

Niro Simone

Corso di  
**Ingegneria di  
Internet e Web**

*Prof. Francesco Lo Presti*

*CdL in Ingegneria Informatica - A.A. 2022/2023*



# INDICE

|   |    |
|---|----|
| <b>Introduzione - Vista Internet</b>          | 1  |
| <i>Rete di accesso</i>                        | 2  |
| <i>Struttura della rete</i>                   | 2  |
| <i>Metodi di commutazione</i>                 | 2  |
| <i>Prestazioni trasferimento pacchetti</i>    | 4  |
| <b>Architettura protocolli di rete</b>        | 5  |
| <i>Modello ISO-OSI</i>                        | 5  |
| <b>Application Layer</b>                      | 8  |
| <i>Architettura delle applicazioni</i>        | 9  |
| <i>Internet transport protocols services</i>  | 10 |
| <i>World Wide Web</i>                         | 10 |
| <i>Protocolli</i>                             | 10 |
| <b>HTTP</b>                                   | 11 |
| <i>Messaggi: richieste e risposte</i>         | 12 |
| <i>Autorizzazioni</i>                         | 13 |
| <i>Cookies</i>                                | 13 |
| <i>Caching</i>                                | 14 |
| <i>HTTP/1.1 to HTTP/3</i>                     | 14 |
| <b>Electronic Mail</b>                        | 15 |
| <i>Formato mail</i>                           | 15 |
| <i>Protocollo SMTP</i>                        | 16 |
| <i>Protocolli MAP</i>                         | 16 |
| <b>DNS</b>                                    | 18 |
| <i>Un database distribuito e gerarchico</i>   | 18 |
| <i>Funzionamento</i>                          | 19 |
| <i>Caching</i>                                | 19 |
| <i>Record</i>                                 | 19 |
| <i>Messaggi</i>                               | 20 |
| <i>Inserimento di record nel database DNS</i> | 21 |
| <b>Distribuzione di file P2P</b>              | 22 |
| <i>Scalabilità dell'architettura P2P</i>      | 22 |
| <i>BitTorrent</i>                             | 24 |

|   |           |
|---|-----------|
| <i>Video su Internet</i>                                    | 25        |
| <i>Streaming HTTP e DASH</i>                                | 25        |
| <i>Reti per la distribuzione di contenuti</i>               | 26        |
| <i>Come funziona la CDN</i>                                 | 27        |
| <b>Livello di trasporto</b>                                 | <b>29</b> |
| <i>Multiplexing e demultiplexing</i>                        | 29        |
| <i>UDP</i>  | 31        |
| <i>Unidirectional Reliable Data Transfer Protocol (RDT)</i> | 32        |
| <i>RDT v1: canale affidabile</i>                            | 32        |
| <i>RDT v2: errore sui bit</i>                               | 32        |
| <i>RDT v3: errore sui bit e perdita di pacchetti</i>        | 34        |
| <i>Go-Back-N</i>  | 36        |
| <i>Selective Repeat</i>                                     | 38        |
| <i>TCP</i>  | 39        |
| <i>Round Trip Time e Timeout</i>                            | 41        |
| <i>Trasferimento dati affidabile</i>                        | 42        |
| <i>Ritrasmissione rapida</i>                                | 44        |
| <i>Controllo di flusso TCP</i>                              | 45        |
| <i>Gestione della connessione TCP</i>                       | 46        |
| <i>Controllo di congestione TCP</i>                         | 47        |
| <i>Controllo di congestione TCP</i>                         | 48        |
| <i>Fairness TCP</i>   | 51        |
| <i>Notifica esplicita di congestione (ECN)</i>              | 52        |
| <b>Network Layer</b>  | <b>53</b> |
| <i>Modelli di servizio</i>                                  | 54        |
| <i>L'interno di un router</i>                               | 55        |
| <i>Elaborazione alle porte di ingresso</i>                  | 56        |
| <i>Struttura di commutazione</i>                            | 57        |
| <i>Elaborazione alle porte di uscita</i>                    | 58        |
| <i>Dove si verifica l'accodamento?</i>                      | 58        |
| <i>Dimensione del buffer</i>                                | 59        |
| <i>Schedulazione dei pacchetti</i>                          | 59        |
| <i>Formato dei datagrammi IPv4</i>                          | 61        |
| <i>Frammentazione dei datagrammi IPv4</i>                   | 62        |
| <i>Indirizzamento IPv4</i>                                  | 63        |

|  |            |
|--|------------|
| <i>Come ottenere l'indirizzo di un host: DHCP</i>                    | 64         |
| <i>NAT (Network Address Translation)</i>                             | 66         |
| <i>Protocollo IPv6</i>   | 68         |
| <i>Inoltro generalizzato e SDN</i>                                   | 70         |
| <i>Esempio del paradigma match-action in OpenFlow</i>                | 72         |
| <b>Network Layer: Control Plane</b>                                  | <b>73</b>  |
| <i>Algoritmi di instradamento</i>                                    | 74         |
| <i>Intradamento "link-state" (LS)</i>                                | 75         |
| <i>Intradamento "distance-vector" (DV)</i>                           | 78         |
| <i>Intradamento DS: modifica dei costi e guasti dei collegamenti</i> | 80         |
| <i>Confronto tra gli intradamenti LS e DV</i>                        | 81         |
| <i>Intradamento interno ai sistemi autonomi</i>                      | 83         |
| <i>OSPF – Open Shortest Path First</i>                               | 83         |
| <i>Intradamento tra ISP: BGP</i>                                     | 85         |
| <i>Selezione del cammino in BGP</i>                                  | 88         |
| <i>Il piano di controllo SDN</i>                                     | 90         |
| <i>Il protocollo OpenFlow in SDN</i>                                 | 91         |
| <b>Link Layer e LAN</b>  | <b>94</b>  |
| <i>Error detection</i>   | 95         |
| <i>Multiple access link</i>  | 95         |
| <i>Protocolli a suddivisione del canale</i>                          | 96         |
| <i>Protocolli ad accesso casuale</i>                                 | 97         |
| <i>ALOHA</i>   | 97         |
| <i>CSMA</i>  | 98         |
| <i>Protocolli a rotazione</i>  | 100        |
| <i>Indirizzi a livello di collegamento e ARP</i>                     | 100        |
| <i>Ethernet</i>  | 103        |
| <i>Switch a livello di collegamento</i>                              | 104        |
| <i>Confronto switch vs. router</i>                                   | 105        |
| <i>LAN Virtuali (VLAN)</i>   | 107        |
| <i>La rete dei data center</i>                                       | 109        |
| <b>Cronaca di una richiesta di una pagina web</b>                    | <b>111</b> |
| <i>DHCP, UDP, IP e Ethernet</i>                                      | 111        |
| <i>DNS e ARP</i>   | 113        |
| <i>Intradamento intra-dominio al server DNS</i>                      | 114        |

|   |     |
|---|-----|
| <i>Interazione client-server: TCP e HTTP</i>                      | 115 |
| <b>Network Security</b>   | 116 |
| <i>Principi di crittografia</i>                                   | 117 |
| <i>Crittografia a chiave simmetrica</i>                           | 117 |
| <i>Cifrari a blocchi</i>  | 118 |
| <i>Cifrari a blocchi concatenati</i>                              | 120 |
| <i>Crittografia a chiave pubblica</i>                             | 121 |
| <i>RSA</i>  | 121 |
| <i>Chiavi di sessione</i>   | 122 |
| <i>Integrità dei messaggi e firma digitale</i>                    | 123 |
| <i>Funzioni hash crittografiche</i>                               | 123 |
| <i>Codice di autenticazione dei messaggi</i>                      | 123 |
| <i>Firme digitali</i>   | 124 |
| <i>Certificazione della chiave pubblica</i>                       | 126 |
| <i>Autenticazione di un punto terminale</i>                       | 126 |
| <i>E-mail sicure</i>  | 129 |
| <i>PGP</i>  | 130 |
| <i>Rendere sicure le connessioni TCP: TLS</i>                     | 131 |
| <i>Sicurezza a livello di rete: IPsec e reti private virtuali</i> | 133 |
| <i>Il datagramma IPsec</i>  | 134 |
| <i>IKE: gestione delle chiavi in IPsec</i>                        | 136 |
| <i>Firewall</i>   | 137 |
| <i>Sistemi di rilevamento delle intrusioni</i>                    | 138 |



## Introduzione - Vista Internet

Internet è un'infrastruttura di telecomunicazione che permette di realizzare applicazioni distribuite mettendo in comunicazione molteplici **apparati terminali**. Gli elementi chiave sono:

- **Host/End-system**: sistema posto agli estremi della rete (*network edge*) che esegue applicazioni di rete. Tali applicazioni usufruiscono dei servizi offerti dalla rete stessa per poter funzionare.
- **Messaggio**: unità informativa scambiata tra applicazioni di rete.
- **Pacchetto**: unità informativa scambiata tra **nodi** della rete. È una porzione di un *messaggio*.
- **Routers**: sistemi interni alla rete che hanno il compito di inoltrare i pacchetti in ingresso al nodo successivo (*next-hop*) nella strada sorgente-destinazione.
- **Servizi**: funzionalità offerte dalla rete ai dispositivi ad essa connessi.
- **Protocolli**: regolano e controllano lo scambio di messaggi nella rete. In particolare, definiscono il formato e l'ordine dei messaggi inviati e ricevuti tra entità della rete e quali sono le azioni intraprese alla trasmissione e alla ricezione di messaggi.

Più nel dettaglio, la struttura della rete prevede: applicazioni ed host (*network edge*), una rete d'accesso (*access networks*) che permette ai singoli nodi di collegarsi al router più vicino (*router d'accesso/di bordo*), ed una rete di transito (*network core*), costituita da numerosi router tra loro interconnessi.

Quando l'apparato trasmettitore deve inviare una sequenza di bit, quest'ultima viene utilizzata per modulare una particolare forma d'onda che attraverserà il **physical link**. Tuttavia, all'interno del mezzo, i segnali sono soggetti a:

- **attenuation**: riduzione della potenza/energia, in quanto assorbita dal mezzo;
- **distortion**: degradazione della forma del segnale (“scompaiono” gli spigoli a causa della limitazione in frequenza che caratterizza i sistemi fisici);
- **noise**: interferenze, sia esterne che interne (termica), sul segnale.

Quindi, il *bit-rate* della trasmissione e la distanza percorribile dall'informazione è influenzata dal mezzo fisico utilizzato. Per il **teorema di Shannon** si ha che

$$\text{bitrate}_{(\max)} = H \cdot \log_2 \left( 1 + \frac{S}{N} \right),$$

dove  $H$  è la banda del segnale in  $\text{Hz}$  e  $\frac{S}{N}$  è il rapporto segnale-rumore.

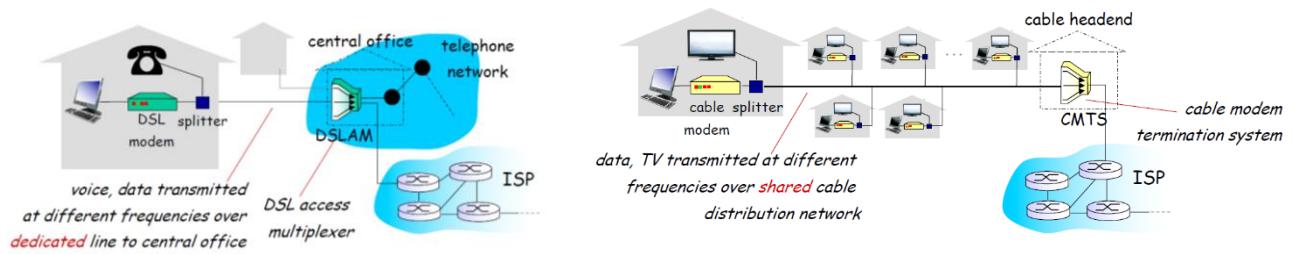
Nel dettaglio, distinguiamo due tipologie di *physical link*:

- **guided link**: il mezzo è un materiale solido (e.g. *Twisted Pair*, ossia una coppia di fili di rame isolati e intrecciati, *coaxial cable* e *fiber optic cable*, ossia un materiale vetroso in grado di trasmettere impulsi luminosi);
- **unguided link**: il mezzo non è solido (e.g. onde radio, ossia onde elettro-magnetiche). In genere subiscono in misura maggiore effetti di eventuali disturbi esterni.

## Rete di accesso

L'accesso alla rete da parte degli *host* può differire per modalità, funzionalità e tecnologie utilizzate. Vediamo alcuni esempi.

- **DSL (Digital Subscriber Line)**: utilizza la **linea telefonica** per le trasmissioni di informazioni. La connessione alla rete avviene attraverso un *modem*. Topologicamente, prima è presente uno *splitter*, il quale ha il compito di smistare pacchetti voce al telefono e pacchetti dati al *modem*. Ne deriva quindi che non è possibile utilizzare entrambi i servizi contemporaneamente.
- **Cavo di rete**: utilizza la **rete televisiva** per la trasmissione di informazione. A differenza dell'accesso DSL, è possibile usufruire contemporaneamente del servizio TV e della rete dati, in quanto viene effettuata **multiplazione in frequenza**.
- **Ethernet**: i dispositivi sono collegati tra loro tramite una **rete cablata ethernet**.
- **Wireless**: i dispositivi si collegano alla rete tramite **access point wireless**.



## Struttura della rete

La rete internet presenta una struttura gerarchica. Infatti, gli utenti sono connessi tra loro tramite una serie di sottoreti, a loro volta interconnesse sia direttamente, sia attraverso altre reti geograficamente di dimensioni maggiori. Alcuni elementi chiave sono:

- **ISP (Internet Service Provider)**: porzione della rete che fornisce connessione internet ad un'area geografica più o meno estesa. Si distinguono:
  - Global ISP: collegano tra loro ISP di livello nazionale;
  - Regional ISP: collegano tra loro ISP di accesso;
  - Access ISP: forniscono accesso internet agli host.
- **IXP (Internet Exchange Point)**: nodi di collegamento tra diversi ISP.

## Metodi di commutazione

La **modalità di commutazione (switching)** descrive come i nodi operano nella rete, ovvero come le informazioni vengono trasferite da un punto ad un altro e quali sono le risorse riservate alla comunicazione. Si distinguono commutazione di circuito e di pacchetto.

La **commutazione di circuito** è un metodo *connection-oriented*. Per ogni comunicazione tra due host vengono riservate/dedicate delle risorse. Sono garantite perciò le prestazioni richieste dagli host, ma occorre gestire esplicitamente l'allocazione delle risorse. La comunicazione risulta divisa in tre fasi:

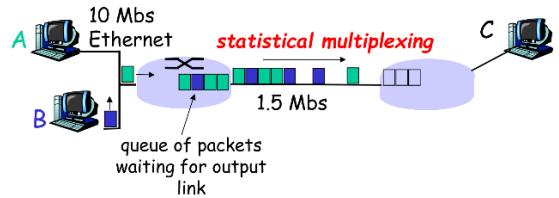
1. instaurazione (*call setup*): allocazione delle risorse;
2. trasferimento dati (*data transfer*): uso delle risorse;
3. rilascio (*call teardown*): rilascio delle risorse.

Dati due host, le risorse allocate individuano un cammino all'interno della rete, il quale sarà utilizzato per tutta la durata della comunicazione. Per aumentare l'efficienza della rete sono utilizzate tecniche di **multiplazione dei canali**, come:

- FDM (*Frequency Division Multiplexing*): ad ogni chiamata viene allocata una frequenza nella banda disponibile;
- TDM (*Time Division Multiplexing*): ad ogni chiamata è allocato uno slot temporale del frame, ovvero di un ciclo.

La **commutazione di pacchetto** è un metodo *connection-less*. Il flusso dati tra host è diviso in **pacchetti**, ognuno dei quali utilizza l'intera capacità dei canali che attraversa. L'allocazione del canale avviene tramite **multiplazione statistica**:

- le risorse del canale sono utilizzate quando richieste;
- non esiste uno schema nella sequenza di pacchetti che attraversano il canale;
- se il traffico di pacchetti supera la capacità del canale (*congestione*), questi vengono messi nella *queue* del *router* che li ha ricevuti, in attesa che si liberino risorse sufficienti. Si può verificare anche perdita di pacchetti nel caso in cui la coda si riempie;
- le risorse della rete sono complessivamente condivise da tutti i suoi utilizzatori.



In particolare, i **router** hanno il compito di inoltrare (*forwarding*) i pacchetti all'interno della rete e distinguiamo:

- **Store & forward**: i router devono attendere l'arrivo di tutti i bit del pacchetto per poterlo ritrasmettere al prossimo nodo. Per tale motivo è conveniente mantenere ridotta la dimensione dei pacchetti;
- **Tabella di inoltro**: per ogni possibile destinatario, indica il *next-hop* a cui inoltrare il pacchetto.

Considerando il trasferimento di un pacchetto (1.500 byte = 12.000 bit):

- col vecchio modem di casa ( $R = 60 \text{ Kbps}$ ) avremmo impiegato  $\frac{L}{R} = \frac{12.000}{60.000} = \frac{1}{5} = 200 \text{ ms}$
- con la nuova tecnologia ( $R = 1.2 \text{ Gbps}$ ) impieghiamo  $\frac{L}{R} = \frac{1.2 \times 10^4}{1.2 \times 10^9} = 10^{-5} = 10 \mu\text{s}$

notiamo come siamo passati da una rete che trasmetteva 5 pacchetti al secondo ad una rete che ne trasmette 100.000.

Le reti a commutazione di pacchetto possono operare in due modalità:

- **circuito virtuale**: simula la commutazione di circuito instaurando una connessione virtuale tra due host. Ogni pacchetto seguirà poi il cammino stabilito dalla connessione instaurata;
- **datagramma**: i pacchetti vengono inoltrati da un nodo ad un altro solamente usando l'indirizzo di destinazione. Ne risulta che i pacchetti scambiati tra una stessa coppia di host possono seguire percorsi differenti ed arrivare fuori ordine.

È importante sottolineare come nella commutazione a pacchetto la probabilità di trovare il canale occupato sia decisamente inferiore rispetto a quella nella commutazione a circuito. In particolare, tale probabilità la si ottiene dal seguente calcolo ( $P$  := probabilità di trovare il canale occupato):

$$\sum_{i=k+1}^N \binom{N}{i} P^i (1-P)^{N-i}$$

Quindi, è in grado di permettere a più utenti di utilizzare la rete.

### Prestazioni trasferimento pacchetti

Le prestazioni di un canale relative alla trasmissione di pacchetti possono essere misurate in termini di:

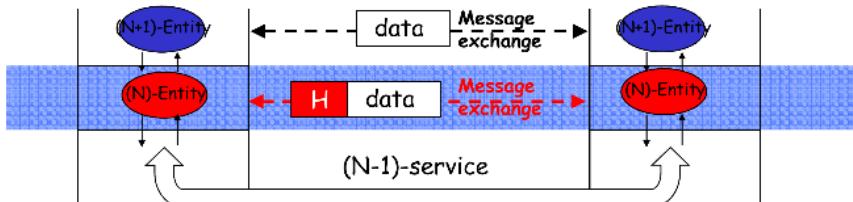
- **ritardo totale di nodo** (*nodal delay*): tempo necessario a spedire un pacchetto dalla sorgente alla destinazione. Può essere espresso come la somma di quattro fattori:
  - $d_{proc}$  **nodal processing**: operazioni di controllo sui bit e determinazione del canale di uscita;
  - $d_{queue}$  **queueing delay**: tempo atteso in coda. Dipende dal livello di congestione della rete;
  - $d_{trans}$  **transmission delay** =  $\frac{L}{R} := \frac{\text{dimensione pacchetti}}{\text{velocità trasmissione}}$ ;
  - $d_{prop}$  **propagation delay** =  $\frac{d}{s} := \frac{\text{lunghezza mezzo fisico}}{\text{velocità propagazione}}$ .
- **perdite** (*loss*): pacchetti scartati dal nodo che precede il canale a causa della coda piena. Tali pacchetti possono essere ritrasmessi in accordo al protocollo utilizzato. L'occorrenza delle perdite è legata sia alla capacità del canale che al traffico presente nella rete.
- **capacità** (*throughput*): velocità di trasferimento dei bit dal mittente al destinatario espressa come  $R$  = bits/time unit. Distinguiamo vari tipi di throughput:
  - **istantaneo**: velocità in un dato istante di tempo;
  - **medio**: velocità media in un dato intervallo di tempo;
  - **peggiore** (*bottleneck*): canale con velocità minore lungo il cammino che collega due host. Determina la velocità massima teorica a cui può avvenire la comunicazione.

Ad ogni modo, il trasferimento dei pacchetti riguarda anche gli attacchi informatici, quali ad esempio gli attacchi DoS, il *packet “sniffing”* e l'*IP spoofing*.

## Architettura protocolli di rete

I sistemi connessi alla rete internet sono caratterizzati da un'**architettura a strati**, ovvero un modello astratto dell'ambiente di comunicazione che struttura e organizza i protocolli necessari alla comunicazione stessa in una sorta di pila ordinata. In particolare, si definiscono:

- **(N)-Strato (layer)**: implementa **(N)-servizi**, ossia insiemi di (N)-funzioni, che vengono forniti allo strato superiore (N+1). Tali funzioni sono realizzate
  - utilizzando azioni proprie dello strato regolate da specifici **protocolli**,
  - sfruttando i servizi offerti dal livello inferiore (N-1).
- **(N)-Entità (entity)**: elementi attivi dell'(N)-strato che implementano e trasportano una specifica (N)-funzione cooperando con altre entità dello stesso strato.
- **(N)-Protocollo (protocol)**: regola la cooperazione tra (N)-entità, specificando
  - la sintassi e la semantica dei messaggi scambiati tra le entità,
  - le modalità con le quali i processi di rete inviano e rispondono ai messaggi.
- **(N)-PDU (Protocol Data Unit)**: messaggio cambiato tra (N)-entità. È costituito da
  - **(N)-PCI (Protocol Control Information)**: **header**, che serve a trasportare le (N)-funzioni;
  - **(N)-SDU (Service Data Unit)**: **payload**, che contiene le informazioni dell'utente dell'(N)-servizio.



Questo permette di suddividere il sistema in diversi sottosistemi indipendenti, modulati e di complessità inferiore, così da facilitare sia la manutenzione che l'aggiornamento del sistema.

Esistono due architetture a strati di riferimento: il modello ISO-OSI (7 livelli) e il modello TCP/IP (5 livelli).

### Modello ISO-OSI

L'**architettura OSI (Open System Interconnection)** è costituita da sette livelli e rispetta i seguenti principi:

- minimizzare il numero di livelli;
- stabilire i confini tra i diversi livelli in cui l'interazione è minima;
- i livelli dovrebbero fare cose diverse e dovrebbero aggiungere qualcosa;
- i livelli dovrebbero essere progettati in modo tale che una modifica non vada ad impattare le interfacce con i livelli adiacenti.

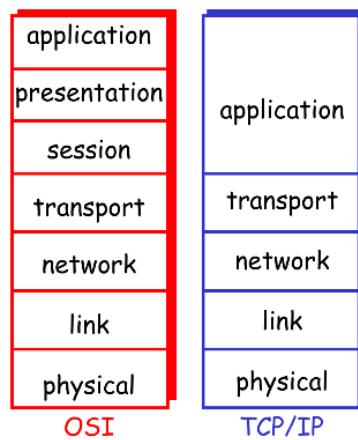
Vediamo più nel dettaglio i vari livelli.

1. **Livello fisico.** Il livello fisico (*Physical*) coordina le funzioni richieste alla trasmissione di flussi di **bit** su un mezzo fisico. Definisce:
  - le caratteristiche delle interfacce e dei mezzi di comunicazione;
  - la rappresentazione dei bit;
  - la velocità di trasmissione;
  - la sincronizzazione dei bit;
  - la configurazione della linea;
  - la modalità di trasmissione.
2. **Livello di collegamento.** Il livello di collegamento (*Data link*) ha il compito di trasformare il canale fisico, di natura inaffidabile, in un canale logico affidabile. Rende utilizzabile il livello precedente ai livelli superiori, in quanto si occupa di:
  - *framing*: divisione dei pacchetti in **frame**;
  - controllo di flusso delle trame (evitare che un nodo invii troppi dati al nodo successivo);
  - controllo di errore (delle trame);
  - controllo di accesso al canale (come comportarsi se si ha collisione);
  - indirizzamento fisico.
3. **Livello di rete.** Il livello di rete (*Network*) ha il compito di consegnare **pacchetti** tra nodo sorgente e nodo destinazione, ovunque essi siano e indipendentemente dal fatto che essi siano collegati da un mezzo fisico o meno. Si occupa di:
  - indirizzamento logico;
  - instradamento dei pacchetti.
4. **Livello di trasporto.** Il livello di trasporto (*Transport*) ha il compito di consegnare i **messaggi** tra due applicazioni. Si occupa di:
  - segmentazione dei messaggi in segmenti;
  - riassemblaggio dei segmenti in messaggi;
  - indirizzamento ai servizi;
  - controllo di connessione (tra host);
  - controllo di flusso dei pacchetti (tra entità logiche e non fisiche come in L2);
  - controllo di errore dei pacchetti (tra entità logiche).
5. **Livello di sessione.** Il livello di sessione (*Session*) ha il compito di gestire la comunicazione tra host. Si occupa di:
  - controllo di dialogo (apertura, chiusura e mantenimento delle sessioni);
  - sincronizzazione.
6. **Livello di presentazione.** Il livello di presentazione (*Presentation*) si occupa della sintassi e della semantica delle informazioni scambiate, rendendo la comunicazione indipendente dalla codifica delle informazioni. Svolge funzioni di traduzione, crittografia e compressione.

**7. Livello di applicazione.** Il livello di applicazione (*Application*) permette all'utente di connettersi alla rete. Si occupa di:

- realizzare un terminale virtuale della rete;
- trasferimento, accesso e amministrazione di files;
- servizi di e-mail;
- servizi di esplorazione remota dei dispositivi.

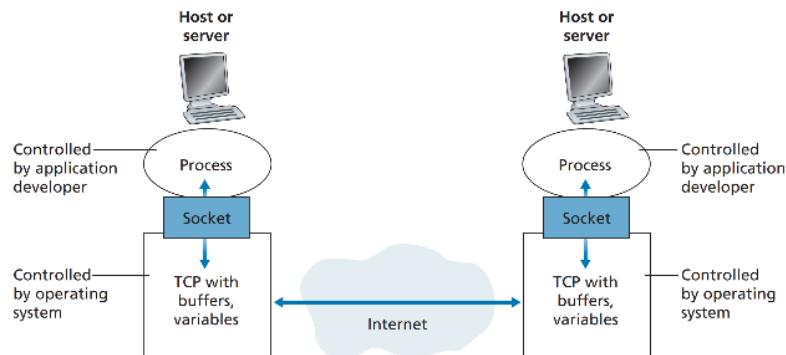
In realtà, dei sette livelli citati, Internet ne utilizza solo cinque, in quanto vengono esclusi i livelli di presentazione e di sessione.



## Application Layer

Il **livello applicativo** permette di far comunicare, tramite la rete, programmi eseguiti su sistemi differenti e distanti tra loro. Vediamo alcune definizioni.

- **Processo:** programma in esecuzione all'interno di un host. Considerati due processi, si ha che:
  - se sono in esecuzione sullo stesso host, comunicano tra loro usando la comunicazione inter-processo (definita dal SO);
  - se sono in esecuzione su host differenti, comunicano tra loro usando lo strato applicativo.
- **User Agent (UA):** software posto tra l'utente e la rete. Implementa:
  - l'interfaccia grafica per presentare le informazioni all'utente;
  - i protocolli di livello applicativo per svolgere le sue funzioni di rete.
- **Applicazione:** insieme di processi distribuiti che comunicano tra loro per offrire un servizio agli utenti. Tutti i processi eseguono gli stessi protocolli per rendere effettiva la comunicazione.
- **Indirizzo IP (Internet Protocol Address):** indirizzo a 32 bit che identifica l'host all'interno della rete.
- **Porta:** indirizzo a 16 bit che identifica un processo all'interno dell'host.
- **Socket:** interfaccia tra livello applicativo e livello di trasporto. Utilizza l'indirizzo IP e il numero di porta per individuare uno specifico processo all'interno della rete.



