- 172.16.0.0 - 172.31.255.255 (/20)
- 192.168.0.0 - 192.168.255.255 (/16)

Gli amministratori di rete possono utilizzarli nelle proprie sottoreti senza la necessità di alcun coordinamento tra di loro, queste sono chiamate "reti private". Una rete privata ha normalmente un router di confine che lo collega all'"Internet pubblico" (che utilizza indirizzi IP pubblici). Le reti private possono riutilizzare gli stessi indirizzi IP, per questo motivo un router di confine non deve instradare ai pacchetti Internet pubblici da/a un IP privato.

Network Address Translation

NAT è una tecnica per abbinare un IP pubblico con uno privato. È una tecnica generica che si trova tra la rete e il livello di trasporto, può essere configurata in vari modi. Il più comune è Network Address Port Translation (NAPT). L'idea è che un router di confine tradurrà i pacchetti IP dallo spazio IP pubblico a quello privato.



Dentro la LAN i dispositivi comunicano con gli indirizzi privati, per comunicare su internet il router che implementa NAT li trasforma

Incoming to the private IP	Outgoing from the public IP
UDP, 192.168.1.2, 8.8.8.8, 12909, 43	UDP, 1.2.3.4, 8.8.8.8, 12909, 43
TCP, 192.168.1.3, 4.3.2.1, 9823, 80	TCP, 1.2.3.4, 4.3.2.1, 9823, 80
TCP, 192.168.1.4, 4.3.2.1, 9823, 80	TCP, 1.2.3.4, 4.3.2.1, 9824, 80

La tabella mappa un pacchetto che arriva all'interfaccia privata del NAT (192.168.1.1 nella figura precedente) a un pacchetto che verrà inviato sull'interfaccia pubblica (1.2.3.4).

Il secondo e il terzo flusso utilizzano la stessa tupla, ad eccezione dell'indirizzo sorgente. **NAT è autorizzato a rimappare la porta di origine per evitare un conflitto nella tabella.**

C'è una tabella inversa per il traffico in entrata, che mappa, ad esempio:

TCP, 4.3.2.1, 1.2.3.4, 80, 9824 → TCP, 4.3.2.1, **192.168.1.4**, 80, **9823**

Il nat è il top ma ha dei problemi

NAT costringe un router a mantenere uno stato (la tabella NAT). L'intero livello IP è progettato per NON avere uno stato.

I dispositivi NAT sono normalmente dispositivi firewall, non semplici router, ciò significa che il NAT deve rilevare l'inizio e la fine di un flusso di traffico.

Questo è ragionevolmente facile con TCP, non con UDP. E cosa succede se la connessione TCP viene uccisa bruscamente? NAT deve memorizzare nella cache il contenuto per qualche tempo prima di rimuoverlo.

I dispositivi NAT riassembreranno i frammenti (perché?)

- Un host dietro un NAT può aprire una connessione TCP verso un host con un IP pubblico, ma non il contrario
- Due host dietro un NAT non possono comunicare tra loro, se non usano un server esterno
- Due host su Internet possono avere lo stesso indirizzo privato, se si trovano su LAN diverse.
- Due host sulla stessa LAN non possono avere lo stesso indirizzo privato.
- Due host su Internet non possono avere lo stesso indirizzo pubblico.

Forwarding Table

La forwarding table mappa un determinato indirizzo di rete con una determinata interfaccia in uscita di un router.

Quando arriva un pacchetto, viene abbinato al contenuto della tabella di inoltro e viene scelta l'interfaccia corrispondente alla linea con la corrispondenza del prefisso di rete più lunga.

Destination Network	Outgoing NIC
0.0.0.0/0	0
124.156.12.0/24	1
124.156.13.0/24	0
124.156.13.128/26	2
200.212.12.127/26	3
200.212.12.64/28	4

Esempio IP Address 1

Possiedo una sottorete 1.2.1.0/24. Posso assegnare a uno dei miei host l'indirizzo 1.2.1.255?

NO! Il NETid è 1.2.1, e l'HOSTid, tutti 1, è per l'indirizzo di trasmissione.

Possiedo una sottorete 1.2.0.0/16. Posso assegnare a uno dei miei host l'indirizzo 1.2.1.0?

SÌ! Il NETid è 1.2, l'indirizzo di rete è 1.2.0.0 (tutti zero HOSTid), quindi 1.2.1.0 è un indirizzo IP valido.

Possiedo una sottorete 1.2.1.0/24. Posso assegnare a uno dei miei host l'indirizzo 1.2.1.0?

NO! Il NETid è 1.2.0 e l'HOSTid tutto zero è per il NETid.

Un'organizzazione ha bisogno di 15 indirizzi IP e deve affittarli da un fornitore di rete. Supponiamo che il fornitore di rete possieda 1.2.0.0/16.

Qual è la lunghezza minima netid che l'organizzazione deve affittare?

Assegna gli indirizzi di rete, inizia la numerazione da "tutti 0", ad esempio, se hai bisogno di un /24, dovresti usare 1.2.0.0/24 (terzo byte impostato su 0). Indica specificamente la gamma consentita di indirizzi.

Per contenere 15 indirizzi hai bisogno di 4 bit, quindi 1.2.0.0/28 sarebbe sufficiente: da 1.2.0.0 a 1.2.0.15 (16 IP, di cui 15 saranno effettivamente utilizzati).

Tuttavia, ogni sottorete ha due indirizzi che non possono essere utilizzati, in questo caso:

- 1.2.0.0 (indirizzo di rete)
- 1.2.0.15 (15 == 00001111) (indirizzo broadcast)

Quindi l'unico modo per avere 15 indirizzi è leasing 1.2.0.0/27 che fornisce 30 indirizzi utilizzabili, di cui solo 15 saranno effettivamente utilizzati.

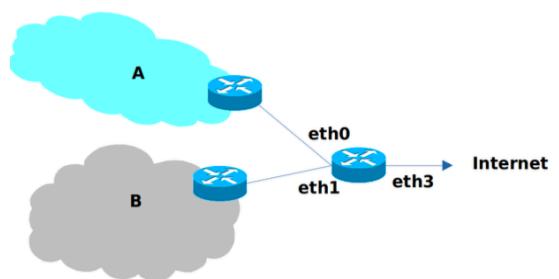
Gli indirizzi disponibili andranno da 1.2.0.0 a 1.2.0.31, di cui 30 sono utilizzabili (hostid da 1 a 30)

Esempio IP Address 2

Un'organizzazione ha due capitoli, A e B. Nella rete A ci sono 14 host, nella rete B ce ne sono 16. Il provider possiede 1.2.0.0/16.

Scopri:

- La maschera di rete minima che l'organizzazione deve noleggiare dal provider.
- Una possibile allocazione e intervalli, questa volta assumere "tutti 1" per l'ID di rete.
- Le tabelle di routing del router connesso a Internet.



Il numero totale di indirizzi IP richiesti è 30, quindi una netmask /27 è sufficiente. Tuttavia, l'amministratore vuole partizionare la rete in due. Quindi ogni sottorete spreca 2 indirizzi IP, quindi ha bisogno di $30 + 2 * 2 = 34$ indirizzi, ciò richiede una rete /26, $2^6 = 64$ indirizzi di cui ne userà solo 30.

Affitta 1.2.255.192/26 (192 == **11000000**) e lo divide in due sottoreti: 1.2.255.192/27 (192 == **11000000**) e 1.2.255.224/27 (224 == **11100000**).

1.2.255.192/27 (192 == **11000000**): Questo fornirà indirizzi da 1.2.255.192 a 1.2.255.223. Questi sono 32 indirizzi di cui l'ultimo e il primo non possono essere utilizzati.

1.2.255.224/27 (224 == **11100000**): Questo fornirà indirizzi da 1.2.255.224 a 1.2.255.255. Questi sono 32 possibili indirizzi di cui il primo e l'ultimo non possono essere utilizzati.

In entrambe le reti abbiamo abbastanza indirizzi IP per gli host e anche per il router.

Tuttavia, gli indirizzi IP costano e l'amministratore non è contento di noleggiare 64 indirizzi e di usarne solo 30.

Una delle reti si adatta a una sottorete /28 e l'altra si adatta a una /27.

Quindi potrebbe affittare due diverse sottoreti.

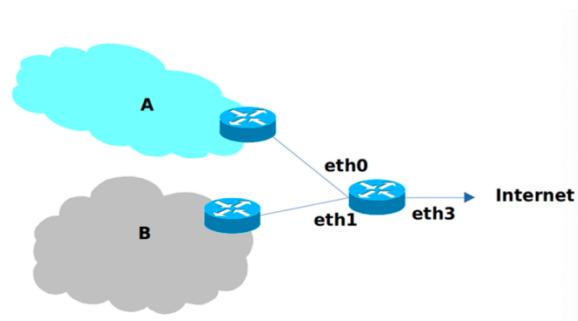
1.2.255.192/27 (192 == **11000000**): Questo fornirà indirizzi da 1.2.255.192 a 1.2.255.223. Questi sono 30 indirizzi utilizzabili per la rete B • 1.2.255.224/28 (224 == **11100000**): questo fornirà indirizzi da 1.2.255.224 a 1.2.255.239.

Questi sono 14 indirizzi utilizzabili per la rete A.

Possiede ancora più indirizzi di quelli di cui ha bisogno ($32+16=48$) ma pagherà meno di prima. Tuttavia, possiede due sottoreti pubbliche, e non solo una, in quanto non copre 1.2.255.240/28 (240 == **11110000**).

The FIB will be:

Destination Network	Out NIC
0.0.0.0/0	eth3
1.2.255.192/27	eth1
1.2.255.224/27	eth0



Quindi un pacchetto con dest. IP 1.2.255.250 verrà inviato al gateway predefinito. Questo è corretto perché l'intervallo 240-255 non fa più parte della rete locale.

Internet Control Message Protocol (ICMP)

ICMP è un protocollo di segnalazione: in genere non contiene informazioni sull'utente. Si usa anche per fare troubleshooting.

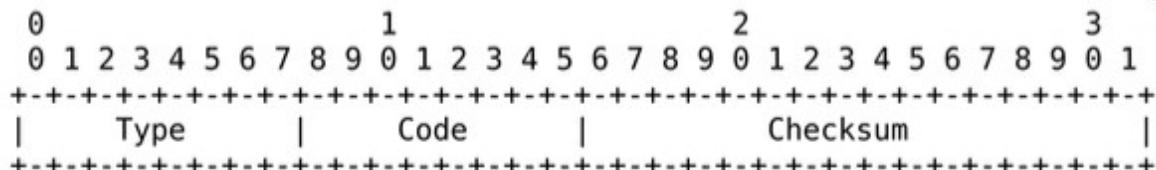
Trasmette messaggi di errore e messaggi relativi al funzionamento e allo stato della rete.

IP e ICMP sono interdipendenti:

- IP dipende da ICMP per la gestione degli errori
- ICMP utilizza un datagram IP per trasmettere messaggi

ICMP non utilizza alcun protocollo a livello di trasporto, quindi **si basa direttamente sul livello di rete**

Header ICMP



Un pacchetto ICMP viene generato quando un altro pacchetto produce un evento anomalo. I primi byte del pacchetto che ha generato l'errore possono essere inclusi nel corpo del pacchetto ICMP

Tipo e codice: identificare il tipo di errore/evento. Alcuni sono:

- **Destination** (3): destinazione non instradabile/la porta di destinazione è chiusa, o è necessaria la frammentazione (Don't Fragment è impostato su 1, ma MTU non è sufficiente)
- **Redirect** (5): il router notifica che c'è un next hop migliore (generalmente ignorato)
- **TTL exceeded** (11): Il pacchetto è rimasto bloccato in un ciclo
- **Echo request/reply** (8/0): dimmi se esisti

Datagramma ICMP

I datagrammi contenenti ICMP sono trattati come qualsiasi altro. L'unica eccezione è che se un messaggio di errore ICMP causa un errore, non viene inviato alcun altro messaggio.