I servizi richiesti dal livello applicativo al livello di trasporto sono:

- **affidabilità:** se viene ammessa oppure no la perdita di messaggi, e in che entità. Infatti, alcune applicazioni (e.g. audio) possono tollerare qualche perdita, mentre altre (e.g. *file transfer*) hanno bisogno di un trasferimento che sia affidabile al 100%;
- **capacità:** velocità di trasmissione necessaria alla trasmissione delle informazioni. Infatti, alcune applicazioni richiedono una quantità minima di banda per funzionare (e.g. multimedia), mentre altre applicazioni (*elastic apps*) utilizzano qualsiasi banda gli capitì;
- **tempismo:** ritardo massimo ammesso per la trasmissione delle informazioni. Infatti, alcune applicazioni (e.g. *internet telephony*, *interactive games*) hanno bisogno di un delay molto basso per funzionare al meglio.

| Application           | Data loss     | Bandwidth                                | Time Sensitive  |
|-----------------------|---------------|--|-----------------|
| file transfer         | no loss       | elastic                                  | no              |
| e-mail                | no loss       | elastic                                  | no              |
| Web documents         | no loss       | elastic                                  | no              |
| real-time audio/video | loss-tolerant | audio: 5kbps-1Mbps<br>video:10kbps-5Mbps | yes, 100's msec |
| stored audio/video    | loss-tolerant | same as above                            | yes, few secs   |
| interactive games     | loss-tolerant | few kbps up                              | yes, 100's msec |
| instant messaging     | no loss       | elastic                                  | yes and no      |

## Architettura delle applicazioni

Esistono diverse possibili **architetture di un'applicazione di rete**.

- **Client-Server:** prevede due tipi di host nella rete:
  - **server:** host sempre attivo e con indirizzo IP fisso, così da essere sempre raggiungibile. Fornisce dei servizi applicativi ai client.
  - **client:** host che ha interesse a comunicare con il server per ottenere dei servizi, che possono indirettamente coinvolgere anche altri utenti. Può essere connesso a intermittenza al server e può avere un indirizzo IP dinamico. Non comunica direttamente con altri client.
- **Peer-to-Peer (P2P):** prevede un unico tipo di host. Gli utenti dell'applicazione comunicano direttamente tra loro. Possono essere connessi a intermittenza e possono cambiare indirizzo. L'assenza di un host centrale noto rende complessa la gestione delle connessioni tra host. Un possibile approccio è il seguente:
  - *Query flooding:* i peers dichiarano i file che possiedono. Un peer chiede ai suoi vicini se possiedono un file. Questi inoltrano la richiesta ai rispettivi vicini. Gli esiti delle richieste tornano al peer richiedente a ritroso. Lo scambio del file avviene direttamente tra peers.
- **Irida:** possiede sia aspetti dell'architettura client-server che P2P. In genere, viene utilizzato l'approccio client-server per gestire le connessioni e l'approccio P2P per erogare i servizi. Considerando un'applicazione che voglia scambiare file tra host, diversi approcci possibili sono:
  - **indice centralizzato:** esiste un server centrale che tiene memoria delle coppie file-indirizzoIP. Ogni client interroga il server per ottenere gli indirizzi IP di host che possiedono il file richiesto. Lo scambio del file avviene direttamente tra host (peer).
  - **overlay gerarchico:** si distingue tra peer super-nodo e peer assegnati ad un super-nodo. Ogni super-nodo tiene traccia dei file di tutti i suoi figli ad esso connesso. Un peer chiede al proprio super-nodo chi possiede il file richiesto. Lo scambio del file avviene direttamente tra peer.

## Internet transport protocols services

Distinguiamo due tipi di protocollo di trasporto:

- **TCP (Transport Control Protocol)**: orientato alla connessione, offre un servizio di comunicazione affidabile (mittente e destinatario si scambiano messaggi senza errori tra le due parti), controllo di flusso (il mittente non può sovraccaricare il buffer del destinatario), controllo di congestione (decide a che velocità inviare i dati nella rete in modo tale che quest'ultima non si congestioni). Non offre: banda minima garantita e temporizzazione.
- **UDP (User Data Protocol)**: servizio inaffidabile di comunicazione che non offre servizio di connessione (possiamo inviare segmenti a destinatari che in realtà non esistono), controllo di flusso (possiamo sovraccaricare il destinatario), controllo di congestione (non viene limitata la velocità). Viene impiegato perché molto semplice e leggero.

I socket di TCP e UDP di base non sono “sicuri” e per sopprimere a questo si fa uso di un livello intermedio detto **TLS (Transport Layer Security)**, il quale rende più sicura la comunicazione in TCP. Quindi, viene aggiunto un “mini strato” tra il livello applicativo e il livello di trasporto.

## World Wide Web

Il **web** è una collezione di risorse e oggetti mantenuti all'interno di server e distribuiti ai client tramite la rete internet. Si definiscono:

- **URL (Uniform Resource Location)**: stringa che permette di individuare una risorsa all'interno della rete. Le tre componenti essenziali che lo costituiscono sono:
  - *protocol*: indica il protocollo da utilizzare;
  - *host*: identifica l'host;
  - *file*: identifica la risorsa richiesta.
- **Pagina**: “contenitore” per risorse e oggetti. È raggiungibile tramite un indirizzo URL.
- **HTML (HyperText Markup Language)**: tipo di pagina web. Può contenere al suo interno molteplici risorse e oggetti, oltre che URL ad altre pagine.

## Protocolli

In generale, i **protocolli del livello applicativo** definiscono:

- i tipi di messaggi scambiati dai processi;
- la sintassi dei diversi tipi di messaggi;
- la semantica di tutti i campi dei messaggi;
- il comportamento dei processi all'invio e alla ricezione dei messaggi.

## HTTP

Il **protocollo HTTP** (*HyperText Transfer Protocol*) definisce le regole con le quali un client richiede una pagina web ad un server ed il server risponde a tale richiesta. Utilizza il *protocollo TCP* e la *porta 80* per svolgere le sue funzioni.

HTTP è un protocollo **stateless** (mantenere lo stato comporterebbe l'utilizzo di risorse, causando un aumento della complessità). Può operare in due modalità differenti:

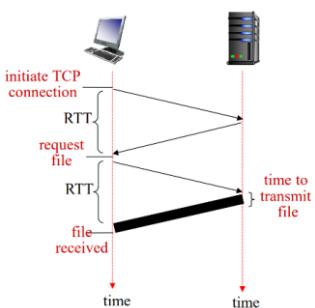
- **non persistente** (v1.0): può inviare un solo oggetto per connessione TCP;
- **persistente** (v1.1): può inviare molteplici oggetti su una stessa connessione TCP.

| HTTP non persistente |  |
|----------------------|--|
| Client               | Inizializza la connessione TCP alla porta 80 verso il server HTTP                |
| Server               | Accetta la connessione TCP alla porta 80 e notifica il client                    |
| Client               | Invia il messaggio di richiesta HTTP sulla connessione TCP                       |
| Server               | Riceve la richiesta ed invia il messaggio di risposta HTTP sulla connessione TCP |
| Server               | Chiude la connessione TCP al termine del trasferimento                           |
| Client               | Riceve il messaggio di risposta dal server, contenente l'oggetto richiesto       |

| HTTP persistente con/senza pipelining |  |
|---------------------------------------|--|
| Client                                | Inizializza la connessione TCP alla porta 80 verso il server HTTP  |
| Server                                | Accetta la connessione TCP alla porta 80 e notifica il client  |
| Client                                | Invia il messaggio di richiesta HTTP sulla connessione TCP   |
| Server                                | Riceve la richiesta ed invia il messaggio di risposta HTTP sulla connessione TCP   |
| Client                                | Riceve il messaggio di risposta dal server, contenente l'oggetto richiesto   |
| Client                                | <b>Senza pipelining:</b> può inviare un altro messaggio di richiesta HTTP al server una volta elaborata l'intera risposta.<br><b>Con pipelining:</b> può inviare un altro messaggio di richiesta HTTP al server appena trova un riferimento ad un altro oggetto all'interno della risposta |
| Server                                | Riceve la richiesta ed invia il messaggio di risposta HTTP sulla connessione TCP   |
| ⟳                                     | Si ripetono i due passi precedenti fintanto che il client chiede nuovi oggetti   |
| Server                                | Chiude la connessione TCP al termine del trasferimento   |

Generalmente, per inviare un messaggio nella rete è necessario un tempo pari al **RTT** (*Round Trip Time*). Considerando il caso in cui si voglia trasmettere un oggetto di dimensione trascurabile, sono necessari 2 RTT, in quanto:

- un RTT è necessario per instaurare la connessione TCP;
- un altro RTT è necessario per trasferire l'oggetto della richiesta HTTP.



Nel caso in cui volessimo inviare molteplici oggetti, sempre di dimensione trascurabile, si ottengono prestazioni diverse a seconda della modalità utilizzata:

- non persistente: 2 RTT/oggetto. Inoltre, il SO deve gestire una connessione TCP per ogni oggetto richiesto;
- persistente senza pipelining: 1 RTT/oggetto + 1 RTT per l'inizializzazione;
- persistente con pipelining: 2 RTT, uno per l'inizializzazione ed uno per il trasferimento degli oggetti.

### Messaggi: richieste e risposte

HTTP prevede due tipi di messaggi, uno di richiesta ed uno di risposta, in formato ASCII. Il *client* invia **messaggi di richiesta** al *server*, la cui struttura prevede:

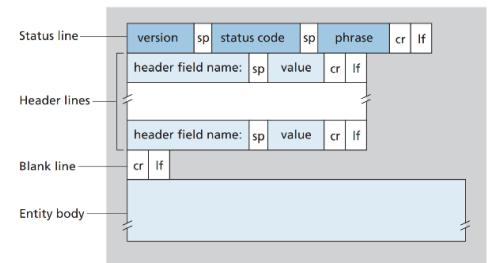
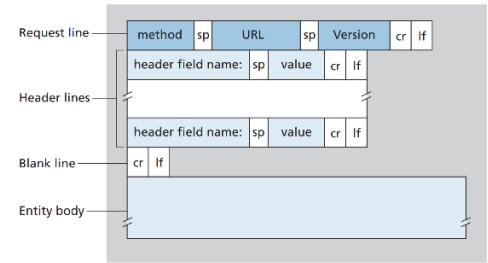
- **Request line:** specifica
  - *Method:* operazione da effettuare sull'oggetto;
  - *URL:* l'indirizzo dell'oggetto della richiesta.
  - *Version:* la versione di HTTP da usare per la richiesta.
- **Header lines:** contiene diverse informazioni sulla richiesta;
- **Entity body:** contiene dati necessari al server per elaborare la richiesta (campi compilati, oggetto da caricare, ...).

In particolare, tra i vari **metodi di richiesta** troviamo:

- **GET:** si richiede l'oggetto specificato nell'URL. Eventuali dati da inviare al server sono contenuti nell'URL stesso (pagine dinamiche);
- **POST:** si richiede l'oggetto specificato nell'URL. Nel campo body sono presenti dati inseriti dall'utente (e.g. campi di un form);
- **HEAD:** si richiede l'oggetto specificato nell'URL. Nel messaggio di risposta però si richiede di non includere l'oggetto stesso;
- **PUT:** si richiede di caricare sul sito l'oggetto nel body all'URL indicato;
- **DELETE:** si richiede di eliminare dal server l'oggetto specificato nell'URL.

Una volta ricevuto il messaggio di richiesta, il *server* invia i **messaggi di risposta** al *client* la cui struttura prevede:

- **Status line:** specifica
  - *Version:* la versione di http da usare per la risposta;
  - *Status code:* numero che indica l'esito della richiesta;
  - *Phrase:* descrive il significato dello *status code*.
- **Header lines:** contiene diverse informazioni sulla risposta;
- **Entity body:** contiene l'oggetto richiesto.



Alcuni **codici di risposta** sono:

- **200 OK**: la richiesta è avvenuta con successo, l'oggetto richiesto è nel corpo del messaggio;
- **301 Moved Permanently**: l'oggetto richiesto è stato spostato. Il suo nuovo indirizzo è specificato nell'header (“**Location: <value>**”);
- **400 Bad Request**: la richiesta non è stata compresa dal server;
- **404 Not Found**: l'oggetto richiesto non è presente nel server;
- **505 HTTP Version Not Supported**: la versione del protocollo utilizzata non è supportata.

Invece, alcuni esempi di **header lines** sono:

- **Host**: nome dell'host nella quale risiede l'oggetto richiesto;
- **User-agent**: informazioni relative al browser utilizzato per effettuare la richiesta;
- **Connection**: viene specificato cosa si vuole della connessione una volta ricevuto il messaggio (e.g. “**close**”, “**keep-alive**”, ...);
- **Accept-language**: viene specificata la preferenza per la lingua;
- **Content-length**: dimensione in byte dell'oggetto nel corpo;
- **Content-type**: indica il tipo di oggetto nel corpo.

## Autorizzazioni

HTTP prevede il controllo di accesso ai contenuti del server tramite **autenticazione del client** (nome e password). Questa è realizzata in maniera stateless, ovvero il client deve includere in ogni messaggio l'autorizzazione ad accedere alle risorse del server. In particolare:

- l'autorizzazione è contenuta nell'header della richiesta HTTP (“**Authorization: <cred>**”);
- se il client non include l'autorizzazione, il server rifiuta l'accesso alla risorsa richiesta ed invia un messaggio di risposta con corpo vuoto per informare il client.

## Cookies

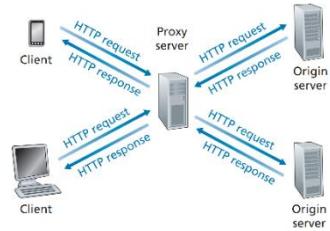
I **cookies** sono elementi del protocollo HTTP che permettono di mantenere delle informazioni sugli utenti che usufruiscono delle funzionalità offerte da un server. Ogni cookie è costituito da quattro componenti:

1. una linea header (“**Set-cookie: <id>**”) nel messaggio di risposta HTTP del server;
2. una linea header (“**Cookie: <id>**”) nel messaggio di richiesta HTTP del client;
3. un **file cookie** mantenuto nel sistema del client e gestito dal browser;
4. un back-end **database di cookie** nel server.

I cookie possono contenere diversi tipi di informazione, come autorizzazioni, stato delle sessioni e/o dati di navigazione. Ogni volta che un utente non identificato si collega, il server fornisce al client un nuovo cookie. Il client deve fare riferimento a tale cookie se vuole mantenere le informazioni in esso memorizzate.

## Caching

Il **caching** è una tecnica che permette di alleggerire il carico dei server e, quindi, il traffico nella rete internet, inserendo un **proxy server** (web cache) all'interno di una sottorete, o tra la sottorete ed il resto della rete. In dettaglio, si configurano i browser dei client in maniera tale da inviare tutte le richieste HTTP al proxy server.



Il *proxy server* alla ricezione di una richiesta HTTP si comporta come segue:

- se è presente una copia dell'oggetto richiesto nella cache, effettua una richiesta con metodo **conditional GET** verso il server di origine (*origin server*) specificando la data alla quale è aggiornata la copia. Si ha che:
  - se la copia è aggiornata, il server di origine invia una risposta HTTP con corpo vuoto ed indicante che la copia è aggiornata (304 – Not Modified),
  - altrimenti, il server di origine invia una risposta HTTP contenente l'oggetto richiesto e viene quindi aggiornata la cache.
- se non è presente una copia dell'oggetto richiesto nella cache, inoltra la richiesta al server di origine, ottiene da esso l'oggetto richiesto, lo salva nella cache e lo inoltra al client.

Si noti che il proxy server si comporta sia da client che da server. Inoltre, permette di ridurre sia i tempi di risposta delle richieste all'interno della sottorete che il traffico attraverso il canale verso la rete internet. Infine, il **conditional GET** assicura che gli oggetti restituiti dal proxy server siano aggiornati, ovvero che le copie cache siano coerenti con gli oggetti presenti sui server.

## HTTP/1.1 to HTTP/3

**HTTP/1.1** ha introdotto la possibilità di utilizzare le **GET Pipelined** su connessioni TCP: il server risponde in ordine alle richieste GET dell'utente (seguendo quindi una politica FCFS), ma correndo il rischio di generare **head-of-line (HOL) blocking** in presenza di oggetti molto grandi.

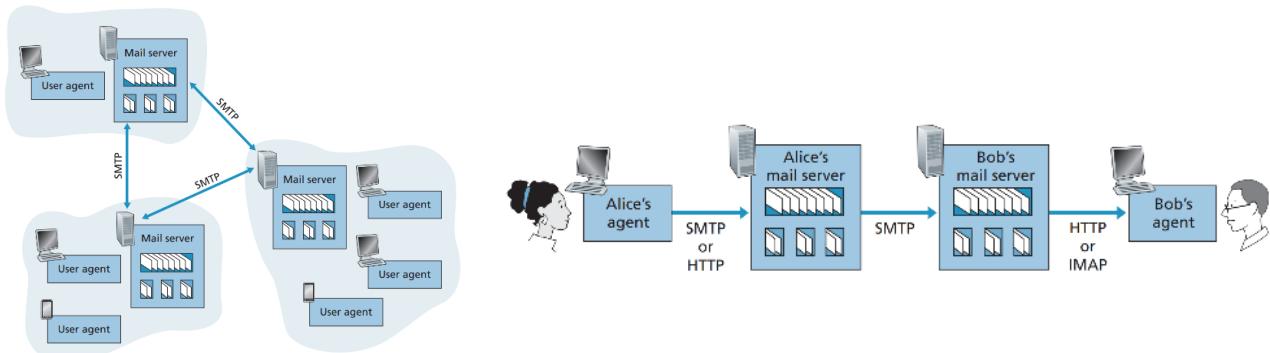
**HTTP/2** ha aumentato la flessibilità con cui il server gestisce l'invio di oggetti al client: l'ordine può essere specificato anche dal client, il server può inviare oggetti non richiesti al client (purché ritenuti opportuni) e gli oggetti stessi vengono scomposti in **frame**. Il tempo totale di trasferimento risulta essere sempre lo stesso, mentre il tempo medio molto più basso.

**HTTP/3** aggiunge la sicurezza, il controllo di errore, il controllo di congestione per oggetto ed utilizza UDP.

# Electronic Mail

Il servizio di posta elettronica, ovvero di messaggistica tramite rete internet, prevede tre componenti principali:

- **mail servers**: server in cui sono memorizzati i messaggi in arrivo agli utenti. Possiedono:
  - una **casella di posta** (*mailbox*) per ogni utente;
  - una **coda di messaggi** in uscita dal server.
- **protocollo SMTP** (*Simple Mail Transfer Protocol*): regola lo scambio di posta tra mail servers;
- **user agents**: permette ai **client** di accedere alla casella di posta. Utilizza un **protocollo MAP** (*Mail Access Protocol*) per comunicare con il mail server:
  - **POP** (*Post Office Protocol*);
  - **IMAP** (*Internet Message Access Protocol*);
  - **HTTP**: si appoggia su un server che fa da da tramite.



Lo **scambio di mail** tra utenti avviene nel modo seguente:

|               |  |
|---------------|--|
| Host A        | Usa il proprio user agent A per comporre l'e-mail e inviarla               |
| User Agent A  | Invia l'e-mail al proprio mail server A utilizzando SMTP o HTTP            |
| Mail Server A | Invia l'e-mail al mail server B utilizzando SMTP                           |
| Mail Server B | Riceve l'e-mail e la salva nella casella di posta dell'utente B            |
| Host B        | Chiude la connessione TCP al termine del trasferimento                     |
| User Agent B  | Riceve il messaggio di risposta dal server, contenente l'oggetto richiesto |

## Formato mail

Le mail sono codificate in ASCII a 7-bit e seguono la seguente struttura:

- **Intestazione (Header)**: specifica
  - destinatario (*To*): casella di posta del destinatario;
  - mittente (*From*): casella di posta del mittente;
  - oggetto (*Subject*): titolo della mail;
  - MIME (*Multipurpose Internet Mail Extensions*): informazioni sul contenuto all'interno del corpo, come ad esempio il tipo, la codifica utilizzata e se il corpo è costituito da più parti. I tipi supportati sono:
    - *text*: testo semplice o HTML;

- *image*: immagine e relativa codifica;
  - *audio*: audio e relativa codifica;
  - *video*: video e relativa codifica;
  - *application*: dati da processare.
- **Linea vuota**: separa intestazione e corpo;
  - **Corpo (Body)**: contiene il messaggio dell'e-mail;
  - **Terminazione [CR LF . CR LF]** (punto a capo).

```

From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data .....
.....base64 encoded data
  
```

## Protocollo SMTP

Il **protocollo SMTP** (*Simple Mail Transfer Protocol*) definisce le regole per trasferire mail verso un **mail server**. Utilizza il protocollo TCP (livello di trasporto) e la **porta 25** per svolgere le sue funzioni.

Il trasferimento di mail si compone di tre fasi:

1. **handshaking**: il server di invio apre una connessione TCP sulla porta 25 verso il server di ricezione ed effettua alcune operazioni di inizializzazione;
2. **trasferimento**: invio di tutte le mail destinate a caselle di posta su server di ricezione;
3. **chiusura**: il server di invio chiude la connessione TCP.

In particolare: il server *mittente* invia **comandi** al server *destinatario* (costituiti da testo ASCII), mentre il server *destinatario* invia **risposte** al server *mittente* (costituite da uno **status code** e una frase esplicativa).

Si noti che il SMTP è molto simile ad HTTP, con le differenze che:

- SMTP lavora in logica *push* e un messaggio SMTP può contenere più oggetti;
- HTTP lavora in logica *pull* e un messaggio HTTP può contenere un solo oggetto.

## Protocolli MAP

I **protocolli MAP** (*Mail Access Protocol*) definiscono le regole con le quali uno User Agent accede ad una casella di posta su un mail server. I due protocolli più utilizzati sono POP3 e IMAP.

- **POP3** (*Post Office Protocol v3*): entra in azione quando lo UA apre una connessione TCP verso il mail server sulla porta 110 e permette autenticazione e download delle e-mail. È un protocollo *stateless*, ossia non mantiene informazioni sulle sessioni precedenti. Quando la connessione TCP è stabilita, POP3 procede in tre fasi:
  - **autorizzazione**: lo UA del client invia nome utente e password (in chiaro) per autenticare l'utente. Il server comunica l'esito dell'autenticazione (OK o ERR);
  - **transazione**: lo UA del client recupera i messaggi, ossia ottiene la lista di tutte le e-mail nella casella di posta (*list*). Per ogni mail in lista, la recupera (*retr*) e la marca per la cancellazione (*dele*). Infine, chiude la sessione;

- **aggiornamento:** fase accessibile solo dopo che il client ha inviato il comando `quit`, che conclude la sessione POP3. Il server di posta rimuove i messaggi marcati per la cancellazione.

È anche possibile omettere il comando di cancellazione dalla sequenza delle operazioni così da permettere il recupero delle e-mail di una stessa casella di posta anche da un altro client. Questo protocollo non fornisce all'utente alcuna procedura per creare cartelle remote e assegnare loro i messaggi.

- **IMAP (*Internet Message Access Protocol*):** permette autenticazione e manipolazione delle e-mail. Un server IMAP associa ogni messaggio arrivato al server ad una cartella. I messaggi in arrivo sono associati alla cartella INBOX del destinatario, che può poi spostare il messaggio in una nuova cartella creata dall'utente e manipolarlo. A differenza di POP3, i server IMAP conservano informazioni di stato sull'utente da una sessione all'altra: per esempio, i nomi delle cartelle e l'associazione tra i messaggi e le cartelle.

## DNS

Il **DNS** (*Domain Name System*) è un servizio in grado di tradurre i nomi degli host nei loro indirizzi IP. Utilizza il protocollo UDP (livello di trasporto) e la **porta 53**.

Oltre alla traduzione degli hostname in indirizzi IP, DNS mette a disposizione altri importanti servizi:

- **Host/mail server aliasing**: il DNS può essere invocato da un'applicazione per ottenere l'**hostname canonico** di un sinonimo, così come l'indirizzo IP dell'host. Risulta particolarmente utile per i mail servers.
- **Load distribution**: nel caso di web server replicati, va associato ad ogni hostname canonico un insieme di indirizzi IP. Il database DNS contiene questo insieme di indirizzi e quando i client effettuano una query DNS per un nome associato a un insieme di indirizzi, il server risponde con l'intero insieme di indirizzi, ma ne varia l'ordinamento ad ogni risposta.

Un database centralizzato su un singolo DNS server non è in grado di adattarsi alla crescita esponenziale della rete (ovvero, non è scalabile). Di conseguenza, il DNS è stato progettato in maniera distribuita, permettendo quindi:

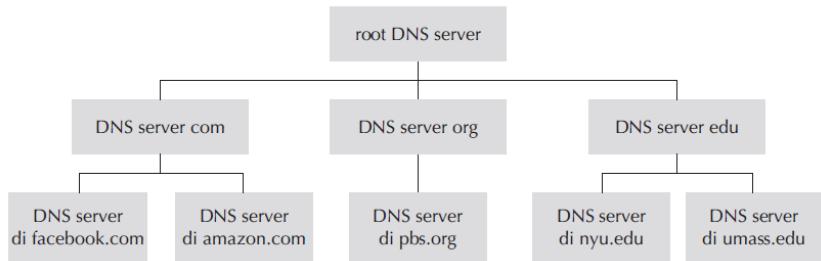
- di evitare di avere un singolo punto di fallimento (*point of failure*) nella rete;
- la distribuzione del traffico di richieste tra più server DNS;
- di evitare ritardi dovuti alla distanza tra i client e l'unico server DNS della rete;
- di semplificare la manutenzione, poiché le informazioni da aggiornare sono divise tra i server DNS.

### Un database distribuito e gerarchico

Il **domain name** è un nome alfanumerico (associato ad un server-host) diviso in diverse parti, dette **livelli di dominio**, separate da un punto e con il livello maggiore a partire da destra.

Per trattare il problema della scalabilità, il DNS utilizza un grande numero di server, organizzati in maniera gerarchica e distribuiti nel mondo. Esistono quattro classi di DNS server:

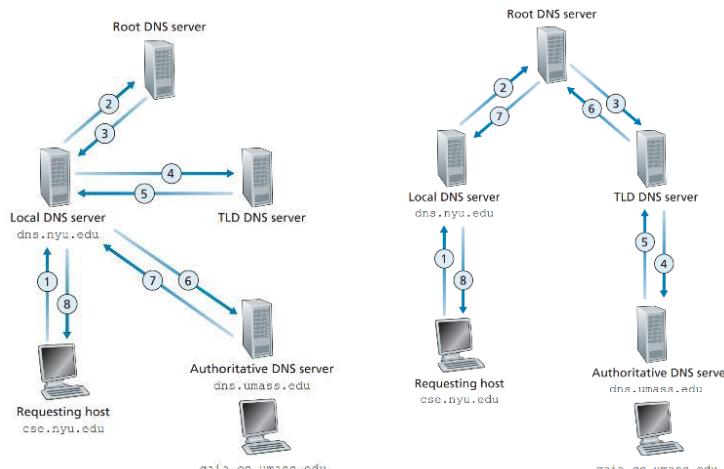
- **Root server**: forniscono gli indirizzi IP dei server TLD. In Internet ne esistono 400 dislocati in tutto il mondo.
- **TLD server** (*Top-Level Domain*): forniscono gli indirizzi IP dei server autoritativi. Si occupano dei domini di primo livello quali com, org, net, edu e gov, e di tutti i domini di primo livello relativi ai vari paesi, come uk, fr, ca e jp.
- **Authoritative DNS server**: ogni organizzazione dotata di host pubblicamente accessibili tramite Internet (quali web server e mail server) deve fornire record DNS pubblicamente accessibili che associno i nomi di tali host a indirizzi IP. Il DNS server autoritativo dell'organizzazione ospita questi record.
- **Local DNS server**: ciascun ISP ha un DNS server locale. Quando un host si connette a un ISP, quest'ultimo gli fornisce un indirizzo IP tratto da uno o più dei suoi DNS server locali.



## Funzionamento

Ogni istanza DNS può operare in due modalità differenti:

- **query iterativa:** il DNS server contattato restituisce
  - l'indirizzo IP del dominio, se questo è noto;
  - l'indirizzo IP del DNS server che conosce il metodo.
- **query ricorsiva:** il DNS server contattato restituisce al client l'indirizzo IP del dominio. Se questo non è noto, si occupa di recuperarlo da altri DNS server.