Il motivo è ovvio: non si vuole creare un ping-pong di messaggi ICMP che non finisce mai.

Protocollo IPv6

Il numero di indirizzi IPv4 è stato presto compreso come troppo piccolo. NAT avrebbe dovuto essere una soluzione temporanea, una soluzione definitiva è IPv6.

IPv6 è ora supportato da tutti i sistemi operativi, ma ancora, dopo 25 anni della sua definizione, la maggior parte degli host Internet utilizza IPv4.

Tuttavia, il suo utilizzo aumenta con il tempo. Tuttavia, ciò non significa che l'utilizzo di IPv4 sia diminuito. Quindi, anche se è chiaro che IPv6 fornisce vantaggi, non è chiaro che sostituirà finalmente IPv4.

La ragione della lenta penetrazione è che IPv6 non è retrocompatibile con IPv4

Formato indirizzo IPv6

Un indirizzo IPv6 è composto da 128 bit ed è scritto in formato esadecimale separato da colonne, come il seguente:

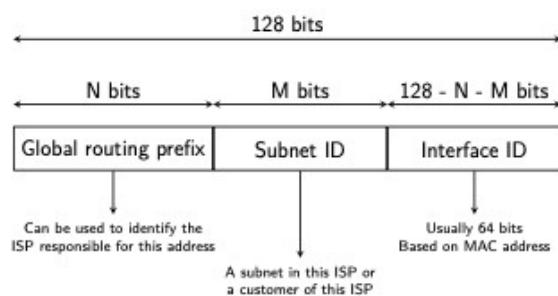
abcd:ef01:2345:6789:abcd:ef01:2345:6789

Nei casi in cui ci sono blocchi di zeri, possono essere compattati:

ff01:0:0:0:0:0:101 è rappresentato come ff01::101

Come in IPv4 l'indirizzo ha un prefisso di una certa lunghezza:

2001:0db8::cd30:0:0:0/60



Disponibilità

Nelle distribuzioni odierne, gli identificatori dell'interfaccia sono sempre larghi 64 bit. Ciò implica che mentre ci sono 2^{128} indirizzi IPv6 diversi, devono essere raggruppati in 2^{64} sottoreti.

Le attuali allocazioni IPv6 fanno parte della sottorete 2000::/3

Le dimensioni tipiche dei blocchi di indirizzi IPv6 sono:

- /32 per un provider di servizi Internet
- /48 per una singola azienda
- /56 per siti di piccoli utenti
- /64 per un singolo utente (ad esempio un utente domestico connesso tramite ADSL)
- /128 nel raro caso in cui è noto che non sarà collegato più di un host

In pratica, possiamo avere $2^{32}-3$ ISP nel mondo, cioè circa una persona ogni 15 nel mondo.

Ogni ISP può allocare $2^{48}-32 = 65536$ grandi aziende

Ogni utente che ottiene /64 può ospitare 18×1018 indirizzi nella sua rete domestica.

Questo è molto, ma chi se ne frega, abbiamo 6×1023 indirizzi IP per metro quadrato della superficie terrestre.

Assegnamento indirizzoIpv6

Considerando quei numeri, non c'è più bisogno di ottimizzare l'assegnazione della sottorete. La maggior parte dello spazio degli indirizzi in una rete sarà comunque inutilizzato, quindi il tipo di ragionamento che abbiamo fatto con la sottorete IPv4 (come ridurre al minimo gli indirizzi e used?) È obsoleto.

Indirizzi speciali

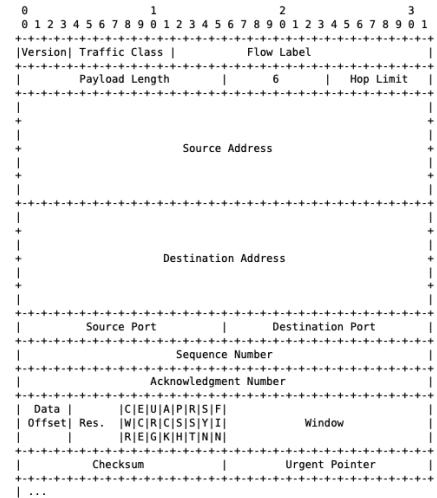
- fc00::/7, unique local unicast, ip privati usato per testare protocolli
- 0:0:0:0:0:0:1, loopback interface
- Ff::/8, multicast address
- Fe80::/10, link local unicast

Header IPv6

- **Next Header:** sostituisce il tipo di protocollo di IPv4 (6=TCP)
- **Hop Limit:** sostituisce il TTL
- **Non esiste un checksum.** Il checksum IP è stato completamente rimosso.
- **Non c'è supporto per la frammentazione.**

Ciò che segue in questo caso è un segmento TCP. IPv6 non influisce sul livello di trasporto, oltre al cambiamento degli indirizzi.

Gli header possono essere più di uno, in modo che l'header successivo punti a un altro header, con il proprio header successivo. Questo rende IPv6 estensibile.



IPsec

IPSec è un insieme di standard che implementano la crittografia e l'autenticazione sui pacchetti IP.

IPSec è stato introdotto in IPv4 come funzionalità opzionale. Con IPv6, il supporto per IPSec è obbligatorio. Questo non significa che ogni router lo usa, significa solo che ogni router deve supportarlo.

Fragmentation

La frammentazione sul percorso fatta dai router è un dolore, perché costringe i router a utilizzare la memoria e il calcolo.

IPv6 ha rimosso il supporto per la frammentazione on-path.

Il mittente può ancora frammentare i pacchetti usando un'intestazione dedicata, ma non ha molto senso, poiché è più facile dividere semplicemente i segmenti in pacchetti più piccoli

ICMPv6

Man mano che il livello IP cambia, anche il protocollo ICMP deve essere aggiornato. Nella maggior parte delle funzionalità di base ICMP rimane lo stesso, una caratteristica rilevante è il codice Packet Too Big ICMP. Questo sostituisce il codice Fragmentation Needed di ICMPv4.

Intradomain Routing

Le reti carrier grade (alto livello di disponibilità e grandezza) di Internet Service Provider (ISP) hanno topologie interne complicate, così come grandi organizzazioni (come le università).

Queste reti sono partionate in diverse sottoreti collegate da router, ognuna con il proprio indirizzamento.

Ci sono **border router** che collegano una rete a un'altra e devono inoltrare i pacchetti da uno all'altro.

Un protocollo di routing **intradomain** implementa il control plane su una rete nello stesso dominio amministrativo, cioè di proprietà di un singolo amministratore che decide le politiche di routing interne da solo.

Un singolo dominio amministrativo ci si aspetta che abbia un numero limitato di sottoreti e router che le collegano. Ordine di decine, o centinaia al massimo.

Questo è abbastanza piccolo da utilizzare un protocollo di **routing link-state**, che sappiamo, può reagire ai guasti più velocemente di un protocollo di routing DV, ma produce più sovraccarico di rete.

Open Shortest Path First (OSPF)

Due dei protocolli più utilizzati sono **Open Shortest Path First (OSPF, RFC 2328)** e IS-IS (RFC 1195).

OSPF segue i principi di base di un protocollo di routing link state, con un paio di modifiche. Può essere usato con IPv4 e IPv6 e la sua logica non cambia.

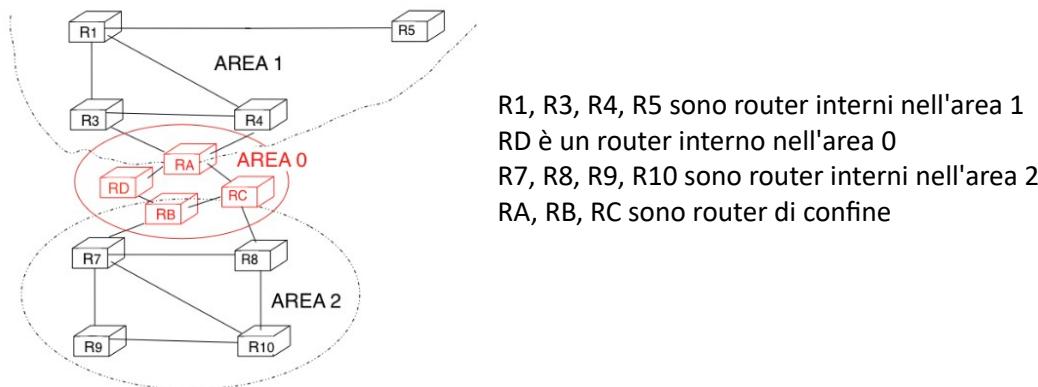
OSPF permette, inoltre, di scegliere quale interfaccia usare e determinare quali collegamenti falliscono.

OSPF Areas

L'amministratore di rete partitiona la rete in diverse aree. Un'area è una parte fisicamente contigua della rete, collegata a qualche altra area da un numero limitato di router. Esistono due tipi di router per OSPF:

- **Router interni**: router che sono collegati solo ad altri router nella stessa area
- **Router di frontiera**: router che appartengono a più di un'area

Alcuni dei router esportano un prefisso di rete, perché sono i gateway di una sottorete.



Backbone Area

L'area zero ha un significato speciale per OSPF, in quanto raccoglie tutti i router di confine e eventualmente alcuni router che non appartengono a un'altra area (come RD).

I router che non appartengono all'area backbone possono raggiungere gli altri solo attraversando l'area backbone.

Ogni rete ha un'area zero e solo una.

Comportamento Link State

All'interno di ogni area non backbone, i router distribuiscono la topologia dell'area scambiando pacchetti LSP con gli altri router dell'area. I router interni non conoscono la topologia di altre aree, ma ogni router sa come raggiungere l'area backbone. All'interno di un'area, i router si scambiano solo pacchetti LSP per tutte le destinazioni raggiungibili all'interno dell'area.

Comportamento DV

Il routing inter-area viene invece eseguito scambiando Distance Vector tra router di confine. Questo aiuta a ridurre il sovraccarico dovuto ai pacchetti di controllo del link state.

Autonomous System

Un **sistema autonomo** (AS) è una raccolta di prefissi di routing IP (Internet Protocol) connessi sotto il controllo di uno o più operatori di rete per conto di una singola entità amministrativa o dominio [Hawkinson e Bates, 1996].

Ci sono 2 tipi di AS:

- **Stub AS**: Questo tipo di AS non fornisce servizi di transito ad altri. Sono clienti finali, come le università, le organizzazioni e la maggior parte delle aziende. Alcuni AS stub si connettono solo a un altro AS (in genere un provider a monte). Sono chiamati single-homed stub ASes. Alcuni AS stub si connettono a più AS e sono chiamati AS stub multi-homed. Un multi-home stub AS non consentirà al traffico da un AS di passare a un altro AS, cioè non fornisce un servizio di transito.
- **Transit AS**: Questo tipo di AS si connette a più AS e offre di instradare i dati da un AS a un altro AS. Fornisce servizi di transito.

In pratica, ogni AS è una raccolta di reti che utilizzano il proprio protocollo di routing interno. Ciò che li rende un AS è che sono **di proprietà e gestiti dalla stessa entità**: un ISP, un'azienda, un governo, un'università...

Quindi non esiste una definizione tecnica rigorosa di AS, fa riferimento a qualche entità amministrativa. Ad ogni AS deve essere assegnato un numero univoco dalla **IANA**, l'Internet Assigned Numbers Authority.

Gli AS hanno un punto di presenza (**POP**), cioè luoghi in cui sono fisicamente accessibili. Gli AS sono responsabili di alcuni prefissi, cioè gestiscono set di indirizzi IP. Questi indirizzi IP devono essere raggiunti dal resto di Internet; quindi, gli AS devono connettersi tra loro.

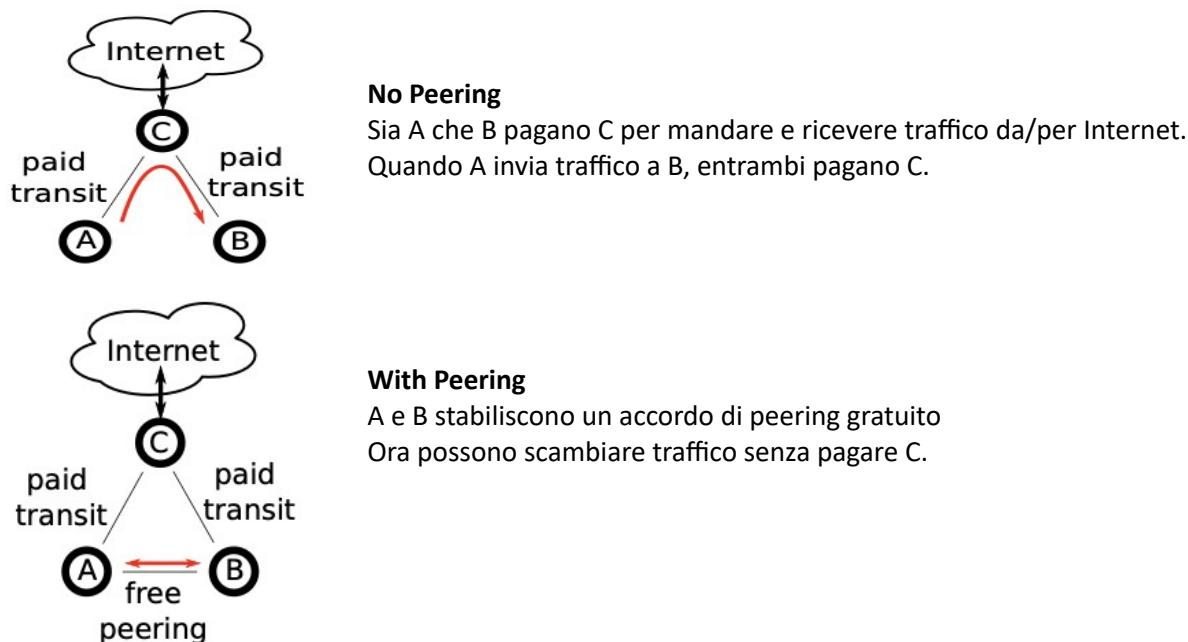
Internet Exchange Points (IXP): un centro dati condiviso in cui gli AS mettono i loro POP e si connettono tra loro. Gli ISP sono più efficienti, poiché con un singolo POP un AS può connettersi a molti altri AS.

Ci sono principalmente due modi di connettere gli AS:

- **Private Peering**: Questa è una connessione diretta punto-punto tra due router di due AS. I proprietari degli AS pagano per l'infrastruttura necessaria per collegare i due AS, cioè: fibra, collegamenti wireless, cavi sottomarini e l'hardware necessario. Il peering privato spesso non è dichiarato pubblicamente, non conosciamo tutto il peering privato (e quindi le connessioni fisiche) che esistono.
- **Transit Agreement**: Un accordo commerciale tra AS A e B in cui B offre di collegare A a Internet. B si comporta come un gateway per A.

Una relazione di transito include generalmente una tassa: l'entità A paga l'entità B in modo che il traffico da A possa raggiungere Internet.

Un accordo di peering (chiamato anche costo condiviso o peering gratuito) è generalmente un accordo gratuito tra A e B in modo che possano scambiare traffico senza costi. Gli accordi di peering sono fatti per risparmiare denaro.



Finché il peering è possibile a causa della vicinanza fisica, è conveniente farlo.

Border Gateway Protocol (BGP)

Mentre all'interno di un singolo AS ogni amministratore di rete può utilizzare il protocollo di routing che preferisce, il routing tra AS deve utilizzare un singolo protocollo. Questo protocollo è **BGP**, il **Border Gateway Protocol**, INTERNET.

Esportare i prefissi

Sappiamo che il ruolo di un protocollo di routing è quello di consentire ai router di scambiare prefissi con altri router, in modo che possano costruire le tabelle di routing su qualsiasi rete esistente.

Nel routing intradominio questo accade effettivamente, un router OSPF di confine espone sempre tutti i prefissi che conosce.

Nel routing interdomain questo non è sempre vero, perché ci sono relazioni commerciali. In generale:

- su una **relazione cliente → provider**, il dominio cliente pubblica al proprio provider i propri prefissi e tutti i percorsi che ha appreso dai propri clienti.
- su una **relazione provider → cliente**, il provider pubblica tutti i percorsi che conosce al suo cliente.
- su una **relazione di peering a costo condiviso** un dominio pubblica solo i suoi percorsi/prefissi interni e i percorsi che ha appreso dai suoi clienti.

BGP Routers

Ogni AS ha bisogno di almeno un router che supporti il protocollo BGP .

I router BGP **annunciano i prefissi di rete**: un annuncio è come dire che la rete con il prefisso X è ospitata nel AS Y, in modo simile a quello che accade con OSPF.

I router BGP annunciano i propri prefissi, ma possono anche annunciare altri prefissi di altri AS.

Un **router di frontiera** eseguirà sia BGP che il protocollo di routing interno (come OSPF).

I demoni di routing (il software che implementa i messaggi di controllo) producono le tabelle di routing.

Le tabelle di routing saranno quindi unite in una forwarding table.

Forwarding Table: una struttura di dati che mappa ogni indirizzo di destinazione (o insieme di indirizzi di destinazione) all'interfaccia in uscita su cui un pacchetto destinato a questo indirizzo deve essere inoltrato per raggiungere la sua destinazione finale. Il router consulta la sua tabella di inoltro per inoltrare ogni pacchetto che gestisce.

Split Horizon

Un AS non annuncia un prefisso al router da cui riceve l'annuncio. Questo elimina il problema di creazione del loop di un collegamento singolo. Inoltre, BGP include anche negli annunci il percorso completo per raggiungere la destinazione

Questa è una differenza chiave rispetto a un protocollo distance vector puro.

Prima di annunciare un prefisso, l'AS controlla che sia già nel percorso: **questo impedisce la creazione di un ciclo più lungo di un collegamento**.

BGP è un Path Vector Protocol

BGP utilizza i principi del routing DV, con tre differenze principali

- in primo luogo, esporta non solo la distanza dalla destinazione, ma l'intero percorso AS, è chiamato protocollo Path Vector
- in secondo luogo, non invia regolarmente aggiornamenti. Invia aggiornamenti solo quando qualcosa cambia o quando un vicino chiede esplicitamente un aggiornamento.
- in terzo luogo, i messaggi BGP UPDATE contengono informazioni solo su alcuni prefissi, non sull'intera tabella di routing. Un AGGIORNAMENTO viene inviato su un prefisso se il percorso è nuovo, uno dei suoi attributi (ad esempio il percorso AS) è cambiato o il percorso è diventato irraggiungibile e deve essere ritirato.

Se, per qualche motivo un AS cessa di ospitare un prefisso, invierà un messaggio di ritiro. La stessa sequenza di router che lo ha annunciato all'inizio, ritirerà anche il loro annuncio. Questo farà scomparire il prefisso da Internet ⇒ buono se un link fallisce

Connessione BGP

Quando due router BGP vogliono comunicare tra loro, prima è necessario configurarli. Entrambi devono accettare connessioni TCP con porta 179.

BGP invia 5 tipi di messaggi:

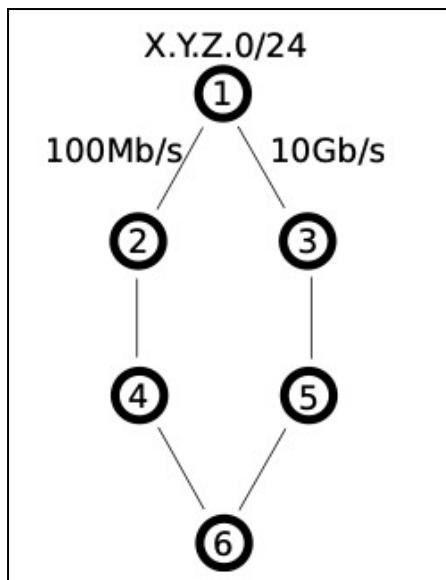
- Messaggio aperto: per stabilire connessioni BGP.
- Messaggio di aggiornamento: per trasferire informazioni di routing tra peer BGP.

- Messaggio Keepalive: per verificare se i colleghi sono ancora raggiungibili.
- Messaggio di notifica: per notificare i colleghi BGP degli errori.
- Messaggio di aggiornamento del percorso: un tipo di messaggio per supportare la capacità di aggiornamento del percorso.

BGP Path Prepending

AS1 è uno stub AS multi-homed, ha due connessioni:

- uno da 10Gb/s, da utilizzare in condizioni normali
- uno da 100Mb/s, da utilizzare solo come backup In che modo AS 1 dice agli altri AS di non usare il secondo?

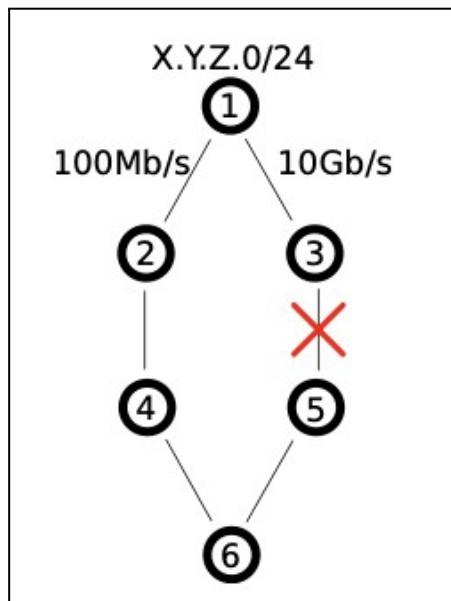


Annuncia il prefisso da X.Y.Z.0/24 a AS 3 con percorso 1

Annuncia il prefisso da X.Y.Z.0/24 a AS 2 con percorso 1,1,1,1,1,1 (**stesso, ripetuto più volte**)

AS6 riceve due diversi aggiornamenti per la rete X.Y.Z.0/24: Uno con percorso 5,3,1, uno con percorso 4,2,1,1,1,1

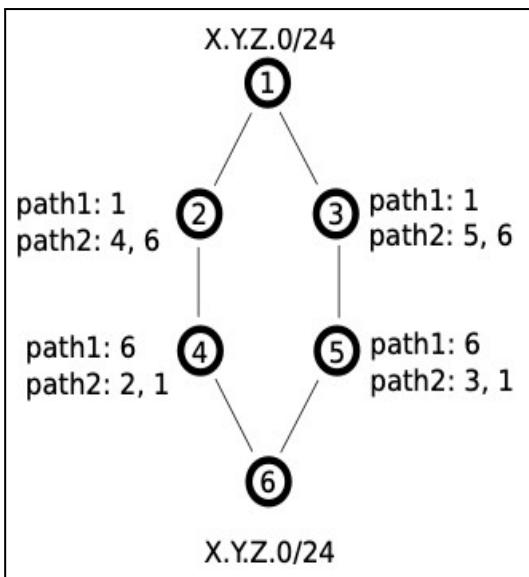
Quindi AS6 utilizzerà il primo (più breve), purché sia disponibile