## Caching

Il DNS server che riceve una risposta DNS (contenente, ad esempio, la traduzione da hostname a indirizzo IP), può mettere in cache le informazioni contenute. Se una coppia hostname/indirizzo IP è nella cache di un DNS server e giunge al server un'altra richiesta con lo stesso hostname, il DNS server può fornire l'indirizzo IP desiderato, anche se non è autoritativo per tale indirizzo.

Dato che gli host e le associazioni tra nome e indirizzo IP non sono in alcun modo permanenti, i DNS server invalidano le informazioni in cache dopo un periodo di tempo fissato (in genere di due giorni).

Inoltre, un DNS server locale può memorizzare in cache gli indirizzi IP dei TLD server, consentendogli di aggirare i root server nella catena di richieste.

## Record

I server che implementano il database distribuito di DNS memorizzano i cosiddetti **resource record** (RR), tra cui quelli che forniscono le corrispondenze tra nomi e indirizzi. Ogni messaggio di risposta DNS trasporta uno o più record di risorse.

Un record di risorsa contiene i seguenti campi:

- *Name* – contiene un nome di dominio, il cui significato dipende dal tipo di record;
- *Value* – in base al tipo di record, contiene o un nome di dominio o un indirizzo IP;
- *Type* – determina il significato dei campi nome e valore. Esistono quattro tipi:
  - A (*Address*): [nome = hostname] e [valore = indirizzo IP];
  - NS (*Name Server*): [nome = dominio] e [valore = hostname del DNS autoritativo che sa come ottenere gli indirizzi IP];
  - CNAME (*Canonical Name*): [nome = alias host] e [valore = nome canonico];
  - MX (*Mail Server*): [nome = alias mail server] e [valore = nome canonico del mail server];
- TTL (*Time to live*) – tempo residuo di vita di un record e determina quando una risorsa vada rimossa dalla cache.

In genere, i record sono inseriti manualmente, ma sono disponibili meccanismi di aggiornamento del database ogni volta che il name server viene a conoscenza di una corrispondenza tra dominio e indirizzo IP.

### Messaggi

Il DNS prevede un **unico formato dei messaggi**, sia per le richieste (*query*) che per le risposte. La semantica dei campi dei messaggi DNS è la seguente.

- I primi 12 byte rappresentano la *sezione di intestazione*, che a sua volta contiene un certo numero di campi. Il primo è un numero di 16 bit che identifica la richiesta. Troviamo poi il campo flag; il primo di questi, il bit di richiesta/risposta, indica se il messaggio è una richiesta (0) o una risposta (1). Un ulteriore bit, chiamato di **richiesta di ricorsione** viene impostato quando un client desidera che il DNS server effettui ricorsione quando non dispone del record. Si imposta un campo da 1 bit di ricorsione disponibile (*recursion-available*) all'interno di una risposta se il DNS server supporta la ricorsione. Nell'intestazione troviamo anche quattro campi il cui nome comincia con “numero di”, che indicano il numero di occorrenze delle quattro sezioni di tipo dati che seguono l'intestazione.
- La *sezione delle domande* contiene informazioni sulle richieste che stanno per essere effettuate. Inoltre, include (1) un campo nome con il nome che sta per essere richiesto e (2) un campo tipo che indica il tipo della domanda sul nome.
- In una risposta proveniente da DNS server, la *sezione delle risposte* contiene i record di risorsa relativi al nome originariamente richiesto.
- La *sezione autoritativa* contiene i record di altri server autoritativi.
- La *sezione aggiuntiva* racchiude altri record utili.

| Identification  |  | Flags                    |  |
|---|--|--------------------------|--|
| Number of questions   |  | Number of answer RRs     | 12 bytes                                   |
| Number of authority RRs   |  | Number of additional RRs |  |
| Questions<br>(variable number of questions)                     |  |                          | Name, type fields for a query              |
| Answers<br>(variable number of resource records)                |  |                          | RRs in response to query                   |
| Authority<br>(variable number of resource records)              |  |                          | Records for authoritative servers          |
| Additional information<br>(variable number of resource records) |  |                          | Additional "helpful" info that may be used |

### Inserimento di record nel database DNS

Supponiamo di aver appena fondato e avviato una società chiamata Network Utopia. La prima cosa che desideriamo fare è registrare il nome di dominio **networkutopia.com** presso un ente di registrazione (*registrar*). Un **registrar** è un'azienda che verifica l'unicità del nome di dominio e lo inserisce nel database DNS.

Quando registriamo il nome di dominio presso un registrar, dobbiamo fornirgli anche i nomi e gli indirizzi IP dei nostri DNS server autoritativi primario e secondario. Supponiamo che i nomi e gli indirizzi IP siano **dns1.networkutopia.com**, **dns2.networkutopia.com**, **212.2.212.1** e **212.212.212.2**. Per ciascuno di questi due server autoritativi l'ente si accernerà dell'inserimento di un record di tipo NS e di tipo A nei TLD server relativi al suffisso com. Più nello specifico, per il server autoritativo primario di **networkutopia.com** il registrar inserirebbe nel sistema DNS i due seguenti record di risorsa:

(networkutopia.com, dns1.networkutopia.com, NS)

(dns1.networkutopia.com, 212.212.212.1, A)

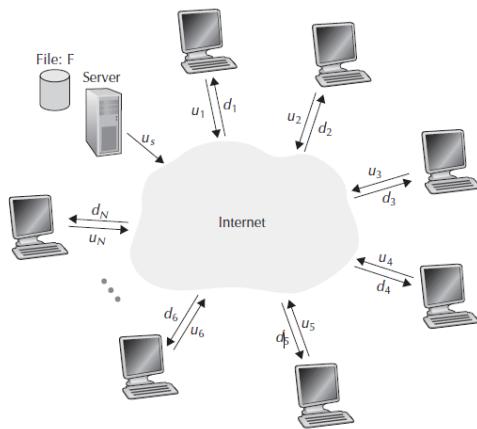
## Distribuzione di file P2P

Nell'architettura peer-to-peer (P2P) ci sono coppie di host connessi in modo intermittente, chiamati *peer*, che comunicano direttamente l'uno con l'altro. I peer non appartengono ai fornitori dei servizi, ma sono computer fissi e portatili controllati dagli utenti.

Esaminiamo la distribuzione di un file voluminoso da un singolo server a un gran numero di host, chiamati peer. In una distribuzione di file client-server, il server deve inviare una copia del file a ciascun peer, ponendo un enorme fardello sul server e consumandone un'elevata quantità di banda. In una distribuzione di file con P2P ciascun peer può ridistribuire agli altri qualsiasi porzione del file abbia ricevuto, aiutando in questo modo il server nel processo di distribuzione.

### Scalabilità dell'architettura P2P

Per confrontare le architetture client-server e P2P e illustrare la scalabilità intrinseca di quest'ultima, considereremo ora un semplice modello quantitativo per la distribuzione di file a un insieme fissato di peer per entrambe le tipologie di architettura.



I server e i peer sono collegati a Internet con collegamenti di accesso:

- $u_s$  è la banda di upload del collegamento di accesso del server;
- $u_i$  è la banda di upload del collegamento di accesso dell' $i$ -esimo peer;
- $d_i$  è la banda di download del collegamento di accesso dell' $i$ -esimo peer;
- $F$  è la dimensione del file da distribuire (in bit);
- $N$  è il numero di peer che vuole una copia del file.

Chiamiamo **tempo di distribuzione** il tempo richiesto perché tutti gli  $N$  peer ottengano una copia del file.

Nell'architettura client-server nessuno dei peer aiuta nella distribuzione del file. Facciamo le seguenti osservazioni.

- Il server deve trasmettere una copia del file a ciascuno degli  $N$  peer, cioè  $NF$  bit. Dato che la banda di upload del server è  $u_s$ , il tempo per distribuire il file deve essere almeno  $NF/u_s$ .
- Sia  $d_{\min}$  la banda di download del peer avente il valore più basso, cioè  $d_{\min} = \min\{d_1, d_2, \dots, d_N\}$ . Il peer con la banda di download più bassa non può ricevere tutti

gli  $F$  bit del file in meno di  $F/d_{min}$  secondi. Quindi, il tempo minimo di distribuzione è almeno  $F/d_{min}$ .

Mettendo assieme queste due osservazioni, otteniamo che:

$$D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\}$$

Consideriamo questo limite inferiore come il tempo di distribuzione effettivo nell'architettura client-server. Quindi, il tempo di distribuzione aumenta linearmente con il numero  $N$  di peer.

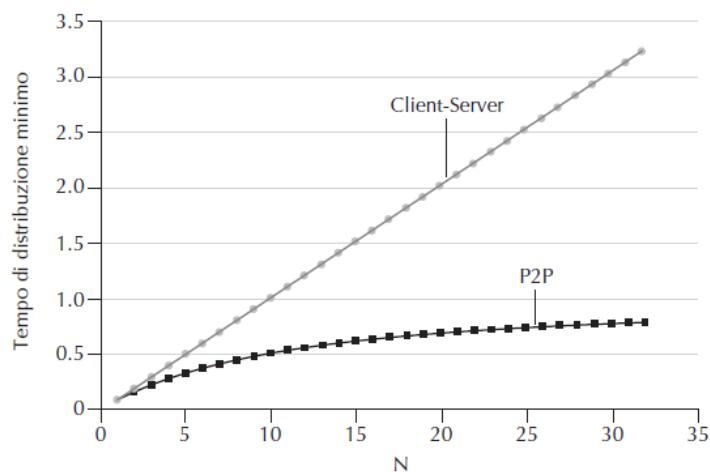
Facciamo adesso un'analogia **analisi per l'architettura P2P**, nella quale ciascun peer assiste il server nella distribuzione del file. In particolare, quando un peer riceve alcuni dati del file, può usare la propria capacità di upload per re-distribuire i dati agli altri peer. Facciamo prima delle osservazioni.

- All'inizio della distribuzione solo il server dispone del file. Per trasmetterlo all'interno della comunità dei peer il server deve inviare ciascun bit del file almeno una volta nel collegamento di accesso. Quindi, il minimo tempo di distribuzione è  $F/u_s$ .
- Come per l'architettura client-server, il peer con la velocità di download più bassa non può ottenere tutti i bit del file in meno di  $F/d_{min}$  secondi. Quindi, il tempo minimo di distribuzione è almeno  $F/d_{min}$ .
- Infine, si osservi che la capacità totale di upload del sistema nel suo complesso è uguale alla velocità di upload del server più quella di ciascun peer, cioè  $u_{tot} = u_s + u_1 + \dots + u_N$ . Il sistema deve consegnare (upload)  $F$  bit a ciascuno degli  $N$  peer, consegnando quindi un totale di  $NF$  bit. Ciò non può essere fatto a una velocità più grande di  $u_{tot}$ . Quindi, il tempo di distribuzione minimo è almeno:  $NF/u_{tot}$ .

Otteniamo così il tempo di distribuzione minimo per l'architettura P2P:

$$D_{P2P} \geq \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_i u_i}\right\}$$

Anche questa volta, consideriamo il limite inferiore come il tempo di distribuzione minimo.



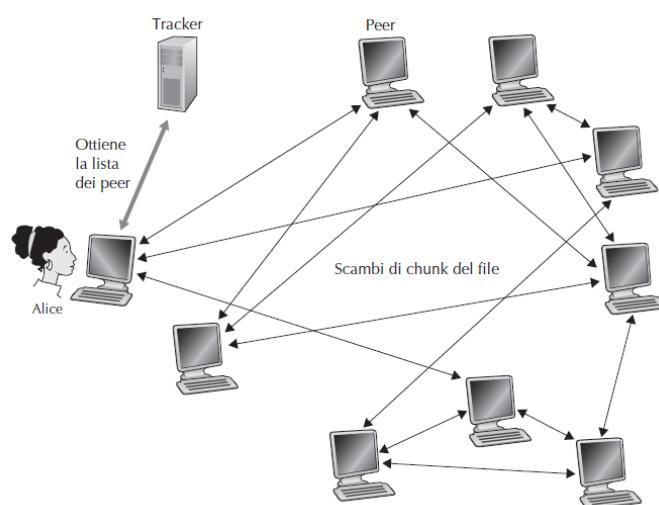
## BitTorrent

**BitTorrent** è un diffuso protocollo P2P per la distribuzione di file. Nel gergo di BitTorrent l'insieme di tutti i peer che partecipano alla distribuzione di un particolare file è chiamato **torrent** (*torrente*). I peer di un torrent scaricano **chunk** (*parti*) del file di uguale dimensione, con una dimensione tipica di 256 kbyte.

Quando un peer entra a far parte di un torrent per la prima volta, non ha chunk del file. Col passare del tempo accumula sempre più parti che, mentre scarica, invia agli altri peer. Una volta che un peer ha acquisito l'intero file, può (egoisticamente) lasciare il torrent o (altruisticamente) rimanere nel torrent e continuare a inviare chunk agli altri peer.

Ciascun torrent ha un nodo di infrastruttura chiamato **tracker**. Quando un peer entra a far parte di un torrent, si registra presso il tracker e periodicamente lo informa che è ancora nel torrent. In questo modo, il tracker tiene traccia dei peer che stanno partecipando al torrent.

Quando un nuovo peer (nell'esempio Alice) entra a far parte di un torrent, il tracker seleziona in modo casuale un sottoinsieme di peer (diciamo 50) dall'insieme dei peer che stanno partecipando a quel torrent, e invia l'indirizzo IP di questi 50 peer ad Alice. Avendo la lista dei peer, Alice cerca di stabilire delle connessioni TCP contemporanee con tutti i peer della lista. Chiamiamo i peer con i quali Alice riesce a stabilire una connessione TCP "peer vicini" (**neighboring peer**). I "peer vicini" a un dato peer cambiano nel tempo.



In un dato istante, Alice avrà un sottoinsieme dei chunk del file e saprà quali chunk hanno i suoi vicini. Con queste informazioni, Alice deve prendere due importanti decisioni: in primo luogo quali chunk deve richiedere per primi ai suoi vicini e, secondariamente, a quali vicini dovrebbe inviare i chunk a lei richiesti. Nel decidere quali chunk richiedere, Alice adotta la tecnica del **rarest first** ("il più raro per primo"). L'idea è determinare quali sono i chunk più rari tra quelli che ancora le mancano, cioè i chunk con il minor numero di copie ripetute tra i suoi vicini, e richiederli per primi.

Per determinare a quali richieste Alice debba rispondere, BitTorrent usa un intelligente algoritmo di trading. L'idea di base è che Alice attribuisca priorità ai vicini che le stanno inviando dati in questo momento alla velocità più alta. Nel gergo di BitTorrent, questi peer

vengono detti ***unchoked*** (“non soffocati” o “non limitati”). Aspetto importante è che ogni 30 secondi sceglie casualmente un vicino in più e gli invia dei chunk. Chiamiamo Bob il peer scelto casualmente. Nel gergo di BitTorrent Bob viene detto ***optimistically unchoked***.

Se i due peer sono soddisfatti degli scambi, l’uno metterà l’altro nella propria lista degli ***unchoked*** e continueranno a effettuare scambi tra loro finché uno non trova un partner migliore.

Tutti gli altri peer, cioè che non ricevono alcun chunk da Alice, sono detti ***choked***. Il meccanismo di incentivazione degli scambi descritto precedentemente viene spesso chiamato ***tit-for-tat*** (“pan per focaccia”).

### Video su Internet

Un **video** è una sequenza di immagini, visualizzate tipicamente a tasso costante di, per esempio, 24 o 30 immagini al secondo. Un’**immagine** non compressa e codificata digitalmente consiste di un array di pixel ognuno dei quali codificato con un numero di bit per rappresentare *luminanza* e *crominanza*. I video possono poi essere compressi in modo da raggiungere un compromesso tra qualità del video e bit rate.

La caratteristica più saliente del video è l’elevato tasso con cui è necessario inviare i bit sulla rete (*bit rate*). La misura in assoluto più importante per lo streaming video è il throughput medio, il cui valore deve essere almeno pari a quello del bit rate del video per avere una riproduzione continua.

La compressione può essere usata anche per creare **versioni multiple** dello stesso video, a livelli di qualità diversi.

### Streaming HTTP e DASH

Nello **streaming HTTP** il video viene semplicemente memorizzato in un server HTTP come un file ordinario con un URL specifico. Quando un utente vuole vedere un video, il client stabilisce una connessione TCP con il server e invia una richiesta GET HTTP per il suo URL. Il server invia il file video, all’interno di un messaggio di risposta HTTP, più velocemente possibile dati il traffico e i protocolli di rete. Sul lato client i byte vengono memorizzati in un buffer dell’applicazione client. Quando il numero di byte nel buffer supera una soglia fissata, l’applicazione client inizia la riproduzione: periodicamente prende i frame video dal buffer, li decomprime e li visualizza all’utente. Quindi, l’applicazione di video streaming consiste nel visualizzare i frame mentre li riceve, memorizzando nel buffer gli ultimi.

C’è un grande svantaggio: i client ricevono la stessa versione codificata del video, nonostante abbiano disponibile una larghezza di banda che può variare sensibilmente da un caso all’altro e nel tempo. Per superare il problema è stato sviluppato un nuovo tipo di streaming basato su HTTP, chiamato **streaming dinamico adattativo su HTTP (DASH, Dynamic Adaptive Streaming over HTTP)**. In DASH, i video vengono codificati in diverse versioni, ognuna avendo un bit rate differente e quindi un differente livello di qualità.

Quando la banda disponibile è elevata, il client seleziona automaticamente i blocchi da una versione con alto bit rate, mentre quando la banda disponibile è poca, seleziona una versione a basso bit rate.

Con DASH, i video nelle varie versioni sono memorizzati nel server http, ognuno a un URL diverso. Il server HTTP ha anche un file di descrizione (detto anche **manifest**), che per ogni versione fornisce il rispettivo URL insieme al bit rate.

### Reti per la distribuzione di contenuti

Forse l'approccio più diretto per le aziende di streaming video in Internet è costruire un unico enorme data center, memorizzare in esso tutti i video e mandarli in streaming direttamente. Tale approccio però presenta tre grandi problemi:

- se il client è lontano dal data center, i pacchetti dal server al client devono percorrere un lungo cammino passando per molti ISP, magari in continenti diversi. Se uno dei collegamenti ha un throughput minore del tasso di consumo del video, anche il throughput totale end-to-end lo sarà, causando all'utente continui e fastidiosi fermi immagine;
- un video molto popolare verrà inviato molte volte sullo stesso collegamento, non solo sprecando banda, ma costringendo anche l'azienda a pagare tutte le volte il suo ISP (collegato al data center) per inviare gli stessi dati;
- un singolo data center rappresenta un singolo punto di rottura: se il data center o il collegamento da esso a Internet si interrompe, l'azienda non sarà più in grado di distribuire alcun video.

Per superare queste problematiche, quasi tutte le maggiori aziende di video streaming usano le **CDN** (*Content Distribution Networks*). Una CDN gestisce server distribuiti in molti posti diversi, memorizza copie dei video e di altri contenuti web nei server e cerca di dirigere le richieste degli utenti al punto della CDN in grado di offrire il servizio migliore. La CDN può essere **privata** (cioè proprietaria del fornitore di contenuti) o **di terze parti** (cioè che distribuisce contenuti per conto di molti fornitori di contenuti).

Le CDN generalmente adottano una delle due politiche di dislocazione dei server che seguono:

- **Enter deep:** entra profondamente nelle reti di accesso degli ISP installando gruppi di server, detti anche cluster, negli ISP di accesso sparsi in tutto il mondo. L'obiettivo è quello di essere vicini agli utenti finali in modo da migliorare il ritardo percepito dall'utente e il throughput, diminuendo il numero da collegamenti e router tra l'utente finale e il cluster CDN da cui riceve i contenuti. Tale approccio, altamente distribuito, pone però grandi sfide nella manutenzione e gestione dei cluster.
- **Bring home:** porta in casa l'ISP costruendo grandi cluster in pochi punti chiave e interconnetterli usando una rete privata ad alta velocità. Invece di entrare negli ISP di accesso, queste CDN pongono ogni cluster in un luogo vicino ai PoP di molti ISP di livello 1. Tale approccio presenta meno problemi di gestione e manutenzione, ma spesso anche una minore qualità di servizio.

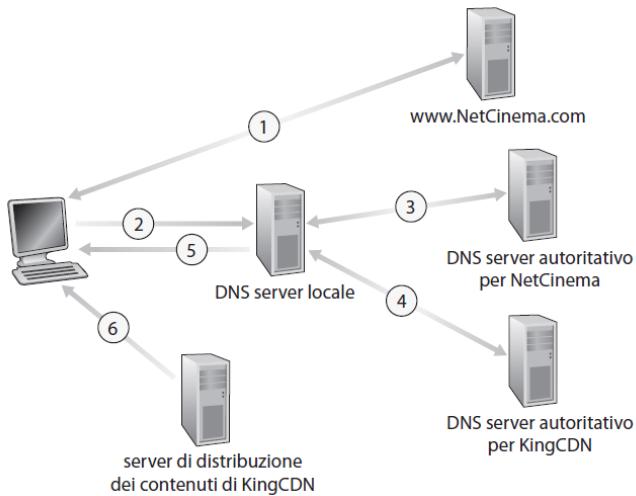
Molte CDN non spingono (push) i loro video verso i cluster, ma usano una semplice strategia per cui se un client richiede un video a un cluster che non lo ha memorizzato, il cluster recupera il video da un archivio centrale o da un altro cluster e ne memorizza localmente una copia mentre lo manda in streaming al client. I video meno richiesti vengono rimossi quando lo spazio disponibile si esaurisce.

### Come funziona la CDN

Quando un browser nell'host di un utente chiede il recupero di uno specifico video identificato da un URL, la CDN deve intercettare la richiesta in modo da poter (1) determinare il cluster di server più appropriato per quel cliente a quell'istante e (2) dirigere la richiesta del client a uno dei server di quel cluster.

Molte **CDN sfruttano il servizio DNS** per intercettare e ridirigere le richieste. Consideriamo il seguente esempio: supponiamo che un fornitore di contenuti, NetCinema, impieghi una CDN di terza parte, KingCDN, per distribuire i suoi video. A ogni video sulla pagina web di NetCinema viene assegnato un URL che include la stringa “video” e un identificatore univoco per tale video. Vengono quindi compiuti sei passi:

1. L'utente visita la pagina web di NetCinema.
2. Quando l'utente seleziona il link <http://video.netcinema.com/6Y7B23V>, il suo host invia una interrogazione DNS a `video.netcinema.com`.
3. Il server locale DNS (LDNS) dell'utente invia l'interrogazione DNS a un DNS server autoritativo per NetCinema, che osserva la stringa “video” nel nome dell'host `video.netcinema.com`. Il DNS server autoritativo per NetCinema per passare l'interrogazione DNS a KingCDN, invece di restituire un indirizzo IP, restituisce all'LDNS il nome di un host nel dominio di KingCDN quale, per esempio, `a1105.kingcdn.com`.
4. Da questo momento, l'interrogazione DNS entra nell'infrastruttura DNS privata di KingCDN. Lo LSDN dell'utente invia quindi una seconda interrogazione, per `a1105.kingcdn.com`, e infine il sistema DNS di KingCDN restituisce gli indirizzi IP di un server di KingCDN all'LDNS. È quindi qui, all'interno del sistema DNS di KingCDN, che viene specificato il server CDN da cui il client riceverà il contenuto.
5. Lo LDNS inoltra l'indirizzo IP del nodo CDN che fornirà il contenuto all'host dell'utente.
6. Il client, una volta ricevuto l'indirizzo IP del server di KingCDN, stabilisce una connessione TCP diretta con il server a quell'indirizzo IP e gli invia una richiesta GET per il video. Nel caso venga impiegato DASH, il server innanzitutto invierà al client il file manifesto con una lista di URL, uno per ogni versione del video, e il client selezionerà in modo dinamico i blocchi da versioni differenti.



Il cuore dell'installazione di una CDN è la **strategia di selezione del cluster**: un meccanismo per dirigere dinamicamente i client a un cluster di server o a un data center della CDN. Vediamo due dei possibili approcci.

- assegnare a un client il **cluster geograficamente più vicino**. Usando un database commerciale di geo-localizzazione, gli indirizzi IP degli LDNS vengono associati a un luogo geografico. Potrebbe non andare bene, perché il cluster più vicino geograficamente potrebbe essere diverso da quello più vicino dal punto di vista del percorso di rete. Inoltre, alcuni utenti sono configurati per usare LDNS remoti; in tal caso, l'LDNS potrebbe trovarsi lontano dal client. Infine, non tiene conto delle variazioni temporali del ritardo di rete e della banda disponibile.
- assegnare a un client il cluster basandosi sulle condizioni di traffico correnti, effettuando **misure in tempo reale** delle prestazioni di ritardo e perdita tra i loro cluster e i loro client. Tuttavia, molti LDNS sono configurati per non rispondere a tali richieste.

## Livello di trasporto

Un protocollo a livello di trasporto mette a disposizione una **comunicazione logica** tra processi applicativi di host differenti. Per “comunicazione logica” si intende, dal punto di vista dell’applicazione, che tutto proceda come se gli host che eseguono i *processi* fossero direttamente connessi.

I protocolli a livello di trasporto sono implementati nei sistemi periferici, ma non nei router della rete. **Lato mittente**, il livello di trasporto converte i messaggi che riceve da un processo applicativo in pacchetti a livello di trasporto, noti come **segmenti**. Questo avviene spezzando (se necessario) i messaggi applicativi in parti più piccole e aggiungendo a ciascuna di esse un’intestazione di trasporto per creare un segmento. Fatto questo, il livello di trasporto passa il segmento al livello di rete, dove viene incapsulato all’interno di un pacchetto a livello di rete (*datagramma*) e inviato a destinazione. **Lato ricevente**, il livello di rete estrae il segmento dal datagramma e lo passa al livello superiore, quello di trasporto. Quest’ultimo elabora il segmento ricevuto, rendendo disponibili all’applicazione destinataria i dati del segmento.

Ricordiamo che il livello di trasporto è collocato esattamente sopra quello di rete nella pila di protocolli. Mentre un protocollo a livello di trasporto mette a disposizione una comunicazione logica tra processi che vengono eseguiti su host diversi, un protocollo a livello di rete fornisce comunicazione logica tra host. La distinzione è sottile, ma importante.

I livelli di trasporto che Internet mette a disposizione sono:

- **UDP (User Datagram Protocol)**, che fornisce un **servizio non affidabile** (*best effort*) e non orientato alla connessione;
- **TCP (Transmission Control Protocol)**, che offre un **servizio affidabile** (e altri servizi come il controllo di congestione) e orientato alla connessione.

### Multiplexing e demultiplexing

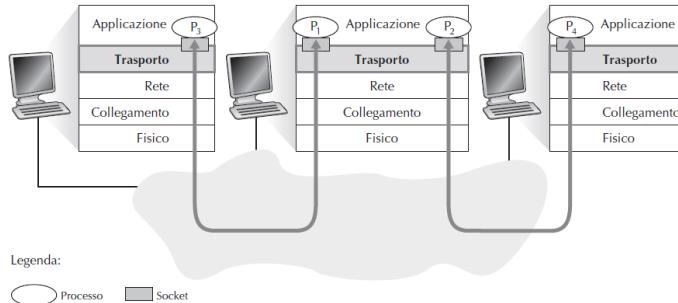
Le entità di trasporto gestiscono la conversione tra messaggi e segmenti e la consegna dei messaggi ai socket nel sistema. Si definiscono:

- **multiplazione**: i messaggi generati dai diversi socket attivi nel sistema vengono eventualmente scomposti, incapsulati ognuno in un segmento e consegnati al livello di rete. Viene effettuata dal mittente;
- **demultiplazione**: i segmenti ricevuti dal livello di rete vengono scapsulati in messaggi, eventualmente ricomposti e consegnati al socket corretto. Viene effettuata dal destinatario.

Il livello di trasporto estende perciò il servizio di consegna di *datagrammi* tra host a livello di rete ad un servizio di consegna di *messaggi* tra applicazioni per il livello applicativo. Il meccanismo con il quale viene effettuata la multiplazione o la demultiplazione dipende dal protocollo di trasporto utilizzato per la comunicazione. In particolare:

- **connection-less** (UDP): socket identificato da una coppia [*dest IP, dest Port*];
- **connection-oriented** (TCP): socket identificato da 4-tupla [*source IP, source Port, dest IP, dest Port*].