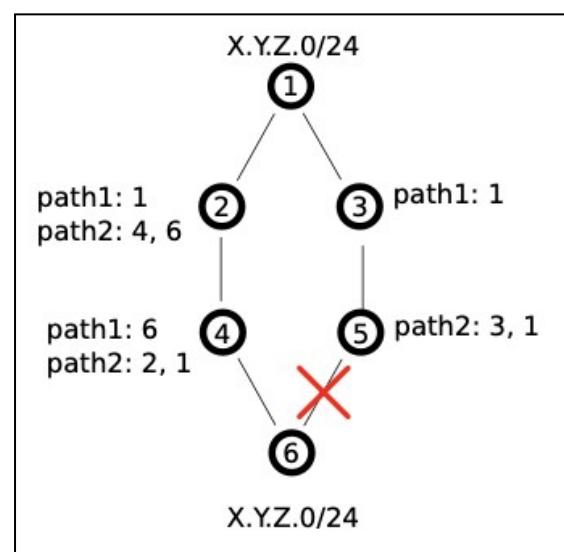
Se il collegamento tra AS5 e AS3 si interrompe, AS5 **ritirerà il prefisso X.Y.Z.0/24**

Quindi AS6 avrà un solo percorso disponibile e utilizzerà quello di backup

Quando il collegamento si sistemerà, il percorso più breve verrà propagato di nuovo e utilizzato. **BGP Anycast**



In questa rete sia AS 1 che 6 annunciano lo stesso prefisso
 Tuttavia, gli altri AS nella rete si divideranno tra quelli che si dirigono verso 1 (AS 2 e 3) e quelli che instradano verso 6 (AS 4 e 5)
Su Internet, lo stesso prefisso può essere annunciato da più di un AS
 Gli altri AS instradano verso quello più vicino
 Questo significa letteralmente che per alcune applicazioni, più di un host Internet ha lo stesso indirizzo IP

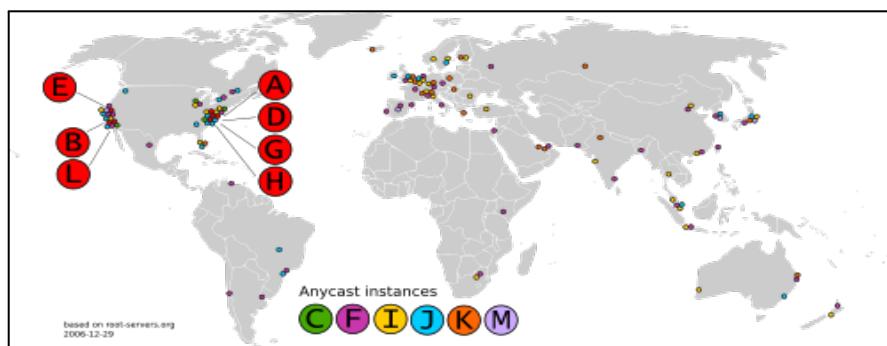


Anycast (percorso verso l'IP più vicino) **consente di avere servizi ridondanti**
 Lo stesso IP X.Y.Z.1 è hostato in due server. Se il percorso migliore fallisce, c'è uno di backup.
 Per questo motivo, Anycast non viene mai utilizzato con un servizio stateful, come una connessione TCP.
 Viene utilizzato per servizi che hanno uno stile request-response.
 Il più noto è DNS.

BGP Anycast for Root Servers

Abbiamo detto che in cima alla gerarchia DNS, ci sono server **root DNS**.

Ce ne sono 13, nominati da A a M. Tuttavia, **ognuno di essi appartiene a una rete annunciata da decine di AS in tutto il mondo**. Quando il tuo browser richiede un dominio, ad esempio, al server A, BGP consegnerà la richiesta alla copia più vicina del server DNS.



I server vengono periodicamente aggiornati e tenuti sincronizzati, quindi il loro database è lo stesso.

Poiché il DNS è un protocollo di richiesta/risposta, non importa se BGP cambia la sua decisione di routing di volta in volta. Ogni richiesta DNS è indipendente da quella precedente, e quindi il sistema funziona.

Se vuoi implementarlo con TCP è più complicato, perché devi mantenere lo stato sincronizzato tra N potenziali server.

IEEE 802.3 - Ethernet qui

Quella che chiamiamo Ethernet è una famiglia di standard progettati a partire dai primi anni '70 a Palo Alto alla Xerox. Il design iniziale è accreditato a Robert Metcalfe.

Gli standard IEEE 802.3 si sono evoluti notevolmente negli ultimi 50 anni, sia nel loro livello fisico che nel loro livello di collegamento dati. Oggi Ethernet è utilizzato principalmente su cavi di rame intrecciati e fibra ottica. Alcune caratteristiche non sono cambiate, incluso l'indirizzo utilizzato per identificare una NIC e il formato dell'intestazione. È il datalink e livello fisico che ad oggi si usa di più per fare comunicazioni a livello di LAN.

Mezzi fisici:

- cavi twisted pair, Questi sono un paio di cavi di rame intrecciati insieme, in ogni cavo c'è più di una coppia (quelli recenti hanno 4 coppie). Gli standard Ethernet si sono evoluti molto negli ultimi 50 anni, supportando velocità sempre più elevate. E così anche i cavi, che sono classificati in categorie. Ogni categoria (cat 5/6/7/8...) ha prestazioni diverse e può quindi essere utilizzata con versioni Ethernet più vecchie o recenti.
- fibra / ecc.

Ethernet offre una consegna di frame unicast, multicast, broadcast. Teoricamente è inaffidabile, ma le moderne Ethernet garantiscono una consegna abbastanza affidabile di frame senza riordinare.

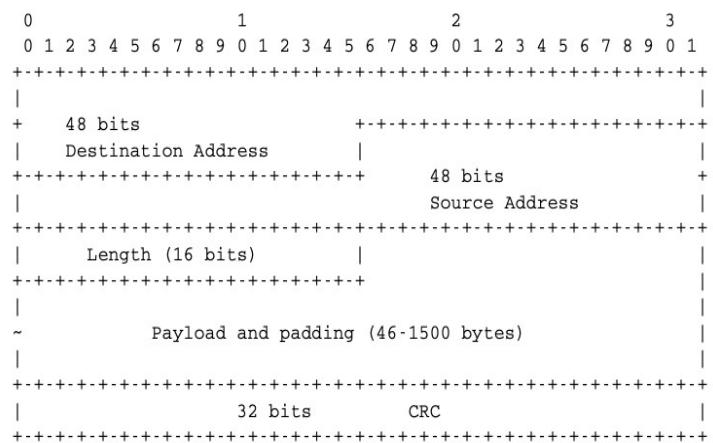
MAC Address

Un indirizzo MAC identifica una NIC al livello di data link. È composto da 48 bit di cui:

- Un bit per il **tipo** (0/1 → unicast/broadcast)
- 24 bit per l'**Identificatore univoco dell'organizzazione (OUI)**, univoco per produttore NIC
- 24 bit univoco per la **NIC**

Un indirizzo MAC dovrebbe quindi essere globalmente univoco. Sono preassegnati alla NIC, non è necessario configurali.

Header frame ethernet



Gli indirizzi sono all'inizio del frame, in particolare, l'indirizzo di destinazione è il primo perché un NIC ricevitore può leggerlo prima e decidere se è interessato a decodificare il resto del frame, o semplicemente ignorarlo.

Il checksum è CRC e non IP checksum come al livello rete

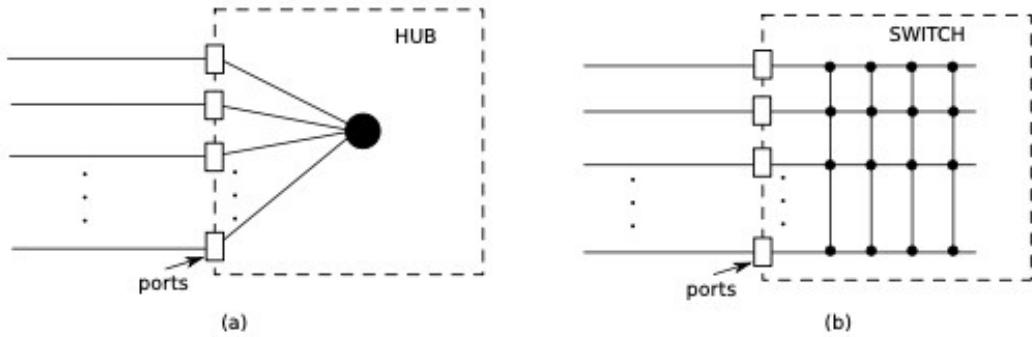
Il CRC è al trailer del frame, contrariamente alle intestazioni IP, questo perché il NIC può calcolare il CRC mentre riceve i dati, e quindi confrontarlo con gli ultimi 32 bit, senza necessariamente memorizzare i dati o il CRC.

Ethernet viene utilizzata principalmente con il protocollo IP, ma non necessariamente. Un livello MAC dovrebbe includere un campo che specifica il protocollo superiore. Nel frame Ethernet iniziale questo campo esisteva (l'EtherType) e ha sostituito il campo Length. Tuttavia, non c'era un campo di lunghezza, poiché il telaio aveva una lunghezza minima fissa e ci deve essere uno spazio tra i pacchetti (nessuna trasmissione) alla fine di un telaio.

Di conseguenza, è stato introdotto un nuovo livello nel modello ISO/OSI, quindi il livello Data Link è diviso in due: il livello MAC (controllo dell'accesso ai media) e il LLC (controllo del collegamento logico). Poiché il ruolo del secondo è marginale, spesso usiamo il nome MAC come sinonimo di livello Data Link. Tuttavia, ad un certo punto il formato a 16 bit EtherType è stato modificato per adattarsi anche alla lunghezza del fotogramma. Quindi la LLC ha perso la maggior parte della sua utilità e viene utilizzata raramente.

Ethernet Switches

Un singolo collegamento può essere esteso per creare un'intera rete con dispositivi che interconnettono i terminali l'uno con l'altro. Questo veniva fatto con gli hub, e ora è fatto con gli switch Ethernet.



Un hub (a) è solo un ripetitore, ogni frame che arriva a una porta viene inviato a tutte le altre porte. Ora è obsoleto. Uno switch (b) comprende lo standard 802.3. Ha un "backplane" che è una logica di controllo che collega selettivamente una porta con un'altra e invia traffico solo sulla porta giusta. Quindi deve imparare su quale porta dovrebbe essere inviato un determinato frame per raggiungere la destinazione, come va fatto?

Nella situazione più semplice la rete è composta da un singolo switch e molti terminali; quindi, lo switch deve imparare dove si trova un certo terminale. Questo si ottiene con un algoritmo di **backward learning**.

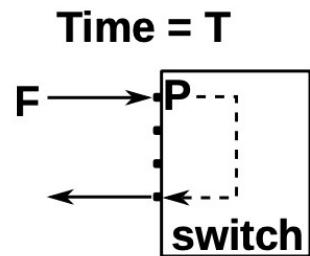
Nella situazione più complessa, ci sono più switch in cascata, e un altro algoritmo deve essere implementato per evitare loop: lo **Spanning Tree** (algoritmo distribuito e protocollo).

Backward Learning

```

1 # Arrival of frame F on port P at time T;
2 # Table : addr->[port, time]; Ports : list of ports
3 src=F.SourceAddress
4 dst=F.DestinationAddress
5 # update the table
6 Table[src] = (P, T)
7 if isUnicast(dst) :
8     if dst in Table:
9         ForwardFrame(F,Table[dst][0])
10    return
11 # multicast, broadcast, or unknown destination
12 for o in Ports :
13     if o!=P :
14         ForwardFrame(F,o)

```



N.B: se una destinazione è unicast lo si vede dal primo bit

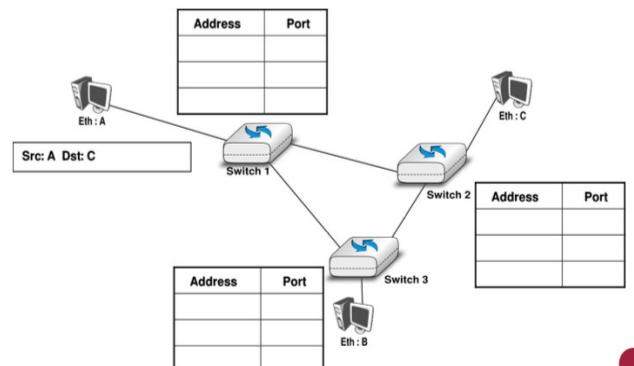
Periodicamente, l'interruttore controlla la tabella e rimuove tutti gli indirizzi obsoleti. Gli indirizzi obsoleti sono quelli che non sono stati utilizzati in un certo intervallo di tempo. Questo ha lo scopo di ripulire la tabella e fare spazio a nuovi indirizzi MAC.

Switching loops

Cosa succede quando questa rete si avvia da zero?

Tutte le tabelle hash sono vuote

- A invia un frame a C, lo switch 1 non sa dove si trova C, quindi instrada a tutte le porte
- Switch 2 non conosce C, quindi invia tutte le porte
- Switch 3 riceve da 1 e 3, non conosce C, quindi invia su tutte le porte
- Switch 1 riceve frame 1 e 2 ...



Ethernet non fornisce un campo TTL: i frame vengono consegnati ma possono viaggiare all'infinito

Uno switching loop può saturare la capacità dei collegamenti, sovraccaricare la CPU degli switch e **uccidere** la tua rete. Non è facile da bloccare, devi rompere il ciclo fisico.

È facile da creare: basta collegare uno switch a due porte su una rete (non farlo a casa). In genere avviene nei loop **non managed**, ovvero gli switch che non implementano STP, e servono solo a instradare.

The Spanning Tree Protocol (STP)

Si presume che ogni **switch managed** abbia un identificatore univoco a 64 bit:

- I primi 16 bit (più significativi) sono determinati dall'amministratore.
- I restanti 48 bit sono un indirizzo MAC determinato dal produttore.

STP costruisce un albero con la radice che è lo switch con l'ID più basso. La configurabilità dei primi bit consente all'amministratore di posizionare la radice vicino al router di frontiera per motivi di prestazioni.

Un SPT non è un albero di costo minimo. Ma copre tutti i nodi di un grafico e non contiene loop.

Port State

Le porte di uno switch che supporta STP possono essere in tre stati: root, designated, blocked.

STP è un protocollo distribuito. Finché tutte le porte non sono impostate su qualche stato, inizialmente le porte non inoltrano il traffico, fino a quando non raggiungono qualche stato

Lo switch root ha tutte le porte impostate su designated. Tutti gli altri switch hanno almeno una porta impostata sullo stato root. Inizialmente tutte le porte sono nello stato blocked

Operazioni semplificate

Ogni switch invia un frame **BPDU** (Bridge Protocol Data Unit) su tutte le sue porte. Il frame contiene vari dati, tra cui <

Rootid, Cost, Senderid >:

- L'ID dell'attuale switch radice designato
- il costo del percorso alla radice (distanza ponderata in base alla capacità del collegamento),
- l'ID del nodo che genera la BPDU

Le BPDU vengono inviate a un indirizzo multicast specifico, non vengono mai inoltrate, sono elaborate solo dagli switch vicini.

Inizialmente ogni switch utilizza il suo ID come ID radice e costo impostato su zero.

Le BPDU possono essere confrontate utilizzando l'ordinamento lessicografico.

Se RootID è differente, è sufficiente rompere i legami.

Quindi se abbiamo

$$\begin{aligned} \text{BPDU} &= <\text{Rootid}, \text{Cost}, \text{Senderid}, p > \\ \text{BPDU}^1 &= <\text{Rootid}^1, \text{Cost}^1, \text{Senderid}^1, p^1 > \end{aligned}$$

Allora

$$\text{if } \text{Rootid} < \text{Rootid}^1 \rightarrow \text{BPDU} < \text{BPDU}^1$$

In caso contrario, confronta il costo e così via...

Quando lo switch A riceve una BPDU dallo switch B, A calcola il costo c:

$$c = \text{Cost} + C_{AB}$$

e il vettore di priorità radice. Questo è memorizzato per ogni porta:

$$V_q = <\text{Rootid}, c, \text{Senderid}, p, q >$$

dove: CAB è il costo del collegamento A-B, q è la porta su cui è stata ricevuta la BPDU. Gli altri parametri provengono dalla BPDU ricevuta da A.

Se V_q è inferiore alla propria BPDU (essenzialmente, il Rootid ricevuto è più piccolo o il costo è inferiore), allora:

- A decide che B diventa lo switch radice e utilizzerà l'ID di B come Rootid nella BPDU che genera d'ora in poi
- La porta q in A diventa la porta radice. Ora A è un figlio di B nell'albero.

Inoltre, A confronterà anche la BPDU ricevuta sulla porta q con la propria BPDU (Nota bene: la propria BPDU Vs la BPDU ricevuta, non Vs il vettore di priorità)

- se la propria BPDU è più piccola, allora q prende lo stato designated: B è un figlio di A.
- Altrimenti, B è un potenziale fratello di A (quindi potrebbe esserci un ciclo) e q viene *blocked*.

Nessun pacchetto di dati viene inviato/accettato dalle porte *blocked*. Le BPDU di tutte le porte vengono invece elaborate continuamente (nel caso in cui qualcosa cambi). Le BPDU vengono inviate solo agli interruttori figli

Port state	Receives BPDUs	Sends BPDU	Handles data frames
Blocked	yes	no	no
Root	yes	no	yes
Designated	yes	yes	yes

Cosa succede se qualcosa cambia?

Il processo viene costantemente ripetuto, con lo switch root che calcola la sua BPDU e la invia alle sue porte designate.

Gli interruttori mantengono un'"età" dell'ultima BPDU, incrementandola ogni secondo.

Quando questo valore supera un massimo, lo switch capisce che la topologia è cambiata: alcuni switch sono failed: il protocollo STP nello switch ricomincia:

Tutte le porte sono impostate su blocked, viene cercata una nuova radice. Fino a quando la topologia non si stabilizza, gli switch non inoltrano il traffico.

The Address Resolution Protocol (ARP)

Ora sappiamo come viene costruito un livello Ethernet e i protocolli necessari per farlo funzionare al livello II. Utilizzando questi algoritmi, NIC A può inviare pacchetti a NIC B su un altro host, in base all'indirizzo MAC di destinazione. Tuttavia, sappiamo che tutto il traffico è trasportato da pacchetti IP. Quindi dobbiamo porci una domanda: ogni volta che un host con un certo IP vuole inviare un pacchetto a un altro IP, quale indirizzo MAC utilizzerà come destinazione? Ci sono due possibili risposte.

Local Destination IP

Supponiamo che l'IP di A sia 192.168.1.10/24, supponiamo che l'IP di destinazione sia 192.168.1.11. Allora A sa che l'host B è nella **stessa rete fisica**. L'IP deve essere raggiungibile **senza routing**, solo attraverso gli switch. Quindi in questo caso A ha bisogno di un modo per scoprire l'indirizzo MAC associato all'IP di B e inviare un frame Ethernet con gli indirizzi di destinazione IP e MAC corretti. Questo problema viene risolto utilizzando l'**Address Resolution Protocol (ARP)**.

A remote IP

Ora A vuole inviare un pacchetto a 1.2.3.4, che non si trova nella stessa sottorete. Il comportamento corretto è quello di inviare il pacchetto utilizzando:

- Nell'intestazione IP(network layer), **l'IP di destinazione sarà 1.2.3.4**
- Nell'header Ethernet (data link), il MAC di destinazione deve essere quello del **default gateway**, al livello 2 i **frame si consegnano sempre al default gateway**, che in base al suo protocollo di routing (es. BGP) capirà qual è il next hop a cui mandarlo.

In che modo A conosce l'indirizzo MAC del router? Questo (tra le altre questioni) è curato dal protocollo **DHCP**.

Due host sulla stessa LAN comunicano attraverso uno switch e gli switch instradano i pacchetti utilizzando gli indirizzi MAC. Le NIC accettano solo frame indirizzati al loro MAC o all'indirizzo di trasmissione Ma in che modo l'host Alice conosce l'indirizzo MAC di Bob, collegato alla stessa LAN?

ARP

Ogni host sulla rete mantiene una **tabella ARP**. La tabella associa un indirizzo IP a un indirizzo MAC corrispondente.

Quando Alice deve inviare un pacchetto a un indirizzo IP nella stessa sottorete della sua NIC, deve esserci una voce valida nella tabella ARP.

Come viene aggiornata la tabella ARP?

Quando 192.168.1.20 ha bisogno di conoscere l'indirizzo MAC che corrisponde a un dato 192.168.1.11, trasmette un pacchetto ARP contenente il suo MAC e indirizzo IP e una richiesta per il MAC della destinazione. Il pacchetto, trasmesso, viene inoltrato da tutti gli switch della rete. La richiesta ARP arriva all'host di destinazione che risponde con un pacchetto unicast diretto al MAC di origine.

La tabella ARP viene aggiornata ad ogni pacchetto ARP ricevuto (sia richieste che risposte)

Header ARP

Hardware type (ethernet)
Protocol type (IPv4)
[...]
Sender hardware address
Sender protocol address
Receiver hardware address
Receiver protocol address

```
?Do I have that hardware type ?  
Yes: (almost definitely)  
?Do I speak that protocol ?  
Yes:  
If the pair <protocol type, sender protocol address> is  
already in my translation table, update the sender  
hardware address field of the entry with the new  
information in the packet.  
?Am I the target protocol address?  
Yes:  
[...]
```

Dynamic Host Configuration Protocol (DHCP)

Quando un host si avvia, ha bisogno di diversi parametri di configurazione, tra cui:

- Un indirizzo IP che può utilizzare: sappiamo che ogni rete ha il proprio netid ma l'host non lo conosce.
- L'indirizzo IP del router che è il gateway della rete.
- L'indirizzo IP del server DNS.

DHCP è un protocollo di gestione che viene utilizzato per configurare gli host di rete all'avvio.

È un protocollo concettualmente semplice, che descriviamo qui perché riguarda la configurazione delle reti locali e non lo scambio di dati. Lo descriveremo concettualmente, non nei dettagli.

DHCP è un protocollo client-server, l'amministratore imposta un server che qualsiasi host può contattare per ricevere la propria configurazione. DHCP utilizza UDP, sulla porta 67, Il protocollo è avviato dal client.

DHCP Discover

Un client appena aggiunto si accende e invia un messaggio di **DHCP Discover**.

- L'IP di destinazione del pacchetto IP è 255.255.255.255, o l'equivalente (limited broadcast) per IPv6
- L'indirizzo IP di origine è impostato su 0.0.0.0 (indirizzo che si usa quando non hai un indirizzo)
- L'indirizzo MAC di origine è impostato sull'indirizzo NIC
- Nel messaggio di scoperta DHCP il client include anche un identificatore client, normalmente l'indirizzo MAC della sua NIC

DHCP Offer

Il server risponderà con un messaggio di offerta, contenente l'ID client (tratto dalla discover), l'indirizzo IP che il server sta offrendo, la subnet mask, la durata del lease (tempo prima che scada).. L'offerta contiene anche l'IP del server.