In entrambi i casi, gli indirizzi IP servono a identificare un socket all'interno della rete, ovvero a quale host appartiene, per effettuare le operazioni di instradamento del livello di rete. Vengono usati all'interno del sistema operativo, ed in particolare dal livello di trasporto, per determinare a quale socket destinare un segmento, ovvero per l'operazione di demultiplicazione vera e propria.



In **UDP** (*connection-less*) ogni **socket** è identificata dalla **coppia [IP, port]**. Viene creata associandole un numero di porta tra quelli disponibili (in maniera casuale o esplicita). Si ha che:

- il **socket mittente** invia segmenti UDP con  $[source\ IP, source\ Port] = [system\ IP, socket\ Port]$ ;
- il **socket destinatario** riceve segmenti UDP con  $[dest\ IP, dest\ Port] = [mittente\ IP, socket\ Port]$ .

L'**host ricevente** utilizza la coppia  $[dest\ IP, dest\ Port]$  del segmento UDP ricevuto per effettuare la demultiplicazione. In particolare, il segmento UDP viene consegnato al socket UDP con numero di porta associato uguale alla porta di destinazione.

In **TCP** (*connection-oriented*) ogni **socket** è identificata dalla **4-tupla**  $[source\ IP, source\ Port, dest\ IP, dest\ Port]$ . Viene creata associandole un numero di porta tra quelli disponibili, in maniera casuale o esplicita. Successivamente:

- il **client** si connette alla socket TCP del server specificando  $[IP\ server, Porta\ server]$ . Il server accetta la connessione se la porta indicata è quella attesa; il socket TCP del client è quindi operativo.
- il **server** attende, e accetta, connessioni da socket TCP di client sulla porta associata al proprio socket. Una volta accettata una connessione, il socket TCP del server è operativo.

L'**host ricevente** utilizza la 4-tupla  $[source\ IP, source\ Port, dest\ IP, dest\ Port]$  del segmento TCP ricevuto per effettuare la demultiplicazione. In particolare, il segmento TCP arriva nel sistema solamente se il numero di porta di sorgente è uguale a quello stabilito in fase di connessione, e viene consegnato alla socket TCP con numero di porta associato uguale alla porta di destinazione.

Si noti che un server possiede una socket TCP per ogni connessione client. Inoltre, possono esserci client che possiedono molteplici connessioni con il server. Per tali motivi una socket TCP viene identificata dall'intera 4-tupla di indirizzi IP e numeri di porta.

## UDP

Il protocollo UDP realizza un trasferimento dati connection-less non affidabile. Ai servizi offerti dal livello di rete aggiunge solamente:

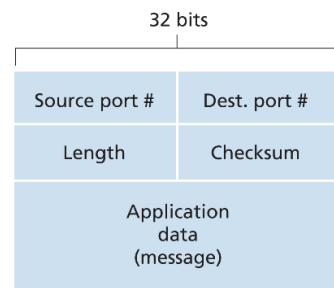
- funzione di multiplazione dei messaggi e demultiplazione dei segmenti;
- controllo di integrità tramite **checksum** dei segmenti.

La semplicità del protocollo permette di ottenere una trasmissione di tipo “best effort”, in quanto:

- possiede un overhead minimo, data l'assenza di una connessione da instaurare (e quindi del relativo stato da mantenere) e di un header della PDU (*Protocol Data Unit*) di dimensioni ridotte;
- può trasmettere messaggi alla massima velocità consentita dal canale di comunicazione utilizzato, data l'assenza di controlli sullo stato di congestione della rete e sul flusso dati.

Un **segmento del protocollo UDP** possiede il seguente formato:

- **Header:** composto da 4 campi a 2 byte
  - *Source port*: numero di porta della sorgente;
  - *Destination port*: numero di porta di destinazione;
  - *Length*: lunghezza in byte del segmento;
  - *Checksum*: bit utilizzati per la rilevazione degli errori.
- **Corpo:** contiene un messaggio.



Il **checksum** è una tecnica che permette di rilevare errori nei bit all'interno di un segmento. Viene generato sommando bit a bit sequenze di 16 bit dal segmento e aggiungendo il riporto generato da tale somma. Il meccanismo di controllo viene implementato come segue:

- il mittente genera il checksum e lo inserisce nel relativo campo del segmento UDP;
- il ricevente rigenera il checksum e lo confronta con quello all'interno del segmento UDP. Se i checksum sono identici, non viene rilevato alcun errore, altrimenti l'errore viene rilevato.

Si noti che il meccanismo di checksum non è completamente affidabile, in quanto possono esserci casi in cui vengono rilevati sia falsi positivi che falsi negativi, sebbene le probabilità che ciò si verifichi siano ridotte.

|            |   |  |
|------------|---|--|
| wraparound | 1 | 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1  |
|            |   | <hr/>  |
|            |   | sum      1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0<br>checksum 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 |

## Undirectional Reliable Data Transfer Protocol (RDT)

Il **protocollo TCP** realizza un trasferimento dati **connection-oriented affidabile**. Utilizza il canale di comunicazione inaffidabile fornito dal livello di rete per fornire un canale di comunicazione affidabile al livello applicativo.

La realizzazione di un canale affidabile è influenzata dalle caratteristiche del canale sottostante sulla quale si basa la comunicazione. Maggiore è l'inaffidabilità di quest'ultimo e maggiore è la complessità del protocollo che realizza il canale affidabile.

I principali fattori che rendono inaffidabile un canale sono:

- errore sui bit: il valore dei bit viene alterato durante il trasferimento delle informazioni.
- perdita di pacchetti: alcuni pacchetti possono essere persi durante la trasmissione.

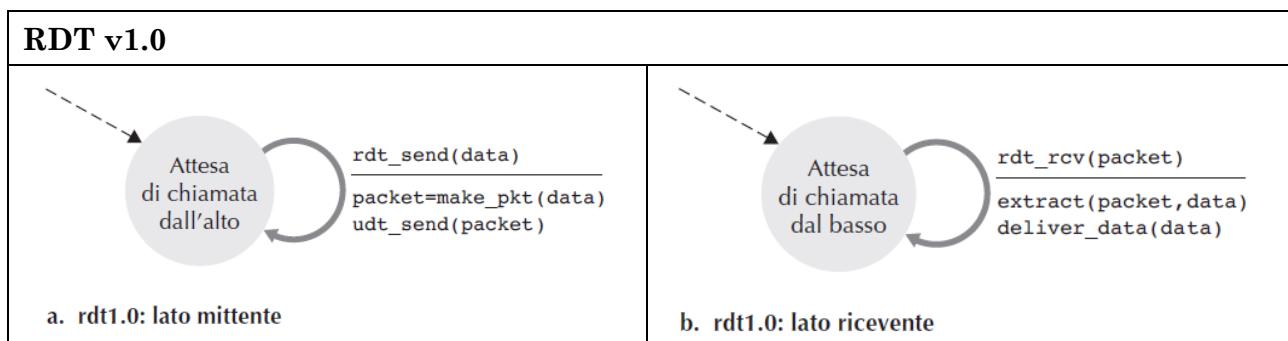
Sia (non esiste) **RDT** (*Reliable Data Transfer*) un protocollo di trasmissione affidabile su canale inaffidabile che espone la seguente interfaccia verso i livelli adiacenti:

- `rdt_send()`: invocata dal livello applicativo quando sono disponibili nuovi dati.
- `udt_sent()`: invocata dal protocollo RDT per trasferire pacchetti sul canale di rete.
- `rdt_rcv()`: invocata dal livello di rete quando arriva un pacchetto dal canale di rete.
- `deliver_data()`: invocata dal protocollo RDT per consegnare dati al livello applicativo.

### RDT v1: canale affidabile

Si supponga che il canale di rete sia completamente affidabile. In tal caso, il protocollo RDT si deve occupare solo di:

- (mittente) incapsulare i dati in pacchetti e inviarli sul canale;
- (destinatario) estrarre i dati dai pacchetti e consegnarli alle applicazioni.



### RDT v2: errore sui bit

Si supponga ora che il canale di rete possa generare errore sui bit trasmessi. Il protocollo RDT deve implementare meccanismi di:

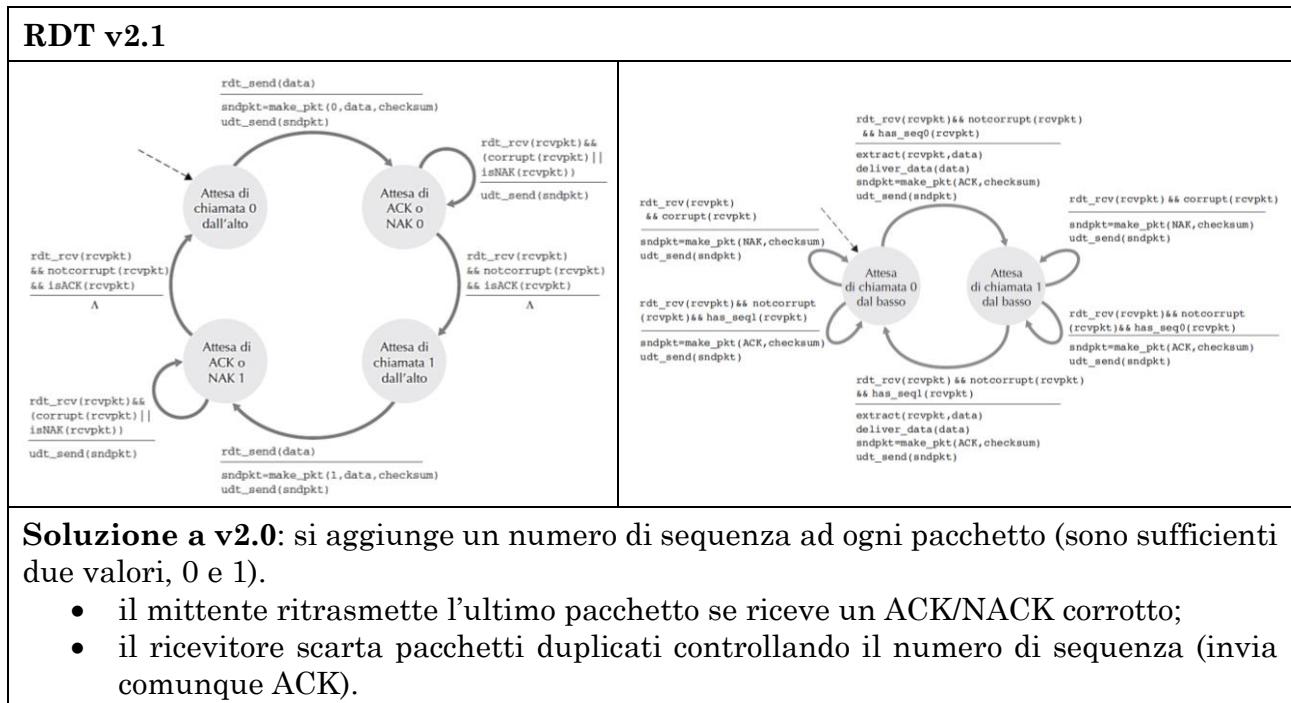
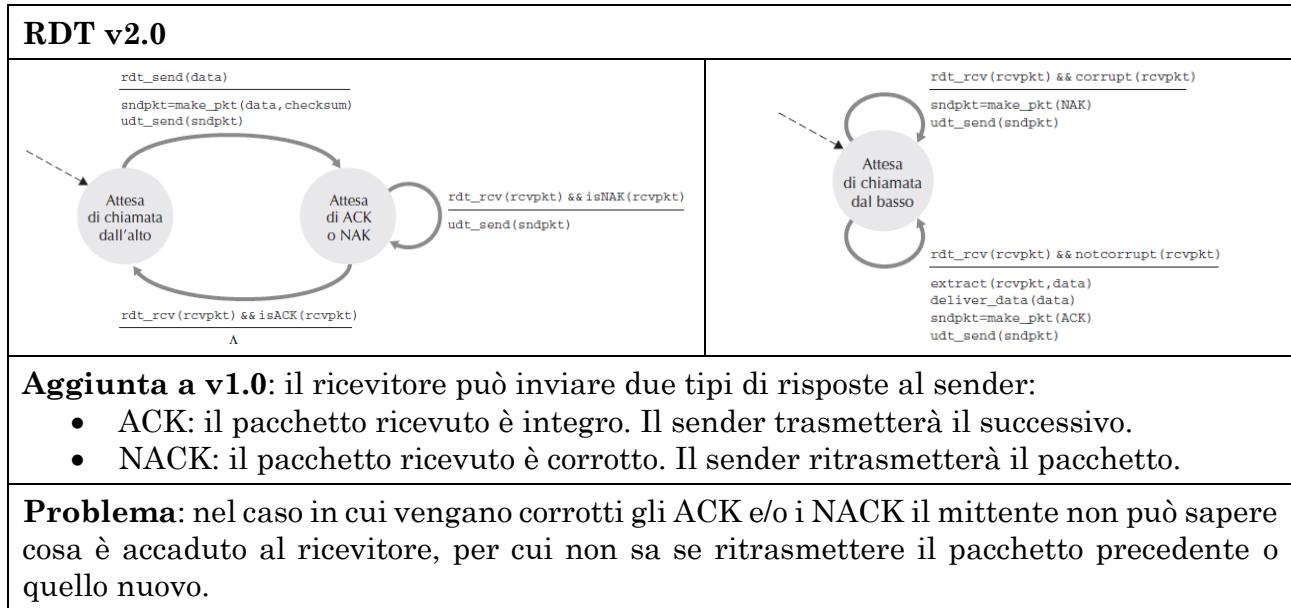
- **rilevazione degli errori**: il destinatario deve essere in grado di rilevare errori nei pacchetti ricevuti. Possono essere utilizzate tecniche come il checksum o basate sulla codifica. Può anche essere possibile applicare tecniche di correzione degli errori.

- **feedback del destinatario:** il destinatario deve potere comunicare al mittente l'esito della ricezione. Ciò viene realizzato tramite risposte di **ACK** o **NACK** inviate dal destinatario al mittente.
- **recupero degli errori/ritrasmissione:** il mittente deve poter ritrasmettere pacchetti ricevuti dal destinatario con errori. È basata sul feedback dal destinatario.

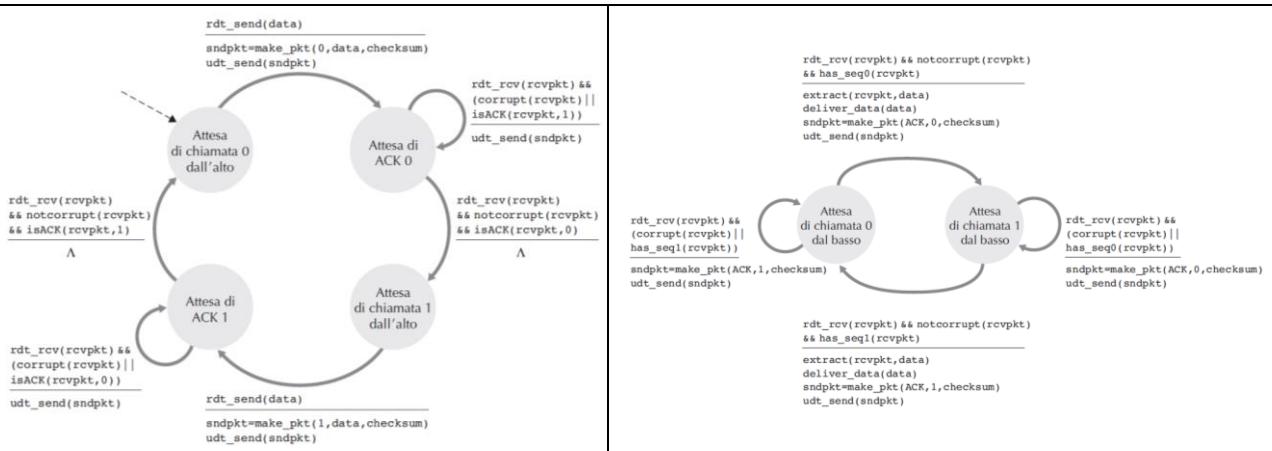
Se indichiamo con  $N$  il numero di trasmissioni per un pacchetto e con  $P$  la probabilità che la ricezione avvenga con errori, possiamo dire che  $N$  si comporta come una geometrica; infatti, la probabilità di avere  $k$  ritrasmissioni di un pacchetto è data da

$$N(k) = p^k \cdot (1 - p)$$

e la sua media è definita da  $E[N] = \frac{1}{1-p}$ .



## RDT v2.2



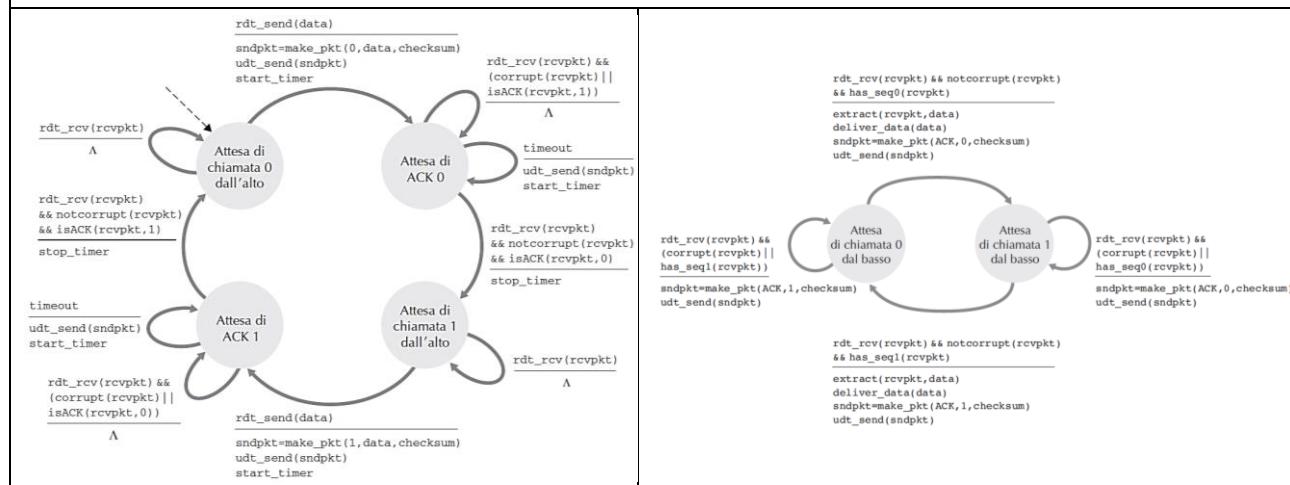
**Miglioramento a v2.1:** il ricevitore invia solamente risposte di ACK, al cui interno viene aggiunto il numero del pacchetto per le quali confermano la ricezione. ACK duplicati equivalgono ad una risposta di NACK per il pacchetto corrente e perciò il sender ritrasmette l'ultimo pacchetto.

## RDT v3: errore sui bit e perdita di pacchetti

Si supponga ora che il canale di rete possa sia generare errore sui bit che perdere pacchetti. Oltre ai meccanismi visti precedentemente, il protocollo RDT deve implementare meccanismi di:

- rilevamento delle perdite di pacchetti:** il mittente deve essere in grado di rilevare la mancata consegna di pacchetti al destinatario. Si utilizzano dei timer per determinare la mancata consegna di un pacchetto, sia questa dovuta alla perdita stessa del pacchetto o alla perdita dell'ACK di risposta. Si noti che il tempo di attesa deve essere almeno pari a un RTT.
- recupero della perdita dei pacchetti/ritrasmissione:** il mittente deve poter ritrasmettere pacchetti la cui ricezione non viene confermata dal destinatario. È basata sulla rilevazione delle perdite.

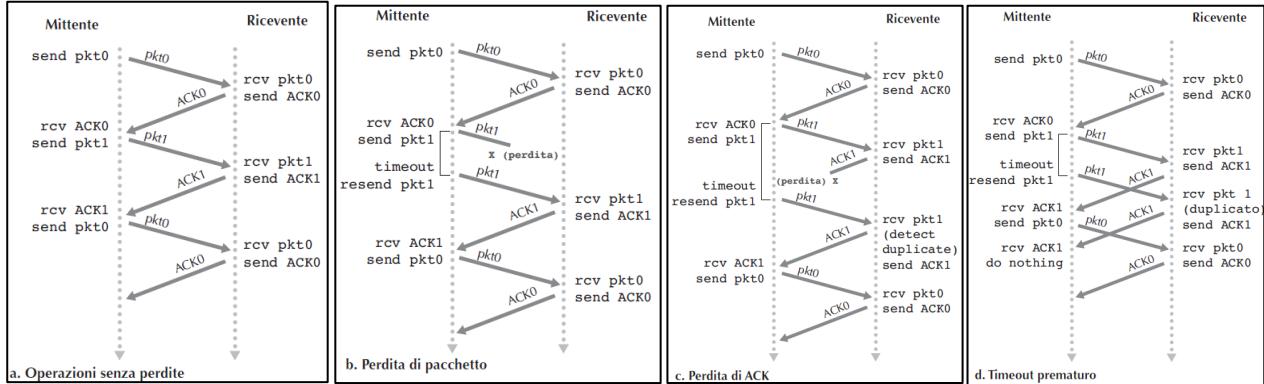
## RDT v3.0



**Aggiunta a v2.2:** viene aggiunto un timer nel mittente per determinare la perdita di pacchetti.

- Il mittente ritrasmette l'ultimo pacchetto se non riceve l'ACK entro la scadenza del timer.
- Nel caso in cui l'ACK arrivi dopo la scadenza del timer, la precedente ritrasmissione ha generato pacchetti duplicati. Tale occorrenza è già gestita dal RDT v2.2 tramite il sequence e ACK number.

Di seguito le varie casistiche.

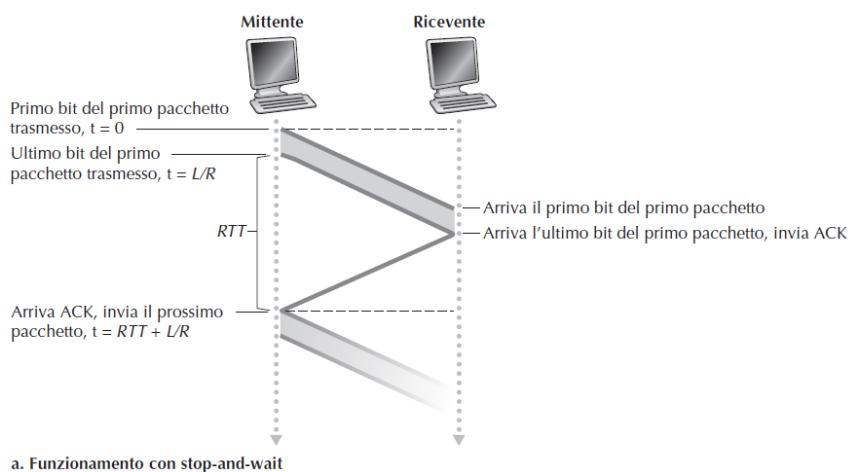


Il protocollo RDT v3.0 è corretto dal punto di vista funzionale, ma difficilmente apprezzabile dal punto di vista prestazionale, in quanto si tratta di un **protocollo stop-and-wait**: il mittente aspetta la risposta del destinatario prima di inviare il pacchetto successivo. Se definiamo l'**utilizzo** del mittente (o del canale) come la frazione di tempo in cui il mittente è stato effettivamente occupato nell'invio di bit sul canale, il protocollo stop-and-wait presenta un triste utilizzo del mittente:

$$U_{\text{mittente}} = \frac{L/R}{RTT+L/R} \rightarrow 0 \text{ se } \frac{L}{R} \rightarrow 0$$

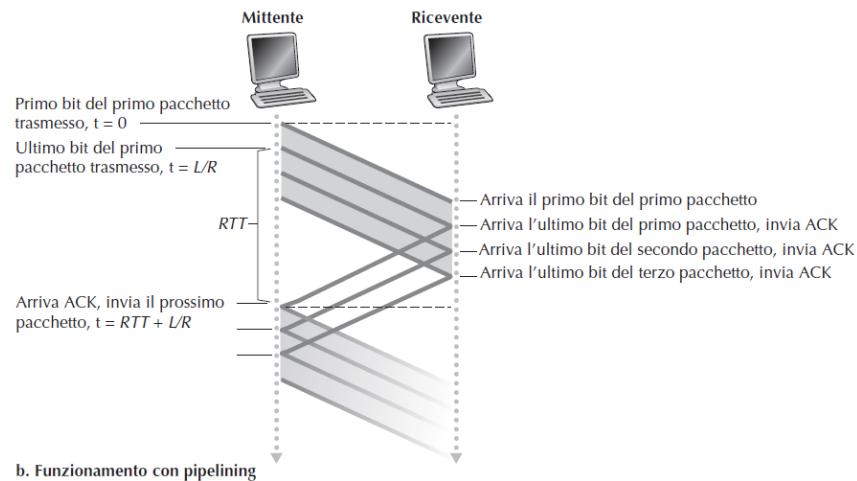
dove:

- $L :=$  dimensione pacchetto
- $R :=$  velocità trasmissione



La soluzione a questo particolare problema è semplice: anziché operare in modalità stop-and-wait, si consente al mittente di inviare più pacchetti senza attendere gli ACK. Questa tecnica è nota come **pipelining**. Per riuscire a realizzare ciò occorre:

- aumentare l'intervallo dei numeri di sequenza utilizzabili (ogni pacchetto in transito deve presentare un numero univoco e ci potrebbero essere più pacchetti in transito ancora in attesa di ACK);
- introdurre dei buffer per il mittente e (forse) per il destinatario.

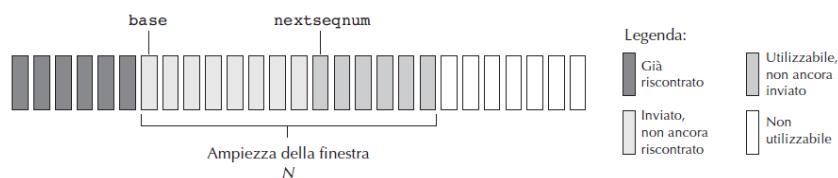


La quantità di numeri di sequenza necessari e i requisiti dei buffer dipendono dal modo in cui il protocollo di trasferimento dati reagisce ai pacchetti smarriti, alterati o troppo in ritardo. Si identificano **Go-Back-N (GBN)** e **Ripetizione selettiva (SR – Selective Repeat)**.

### Go-Back-N

Il **protocollo GBN** realizza trasmissione pipelined affidabile utilizzando il meccanismo degli ACK cumulativi: il mittente può trasmettere più pacchetti senza dover attendere alcun ACK, ma non può avere più di un dato numero massimo consentito  $N$  di pacchetti (se disponibili) in attesa di ACK nella pipeline.

Se definiamo **base** come il numero di sequenza del pacchetto più vecchio che non ha ancora ricevuto un ACK e **nextseqnum** il più piccolo numero di sequenza inutilizzato (ossia il numero di sequenza del prossimo pacchetto da inviare), allora possiamo identificare i seguenti intervalli di numeri di sequenza:



L'intervallo di numeri di sequenza ammissibili per i pacchetti trasmessi, ma che non hanno ancora ricevuto alcun ACK, può essere visto come una finestra di dimensione  $N$  sull'intervallo dei numeri di sequenza. Quando il protocollo è in funzione, questa finestra trasla lungo lo spazio dei numeri di sequenza. Per questo motivo,  $N$  viene spesso chiamato **ampiezza della finestra** (*window-size*) e il protocollo GBN viene detto **protocollo a finestra scorrevole** (*sliding-window protocol*).

L'**header** dei pacchetti contiene un campo a  $k$  bit per indicare il relativo **numero di sequenza**. Ne deriva che i numeri di sequenza rappresentabili sono compresi nell'intervallo  $[0, 2^k - 1]$  e che la grandezza massima della finestra è  $N = 2^k$ .