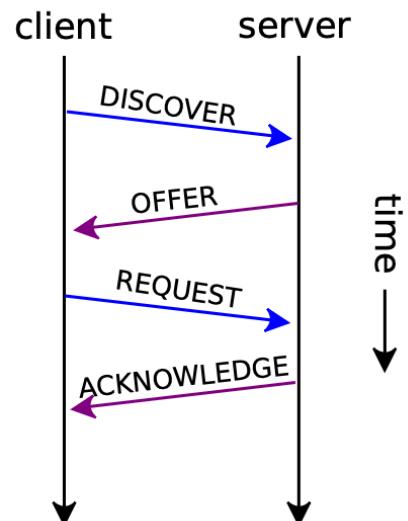
Ci possono essere più server nella rete che possono rispondere alla stessa offerta, per motivi di ridondanza.

DHCP Request

Il client invierà quindi un messaggio broadcast (voglio che tutti i server sappiano che sto accettando una offerta) di richiesta DHCP, che richiede l'IP offerto dal server. Al livello MAC e IP, l'indirizzo di destinazione è l'indirizzo broadcast, ma il corpo del messaggio contiene l'indirizzo IP del server. Qualsiasi server DHCP che riceve una richiesta controllerà l'indirizzo del server incluso nel corpo del pacchetto e deciderà se elaborare o ignorare il messaggio.

DHCP Acknowledgment

Il server risponderà infine con un messaggio di conferma. Questo è destinato all'IP richiesto dal client e contiene anche la durata del contratto di locazione e qualsiasi altra informazione di configurazione: normalmente il gateway predefinito e il server DNS.



IEEE 802.11 - Wi-Fi

Un altro standard dell'IEEE. Lo standard descrive il livello fisico e MAC dello stack ISO/OSI per le comunicazioni wireless LAN: modulazione, codifica, accesso al supporto.

Il processo di standardizzazione è lento, quindi i produttori hanno creato un'organizzazione chiamata "alleanza Wi-Fi". L'alleanza Wi-Fi produce versioni preliminari e ridimensionate degli standard prima che siano finalizzati. Ad esempio, limitato a quelle caratteristiche che sono considerate stabili.

Un fornitore può chiedere di certificare il suo prodotto, che dovrà superare i test per ottenere il "badge" dell'alleanza Wi-Fi. Tecnicamente, parliamo di reti 802.11, commercialmente di reti Wi-Fi.

Ancora una volta, per sottolineare le differenze tra le generazioni di tecnologia, il marchio Wi-Fi utilizza i numeri.

802.11ac è stato indicato come Wi-Fi 5. Aveva due versioni, onda 1 e onda 2.

802.11ax è indicato come Wi-Fi 6.

Questo è solo waffle di marketing.

Livello Fisico

La versione più comune di WI-FI usa:

- Frequenze: tra 2.4 o 5 GHz
- Bitrate: 11 - 6700 Mbps
- Range: circa 50m al chiuso e 300 m all'esterno, ma i dispositivi professionali consentono chilometri.
- Supporta la mobilità

Il Wi-Fi normalmente funziona a frequenze senza licenza, cioè frequenze per le quali non è necessaria una licenza per emettere segnali, le cosiddette bande ISM (Industrial, Scientifica e Medica). Le bande ISM utilizzate dal Wi-Fi sono principalmente due:

- 2,4 - 2,4835 (83 MHz, comunemente nota come banda a 2,4 GHz)
- 5,17 - 5,33; 5,49 - 5,710 (160 + 220 MHz, comunemente nota come banda a 5 GHz)

Le bande ISM non sono utilizzate solo dal Wi-Fi, quindi subiscono l'interferenza di altri dispositivi. Per questo motivo, e per motivi di sicurezza, la potenza massima di trasmissione è limitata dalla legge

La banda utilizzata dal Wi-Fi è divisa in **canali**, cioè sottobande che possono essere utilizzate (quasi) in modo indipendente

Per la banda a 2,4 GHz vengono utilizzati 11 canali, per la banda a 5 GHz vengono utilizzati 19 canali (i canali possono essere di più, ma quelli utilizzabili dipendono dal paese e dalle leggi).

A 5 GHz 802.11ac (Wi-Fi 5, il più utilizzato) utilizza canali di 20 MHz e ne definisce 30. In Europa puoi usare 19 su 30 a causa delle restrizioni legali.

Questo rende questa banda più attraente, poiché è più improbabile che tu abbia interferenze da un AP nelle vicinanze.

La larghezza di banda disponibile è limitata dalle normative. La banda è divisa in canali che possono essere aggregati. L'aggregazione dei canali aumenta la larghezza di banda disponibile, ma aumenta anche le possibilità di interferenza con le reti vicine.

Livello MAC

Tipi di traffico

Ci sono 3 tipi di frame a livello MAC Wi-Fi:

- **Management Frame**

Tutto ciò che non trasporta traffico ma è necessario per gestire le comunicazioni.

- frame di autenticazione.
- frame di **associazione**.

- **Beacon Frames**: contengono informazioni sulla rete

Fino a poco tempo fa, non autenticato o crittografato.

- **Control Frame**

Vengono utilizzati per gestire l'accesso al canale.

- frame **RTS/CTS**.

- **ACK Frames.**
Non crittografato e autenticato.
- **Data frames**
trasportano effettivamente il traffico, sono crittografati.

Definizioni

Basic Service Set (BSS): È l'unità più piccola che costituisce una rete 802.11. Un BSS può essere di due tipi, ci concentriamo sul modello "Infrastrutturato" che è il più comune. In questo modello c'è un Access Point e alcune stazioni connesse

Tutto il traffico scambiato è sempre tra stazioni e AP, le stazioni non comunicano direttamente tra loro. L'AP diventa così un "**centro stellare**".

Più BSS possono essere collegati tra loro utilizzando un **DS** (Distribution Service), che è un collegamento di rete tra i vari AP che creano i singoli BSS.

Il collegamento è normalmente un collegamento gigabit cablato. Parliamo quindi di un Extended Service Set (**ESS**). Normalmente le stazioni collegate a un ESS sono sullo stesso segmento di rete IP, quindi possono comunicare tra loro, e gli AP condividono lo stesso SSID (ma non è obbligatorio).

Entrare in una rete

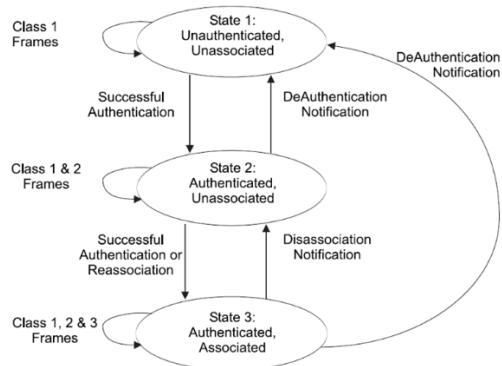
Ogni AP emette frame su base periodica (frame Beacon, normalmente 10 volte al secondo) contenenti informazioni BSS, tra cui il BSSID, l'SSID e molte altre informazioni sulla rete.

Una stazione che vuole unirsi alla rete deve passare attraverso un processo in tre fasi:

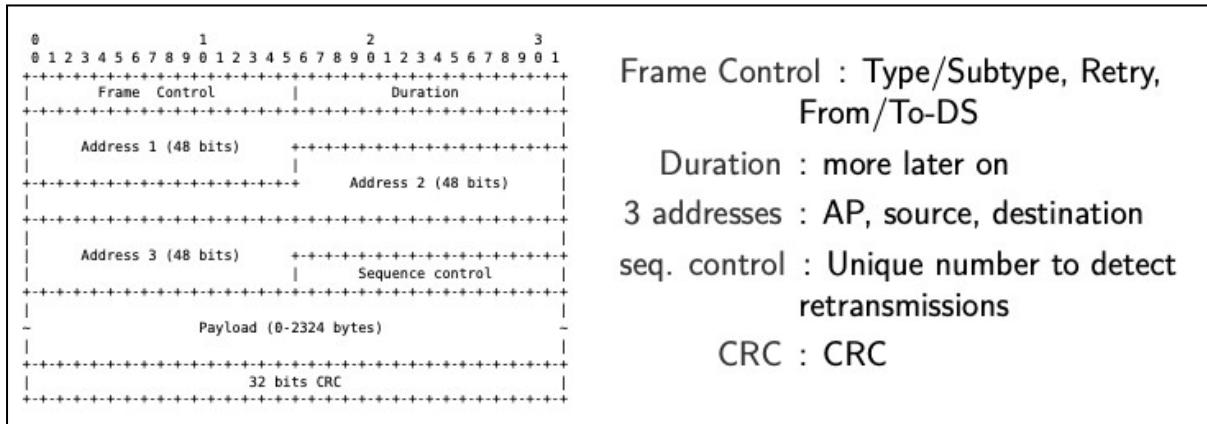
- **Scansione:**
 - Passiva: la stazione ascolta per un breve tempo su ogni canale aspettando per un beacon frame
 - Attiva: la stazione manda una Probe-request per trovare gli AP, non si fa normalmente.
- **Autenticazione e Associazione**

L'autenticazione e l'associazione sono due fasi che il client fa solo una volta. Sono utilizzate per creare un canale sicuro tra ogni client e un singolo AP di quelli che si trovano nell'ESS. Ci sono molte diverse modalità di autenticazione, non entriamo nei dettagli.

Macchina di stato



MAC Header Wi-Fi



Controllo degli accessi

L'obiettivo del livello MAC è coordinare l'accesso a un supporto condiviso.

Sappiamo che se un terminale riceve due segnali contemporaneamente, si sommano e difficilmente possono essere interpretati separatamente. Quando si utilizzano supporti fisici condivisi, le stazioni non possono trasmettere allo stesso tempo, devono evitare collisioni

Collisione: una situazione in cui una radio riceve due o più frame contemporaneamente. Questo normalmente implica che nessuno di loro viene ricevuto correttamente.

Le collisioni si verificano al ricevitore, il mittente non saprà se il frame è stato ricevuto correttamente.

Per confermare che un frame è stato consegnato correttamente, il destinatario invia un pacchetto di conferma (ACK).

Trasmissione non riuscita: un mittente presume che la trasmissione non abbia avuto successo (c'è stata una collisione) se dopo un certo tempo dopo la trasmissione, non ha ricevuto un ACK.

Carrier Sensing: un modo di base per evitare la collisione è, quando un terminale ha bisogno di trasmettere, controllare prima di trasmettere, se qualcun altro sta trasmettendo.

CSMA/CA (Carrier Sensing Multiple Access / Collision Avoidance): 802.11 usa CSMA/CA come livello MAC, è un parente stretto di ALOHA.

Efficienza del MAC

Quando la rete è in saturazione, la situazione ideale è che il canale sia sempre occupato, inviando nuovi dati. Se per qualche motivo il canale è idle, o sta inviando nuovamente i dati che sono stati inviati in precedenza, la rete non sfrutterà tutta la capacità.

Esempio: se una rete con una capacità massima di 1Gb/s trasmette solo il 50% delle volte, equivale a una rete con 500Mb/s che trasmette il 100% del tempo.

Quindi un obiettivo fondamentale del MAC è fornire la massima efficienza, cioè massimizzare il tempo in cui il canale è occupato. Tuttavia, i terminali devono coordinarsi e questo coordinamento richiede di rimanere liberi per qualche tempo o di inviare e ricevere alcuni messaggi (di controllo) al fine di evitare collisioni. Pertanto, l'efficienza del 100% **non può mai essere raggiunta**.

Ci sono due modi in cui il MAC può funzionare: **PCF** e **DCF**.

PCF: Funzione di coordinamento dei punti

Il PCF è una modalità di coordinamento centralizzato in cui l'AP invia ai terminali un programma in cui alloca un intervallo di tempo per la trasmissione di ciascun terminale.

PCF implementa una sorta di **TDMA**, con tutti i suoi difetti, tra i quali, non esiste un modo ottimale per condividere le risorse, e le risorse possono essere assegnate e inutilizzate. (non si usa)

DCF: funzione di coordinamento distribuito

DCF implementa uno schema di accesso casuale, un modello simile ad **ALOHA**.

Si basa sulla definizione di intervalli, tempi di attesa e politiche di ritrasmissione. Ci sono anche in questo caso, due versioni, entrambe obbligatorie nello standard.

DCF

DCF: Carrier Sensing e Static Reservation

Ogni nodo che intende trasmettere un frame, in primo luogo, ascolta il canale.

Se il canale è libero attende un intervallo di tempo definito **DIFS** (DCF Interframe Space, Il valore di DIFS dipende dallo standard, in 802.11n/ac è 16 μs). Se il canale rimane idle (libero) per DIFS la stazione trasmette il suo frame.

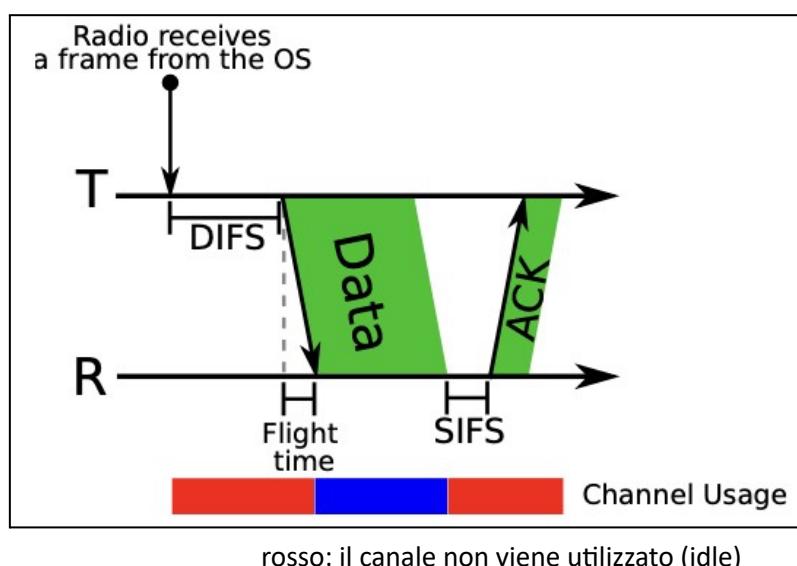
La stazione ricevente, riceve i frame e calcola un **checksum**:

- se il checksum è corretto attende un breve intervallo di tempo, chiamato **SIFS** (short-IFS), e invia un ACK.
- Se il pacchetto contiene errori, non invia l'ACK

Esempio di MAC semplice: trasmettitore singolo

La radio nel terminale T riceve un fotogramma dal sistema operativo da trasmettere. Questo evento è imprevedibile, dipende dalle applicazioni in esecuzione su T. La destinazione è il terminale R.

T attende i DIFS e invia il frame, che arriva al ricevitore dopo un certo flight time. Se non ci sono errori, il ricevitore risponde con un ACK.



rosso: il canale non viene utilizzato (idle)

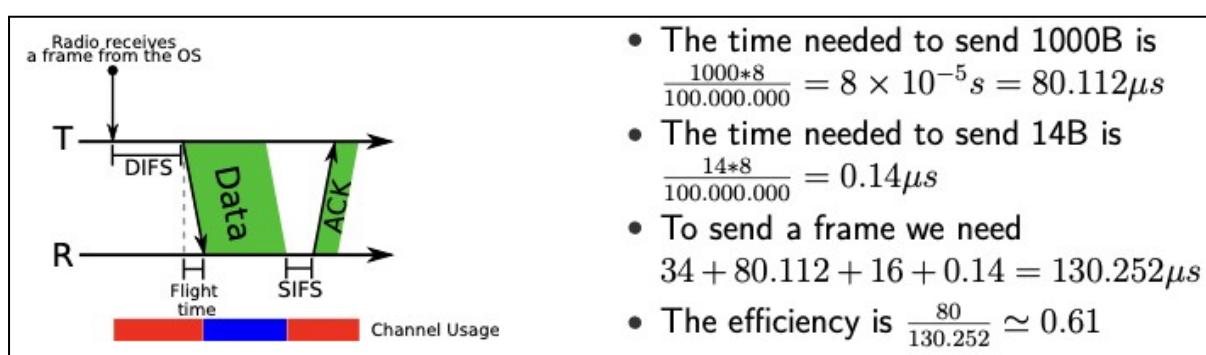
L'obiettivo del livello MAC è avere il rapporto blu/blu+rosso più vicino a 1 possibile.

Calcolo dell'efficienza

Immagina che la velocità di comunicazione sia di 100Mb/s. T deve inviare un frame di 1000B

Un frame ACK è lungo 14B (l'header del frame è più corto dei frame di dati) Ignora il tempo di volo, supponi 802.11ac a 5GHz

Qual è l'efficienza MAC dello schema mostrato prima?



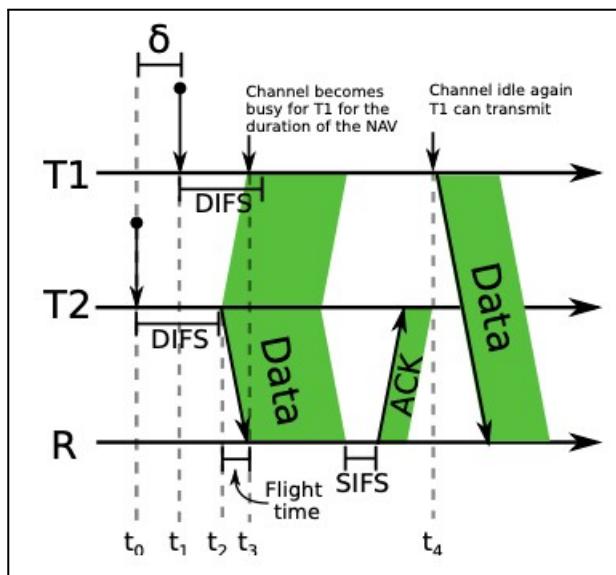
Neanche in questo esempio è vicina al 100% (normale che sia così).

Esempio di MAC semplice: due trasmettitori

Ogni frame nel suo header include un **NAV**, network allocation vector, che è un campo che specifica quanto tempo ci vorrà per inviare il pacchetto e ricevere l'ACK.

Tutti i terminali nell'intervallo di ricezione leggono l'header del frame (anche se non diretto a loro), controllano il NAV e sanno che non saranno in grado di trasmettere per quel periodo di tempo.

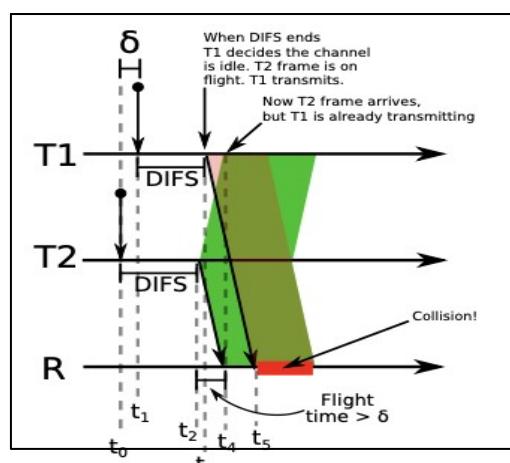
Se il pacchetto non è diretto a loro, possono spegnere la radio (e quindi risparmiare energia) fino alla fine del tempo NAV. In pratica, quando una stazione invia un fotogramma riserva il canale per un certo periodo.



Collisione (2 trasmettitori)

Se δ è troppo piccolo (meno del tempo di volo), quando DIFS termina, T1 considera ancora il canale inattivo e inizia a trasmettere. Viene generata una collisione sul ricevitore, R non invierà un messaggio ACK

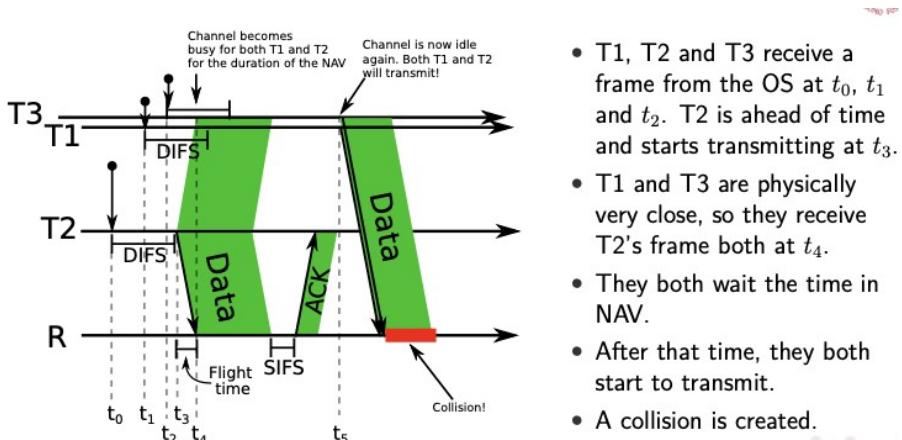
Poiché sia T1 che T2 non ricevono un ACK, sanno che il frame non è stato ricevuto correttamente.



Questo è un caso molto sfortunato, poiché l'onda impiega 3 ns (nanosecondi) per viaggiare di un metro.

Per avere una collisione, T1 e T2 devono ricevere un pacchetto dal sistema operativo, quasi contemporaneamente. Devono essere sincronizzati.

Esempio di MAC semplice: tre trasmettitori



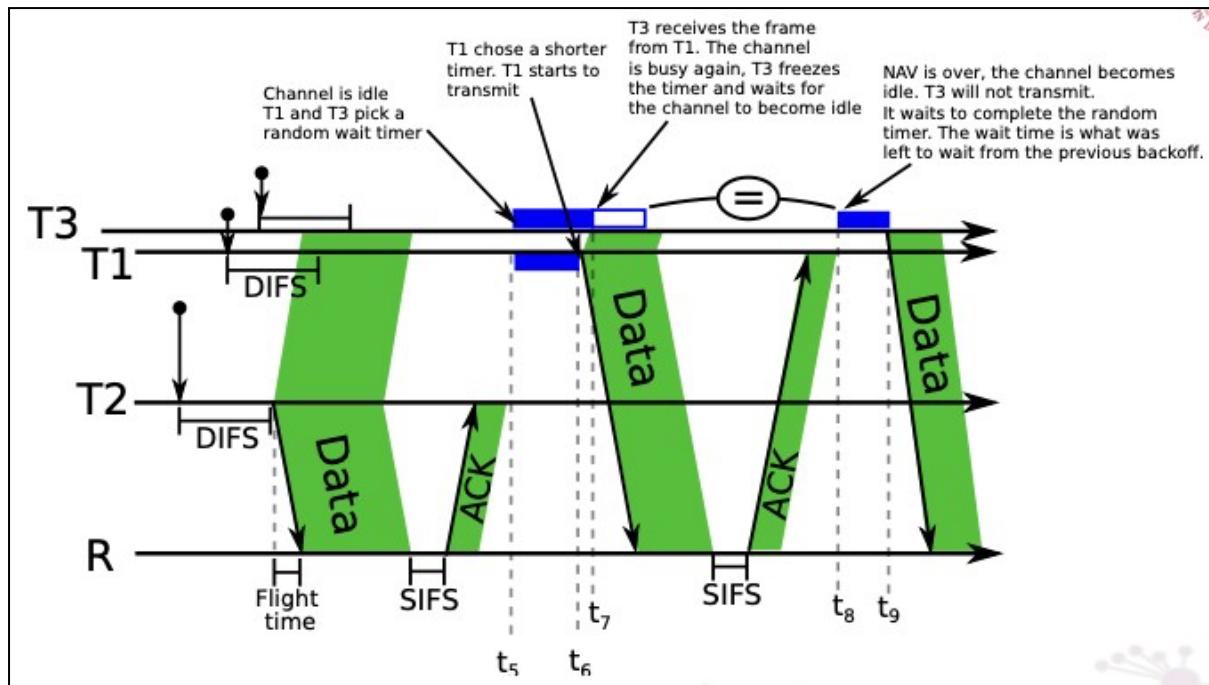
- T1, T2 and T3 receive a frame from the OS at t_0, t_1 and t_2 . T2 is ahead of time and starts transmitting at t_3 .
- T1 and T3 are physically very close, so they receive T2's frame both at t_4 .
- They both wait the time in NAV.
- After that time, they both start to transmit.
- A collision is created.