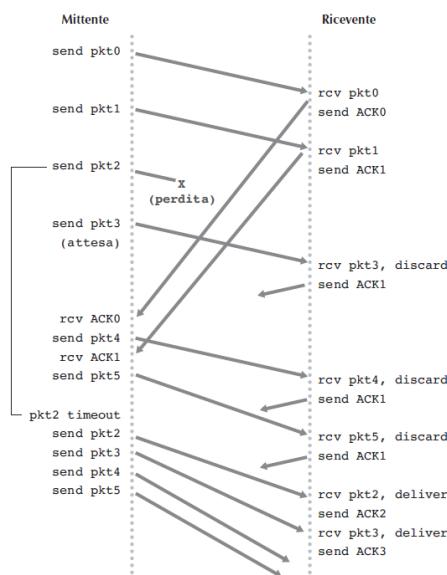
Il *mittente* deve rispondere a tre tipi di evento.

- **Invocazione dall'alto:** quando dall'alto si chiama `rdt_send()`, come prima cosa il mittente controlla se la finestra sia piena, ossia se vi siano  $N$  pacchetti in sospeso senza ACK.
  - Se la finestra non è piena, crea e invia un pacchetto e le variabili vengono aggiornate di conseguenza;
  - altrimenti, il mittente restituisce i dati al livello superiore o li bufferizza.
- **Ricezione di un ACK:** l'ACK del pacchetto con il numero di sequenza  $n$  verrà considerato un **acknowledgment cumulativo**, che indica che tutti i pacchetti con un numero di sequenza minore o uguale a  $n$  sono stato correttamente ricevuti dal destinatario.
- **Evento di timeout:** poiché si usa un contatore per risolvere il problema di pacchetti dati o acknowledgment persi, quando si verifica un timeout il mittente invia nuovamente *tutti* i pacchetti spediti che ancora non hanno ricevuto un ACK. Se si riceve un ACK, ma ci sono ancora pacchetti aggiuntivi trasmessi e non riscontrati, il timer viene fatto ripartire. Se, invece, non ci sono pacchetti in sospeso in attesa di acknowledgment, il contatore viene fermato.

Per quanto riguarda il *destinatario*, esso non ha bisogno di un buffer, in quanto accetta solamente pacchetti ricevuti in ordine e non quelli che giungono fuori sequenza. L'unica parte delle informazioni che il destinatario deve memorizzare è il numero di sequenza del successivo pacchetto nell'ordine (**expectedseqnum**).

Ovviamente, lo svantaggio di eliminare un pacchetto ricevuto correttamente è che la sua ritrasmissione potrebbe andare persa o essere alterata, il che richiederebbe ulteriori ritrasmissioni.

La figura di seguito mostra come opera il protocollo GBN con una finestra di 4 pacchetti.

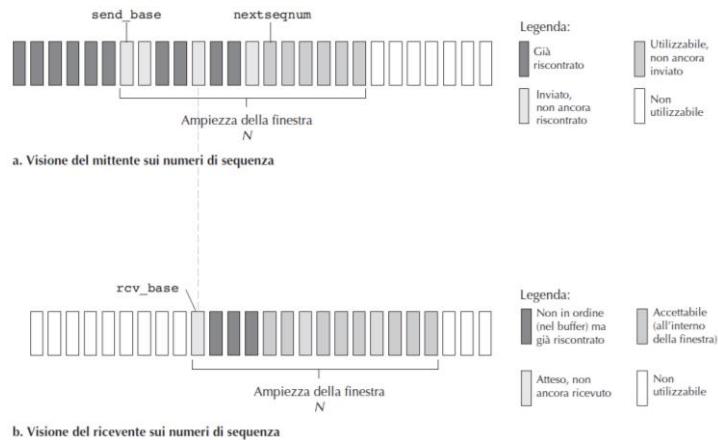


## Selective Repeat

Ci sono scenari in cui GBN ha problemi di prestazioni. In particolare, quando l'ampiezza della finestra e il prodotto tra larghezza di banda e ritardo sono entrambi grandi, nella pipeline si possono trovare numerosi pacchetti. Un errore su un solo pacchetto può pertanto provocare un elevato numero di ritrasmissioni, in molti casi inutili. Al crescere della probabilità di errore sul canale, la pipeline può saturarsi a causa di queste ritrasmissioni non necessarie.

I **protocolli a ripetizione selettiva (SR – Selective Repeat)** evitano le ritrasmissioni non necessarie facendo ritrasmettere al mittente solo quei pacchetti su cui esistono sospetti di errore (ossia, smarrimento o alterazione).

Il destinatario SR invia un riscontro per i pacchetti correttamente ricevuti sia in ordine sia fuori sequenza. Questi vengono memorizzati in un buffer finché non sono stati ricevuti tutti i pacchetti mancanti (ossia quelli con numeri di sequenza più bassi), momento in cui un blocco di pacchetti può essere trasportato in ordine al livello superiore.



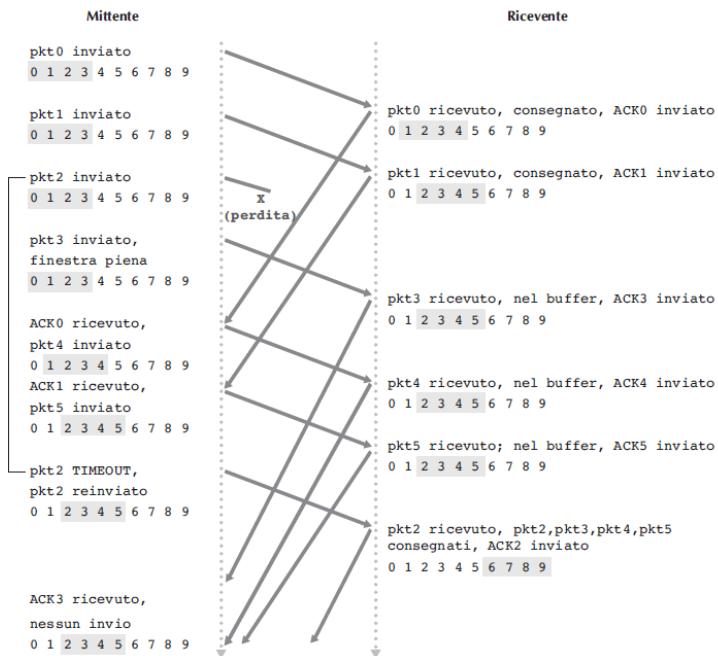
Vediamo eventi e azioni di un *mittente SR*:

- **Dati ricevuti dall'alto:** quando si ricevono dati dall'alto, il mittente SR controlla il successivo numero di sequenza disponibile per il pacchetto:
  - se è all'interno della finestra del mittente, i dati vengono impacchettati e inviati;
  - altrimenti, sono salvati nel buffer o restituiti al livello superiore per una successiva ritrasmissione, come in GBN.
- **Timeout:** vengono usati ancora i contatori per cauterarsi contro la perdita di pacchetti. Ora però ogni pacchetto deve avere un proprio timer logico, dato che al timeout sarà ritrasmesso un solo pacchetto.
- **ACK ricevuto:** se si riceve un ACK, il mittente SR etichetta tale pacchetto come ricevuto, ammesso che sia nella finestra. Se il numero della sequenza è uguale a `send_base`, la base della finestra si muove verso il pacchetto che non ha ricevuto ACK con il più piccolo numero di sequenza. Se la finestra si sposta e ci sono pacchetti non trasmessi con numero di sequenza che ora cade all'interno della finestra, questi vengono trasmessi.

Vediamo eventi e azioni di un *destinatario SR*:

- **Pacchetto con numero di sequenza in  $[rcv\_base, rcvbase+N-1]$  ricevuto correttamente:** il pacchetto ricevuto è all'interno della finestra del ricevente e al mittente viene restituito un pacchetto di ACK selettivo.
  - Se il pacchetto non era già stato ricevuto viene inserito nel buffer.
  - Se presenta un numero di sequenza uguale alla base della finestra di ricezione, allora questo pacchetto e tutti i pacchetti nel buffer aventi numeri consecutivi (a partire da  $rcv\_base$ ) vengono consegnati al livello superiore.
- **Pacchetto con numero di sequenza in  $[rcv\_base-N, rcv\_base-1]$  ricevuto correttamente:** in questo caso si deve generare un ACK, anche se si tratta di un pacchetto che il ricevente ha già riscontrato.
- **Altrimenti** si ignora il pacchetto.

Di seguito il funzionamento di SR.



La mancanza di sincronizzazione tra le finestre del mittente e del destinatario ha conseguenze importanti quando abbiamo a che fare con un intervallo finito di numeri di sequenza. Il risultato importante e da tenere a mente è che la finestra deve avere ampiezza inferiore o uguale alla metà dello spazio dei numeri di sequenza dei protocolli SR ( $N = 2^k - 1 \equiv [\mathcal{N}/2]$ , dove  $\mathcal{N}$  := cardinalità numeri di sequenza).

## TCP

TCP viene detto **orientato alla connessione** in quanto, prima di effettuare lo scambio dei dati, i processi devono effettuare l'handshake, ossia devono inviarsi reciprocamente alcuni segmenti preliminari per stabilire i parametri del successivo trasferimento dati.

Una connessione TCP:

- offre un **servizio full-duplex**: i dati a livello di applicazione possono fluire dal processo A al processo B nello stesso momento in cui fluiscono in direzione opposta;

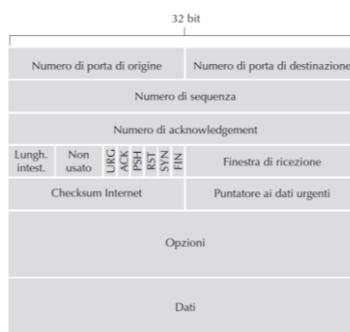
- è **punto a punto**, ossia ha luogo tra un singolo mittente e un singolo destinatario;
- dirige i dati al **buffer di invio** della connessione, uno dei buffer riservato durante l'handshake da cui, di tanto in tanto, preleverà blocchi di dati e li passerà al livello di rete.

La massima quantità di dati prelevabili e posizionabili in un segmento viene limitata dalla **dimensione massima di segmento** (MSS, *Maximum Segment Size*), impostata determinando prima la lunghezza del frame più grande che può essere inviato a livello di collegamento dall'host mittente locale (MTU, *Maximum Transmission Unit*) e poi scegliendo un MSS tale che il segmento TCP (una volta incapsulato in un datagramma IP) stia all'interno di un singolo frame a livello di collegamento.

TCP accoppia ogni blocco di dati dal client a una intestazione TCP, andando pertanto a formare **segmenti TCP**. Questi vengono passati al sottostante livello di rete, dove sono incapsulati separatamente nei datagrammi IP a livello di rete che vengono poi immessi nella rete. Quando all'altro capo TCP riceve un segmento, i dati del segmento vengono memorizzati nel buffer di ricezione della connessione TCP. L'applicazione legge il flusso di dati da questo buffer.

Il segmento TCP consiste di **campi di intestazione** e di un **campo contenente un blocco di dati** proveniente dall'applicazione. L'intestazione include i seguenti campi:

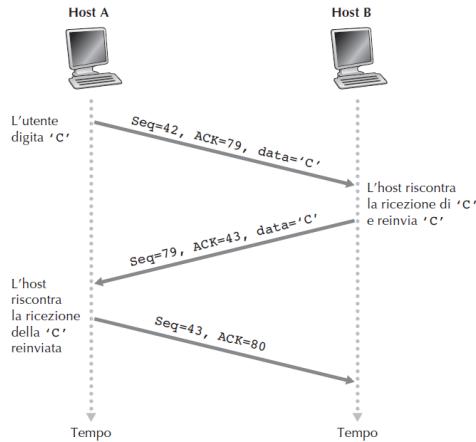
- **numeri di porta di origine e di destinazione**: utilizzati per il multiplexing/demultiplexing dei dati da e verso le applicazioni del livello superiore.
- **checksum**;
- **numero di sequenza** e **numero di ACK**, entrambi di 32 bit che rappresentano, rispettivamente, il numero di sequenza del primo byte nel corpo e il numero di sequenza del prossimo byte atteso;
- **finestra di ricezione**, di 16 bit, per il controllo di flusso.
- **lunghezza dell'intestazione**, di 4 bit, che specifica la lunghezza dell'intestazione TCP in multipli di 32 bit. L'intestazione TCP ha lunghezza variabile a causa del campo delle opzioni TCP;
- **opzioni**, opzionale, utilizzato quando mittente e destinatario negoziano la MSS;
- **flag**, di 6 bit. RST, SYN e FIN per impostare e chiudere la connessione, PSH che ha valore 1 se il destinatario deve inviare immediatamente i dati al livello superiore, URG per indicare la presenza di dati che l'entità mittente ha marcato come "urgenti" (la posizione dell'ultimo byte dei dati urgenti è denotata dal relativo puntatore).



Riguardo i numeri di sequenza, questi si applicano al flusso di byte trasmessi e non alla serie di segmenti trasmessi. Il **numero di sequenza per un segmento** è pertanto il numero nel flusso di byte del primo byte del segmento.

Per quanto riguarda i numeri di acknowledgment: *il numero di ACK che l'Host A scrive nei propri segmenti è il numero di sequenza del byte successivo che l'Host A attende dall'Host B.*

Dato che TCP effettua l'ACK solo dei byte fino al primo byte mancante nel flusso, si dice che tale protocollo offre **acknowledgment cumulativo**.



### Round Trip Time e Timeout

TCP utilizza un **meccanismo di timeout** e ritrasmissione per recuperare i segmenti persi. Chiaramente, il timeout dovrebbe essere più grande del RTT, ma di quanto deve essere maggiore? E, innanzitutto, come dovrebbe essere stimato RTT? Si dovrebbe associare un timer a ogni segmento non riscontrato?

L'RTT misurato di un segmento, denotato come **SampleRTT**, è la quantità di tempo che intercorre tra l'istante di invio del segmento e quello di ricezione dell'ACK del segmento. In ogni istante di tempo, SampleRTT viene valutato per uno solo dei segmenti trasmessi e per cui non si è ancora ricevuto acknowledgment, il che comporta approssimativamente la misurazione di un nuovo valore di SampleRTT a ogni RTT.

Ovviamente, i campioni variano da segmento a segmento in base alla congestione nei router e al diverso carico sui sistemi periferici. Per effettuare una stima, risulta naturale calcolare una media dei valori di SampleRTT, che TCP chiama **EstimatedRTT**. Quando si ottiene un nuovo SampleRTT, TCP aggiorna EstimatedRTT secondo la formula

$$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}$$

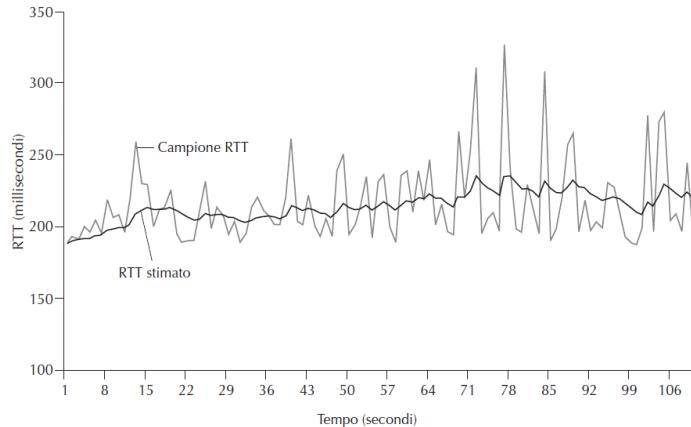
Il valore raccomandato per  $\alpha$  è  $0,125 = 1/8$ .

Tale media attribuisce maggiore importanza ai campioni recenti rispetto a quelli vecchi. In statistica, una media costruita in tal modo è detta **media mobile esponenziale ponderata**.

Oltre ad avere una stima di RTT, è anche importante possedere la misura della sua variabilità. La variazione di RTT, **DevRTT**, stima di quanto **SampleRTT** generalmente si discosta da **EstimatedRTT**:

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

Se i valori di **SampleRTT** presentano fluttuazioni limitate, allora **DevRTT**, sarà piccolo; in caso di notevoli fluttuazioni, **DevRTT** sarà grande. Il valore suggerito per  $\beta$  è  $0,25 = 1/4$ .



A questo punto, possiamo dire che l'intervallo di timeout non può essere inferiore a quello di **EstimatedRTT**, altrimenti verrebbero inviate ritrasmissioni non necessarie. Ma l'intervallo stesso non dovrebbe essere molto maggiore di **EstimatedRTT**, altrimenti TCP non ritrasmetterebbe rapidamente il segmento perduto. È pertanto auspicabile impostare il timeout a **EstimatedRTT** più un certo margine che dovrebbe essere grande quando c'è molta fluttuazione nei valori di **SampleRTT** e piccolo in caso contrario. Tutti questi aspetti vengono presi in considerazione dal modo in cui TCP determina l'**intervallo di timeout di ritrasmissione**:

$$TimeoutInterval = EstimatedRTT + 4 \cdot DevRTT$$

### Trasferimento dati affidabile

Vedremo ora come TCP offra il **trasferimento affidabile dei dati** in due passi successivi.

Dapprima presentiamo una descrizione molto semplificata di un mittente TCP che fa uso solo di timeout per recuperare i segmenti persi e poi una più completa che utilizza anche gli ACK duplicati.

```
/* Ipotizziamo che il mittente non subisca imposizioni dal controllo
di flusso o di congestione TCP, che i dati dall'alto abbiano
dimensione inferiore a MSS e che il trasferimento dati avvenga in
un'unica direzione */

NextSeqNum = InitialSeqNumber
SendBase = InitialSeqNumber

loop (per sempre) {
    switch (evento) {

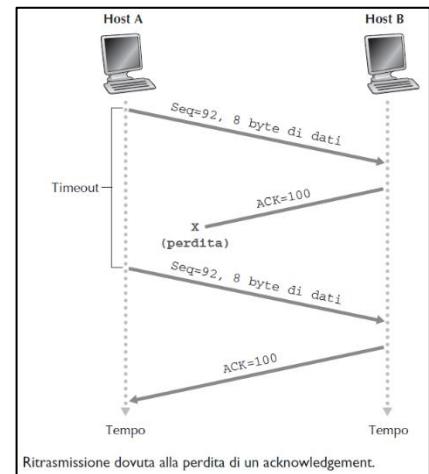
        evento: dati ricevuti dall'applicazione a livello superiore crea
        il segmento TCP con numero di sequenza NextSeqNum
        if (il timer attualmente non è in funzione)
            avvia il timer
        passa il segmento a IP
        NextSeqNum = NextSeqNum + lunghezza(dati)
        break;

        evento: timeout del timer
        ritrasmetti il segmento che non ha ricevuto ACK con il più
        piccolo numero di sequenza
        avvia il timer
        break;

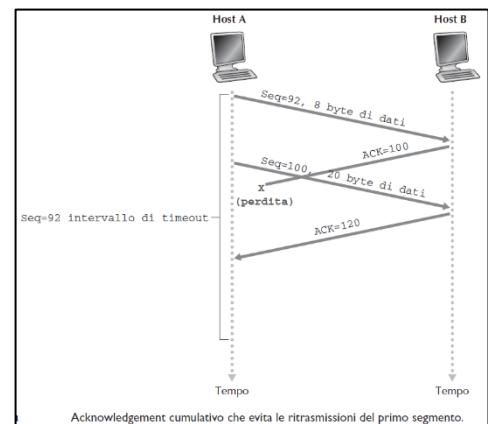
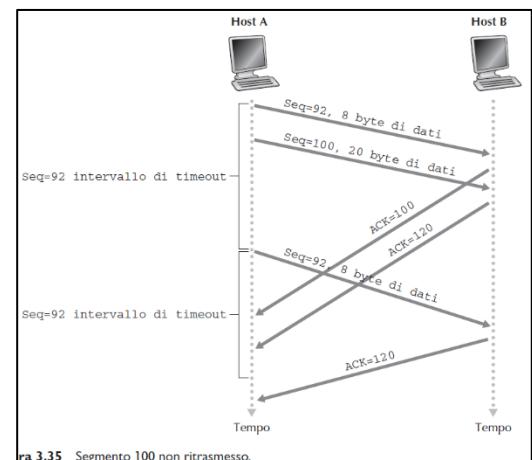
        evento: ACK ricevuto, con valore del campo ACK pari a y
        if (y > SendBase) {
            SendBase = y
            if (esistono attualmente segmenti senza ACK)
                avvia il timer
        }
        break;
    } /* fine del loop */
}
```

Consideriamo ora qualche semplice scenario di esecuzione TCP.

- L'Host A spedisce un segmento all'Host B. Supponiamo che questo segmento abbia numero di sequenza 92 e contenga 8 byte di dati. Dopo aver inviato il segmento, A attende un segmento da B con numero di acknowledgment 100. Sebbene il segmento in questione sia stato ricevuto da B, l'acknowledgment sul percorso inverso viene smarrito. In questo caso si verifica l'evento di timeout e l'Host A ritrasmette lo stesso segmento. Ovviamente, quando l'Host B riceve la ritrasmissione, rileva dal numero di sequenza che il segmento contiene dati che sono già stati ricevuti. Quindi, l'Host B scarta i byte del segmento ritrasmesso.



- L'Host A invia due segmenti. Il primo ha numero di sequenza 92 e 8 byte di dati, il secondo ha numero di sequenza 100 e 20 byte di dati. Supponiamo che entrambi arrivino intatti a B e che questo invii due acknowledgment separati per i segmenti, il primo numerato 100, il secondo 120. Supponiamo ora che nessuno degli acknowledgment arrivi all'Host A prima del timeout. Quando si verifica il timeout, l'Host A rispedisce il primo segmento con numero di sequenza 92 e riavvia il timer. Fino a quando l'ACK del secondo segmento non arriva prima del nuovo timeout, il secondo segmento non sarà ritrasmesso.
- L'Host A invia due segmenti, esattamente come nel secondo esempio. L'acknowledgment del primo segmento viene perso nella rete ma, appena prima dell'evento di timeout, l'Host A riceve un acknowledgment con numero 120. È, pertanto, a conoscenza che l'Host B ha ricevuto tutto fino al byte 119; quindi non rispedisce nessuno dei due segmenti.



In tutti i casi in cui si verifica un timeout, TCP ritrasmette il segmento con il più basso numero di sequenza che non abbia ancora ricevuto acknowledgment. Tuttavia, ogni volta che questo si verifica, TCP imposta il successivo intervallo di timeout al doppio del valore precedente, anziché derivarlo dagli ultimi EstimatedRTT e DevRTT.

Di conseguenza, gli intervalli crescono esponenzialmente a ogni ritrasmissione. Tuttavia, tutte le volte che il timer viene avviato dopo uno degli altri due eventi possibili (ossia la ricezione di dati dell'applicazione superiore e la ricezione di un ACK), il `TimeoutInterval` viene ricavato dai più recenti valori di `EstimatedRTT` e `DevRTT`.

### Ritrasmissione rapida

Fortunatamente, il mittente può in molti casi rilevare la perdita dei pacchetti ben prima che si verifichi l'evento di timeout grazie agli **ACK duplicati** relativi a un segmento il cui ACK è già stato ricevuto dal mittente.

La tabella che segue riassume la politica di generazione degli ACK di TCP.

| Destinatario TCP   |  |
|--|--|
| Evento   | Azione   |
| Arrivo in ordine di un segmento con #seq atteso && tutti i segmenti già riscontrati    | L'ACK viene ritardato di un massimo di 500ms in attesa di un altro segmento.<br>Se nessun altro segmento arriva, viene inviato l'ACK.            |
| Arrivo in ordine di un segmento con #seq atteso && un altro segmento attende riscontro | Viene immediatamente inviato un ACK cumulativo per questo segmento ed il precedente.<br>(È legato all'evento precedente).                        |
| Arrivo fuori ordine di un segmento con #seq maggiore di quello atteso (gap)            | Viene immediatamente inviato un ACK duplicato che indica il #seq per il prossimo byte/segmento atteso.<br>(È legato al <i>fast retransmit</i> ). |
| Arrivo di un segmento che riempie, parzialmente o interamente, il gap                  | Se i segmenti fuori ordine vengono bufferizzati e il segmento arrivato riempie l'inizio del gap, invia un ACK cumulativo.                        |

Quando il destinatario TCP riceve un segmento con numero di sequenza superiore al successivo numero di sequenza atteso e in ordine, rileva un buco nel flusso di dati, ossia un segmento mancante. Tale vuoto potrebbe essere il risultato di segmenti persi o riordinati all'interno della rete. Il destinatario non può inviare un acknowledgment negativo esplicito al mittente, dato che TCP non lo prevede, ma si limita a mandare nuovamente un ACK relativo all'ultimo byte di dati che ha ricevuto in ordine (duplicando così un ACK).

Dato che in molti casi il mittente invia un gran numero di segmenti, se uno di questi viene smarrito ci saranno probabilmente molti ACK duplicati. Se il mittente TCP riceve tre ACK duplicati per lo stesso dato, considera questo evento come indice che il segmento che lo segue è andato perduto.

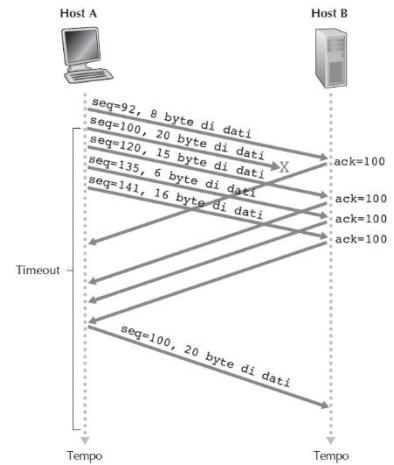
Nel caso in cui siano stati ricevuti tre ACK duplicati, il mittente TCP effettua una **ritrasmissione rapida** (*fast retransmit*), rispedendo il segmento mancante prima che scada il timer.

In TCP con trasmissione rapida il seguente frammento di codice sostituisce l'evento di ACK ricevuto nella prima figura del paragrafo “Trasferimento dati affidabile”.

```

evento: ACK ricevuto, con valore del campo ACK pari a y
    if (y > SendBase) {
        SendBase = y
        if (esistono attualmente segmenti che non hanno
ricevuto un ACK)
            avvia il timer
    } else { /* un ACK duplicato per un segmento */
        incrementa il numero di ACK duplicati ricevuti per y
    if (numero di ACK duplicati ricevuti per y == 3) {
        /* ritrasmissione rapida */
    rispedisci il segmento con numero di sequenza y
    }
}
break;

```



Sulla destra vediamo invece un esempio dove viene perduto il secondo segmento e ritrasmesso prima che il timer scada.

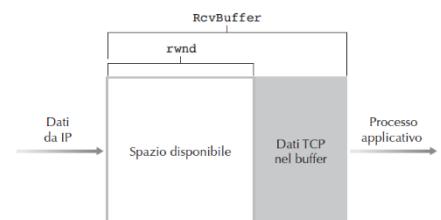
### Controllo di flusso TCP

Quando la connessione TCP riceve byte corretti e in sequenza, li posiziona nel **buffer di ricezione**. Il processo applicativo associato legge i dati da questo buffer, ma non necessariamente nell'istante in cui arrivano. Infatti, l'applicazione ricevente potrebbe essere impegnata in qualche altro compito e potrebbe non leggere i dati fino a un istante di tempo molto successivo al loro arrivo. Se l'applicazione è relativamente lenta nella lettura dei dati può accadere che il mittente mandi in **overflow** il buffer di ricezione inviando molti dati troppo rapidamente.

TCP offre un **servizio di controllo di flusso** (*flow-control service*) alle proprie applicazioni per evitare che il mittente saturi il buffer del ricevente (paragona la frequenza di invio del mittente con quella di lettura dell'applicazione ricevente).

TCP offre controllo di flusso facendo mantenere al mittente una variabile chiamata **finestra di ricezione** (*receive window*) che, in sostanza, fornisce al mittente un'indicazione dello spazio libero disponibile nel buffer del destinatario. Dato che TCP è full-duplex, i due mittenti mantengono finestre di ricezione distinte.

Supponiamo che l'Host A stia inviando un file di grandi dimensioni all'Host B su una connessione TCP. Quest'ultimo alloca un buffer di ricezione per la connessione, la cui dimensione è denotata come **RcvBuffer**. Di tanto in tanto, il processo applicativo nell'Host B legge dal buffer. La finestra di ricezione, indicata con **rwnd**, viene impostata alla quantità di spazio disponibile nel buffer. L'Host B comunica all'Host A quanto spazio disponibile sia presente nel buffer della connessione, scrivendo il valore corrente di **rwnd** nel campo apposito dei segmenti che manda ad A.



Mantenendo la quantità di dati senza acknowledgment sotto il valore di **rwnd**, si garantisce che l'Host A non mandi in overflow il buffer di ricezione dell'Host B. Quindi, l'Host A si assicura che per tutta la durata della connessione sia rispettata la disuguaglianza

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

Le specifiche TCP richiedono che l'Host A continui a inviare segmenti con un byte di dati quando la finestra di ricezione di B è zero. Il destinatario risponderà a questi segmenti con un acknowledgment. Prima o poi il buffer inizierà a svuotarsi e i riscontri conterranno un valore non nullo per `rwnd`.

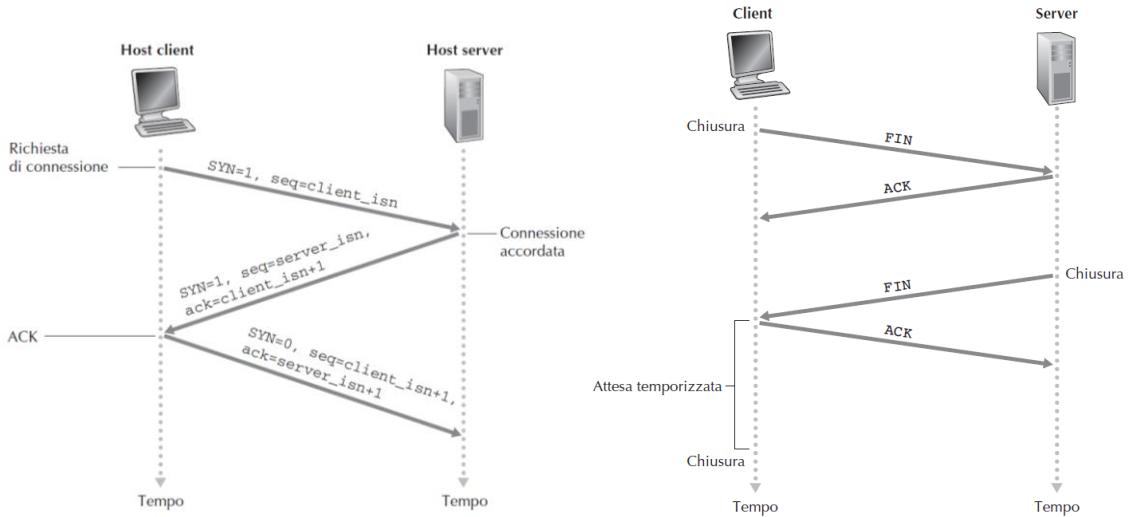
### Gestione della connessione TCP

Consideriamo innanzitutto come viene stabilita una connessione TCP. Supponiamo che un processo in esecuzione in un host (client) voglia inizializzare una connessione con un altro processo in un altro host (server).

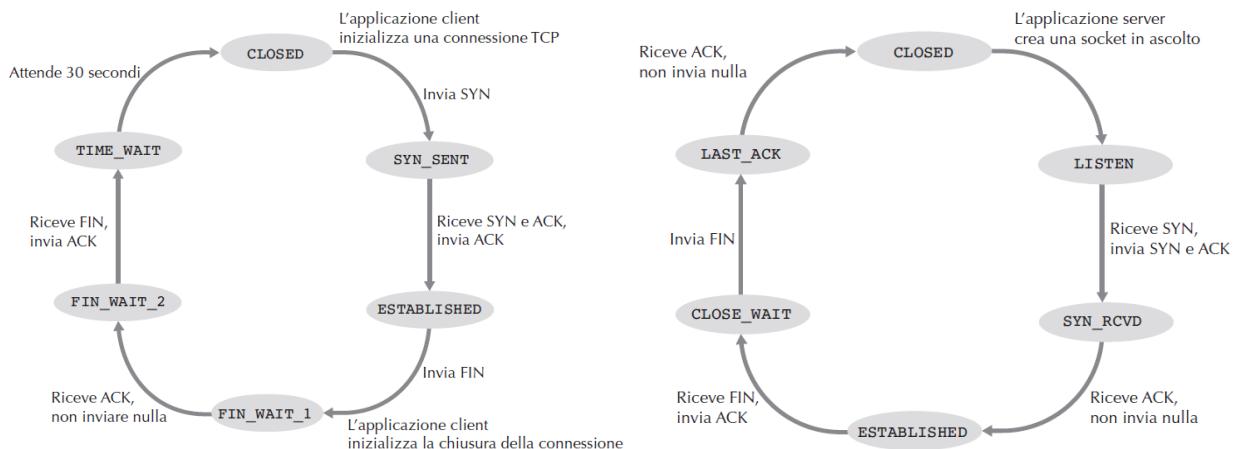
1. TCP lato client invia uno speciale segmento al TCP lato server. Questo segmento speciale non contiene dati a livello applicativo, ma uno dei bit nell'intestazione del segmento, il bit SYN, è posto a 1. Per questo motivo, tale segmento viene detto **segmento SYN**. Inoltre, il client sceglie a caso un numero di sequenza iniziale (`client_isn`) e lo pone nel campo numero di sequenza del segmento SYN iniziale. Quest'ultimo viene incapsulato in un datagramma IP e inviato al server.
2. Quando il datagramma IP contenente il segmento TCP SYN arriva all'host server (ammesso che arrivi), il server estrae il segmento dal datagramma, alloca i buffer e le variabili TCP alla connessione e invia un segmento di connessione approvata al client TCP. Anche questo segmento non contiene dati a livello applicativo, ma nella sua intestazione vi sono tre informazioni importanti. Innanzitutto, il bit SYN è posto a 1. In secondo luogo, il campo ACK assume il valore `client_isn + 1`. Infine, il server sceglie il proprio numero di sequenza iniziale (`server_isn`) e lo pone nel campo del numero di sequenza. Il segmento di connessione approvata viene talvolta detto **segmento SYNACK**.
3. Alla ricezione del segmento SYNACK, anche il client alloca buffer e variabili alla connessione. L'host client invia quindi al server un altro segmento in risposta al segmento di connessione approvata del server. Tale operazione viene svolta dal client ponendo il valore `server_isn + 1` nel campo ACK dell'intestazione del segmento TCP. Il bit SYN è posto a zero, dato che la connessione è stata stabilita. In questo terzo passo il campo dati del segmento può contenere informazioni che vanno dal client verso il server.

Ciascuno dei due processi che partecipano alla connessione può terminarla. E, in tal caso, le "risorse" negli host (ossia buffer e variabili) vengono deallocate.

Supponiamo che il client decida di **chiudere la connessione**. Il processo applicativo client invia un comando di chiusura, che forza il client TCP a inviare un segmento TCP speciale al processo server. Nell'intestazione di tale segmento troviamo il **bit FIN** con valore 1. Quando il server riceve questo segmento, risponde inviando un ACK al client. Il server spedisce quindi il proprio segmento di shutdown, con il bit FIN uguale a 1. Infine, il client manda a sua volta un acknowledgment a quest'ultimo segmento del server. A questo punto, tutte le risorse degli host risultano deallocate.



Nell'arco di una connessione TCP, i protocolli TCP in esecuzione negli host attraversano vari **stati TCP**. (Stati visitati da un client a sinistra e stati visitati da un server a destra)



Consideriamo ora cosa succede quando un host riceve un segmento TCP i cui numeri di porta o il cui indirizzo IP di origine non corrispondono ad alcuna socket attiva nell'host. In tal caso invierà al mittente un **segmento speciale di reset** con il bit RST impostato a 1, allo scopo di comunicare alla sorgente "non ho una socket per quel segmento".

Quando un host riceve un pacchetto UDP il cui numero di porta di destinazione non corrisponde ad una socket UDP attiva, invia uno speciale datagramma ICMP.

### Controllo di congestione TCP

La ritrasmissione di pacchetti tratta quindi un sintomo della congestione di rete (la perdita di uno specifico segmento a livello di trasporto), ma non le cause della congestione di rete: il tentativo da parte di troppe sorgenti di inviare dati a ritmi troppo elevati.

Principalmente, si identificano due orientamenti al controllo della congestione utilizzati nella pratica:

- **controllo di congestione end-to-end**: il livello di rete non fornisce supporto esplicito al livello di trasporto per il controllo di congestione la cui presenza deve essere dedotta dai sistemi periferici sulla base dell'osservazione del comportamento della rete (per esempio, perdita di pacchetti e ritardi). TCP deve necessariamente

- utilizzare questo approccio, dato che il livello IP non offre feedback relativamente alla congestione della rete;
- **controllo di congestione assistito dalla rete**: i componenti a livello di rete (ossia i router) forniscono un feedback esplicito al mittente sullo stato di congestione della rete.

### Controllo di congestione TCP

L'approccio scelto da TCP consiste nell'imporre a ciascun mittente un limite alla velocità di invio sulla propria connessione in funzione della congestione di rete percepita. Se il mittente TCP si accorge di condizioni di scarso traffico sul percorso che porta alla destinazione, incrementa il proprio tasso trasmissivo; se, invece, percepisce traffico lungo il percorso, lo riduce.

Abbiamo visto che gli estremi di una connessione TCP gestiscono un buffer di ricezione, uno di invio e diverse variabili, tra cui `LastByteRead` e `rwnd`. Il meccanismo di **controllo di congestione TCP** fa tener traccia agli estremi della connessione di una variabile aggiuntiva: la **finestra di congestione** (*congestion window*), indicata con `cwnd`, che impone un vincolo alla velocità di trasmissione di traffico sulla rete da parte del mittente. Nello specifico, la quantità di dati che non hanno ancora ricevuto acknowledgment inviata da un mittente non può eccedere il minimo tra i valori di `cwnd` e `rwnd`, ossia:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{cwnd}, \text{rwnd}\}$$

Assumiamo che il buffer di ricezione sia sufficientemente capiente da poter ignorare il vincolo della finestra di ricezione e che il mittente abbia sempre dati da inviare, per cui la finestra di congestione sia sempre completamente in uso. Di conseguenza, approssimativamente all'inizio di ogni RTT, il vincolo consente al mittente di trasmettere `cwnd` byte di dati sulla connessione; al termine del RTT il mittente riceve gli acknowledgment relativi ai dati. Quindi, la velocità di invio del mittente è approssimativamente `cwnd/RTT` byte/s. Modificando il valore di `cwnd`, il mittente può regolare la velocità di invio dei dati sulla propria connessione.

Consideriamo ora come il mittente TCP percepisce la presenza di congestione sul cammino verso la destinazione. Definiamo **“evento di perdita”** per il mittente TCP l'occorrenza o di un timeout o della ricezione di tre ACK duplicati da parte del destinatario. Il mittente considera questo evento come un'indicazione di congestione sul percorso tra sé e il destinatario.

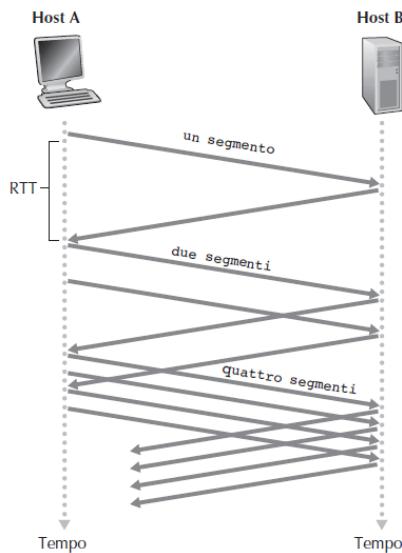
Il mittente TCP utilizza gli acknowledgment per aumentare l'ampiezza della propria finestra di congestione (e, di conseguenza, la velocità trasmissiva). Se gli acknowledgment arrivano con frequenza relativamente bassa (e.g. se il percorso end-to-end presenta ritardi elevati o transita per un collegamento lento), allora la finestra di congestione verrà ampliata piuttosto lentamente. Se, invece, gli acknowledgment giungono con una frequenza alta, allora la finestra di congestione verrà ampliata più rapidamente. Per queste dinamiche, si dice che TCP è **auto-temporizzato** (*self-clocking*).

Come fanno i mittenti TCP a determinare la loro velocità di trasmissione in modo da non congestionare la rete, ma allo stesso tempo utilizzare tutta la banda disponibile?

- un segmento perso implica congestione, quindi i tassi di trasmissione del mittente TCP dovrebbero essere decrementati quando un segmento viene perso;
- un acknowledgment indica che la rete sta consegnando i segmenti del mittente al ricevente e quindi il tasso di trasmissione del mittente può essere aumentato quando arriva un acknowledgment non duplicato;
- rilevamento della larghezza di banda.

L'**algoritmo di controllo di congestione di TCP** presenta tre componenti o fasi principali: (1) *slow start*, (2) *congestion avoidance* e (3) *fast recovery*.

- ***Slow start***. Quando si stabilisce una connessione TCP, il valore di *cwnd* viene in genere inizializzato a 1 MSS, il che comporta una velocità di invio iniziale di circa MSS/RTT. Durante la fase iniziale, detta *slow start*, il valore di *cwnd* parte da 1 MSS e si incrementa di 1 MSS ogni volta che un segmento trasmesso riceve un ACK. Nello specifico, TCP invia il primo segmento e attende un riscontro. Se il segmento riceve un acknowledgment prima che si verifichi un evento di perdita, il mittente incrementa la finestra di congestione di 1 MSS e invia due segmenti di dimensione massima. Questi segmenti ricevono a loro volta degli acknowledgment e il mittente incrementa la finestra di congestione di 1 MSS per ciascuno di essi portandola a 4 MSS e così via. Quindi, in TCP, la velocità di trasmissione parte lentamente, ma **cresce in modo esponenziale** durante la fase di slow start.

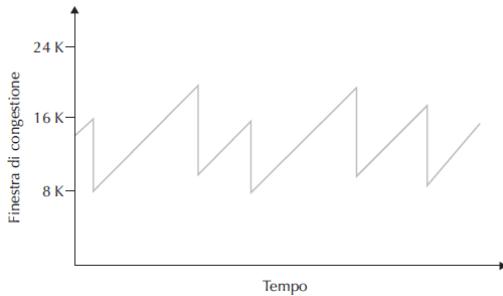


La fase di slow start può terminare in tre modi:

- se c'è un evento di perdita (e, quindi, una congestione) indicato da un evento di timeout, il mittente TCP pone il valore di *cwnd* pari a 1 e inizia di nuovo il processo di slow start. Inoltre, pone il valore di una seconda variabile di stato, ***ssthresh*** (forma contratta per “slow start threshold”) a  $cwnd/2$ : metà del valore che aveva la finestra di congestione quando la congestione è stata rilevata;
- quando il valore di *cwnd* è pari a *ssthresh*, la fase di slow start termina e TCP entra in modalità di *congestion avoidance*;
- quando vengono rilevati tra acknowledgment duplicati, nel qual caso TCP opera una ritrasmissione rapida ed entra nello stato di *fast recovery*.

- **Congestion avoidance.** Quando TCP entra nello stato di *congestion avoidance*, il valore di `cwnd` è circa la metà di quello che aveva l'ultima volta in cui era stata rilevata la congestione. Quindi, invece di raddoppiare il valore di `cwnd` ogni RTT, TCP adotta un approccio più conservativo, incrementando `cwnd` di 1 MSS ogni RTT.  
Ma quando finisce l'incremento lineare durante la congestion avoidance?
  - Quando si verifica un timeout il valore di `cwnd` è posto uguale a 1 MSS e il valore di `ssthresh` viene impostato alla metà del valore di `cwnd` al momento del timeout;
  - In caso di acknowledgment duplicati TCP dimezza il valore di `cwnd` (aggiungendo 3 MSS per tenere conto dei duplicati ricevuti) e imposta il valore di `ssthresh` pari a metà del valore di `cwnd` al momento del ricevimento dei tre ACK duplicati. Infine, TCP entra nello stato di *fast recovery*.
- **Fast recovery.** Durante la fase di *fast recovery* il valore di `cwnd` è incrementato di 1 MSS per ogni ACK duplicato ricevuto relativamente al segmento perso che ha causato l'entrata di TCP nello stato di fast recovery. Infine, quando arriva un ACK per il segmento perso, TCP entra nello stato di *congestion avoidance* dopo aver ridotto il valore di `cwnd`.  
Se si verifica un timeout vi è invece una transizione dallo stato di fast recovery a quello di *slow start*: il valore di `cwnd` è posto a 1 MSS e il valore di `ssthresh` è impostato a metà del valore di `cwnd` nel momento in cui si è riscontrato l'evento di perdita.

Il controllo di congestione di TCP è spesso indicato come una forma di controllo di congestione **incremento additivo, decremento moltiplicativo** (AIMD, *Additive-Increase Multiplicative-Decrease*). Il controllo di congestione AIMD dà luogo al comportamento a dente di sega mostrato in figura:



Le **prestazioni ideali** del protocollo TCP a regime, ovvero in fase di *congestion avoidance*, sono date da ( $W :=$  dimensione finestra per la quale avviene perdita):

$$\text{throughput} = \frac{3}{4} \cdot \frac{W \text{ bytes}}{RTT \text{ s}}$$

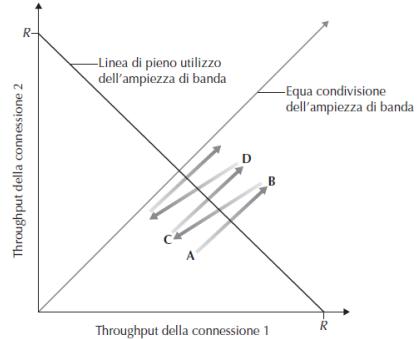
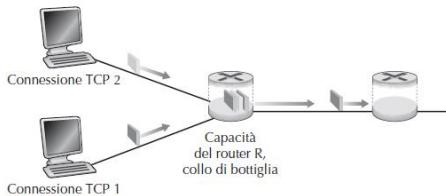


Si noti che il valore di  $0.75 \cdot W$  rappresenta la dimensione media della finestra di invio.

## Fairness TCP

Consideriamo  $K$  connessioni TCP, ciascuna con un differente percorso end-to-end, ma che passano tutte attraverso un collegamento con capacità trasmissiva di  $R$  bps che costituisce il “collo di bottiglia” del sistema. Si dice che un meccanismo di controllo di congestione è **equo (fair)** se la velocità trasmissiva media di ciascuna connessione è approssimativamente  $R/K$ ; in altre parole, ciascuna connessione ottiene la stessa porzione di banda del collegamento.

Consideriamo il caso semplice di due connessioni TCP che condividono un collegamento con capacità trasmissiva  $R$ . Assumiamo che le connessioni abbiano gli stessi valori di MSS e RTT (e pertanto abbiano uguali finestre di congestione e lo stesso throughput), che debbano trasmettere una gran quantità di dati e che non vi siano altre connessioni TCP o datagrammi UDP che attraversano questo collegamento condiviso. Inoltre, ignoriamo la fase di slow start e assumiamo che le connessioni operino in modalità di congestion avoidance (AIMD) per tutto il tempo.



Il grafico traccia il throughput delle due connessioni. Se TCP sta suddividendo la larghezza di banda del collegamento in modo uguale tra le due connessioni, allora il throughput dovrebbe cadere sulla bisettrice del primo quadrante. L’obiettivo dovrebbe essere ottenere **throughput situati vicino all’intersezione tra la bisettrice e la linea di massimo utilizzo della banda**.

Supponiamo che le dimensioni della finestra TCP siano tali che a un certo istante di tempo le connessioni 1 e 2 raggiungano i throughput corrispondenti al punto A della figura. Dato che la porzione di banda del collegamento utilizzata congiuntamente è minore di  $R$ , non si verificheranno perdite e le due connessioni aumenteranno la loro finestra di 1 MSS per RTT come risultato dell’algoritmo di congestion avoidance. Di conseguenza, il throughput congiunto procede lungo la semiretta a  $45^\circ$  (pari incremento per entrambe le connessioni) uscente dal punto A. La banda congiunta potrebbe essere maggiore di  $R$ , e si potrebbe quindi verificare una perdita di pacchetti. Se le connessioni 1 e 2 subiscono una perdita di pacchetti quando raggiungono i throughput indicati dal punto B, allora decrementeranno le loro finestre di un fattore 2. I throughput raggiunti di conseguenza si troveranno pertanto sul punto C, e così via.

Nella pratica, le applicazioni client/server ottengono porzioni di banda assai diverse. In particolare, è stato dimostrato che quando più connessioni condividono un collo di bottiglia, quelle con RTT inferiore sono in grado di acquisire più rapidamente larghezza di banda su un particolare collegamento, non appena la banda si libera.

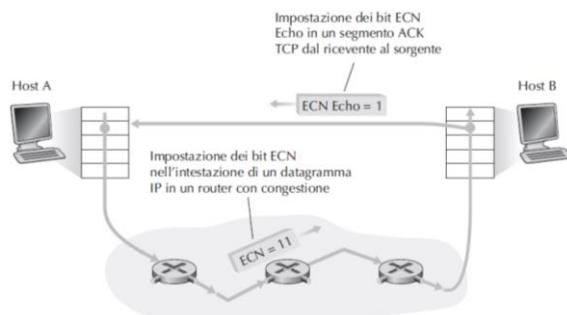
Dal punto di vista TCP, le applicazioni multimediali che fanno uso di UDP non sono fair: non cooperano con altre né adeguano la loro velocità trasmissiva in modo appropriato. Dato che il controllo di congestione di TCP diminuisce il proprio tasso trasmissivo come risposta alla congestione crescente (e alle perdite), mentre UDP no, le sorgenti UDP possono soffocare il traffico TCP.

Anche se potessimo forzare il traffico UDP a comportarsi in modo equo, il problema della fairness non sarebbe tuttavia completamente risolto, perché nulla può impedire a un'applicazione basata su TCP di usare più connessioni in parallelo: queste ottengono una porzione di banda maggiore sui collegamenti congestionati. Consideriamo per esempio un collegamento di capacità  $R$  cui accedono nove applicazioni client/server, ciascuna delle quali utilizza una connessione TCP. Se giunge un'altra applicazione con una connessione TCP, allora la frequenza trasmissiva di tutte le applicazioni sarà circa uguale a  $R/10$ . Ma se, invece, la nuova applicazione usa 11 connessioni TCP in parallelo, allora otterrà un'allocazione iniqua superiore a  $R/2$ .

### Notifica esplicita di congestione (ECN)

Più di recente, sono state implementate estensioni per IP e TCP che permettono alla rete di segnalare esplicitamente una congestione a mittente e ricevente TCP. Questa forma di **controllo della congestione assistito dalla rete** è nota come **notifica esplicita di congestione** (*Explicit Congestion Notification*).

A livello di rete vengono utilizzati due bit (quindi quattro possibili valori) nel campo Tipo di Servizio dell'intestazione IP. Se un router è congestionato, imposta tali bit e invia il pacchetto IP contrassegnato al destinatario, che quindi informa il mittente. I **bit ECN** vengono inoltre utilizzati dal mittente per segnalare che mittente e ricevente sono abilitati all'uso di ECN.



Come mostrato in figura, quando il destinatario TCP riceve un'indicazione di congestione, ne informa il mittente TCP impostando il **bit ECE** (*Explicit Congestion notification Echo*) all'interno di un segmento ACK. Il mittente TCP reagisce dimezzando la finestra di congestione, esattamente come farebbe in caso di perdita di un segmento usando il meccanismo di ritrasmissione rapida, e imposta il **bit CWR** (*Congestion Window Reduced*) nell'intestazione del successivo segmento che invia al ricevente.

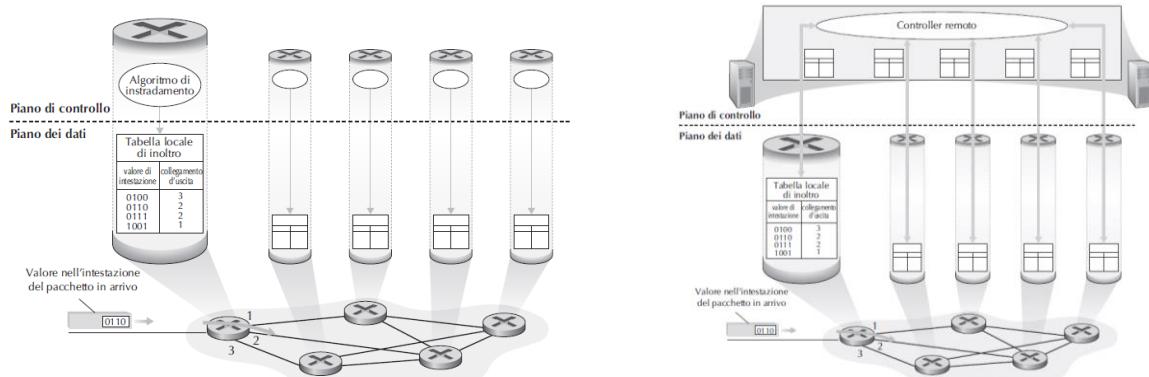
## Network Layer

Il ruolo principale del **livello di rete** è piuttosto semplice: trasferire pacchetti da un host a un altro. Per fare questo è possibile identificare due importanti funzioni.

- **Forwarding** (inoltro). Quando un router riceve un pacchetto, lo deve trasferire sull'appropriato collegamento di uscita. Un pacchetto può anche essere bloccato (per esempio se inviato da un mittente malevolo o destinato a un host vietato) o duplicato e inviato su più collegamenti di uscita.
- **Routing** (instradamento). Il livello di rete deve determinare il percorso che i pacchetti devono seguire tramite **algoritmi di instradamento** (*algoritmi di routing*). La funzione di instradamento è implementata nel piano di controllo del livello di rete.

Con **inoltro** faremo quindi riferimento all'azione locale con cui il router trasferisce i pacchetti da un'interfaccia di ingresso a quella di uscita. Con **instradamento** indicheremo, invece, il processo globale di rete che determina i percorsi dei pacchetti nel loro viaggio dalla sorgente alla destinazione.

Per inoltrare i pacchetti, i router estraggono da uno o più campi dell'intestazione i loro valori che utilizzano come indice della **tavella di inoltro** (*forwarding table*), un elemento chiave di qualsiasi router. Il risultato indica a quale interfaccia di uscita il pacchetto debba essere diretto.



Esiste anche un approccio alternativo in cui un controller remoto, separato fisicamente dai router, calcola e distribuisce le tabelle di inoltro a tutti i router. Il controller remoto potrebbe essere implementato in un data center remoto con elevata affidabilità e ridondanza e potrebbe essere gestito da un ISP o da una terza parte. Come potrebbero comunicare tra loro i router e il controller remoto? Scambiandosi messaggi contenenti le tabelle di inoltro e altre informazioni di instradamento. Il piano di controllo mostrato nella figura sopra a destra è il cuore del **software-defined networking** (SDN), nel quale la rete è “software-defined” perché il controller che calcola le tabelle di inoltro e interagisce coi router è implementato in software.

## Modelli di servizio

Consideriamo ora alcuni servizi che il livello di rete potrebbe offrire.

- **Consegna garantita.** Questo servizio assicura che il pacchetto giunga, primo o poi, alla propria destinazione.
- **Consegna garantita con ritardo limitato.** Questo servizio non solo garantisce la consegna del pacchetto, ma anche il rispetto di un limite di ritardo specificato (per esempio 100ms).
- **Consegna ordinata.** Questo servizio garantisce che i pacchetti giungano alla destinazione nell'ordine in cui sono stati inviati.
- **Banda minima garantita.** Questo servizio a livello di rete emula il comportamento di un collegamento trasmissivo con il bit rate specificato tra host di invio e di destinazione, anche se l'effettivo percorso end-to-end può attraversare diversi collegamenti fisici. Finché l'host di invio trasmette bit (sotto forma di pacchetti) a un tasso inferiore al bit rate specificato, non si verifica perdita di pacchetti.
- **Servizi di sicurezza.** Il livello di rete dell'host sorgente può cifrare tutti i datagrammi inviati; il livello di rete nell'host di destinazione avrà il compito di decifrarli. Con questo tipo di servizio, la riservatezza viene fornita a tutti i segmenti del livello di trasporto.