Questo non è un evento improbabile perché T1 e T3 ricevono il fotogramma (quasi) allo stesso tempo. Il problema non è più dovuto a δ . La static reservation dovuta al NAV sta effettivamente sincronizzando T1 e T3. Poiché una LAN wireless è fisicamente piccola, tutti i nodi ricevono i frame quasi contemporaneamente, quindi dopo la fine del frame. Se hanno dati da inviare, si scontreranno in modo deterministico.

Per evitare la trasmissione sincronizzata, è necessario introdurre un'attesa casuale prima della trasmissione. Quando il canale ritorna idle, ogni stazione attende un **tempo casuale nell'intervallo [0, CW]**.

Essendo casuale, sarà diverso per ogni stazione. Quindi possono accadere due cose:

- Per T1 il timer scade e il canale è idle. T1 può trasmettere
- Prima che il timer di T3 scada, T3 riceve un frame da T1, il canale è di nuovo occupato. T3 non può trasmettere.



Si noti che questo approccio non elimina la possibilità di collisioni • Se la differenza tra i timer casuali è inferiore al tempo di volo, le collisioni sono possibili nell'intervallo $t_6 - t_7$.

La probabilità di una collisione è più alta se diversi terminali trasmettono.

La probabilità di collisione diminuisce se estendiamo CW, perché i timer casuali sono scelti su un intervallo più grande.

Dopo ogni errore il mittente proverà di nuovo a inviare il frame, ma deve ridurre la probabilità di collisione. Sceglie un timer casuale su una finestra più grande.

Quando una trasmissione fallisce, il terminale cambia la finestra dell'intervallo casuale. CW viene aggiornato su ogni trasmissione non riuscita con la seguente regola ricorsiva:

$$CW_1 = CW_{min}$$

$$CW_i = 2 \times CW_{i-1}$$

$$CW_i < CW_{max}$$

Dove CW_{min} e CW_{max} sono parametri di sistema.

Quindi al tentativo i , $CW_i = 2^i \times CW$,

La probabilità che due random timer siano molto simili diminuisce.

Si chiama **Exponential Backoff Strategy**: ad ogni fallimento l'attesa casuale massima sarà maggiore.

Quando una trasmissione ha esito positivo, CW viene reimpostato: $CW = CW_{min}$.

Dopo un numero massimo di collisioni, il mittente si arrende e lascia cadere il frame. Il backoff si verifica quando:

- T1 deve trasmettere ma il canale è occupato.
- T1 ha appena completato una trasmissione e vuole trasmettere un altro fotogramma o riprovare dopo una trasmissione non riuscita.

Il backoff esponenziale binario introduce la casualità nel sistema al fine di rendere più improbabile che due stazioni creino una collisione. Tuttavia, ci sono momenti in cui i terminali hanno alcuni dati da trasmettere, ma non lo fanno.

In quei momenti la rete è inattiva, quindi la rete, anche sotto saturazione, non trasmette il 100% del tempo. I timer casuali riducono le collisioni ma sprecano le risorse

Meccanismo RTS / CTS

Se due terminali di trasmissione non sono in grado di comunicare tra loro, DCF non funziona. Infatti, nessuno dei due può fare il rilevamento del canale e notare che il canale è occupato quando l'altro sta trasmettendo. Le collisioni si verificheranno sull'AP perché entrambi i trasmettitori pensano di avere un canale chiaro.

Per risolvere il problema del nodo nascosto, DCF consente una prenotazione esplicita.

Prima di inviare un frame di dati, T1 invia un frame RTS (Request to send). L'RTS contiene un NAV abbastanza lungo da consentire l'invio di tutti i dati più l'ACK.

Se l'AP riceve correttamente l'RTS risponde con un CTS (Clear to Send), dopo SIFS, il CTS sostituisce l'ACK e contiene il NAV necessario per inviare tutti i dati incluso l'ACK.

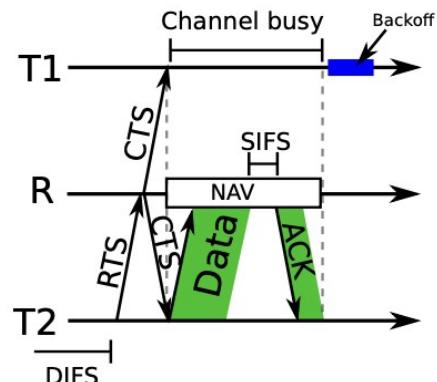
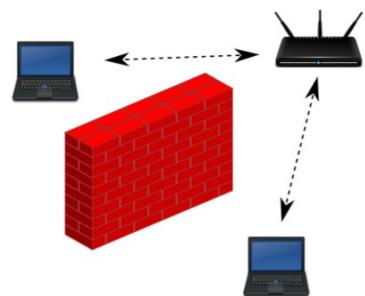
In una rete WI-Fi ci può essere il problema di un nodo nascosto, come per esempio la presenza di un muro.

Non tutte le stazioni hanno ricevuto l'RTS, poiché i terminali non sono in grado di ricevere frame l'uno dall'altro. Ma tutti i terminali devono essere nel raggio di comunicazione con l'AP; quindi, tutti i terminali ricevono il CTS.

Tutti i terminali sanno quindi che il canale è occupato per il tempo NAV.

L'handshake RTS/CTS aumenta il tempo di attesa prima della trasmissione. RTS/CTS riduce ulteriormente il bit rate (aumenta il tempo in cui nessuno trasmette i dati). Sono normalmente utilizzati per inviare frame di grandi dimensioni, cioè quei fotogrammi che mantengono il canale occupato a lungo, e se avessero una collisione, quel tempo sarebbe sprecato. Normalmente i driver hanno una soglia al di sopra della quale usano RTS/CTS.

Si noti anche che il pacchetto RTS può scontrarsi! Le piccole dimensioni dell'RTS lo rendono meno probabile.



Stima del Bitrate

Nyquist

Con 802.11ac, la più alta modulazione/codifica è 256-QAM, che corrisponde a 8 bit per simbolo.

802.11ac funziona solo a 5GHz, quindi per il teorema di Nyquist il bit-rate massimo ottenibile è dato dalla larghezza di banda per il numero di bit per simbolo, consideriamo la larghezza del canale più grande:

$$2 * 8 * (5330 - 5170) = 2 * 8 * 160 = 2560 \text{ Mbit/s}$$

Shannon

Non abbiamo le conoscenze teoriche per stimare l'SNR, quindi ti darò due numeri (probabilmente ottimisti):

- SNR a 10m: $SNR = 10^{5,1} = 125892 \rightarrow C = B \log_2(1 + SNR) = 160 \log_2(1 + 125892) \cong 2710 \text{ Mbit/s}$
- SNR a 20m: $SNR = 10^{4,5} = 31622 \rightarrow C = B \log_2(1 + SNR) = 160 \log_2(1 + 31622) \cong 2416 \text{ Mbit/s}$

Capacità Massima

In questo esempio abbiamo una capacità teorica massima di 2560 Mb/s. Questo è il limite superiore del bit-rate massimo con questo supporto fisico.

A 10 m possiamo raggiungere, con i parametri dati, fino a 2710 Mb/s. Questo è il limite superiore di qualsiasi supporto fisico con l'SNR dato, ed è superiore a 2560.

Quindi, alla fine, raggiungiamo 2560, non di più.

Modulating Coding Scheme (MCS)

MCS è una coppia di valori, **modulazione** e **codifica**, che è mappata in un numero (MCS0-MCS9)

Abbiamo visto che la modulazione rappresenta il numero di bit per simbolo.

La codifica rappresenta quanti di questi bit vengono effettivamente utilizzati per i dati. Un certo numero di bit è tipicamente usato per la ridondanza, simile al controllo della parità. È solo un fattore di moltiplicazione del bit-rate ottenuto dalla modulazione.

Data la modulazione e la codifica, e conoscendo tutti i dettagli del MAC (DIFS, SIFS, Guard Times) puoi effettivamente stimare il throughput massimo.

In realtà c'è un altro parametro che deve essere valutato, il numero di flussi spaziali. Abbiamo visto che in un sistema MIMO, è possibile inviare più di un flusso contemporaneamente, se il dispositivo è dotato di più di un'antenna. Questi flussi possono essere utilizzati per inviare gli stessi dati (e quindi aumentare l'SNR) o dati diversi (e avere trasmissioni di dati parallele)

Dinamicamente la nostra radio cerca di capire qual è l'MCS migliore da utilizzare in un dato istante, adattandosi alla situazione attuale e al SNR. Normalmente usano qualche approccio try/error, in cui inviano frame a vari MCS e scelgono il migliore.

Adaptive Coding and Modulation: Una strategia che cerca di selezionare il miglior MCS date le attuali condizioni del canale

Clear Channel Assignment (CCS)

Abbiamo detto che 802.11ac ti consente di aggregare i canali in modo dinamico, fotogramma per fotogramma. Come succede?

Ogni AP, nei frame Beacon, indica un canale primario a 20/40/80 MHz. Quando un terminale vuole trasmettere un frame utilizzando più canali, invia un RTS su ogni canale che desidera utilizzare. L'AP risponderà con un CTS su ogni canale su cui ha ricevuto l'RTS. A quel punto, come nel caso precedente, tutti i canali per l'intero NAV sono occupati anche per tutte le altre stazioni.

Condivisione Canale

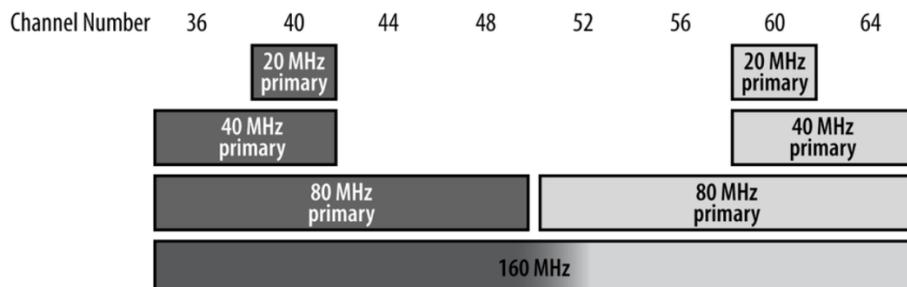


Figure 3-6. Coexistence of multiple networks in the same frequency space