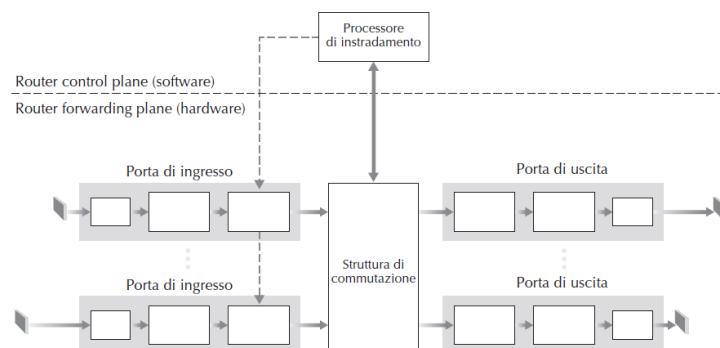
Il livello di rete di Internet mette a disposizione un solo servizio, noto come **servizio best-effort**, ossia “col massimo impegno possibile”. Con questo servizio, non c’è garanzia che i pacchetti vengano ricevuti nell’ordine in cui sono stati inviati, così come non è garantita la loro eventuale consegna. Non c’è garanzia sul ritardo end-to-end, così come non c’è garanzia su una larghezza di banda minima.

Vale la pena menzionare altri termini che vengono spesso impiegati in modo intercambiabile. Riserveremo il termine **commutatore di pacchetto** (*packet switch* o *switch*) per indicare un generico dispositivo che si occupa del trasferimento dei pacchetti da un’interfaccia in ingresso a quella in uscita, in base al valore dei campi nell’intestazione del pacchetto. Alcuni commutatori di pacchetto, chiamati **commutatori a livello di collegamento** (*link-layer switch*) stabiliscono l’inoltro in relazione al valore del campo del frame a livello di collegamento (livello 2). Altri, chiamati **router**, prendono le decisioni di inoltro basandosi sul valore nel campo a livello di rete. I router sono quindi dispositivi a livello di rete (livello 3). Dato che ci concentreremo sul livello di rete, useremo il termine *router* anziché *switch*.

## L'interno di un router

Presentiamo una visione ad alto livello di una generica architettura di router in cui si possono identificare quattro componenti.

- **Porte di ingresso (input port).** Svolgono le funzioni a livello fisico di terminare di un collegamento in ingresso al router; svolgono anche funzioni a livello di collegamento (rappresentate dai riquadri centrali delle porte di ingresso e di uscita), necessarie per inter-operare con le analoghe funzioni all'altro capo del collegamento di ingresso. Svolgono inoltre la cruciale funzione di ricerca, in modo che il pacchetto inoltrato nella struttura di commutazione del router esca sulla porta di uscita corretta. I pacchetti di controllo, per esempio quelli che trasportano informazioni sul protocollo di instradamento, sono inoltrati dalla porta di ingresso al processore di instradamento.
- **Struttura di commutazione (switching fabric).** Connnette fisicamente le porte di ingresso a quelle di uscita ed è interamente contenuta all'interno del router.
- **Porte di uscita (output port).** Memorizzano i pacchetti che provengono dalla struttura di commutazione e li trasmettono sul collegamento in uscita, operando le funzionalità necessarie del livello di collegamento e fisico. Nei collegamenti bidirezionali, che trasportano traffico in entrambe le direzioni, la porta di uscita verso un collegamento è solitamente accoppiata alla porta di ingresso di quel collegamento sulla stessa scheda di collegamento (detta anche *line card*).
- **Processore di instradamento (routing processor).** Esegue le funzioni del piano di controllo. Nei router tradizionali, esegue i protocolli di instradamento, gestisce le tabelle di inoltro e le informazioni sui collegamenti attivi, ed elabora la tabella di inoltro per il router. Nei router SDN, il processore di instradamento è responsabile della comunicazione con il controller remoto, in modo da ricevere le occorrenze della tabella di inoltro e installarle alle porte di ingresso. Inoltre, effettua operazioni di gestione di rete.



Mentre il piano dei dati opera sulla scala temporale dei nanosecondi, le funzioni di controllo del router operano sulla scala temporale dei millisecondi o dei secondi. Le funzioni del **piano di controllo** sono solitamente implementate via software ed eseguite sul processore di instradamento.

### Elaborazione alle porte di ingresso

Come già detto, la funzione di terminazione (elettrica) della linea e l'elaborazione a livello di collegamento implementano rispettivamente il livello fisico e di collegamento associati a un singolo collegamento di ingresso al router. **L'elaborazione effettuata alla porta di ingresso** è centrale per la funzionalità del router: è qui che, utilizzando le informazioni della **tabella di inoltro**, viene determinata la porta di uscita a cui dirigere un pacchetto attraverso la struttura di commutazione.

La tabella di inoltro viene elaborata e aggiornata dal processore di instradamento o ricevuta da un controller SDN remoto. Una sua copia conforme è di solito memorizzata su ciascuna porta di ingresso. Essendoci copie locali della tabella di inoltro, la decisione relativa può essere presa dalle porte di ingresso, senza invocare il processore di instradamento centralizzato.

Consideriamo il caso “più semplice” in cui l'**inoltro** è **basato sull'indirizzo di destinazione**. Supponiamo che tutti gli indirizzi di destinazione siano a 32 bit, che il nostro router abbia quattro collegamenti, numerati da 0 a 3, e che i pacchetti debbano essere inoltrati verso le interfacce di collegamento come segue:

| Intervallo degli indirizzi di destinazione |          |          |          |          | Interfaccia |
|--|----------|----------|----------|----------|-------------|
| da   | 11001000 | 00010111 | 00010000 | 00000000 | 0           |
| a  | 11001000 | 00010111 | 00010111 | 11111111 |             |
| da   | 11001000 | 00010111 | 00011000 | 00000000 | 1           |
| a  | 11001000 | 00010111 | 00011000 | 11111111 |             |
| da   | 11001000 | 00010111 | 00011001 | 00000000 | 2           |
| a  | 11001000 | 00010111 | 00011111 | 11111111 |             |
| altrimenti                                 |          |          |          |          | 3           |

Potremmo, per esempio, avere la seguente tabella:

| Corrispondenza di prefisso |          |          | Interfaccia |
|----------------------------|----------|----------|-------------|
| 11001000                   | 00010111 | 00010    | 0           |
| 11001000                   | 00010111 | 00011000 | 1           |
| 11001000                   | 00010111 | 00011    | 2           |
| altrimenti                 |          |          | 3           |

Con questa struttura il router confronta un **prefisso** dell'indirizzo di destinazione del pacchetto con una riga della tabella; se c'è corrispondenza, il router inoltra il pacchetto al collegamento associato. Quando si verificano corrispondenze multiple, il router adotta la **regola di corrispondenza a prefisso più lungo**; in altre parole, viene determinata la corrispondenza più lunga all'interno della tabella e i pacchetti vengono inoltrati all'interfaccia di collegamento associata.

Una volta determinata la porta di output di un pacchetto, esso può essere inviato alla struttura di commutazione. In alcune architetture di router un pacchetto può essere temporaneamente fermato prima di entrare nella struttura di commutazione quando essa è utilizzata da altre porte di input. Un pacchetto bloccato verrà accodato sulla porta di input e quindi schedulato per il passaggio alla struttura di commutazione più tardi.

## Struttura di commutazione

La **struttura di commutazione** (*switching fabric*) rappresenta il vero e proprio cuore dei router, attraverso il quale i pacchetti vengono commutati (ossia inoltrati) dalla porta di ingresso alla porta di uscita. La commutazione può essere ottenuta in vari modi.

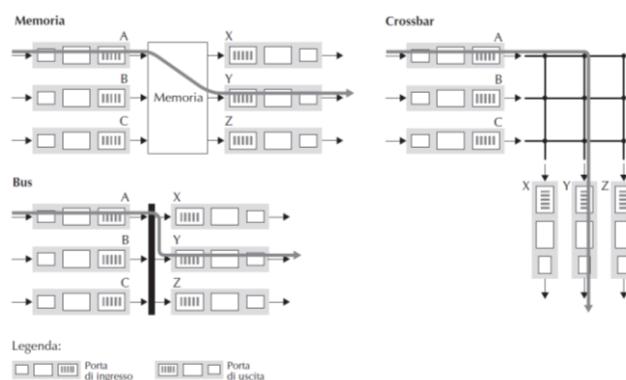
- **Commutazione in memoria.** La commutazione viene effettuata sotto il controllo diretto della CPU (processore di instradamento). Quando sopraggiunge un pacchetto, la porta di ingresso ne segnala l'arrivo tramite interrupt e quindi lo copia nella memoria del processore di instradamento che procede a estrarre dall'intestazione l'indirizzo di destinazione. Quindi, individua tramite la tabella di inoltro l'appropriata porta di uscita nel cui buffer copia il pacchetto.

Due pacchetti non possono essere inoltrati contemporaneamente, anche se hanno differenti porte di destinazione, perché può essere effettuata solo un'operazione alla volta di scrittura/lettura in memoria tramite il bus di sistema.

- **Commutazione tramite bus.** le porte di ingresso trasferiscono un pacchetto direttamente alle porte di uscita tramite un bus condiviso e senza intervento da parte del processore di instradamento. Questo viene tipicamente fatto aggiungendo un'etichetta interna di commutazione (intestazione) al pacchetto che indica la porta locale di output alla quale il pacchetto deve essere trasferito quando viene trasmesso sul bus. Il pacchetto viene ricevuto da tutte le porte di output, ma solo la porta corrispondente all'etichetta lo raccoglierà.

Se più pacchetti arrivano contemporaneamente al router, ognuno su una porta di input diversa, tutti tranne uno dovranno aspettare, dato che sul bus si può trasferire soltanto un pacchetto alla volta. Poiché ciascun pacchetto deve attraversare il bus, la larghezza di banda della commutazione è limitata da quella del bus.

- **Commutazione attraverso rete di interconnessione.** Quando un pacchetto giunge a una porta di ingresso A e deve essere inoltrato alla porta Y, il controller chiude l'incrocio di A e Y e la porta A invia il pacchetto sul suo bus e solo il bus Y lo riceverà. Si noti che un pacchetto dalla porta B può essere inoltrato a X nello stesso tempo, perché i pacchetti A-Y e B-X usano bus di input e output diversi. È quindi in grado di inoltrare più pacchetti in parallelo; tuttavia, se due pacchetti provenienti da due diverse porte di input sono destinati alla stessa porta di output, uno dovrà accodarsi alla porta di input, perché solo un pacchetto alla volta può essere inoltrato su uno specifico bus.



## Elaborazione alle porte di uscita

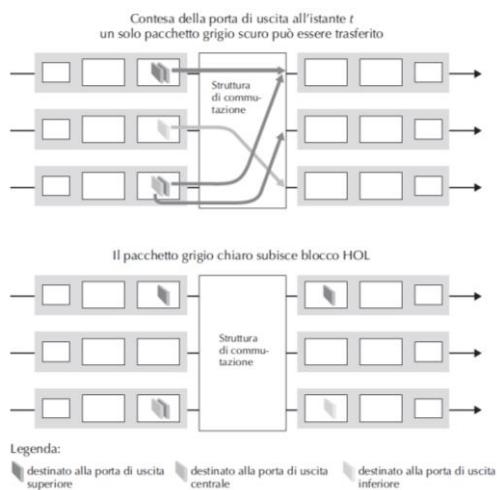
L'**elaborazione alle porte di uscita** prende i pacchetti dalla memoria della porta di uscita e li trasmette sul collegamento di uscita. Questo comprende selezionare e togliere dalla coda i pacchetti per la trasmissione e operare le necessarie funzioni a livello di collegamento e fisico.

## Dove si verifica l'accodamento?

Si possono formare code di pacchetti sia presso le porte di ingresso sia presso quelle di uscita. Il luogo e la lunghezza della coda (sia alle porte di input che di output) dipendono dalla quantità di traffico di rete, dalle velocità relative della struttura di commutazione e dalla linea. Quando queste code crescono, la memoria del router può esaurirsi e quindi può avvenire una perdita di pacchetti nel caso non vi sia memoria per immagazzinare quelli in arrivo.

Ma che cosa avviene se la struttura di commutazione non è sufficientemente rapida (rispetto alle linee in ingresso) nel trasferire tutti i pacchetti in arrivo senza ritardo? In questo caso può verificarsi **accodamento anche alle porte di ingresso**.

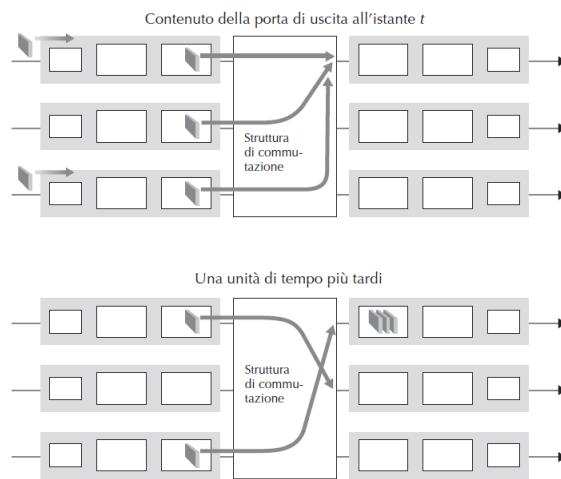
La figura mostra un esempio in cui due pacchetti in testa alle rispettive code di ingresso sono destinati alla stessa porta di uscita, in alto a destra. Supponiamo che la struttura di commutazione scelga di trasferire il primo pacchetto della coda in alto a sinistra. In questo caso, il pacchetto nella coda in basso a sinistra deve attendere, così come quello grigio più chiaro, che si trova dietro di lui, anche se non si verifica contesa per la porta di uscita centrale (che rappresenta la destinazione del pacchetto di colore chiaro). Questo fenomeno è noto come **blocco in testa alla coda (HOL, head-of-the-line blocking)**: un pacchetto nella coda di ingresso deve attendere il trasferimento attraverso la struttura (anche se la propria porta di destinazione è libera) in quanto risulta bloccato da un altro pacchetto che lo precede.



Ma che **cosa avviene alle porte di uscita?** Dato che la porta di uscita può trasmettere un solo pacchetto in un intervallo prestabilito (tempo di trasmissione del pacchetto), gli  $N$  pacchetti in arrivo dovranno mettersi in coda per la trasmissione sul collegamento in uscita. Di conseguenza, è possibile che giungano altri  $N$  pacchetti nel periodo necessario per trasmettere solo uno degli  $N$  pacchetti già accodati, e così via. Quindi è possibile che si

formino code di pacchetti alle porte di uscita anche quando la struttura di commutazione è  $N$  volte più rapida delle velocità di linea delle porte.

Il numero di pacchetti in coda può continuare a crescere fino a esaurire lo spazio di memoria sulla porta di uscita. In assenza di sufficiente memoria per inserire nel buffer il nuovo pacchetto in ingresso occorrerà stabilire se scartarlo (politica nota come ***drop-tail***, eliminazione in coda) o se rimuoverne uno o più, fra quelli già in coda, per far posto al nuovo arrivato. In alcuni casi può risultare vantaggioso eliminare un pacchetto (o marcarne l'intestazione) prima che il buffer sia pieno, al fine di fornire al mittente un segnale di congestione.



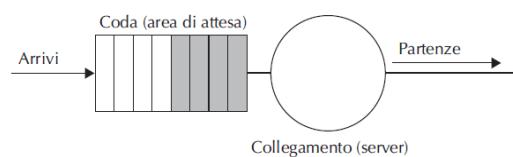
### Dimensione del buffer

Quando vi sono più pacchetti accodati sulle porte di uscita, uno **schedulatore di pacchetti** (*packet scheduler*) deve stabilire in quale ordine trasmetterli. Dato che i buffer nei router sono necessari per assorbire le fluttuazioni del traffico, sorge la domanda su *quanto* debba essere la loro capacità. Per molti anni la regola empirica usata per dimensionare i buffer era che la quantità di memoria riservata al buffer ( $B$ ) dovesse essere uguale a un RTT medio (per esempio 250 ms) per la capacità del collegamento ( $C$ ), ossia  $B = RTT \times C$ . Recenti studi teorici e sperimentali, tuttavia, suggeriscono che quando c'è un gran numero di flussi TCP ( $N$ ) che passano attraverso un collegamento, la quantità di buffer necessaria sia  $B = RTT \times \frac{C}{\sqrt{N}}$ .

### Schedulazione dei pacchetti

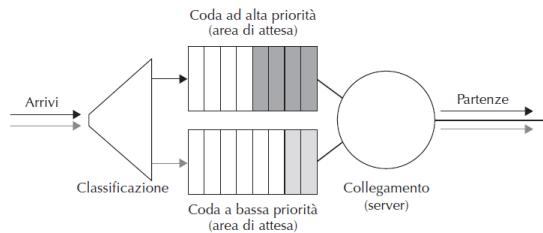
Torniamo ora alla domanda su come determinare l'ordine con cui i pacchetti vengono trasmessi sulla porta di uscita.

- **FIFO (First-Come-First-Out).** I pacchetti che arrivano alla coda di uscita del collegamento aspettano di essere trasmessi se quest'ultimo è occupato nella trasmissione di un altro pacchetto. I pacchetti vengono quindi trasmessi nello stesso ordine con cui sono arrivati in coda.



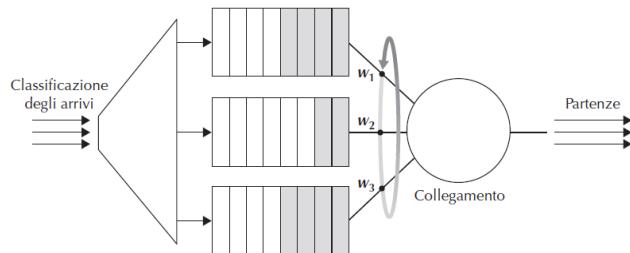
- **Priority queuing.** I pacchetti sono classificati in base a classi di priorità. In pratica, un operatore di rete può configurare una coda in modo che i pacchetti con informazioni di gestione di rete (identificati per esempio tramite il numero di porta UDP/TCP sorgente o di destinazione) abbiano la priorità rispetto al traffico utenti. Ciascuna classe di priorità ha la propria coda. I pacchetti non vengono più trasmessi nell'ordine generale di arrivo, ma selezionando di volta in volta il pacchetto dalla coda non vuota con priorità più alta. Invece, la scelta fra i pacchetti di una classe è di solito effettuata seguendo la strategia FIFO.

Nella modalità di **accodamento a priorità non prelazionabile** la trasmissione dei pacchetti non può essere interrotta una volta iniziata.



- **Round Robin e WFQ.** Anche nella modalità di accodamento round robin (*round robin queuing*), i pacchetti sono suddivisi in classi, ma senza una rigida priorità di servizio. Infatti, è prevista una sorta di sequenza ciclica tra le diverse classi. La forma più semplice di round robin prevede l'alternanza della trasmissione: prima viene inviato un pacchetto della classe 1 e poi uno della classe 2, quindi nuovamente un pacchetto di classe 1, e così via. Nella **modalità conservativa** (*work-conserving round robin*), il collegamento non resta mai inattivo fintanto che ci sono pacchetti, di qualsiasi classe, da trasmettere: se la coda di una classe è vuota viene immediatamente consultata quella successiva.

Nell'**accodamento equo ponderato WFQ** (*Weighted Fair Queuing*) i pacchetti in arrivo sono classificati e accodati in base alla classe. Anche WFQ offre un servizio di tipo ciclico, per cui, nel caso di tre categorie, servirà prima la classe 1, poi la 2 e infine la 3; per ricominciare quindi dal primo gruppo. Inoltre, anche WFQ è conservativa e pertanto quando la coda di una classe è vuota si sposta immediatamente a quella successiva. Diversamente da round robin, con WFQ le varie classi possono ricevere un servizio differenziato. In particolare, a ciascuna classe  $i$  è assegnato un peso  $w_i$ . In qualsiasi momento in cui nella coda  $i$  siano presenti pacchetti che debbano essere spediti, WFQ garantisce che questi ricevano una frazione di servizio pari a:  $w_i / \sum w_j$ , dove al denominatore è indicata la somma effettuata su tutte le classi che hanno pacchetti da trasmettere. Quindi, per un collegamento con capacità trasmissiva  $R$ , la classe  $i$  avrà sempre un rendimento almeno pari a  $R \cdot w_i / \sum w_j$ .

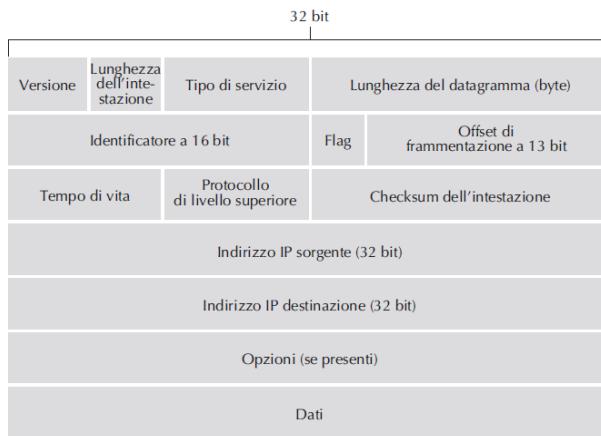


## Formato dei datagrammi IPv4

Ricordiamo che il pacchetto a livello di rete è noto come *datagramma*. I principali campi dei datagrammi IPv4 sono i seguenti:

- **Numero di versione.** Quattro bit che specificano la versione del protocollo IP del datagramma e che consentono al router la corretta interpretazione del datagramma.
- **Lunghezza dell'intestazione (*header length*).** Quattro bit che indicano dove iniziano effettivamente i dati del datagramma. La maggior parte dei datagrammi IP non contiene opzioni, pertanto il tipico datagramma IP ha un'intestazione di 20 byte.
- **Tipo di servizio (TOS *Type Of Service*).** Bit per distinguere diversi tipi di datagrammi (e.g. quelli che richiedono basso ritardo, alto throughput o affidabilità).
- **Lunghezza del datagramma.** Rappresenta la lunghezza totale del datagramma IP, intestazione più dati, misurata in byte. È un campo lungo 16 bit e quindi la massima dimensione dei datagrammi IP è 65.535 byte, anche se raramente questi superano i 1500 in modo da non superare la lunghezza massima del campo dati dei frame Ethernet.
- **Identificatore, flag, offset di frammentazione.** Tre campi che servono per la cosiddetta *frammentazione*.
- **Tempo di vita (TTL *Time To Live*).** Usato per assicurare che i datagrammi non restino in circolazione per sempre nella rete (e.g. a causa di un instradamento ciclico). Questo campo viene decrementato di un'unità ogni volta che il datagramma è elaborato da un router; quando raggiunge 0, il datagramma deve essere scartato.
- **Protocollo.** Indica lo specifico protocollo a livello di trasporto al quale vanno passati i dati del datagramma. Per esempio, il valore 6 indica che i dati sono destinati a TCP, mentre il valore 17 designa UDP.
- **Checksum dell'intestazione.** Consente ai router di rilevare gli errori sui bit dei datagrammi ricevuti. È calcolato trattando ogni coppia di byte dell'intestazione come numeri che sono poi sommati in complemento a 1.
- **Indirizzi IP sorgente e destinazione.** Quando un host crea un datagramma, inserisce il proprio indirizzo IP nel campo indirizzo IP dell'origine e quello della destinazione nel campo indirizzo IP di destinazione.
- **Opzioni.** Questi campi consentono di estendere l'intestazione IP.
- **Dati (*payload*).** Nella maggior parte dei casi, contiene il segmento a livello di trasporto (TCP o UDP) da consegnare alla destinazione; tuttavia, può trasportare anche altri tipi di dati, quali i messaggi ICMP.

I datagrammi IP hanno 20 byte di intestazione (*header*), escludendo le opzioni. I datagrammi non frammentati che trasportano segmenti TCP hanno 40 byte complessivi di intestazione: 20 di intestazione IP più 20 di intestazione TCP, assieme al messaggio di livello applicativo.



### Frammentazione dei datagrammi IPv4

La massima quantità di dati che un frame a livello di collegamento può trasportare è detta **unità massima di trasmissione (MTU, maximum transmission unit)**. Dato che, per il trasporto da un router a un altro, i datagrammi IP sono incapsulati in frame a livello di collegamento, la MTU di questo protocollo pone un limite rigido alla lunghezza dei datagrammi IP. Tale limite non costituisce un problema in sé, ma la difficoltà nasce dal fatto che le tratte del percorso tra mittente e destinatario possono utilizzare differenti protocolli a livello di collegamento e possono avere differenti MTU.

Supponiamo di ricevere un datagramma IP da un collegamento: dalla nostra tabella di inoltro determiniamo il collegamento di uscita e scopriamo che questo ha MTU inferiore alla lunghezza del datagramma IP. Come dividiamo il datagramma nel campo dati del frame a livello di collegamento? La soluzione consiste nel **frammentare i dati** del datagramma IP in due o più datagrammi IP più piccoli, detti **frammenti**, e quindi trasferirli sul collegamento di uscita. I frammenti dovranno però essere riassemblati prima di raggiungere il livello di trasporto alla destinazione.

Quando un host di destinazione riceve una serie di datagrammi dalla stessa origine, deve individuare i frammenti, determinare quando ha ricevuto l'ultimo e stabilire come debbano essere assemblati per formare il datagramma originario.

Quando crea un datagramma, l'host lo contrassegna con un numero identificativo e con gli indirizzi di sorgente e di destinazione. Generalmente, l'host di invio incrementa l'identificativo di ciascun datagramma che invia. Quando il router frammenta il datagramma, contrassegna i frammenti con gli indirizzi di sorgente e di destinazione e con l'identificatore numerico del datagramma originario. Quando la destinazione riceve una serie di datagrammi dallo stesso host mittente, può esaminare gli identificatori per individuare i frammenti di uno stesso datagramma.

## Indirizzamento IPv4

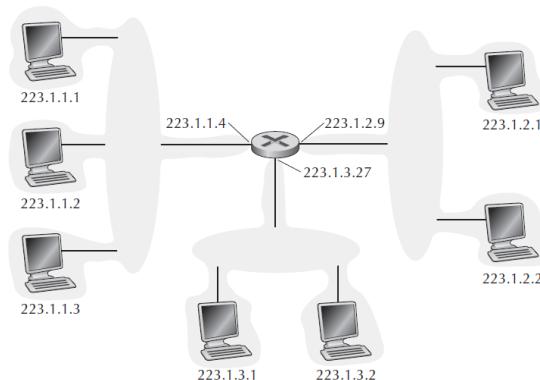
Dato che host e router sono in grado di inviare e ricevere datagrammi, IP richiede che tutte le interfacce abbiano un proprio indirizzo IP. Pertanto, l'indirizzo IP è tecnicamente associato a un'interfaccia, anziché all'host o al router che la contiene.

Gli **indirizzi IP** sono **lunghi 32 bit** (4 byte) e quindi ci sono in totale  $2^{32}$  indirizzi IP, cioè circa 4 miliardi. Tali indirizzi sono solitamente scritti nella cosiddetta **notazione decimale puntata** (*dotted-decimal notation*), in cui ciascun byte dell'indirizzo viene indicato in forma decimale ed è separato con un punto dagli altri byte dell'indirizzo. Ad esempio, l'indirizzo 193.32.216.9 in notazione binaria diventa

11000001 00100000 11011000 00001001

Ogni interfaccia di host e router di Internet ha un indirizzo IP globalmente univoco. Tuttavia, tali indirizzi non possono essere scelti in modo arbitrario. Una parte dell'indirizzo di un'interfaccia è determinata dalla sottorete cui è collegata.

La figura a seguire mostra un router (con tre interfacce) che connette sette host.



Per IP, questa rete che interconnette tre interfacce di host e l'interfaccia di un router forma una **sottorete**. Nella letteratura relativa a Internet le sottoreti sono anche chiamate *reti IP* o semplicemente *reti*. IP assegna a questa sottorete l'indirizzo 223.1.1.0/24, dove la notazione /24, detta anche **maschera di sottorete** (*subnet mask*), indica che i 24 bit più a sinistra dell'indirizzo definiscono l'indirizzo della sottorete.

In generale, possiamo usare la seguente procedura per definire le sottoreti di un sistema. *Per determinare le sottoreti si sgancino le interfacce da host e router in maniera tale da creare isole di reti isolate delimitate dalle interfacce. Ognuna di queste reti isolate viene detta sottorete (subnet).*

La strategia di assegnazione degli indirizzi Internet è detta **classless interdomain routing CIDR** (si pronuncia come l'inglese *cider*). Essa generalizza la nozione di indirizzamento di sottorete. L'indirizzo IP viene diviso in due parti e mantiene la forma decimale puntata  $a.b.c.d/x$ , dove  $x$  indica il numero di bit nella prima parte dell'indirizzo. Gli  $x$  bit più a sinistra di un indirizzo della forma  $a.b.c.d/x$  costituiscono la porzione di rete dell'indirizzo IP e sono spesso detti **prefisso** (di rete) dell'indirizzo. I rimanenti  $32 - x$  bit di un indirizzo possono essere usati per distinguere i dispositivi interni dell'organizzazione,

che hanno tutti lo stesso prefisso di rete. Saranno quindi i router della rete interna che utilizzeranno i restanti bit dell'indirizzo per indirizzarli al dispositivo destinatario.

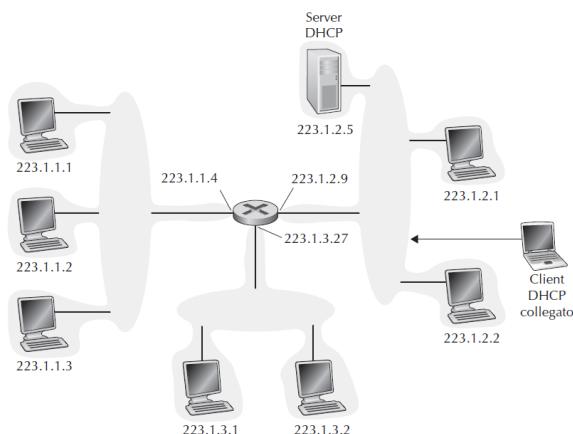
Prima dell'adozione di CIDR, le parti di rete di un indirizzo IP dovevano essere lunghe 8, 16 o 24 bit. Tale schema di indirizzamento era noto come **classful addressing**, dato che le sottoreti con indirizzi di sottorete da 8, 16 e 24 bit erano note rispettivamente come **reti di classe A, B e C**. Il requisito che la parte di sottorete di un indirizzo IP fosse lungo esattamente 1, 2 o 3 byte si rivelò presto problematico nel supportare il numero di organizzazioni in rapida crescita con sottoreti di piccole e medie dimensioni. Una sottorete di classe C (/24) poteva ospitare solo fino a  $2^8 - 2 = 254$  host (due dei 256 indirizzi sono riservati per usi speciali): troppo pochi per molte organizzazioni. D'altro canto, alcune sottoreti di classe B (/16), che possono avere 65.634 host, risultavano sovradimensionate.

Costituirebbe una lacuna non menzionare un altro tipo di indirizzo IP, il cosiddetto indirizzo IP broadcast 255.255.255.255. Quando un host emette un datagramma con destinazione 255.255.255.255, il messaggio viene consegnato a tutti gli host sulla stessa sottorete.

### Come ottenere l'indirizzo di un host: DHCP

Gli indirizzi degli host possono essere configurati manualmente, ma più spesso questo compito è svolto utilizzando il **dynamic host configuration protocol (DHCP)**. DHCP consente a un host di ottenere un indirizzo IP in modo automatico, così come di apprendere informazioni aggiuntive, quali la sua maschera di sottorete, l'indirizzo del router per uscire dalla sottorete (spesso detto *router di default* o *gateway*) e l'indirizzo del suo DNS server locale. L'amministratore di rete può configurare DHCP di modo che un dato host riceva un indirizzo IP persistente, oppure in modo da assegnare a ciascun host che si connette un **indirizzo IP temporaneo**.

DHCP viene spesso detto protocollo **plug-and-play** o **zero-conf** (*zero-configuration*) per la sua capacità di automatizzare la connessione degli host alla rete. È un protocollo client-server; la figura a seguire mostra un server DHCP collegato alla sottorete 223.1.2/24, con il router che opera da agente di relay per i client collegati nelle sottoreti 223.1.1/24 e 223.1.3/24.

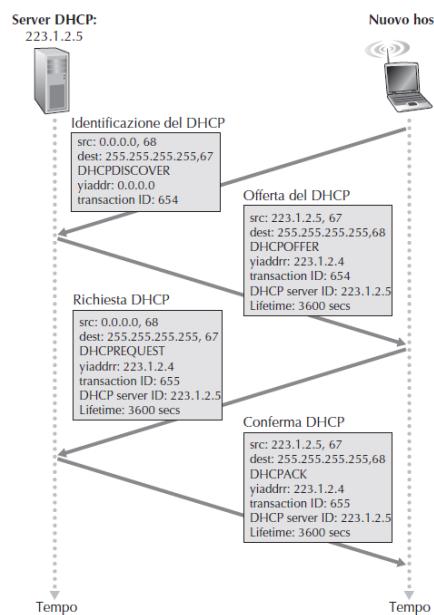


Per i nuovi host, il protocollo DHCP si articola in quattro punti:

- **Individuazione del server DHCP.** Il primo compito di un host appena collegato è l'identificazione del server DHCP con il quale interagire. Questa operazione è svolta

utilizzando un messaggio **DHCP discover**, che un client invia in un pacchetto UDP attraverso la porta 67. Il pacchetto UDP è incapsulato in un datagramma IP. Il client DHCP crea un datagramma IP contenente il suo messaggio DHCP con l'indirizzo IP di destinazione broadcast di 255.255.255.255 e l'indirizzo IP sorgente di 0.0.0.0, cioè “questo host”. Il client DHCP inoltra il datagramma IP al suo livello di collegamento, che invia il frame in broadcast a tutti i nodi collegati alla sottorete.

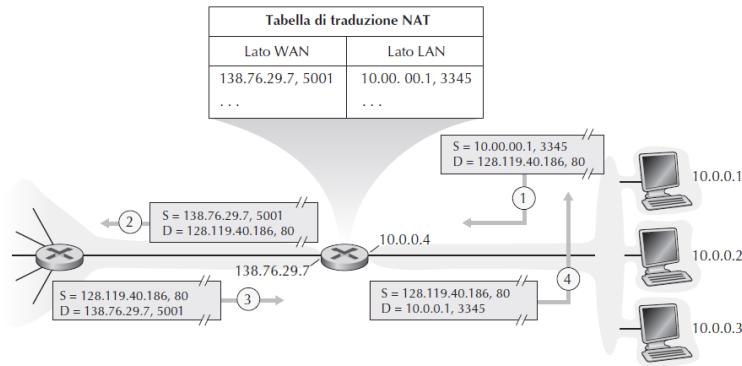
- **Offerta del server DHCP.** Un server DHCP, che riceve un messaggio di identificazione, risponde al client con un messaggio **DHCP offer**, che viene inviato in broadcast a tutti i nodi della sottorete, usando di nuovo l'indirizzo IP broadcast 255.255.255.255. Ciascun messaggio di offerta server contiene l'ID di transazione del messaggio di identificazione ricevuto, l'indirizzo IP proposto al client, la maschera di sottorete e la durata della concessione (**lease time**) dell'indirizzo IP (il lasso di tempo durante il quale l'indirizzo IP sarà valido).
- **Richiesta DHCP.** Il client appena collegato sceglierà tra le offerte dei server e risponderà con un messaggio **DHCP request**, che riporta i parametri di configurazione.
- **Conferma DHCP.** Il server risponde al messaggio di richiesta DHCP con un messaggio **DHCK ACK**, che conferma i parametri richiesti.



Quando il client riceve il DHCP ACK, l'interazione è completata e il client può utilizzare l'indirizzo IP fornito da DHCP per la durata della connessione.

## NAT (Network Address Translation)

La figura a seguire mostra l'attività di un **router abilitato al NAT**, con un'interfaccia che fa parte della rete domestica (sulla destra della figura). Come visto in precedenza, le quattro interfacce della rete domestica hanno lo stesso indirizzo di sottorete, 10.0.0.0/24. Lo spazio di indirizzamento 10.0.0.0/8 è una delle tre parti dello spazio di indirizzi IP riservato alle **reti private**, o **realm** (*realm*) con indirizzi privati: ossia, una rete i cui indirizzi hanno significato solo per i dispositivi interni. Ma se gli indirizzi privati hanno significato solo all'interno di una data rete, come viene gestito l'indirizzamento dei pacchetti relativi all'Internet globale, in cui gli indirizzi sono necessariamente univoci? La risposta è il NAT.



I router abilitati al **NAT** non appaiono come router al mondo esterno, ma si comportano come un unico dispositivo con un unico indirizzo IP. Nella figura tutto il traffico che lascia il router domestico verso Internet ha l'indirizzo IP di origine 138.76.29.7 e tutto il traffico in entrata deve avere lo stesso indirizzo come destinazione. In sostanza, il router abilitato al NAT nasconde i dettagli della rete domestica al mondo esterno.

Se tutti i datagrammi in arrivo al router NAT della rete geografica hanno lo stesso indirizzo IP di destinazione, allora come apprende il router a quale host interno dovrebbe essere inoltrato un determinato datagramma? Il trucco consiste nell'utilizzare una **tavella di traduzione NAT** (*NAT translation table*) nel router NAT e nell'includere nelle righe di tale tabella i numeri di porta oltre che gli indirizzi IP.

Le NAT, tuttavia, **violano il principio end-to-end** dell'architettura a strati: il ruoter, un sistema di livello di rete (livello 3), modifica informazioni riguardanti il livello superiore (livello 4), ovvero il numero di porta dei pacchetti che lo attraversano. Infatti, vengono utilizzati i numeri di porta, propri del livello di trasporto, per indirizzare hosts (e quindi svolgere una funzione del livello di rete). Tale problematica è conosciuta come **NAT traversal problem** e genera difficoltà nell'instaurazione di connessioni da parte di sistemi nella **WAN** (*Wide Area Network*) verso sistemi all'interno della **LAN** (*Local Area Network*). Alcune soluzioni sono le seguenti:

- **Configurazione statica della NAT:** si configura manualmente la tabella di traslazione NAT per soddisfare le esigenze delle applicazioni server eseguite all'interno della LAN.
- **UPnP - IGD (Universal Plug and Play - Internet Gateway Device):** protocollo che permette ai sistemi nella LAN di imparare indirizzi IP pubblici e di modificare la tabella di traslazione NAT.

- **Relaying**: tutti i client si collegano ad un server di *relay*, il quale svolge il ruolo di tramite nella comunicazione. Il server conosce l'indirizzamento, relativo alla NAT o pubblico, di tutti i suoi client.

Nonostante queste problematiche, NAT è diventato un componente importante di Internet, così come altri dispositivi (**middlebox**) che operano a livello di rete, ma hanno funzionalità diverse dai router. I *middlebox* non eseguono il tradizionale inoltro dei pacchetti, ma effettuano altre funzioni, tra le quali NAT, bilanciamento di flussi e firewall.

## Protocollo IPv6

Per soddisfare l'esigenza di un **grande spazio di indirizzamento** venne sviluppato un nuovo protocollo: **IPv6**. La figura seguente mostra il formato dei datagrammi e illustra i cambiamenti più significativi rispetto a IPv4.



- **Indirizzamento esteso.** IPv6 aumenta la dimensione dell'indirizzo IP da 32 a 128 bit, in modo che gli indirizzi IP diventino praticamente inesauribili.
- **Intestazione ottimizzata di 40 byte.** Una serie di campi IPv4 è stata eliminata o resa opzionale. La risultante intestazione a 40 byte e a lunghezza fissa consente una più rapida elaborazione dei datagrammi IP, mentre una nuova codifica delle opzioni ne consente l'elaborazione in maniera più flessibile.
- **Etichettatura dei flussi.** IPv6 presenta una definizione elusiva di **flusso** (*flow*), in quanto consente "l'etichettatura di pacchetti che appartengono a flussi particolari per i quali il mittente richiede una gestione speciale, come una qualità di servizio diversa da quella di default o un servizio in tempo reale". Non sono considerate come flusso le applicazioni più tradizionali, quali il trasferimento file e la posta elettronica.
- **Versione.** Campo a 4 bit che identifica il numero di versione IP.
- **Classe di traffico.** Campo a 8 bit, simile al campo TOS di IPv4, che può essere utilizzato per attribuire priorità a determinati datagrammi all'interno di un flusso o provenienti da specifiche applicazioni rispetto a quelli di altri servizi.
- **Etichetta di flusso.** Campo a 20 bit utilizzato per identificare un flusso di datagrammi.
- **Lunghezza del payload.** Valore a 16 bit trattato come un intero senza segno che indica il numero di byte nel datagramma IPv6 che seguono l'intestazione a lunghezza fissa di 40 byte.
- **Intestazione successiva.** Campo che identifica il protocollo a cui verranno consegnati i contenuti (campo dati) del datagramma.

- **Limite di hop.** Il contenuto di questo campo è decrementato di 1 da ciascun router che inoltra il datagramma. Quando il suo valore raggiunge 0, il datagramma viene eliminato.
- **Indirizzi sorgente e destinazione.**
- **Dati.**

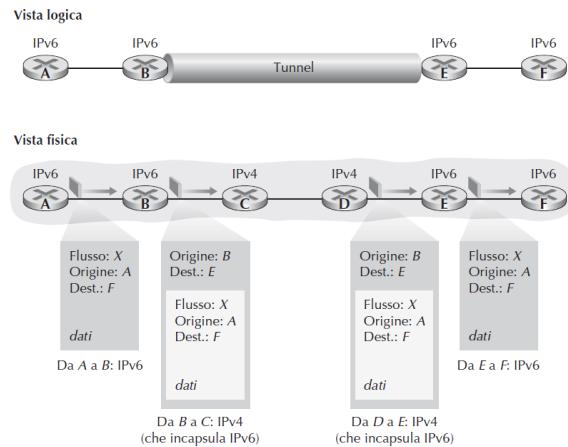
Possiamo quindi notare che alcuni campi sono stati eliminati.

- **Frammentazione/riassemblaggio.** IPv6 non consente frammentazione né riassemblaggio sui router intermedi; queste operazioni possono essere effettuate soltanto da sorgente o destinazione. Se un router riceve un datagramma IPv6 che risulta troppo grande per essere inoltrato sul collegamento di uscita, non fa altro che eliminarlo e inviare al mittente un messaggio d'errore ICMP “Pacchetto troppo grande”.
- **Checksum dell'intestazione.** I protocolli Internet a livello di trasporto (e.g. TCP e UDP) e di collegamento (e.g. Ethernet) calcolano un loro checksum; quindi, questa funzionalità è stata ritenuta talmente ridondante nel livello di rete da decidere di rimuoverla.
- **Opzioni.** Questo campo non è del tutto scomparso, ma è una delle possibili intestazioni successive cui punta l'intestazione IPv6. In altre parole, proprio come le intestazioni del protocollo TCP o UDP possono rappresentare l'intestazione successiva all'interno di un pacchetto IP, lo stesso accade anche per il campo Opzioni. Grazie all'eliminazione di tale campo, la lunghezza dell'intestazione IP è fissata a 40 byte.

Approfondiamo ora il **passaggio da IPv4 a IPv6**. Mentre i nuovi sistemi IPv6 sono retrocompatibili, ossia sono in grado d'inviare, instradare e ricevere datagrammi IPv4, i sistemi IPv4 esistenti non sono in grado di gestire datagrammi IPv6.

L'approccio alla transizione da IPv4 a IPv6 più diffusamente adottato è noto come **tunneling**. Supponiamo che due nodi IPv6 (B ed E nella figura) vogliano utilizzare datagrammi IPv6, ma siano connessi da un insieme di router intermedi IPv4, che chiameremo **tunnel**. Il Nodo B, al lato di invio del tunnel, prende l'*intero* datagramma IPv6 pervenutogli da A e lo pone nel campo dati di un datagramma IPv4. Quest'ultimo viene quindi indirizzato al Nodo E, al lato di ricezione del tunnel e inviato al primo nodo nel tunnel (C). I router IPv4 intermedi instradano il datagramma IPv4, come farebbero per qualsiasi altro datagramma, ignari che questo contenga un datagramma IPv6 completo. Il nodo IPv6, sul lato di ricezione del tunnel, riceverà quindi il datagramma IPv4, determinerà che questo contiene uno IPv6 osservando che il valore del campo numero di protocollo nel pacchetto

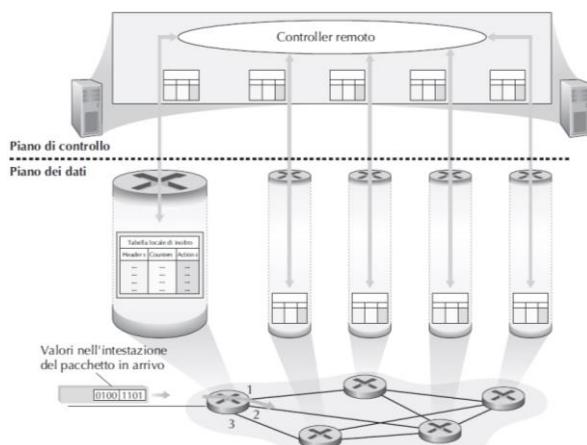
IPv4 è 41 corrispondente a payload IPv4, lo estrarrà e lo instraderrà esattamente come se l'avesse ricevuto da un nodo IPv6 adiacente.



### Inoltro generalizzato e SDN

Ricordiamo che l'inoltro basato sulla destinazione è caratterizzato da due passi: ricerca dell'indirizzo IP di destinazione (*match*), quindi invio del pacchetto attraverso la struttura di commutazione a una specifica porta di uscita (*action*).

Nell'**inoltro generalizzato**, una tabella match-action generalizza il concetto di tabella di inoltro basata sulla destinazione. Poiché le decisioni di inoltro possono essere effettuate utilizzando indirizzi di sorgente e destinazione del livello di rete e/o del livello di collegamento, i dispositivi di inoltro mostrati nella figura seguente sono denominati **packet switch** piuttosto che "router" di livello 3 o "switch" di livello 2.



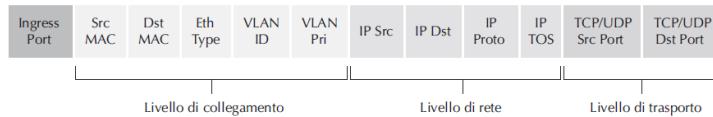
La trattazione seguente sull'inoltro generalizzato sarà basata su **OpenFlow**. Ogni occorrenza (riga) in una tabella di inoltro match-action, nota come **tabella dei flussi (flow table)** in OpenFlow, contiene quanto segue.

- *Un insieme di valori dei campi dell'intestazione* con i quali il pacchetto entrante verrà confrontato. Un pacchetto che non abbia riscontro in alcuna occorrenza della tabella dei flussi può essere scartato o inviato a un controller remoto per ulteriori elaborazioni.
- *Un insieme di contatori* che vengono aggiornati quando i pacchetti vengono associati a un'occorrenza nella tabella dei flussi. Tali contatori possono contenere il numero di

pacchetti associati a tale occorrenza e l'informazione temporale sull'ultimo aggiornamento dell'occorrenza.

- *Un insieme di azioni* che devono essere intraprese quando un pacchetto è associato a un'occorrenza della tabella dei flussi. Tali azioni potrebbero essere l'inoltro di un pacchetto a una data porta di uscita, lo scarto del pacchetto, l'effettuazione delle copie del pacchetto e il loro invio a più porte di uscita e/o la riscrittura di alcuni campi dell'intestazione.

La figura a seguire mostra gli undici campi dell'intestazione di un pacchetto e l'ID della porta di ingresso che possono essere confrontati in una regola match-action di OpenFlow.



Come prima osservazione notiamo che l'astrazione del match di OpenFlow permette che un match venga effettuato sui campi delle intestazioni di tre livelli di protocollo (in contraddizione con il *principio di stratificazione*). Compiendo la funzione di inoltro sulla base dell'indirizzo Ethernet piuttosto che dell'indirizzo IP, possiamo dire che un dispositivo abilitato a OpenFlow può funzionare sia come un router, dispositivo di livello 3 che inoltra datagrammi, sia come uno switch, dispositivo di livello 2 che inoltra frame.

Si osservi infine che non tutti i campi dell'intestazione IP possono essere confrontati. Per esempio, OpenFlow non permette il confronto sulla base di TTL o lunghezza del datagramma.

Di seguito alcuni esempi.

#### Destination-based forwarding:

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | VLAN Pri | IP Src | IP Dst | IP Prot  | IP ToS | TCP s-port | TCP d-port | Action |
|-------------|---------|---------|----------|---------|----------|--------|--------|----------|--------|------------|------------|--------|
| *           | *       | *       | *        | *       | *        | *      | *      | 51.6.0.8 | *      | *          | *          | port6  |

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

#### Firewall:

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | VLAN Pri | IP Src | IP Dst | IP Prot | IP ToS | TCP s-port | TCP d-port | Action |
|-------------|---------|---------|----------|---------|----------|--------|--------|---------|--------|------------|------------|--------|
| *           | *       | *       | *        | *       | *        | *      | *      | *       | *      | *          | *          | drop   |

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

| Switch Port | MAC src | MAC dst | Eth type | VLAN ID | VLAN Pri | IP Src      | IP Dst | IP Prot | IP ToS | TCP s-port | TCP d-port | Action |
|-------------|---------|---------|----------|---------|----------|-------------|--------|---------|--------|------------|------------|--------|
| *           | *       | *       | *        | *       | *        | 128.119.1.1 | *      | *       | *      | *          | *          | drop   |

Block (do not forward) all datagrams sent by host 128.119.1.1

#### Layer 2 destination-based forwarding:

| Switch Port | MAC src | MAC dst               | Eth type | VLAN ID | VLAN Pri | IP Src | IP Dst | IP Prot | IP ToS | TCP s-port | TCP d-port | Action |
|-------------|---------|-----------------------|----------|---------|----------|--------|--------|---------|--------|------------|------------|--------|
| *           | *       | 22:A7:23:<br>11:E1:02 | *        | *       | *        | *      | *      | *       | *      | *          | *          | port3  |

layer 2 frames with destination MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

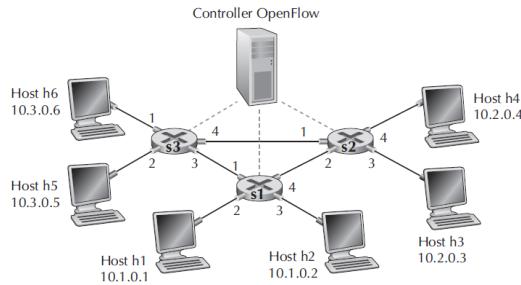
Come mostrato due figure fa, ogni occorrenza della tabella dei flussi ha una lista di azioni che determinano l'elaborazione da effettuare su un pacchetto che le corrisponde. In caso vi siano molteplici azioni, vengono effettuate nell'ordine specificato nella lista.

Di seguito sono elencate alcune delle più importanti azioni possibili.

- Inoltro (*forwarding*). Un pacchetto in entrata può essere inoltrato ad una particolare porta di uscita, inviato in broadcast a tutte le porte eccetto quella da cui è entrato o inviato in multicast ad un insieme selezionato di porte. Il pacchetto può essere incapsulato ed inviato al controller remoto del dispositivo. Il controller può quindi effettuare alcune azioni sul pacchetto, quali installare nuove occorrenze nella tabella dei flussi o restituire il pacchetto al dispositivo per effettuare l'inoltro in base alle regole aggiornate della tabella dei flussi.
- Scarto (*dropping*). Un'occorrenza della tabella dei flussi senza azioni indica che il pacchetto dovrebbe essere scartato.
- Modifica dei campi (*modify-field*). I valori nei dieci campi dell'intestazione del pacchetto possono essere riscritti prima che il pacchetto venga inoltrato alla porta di uscita selezionata.

### Esempio del paradigma match-action in OpenFlow

Applichiamo ora l'inoltro generalizzato nel contesto della rete di esempio mostrata in figura, costituita da 6 host e 3 packet switch, ognuno con 4 interfacce locali numerate da 1 a 4.



Supponiamo che i pacchetti da *h5* o *h6* destinati a *h3* o *h4* debbano essere inoltrati da *s3* a *s1* e quindi da *s1* a *s2*, evitando quindi l'uso del collegamento tra *s3* e *s2*. L'occorrenza della tabella dei flussi in *s1* sarebbe:

| s1 Flow Table (Example 1)                                |            |
|--|------------|
| Match  | Action     |
| Ingress Port = 1 ; IP Src = 10.3.*.* ; IP Dst = 10.2.*.* | Forward(4) |
| ...  | ...        |

Ovviamente abbiamo anche bisogno di un'occorrenza della tabella dei flussi in *s3* in modo che i datagrammi da *h5* o *h6* siano inoltrati a *s1* sull'interfaccia di uscita 3:

| s3 Flow Table (Example 1)             |            |
|---------------------------------------|------------|
| Match                                 | Action     |
| IP Src = 10.3.*.* ; IP Dst = 10.2.*.* | Forward(3) |
| ...                                   | ...        |

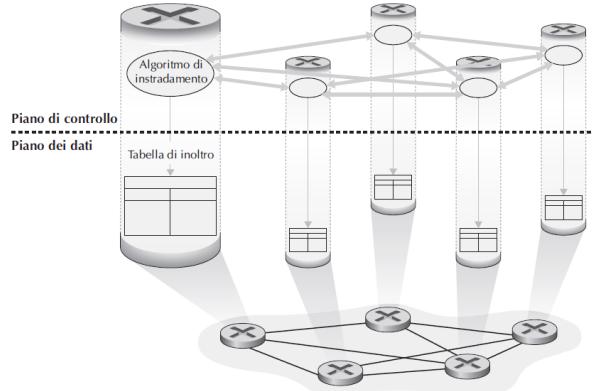
Infine, abbiamo bisogno di un'occorrenza della tabella dei flussi in *s2* in modo che i datagrammi inviati da *s1* siano inviati alla loro destinazione, *h3* o *h4*:

| s2 Flow Table (Example 1)            |            |
|--------------------------------------|------------|
| Match                                | Action     |
| Ingress port = 2 ; IP Dst = 10.2.0.3 | Forward(3) |
| Ingress port = 2 ; IP Dst = 10.2.0.4 | Forward(4) |
| ...                                  | ...        |

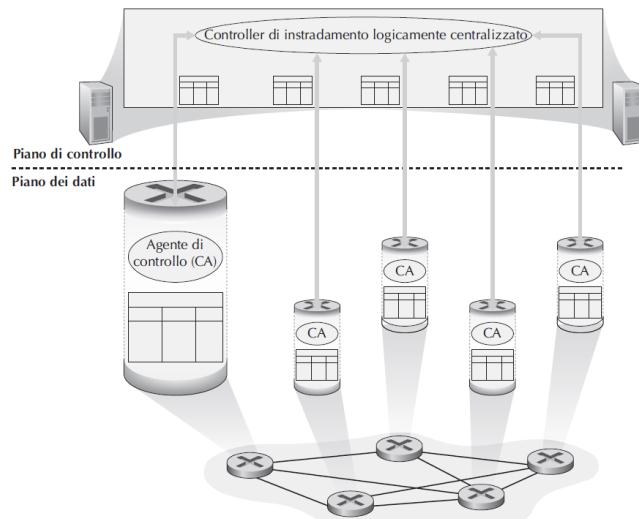
## Network Layer: Control Plane

In questo capitolo studieremo come le tabelle di inoltro e dei flussi vengono calcolate, mantenute e installate. Esistono due possibili approcci.

- **Controllo locale.** L'algoritmo di instradamento viene eseguito su ogni singolo router, all'interno del quale vengono effettuate sia le funzioni di inoltro (piano dei dati) che quelle di instradamento (piano di controllo). Ogni router ha una componente di instradamento che comunica con le componenti di instradamento degli altri router per calcolare la propria tabella di inoltro.



- **Controllo logicamente centralizzato.** Il controller logicamente centralizzato calcola e distribuisce le tabelle di inoltro che devono essere utilizzate da ogni router. L'astrazione del match-action permette che il router effettui l'inoltro IP tradizionale insieme a molte altre funzioni (distribuzione del carico, firewalling e NAT) che prima erano implementate in middlebox separate.



Il controller interagisce con l'agente di controllo (*CA, control agent*) in ogni router tramite un protocollo che configura e gestisce la tabella dei flussi del router. Tipicamente, il CA ha funzionalità minime: comunica con il controller ed esegue quello che il controller gli ordina. Al contrario degli algoritmi di instradamento, gli agenti di controllo non interagiscono direttamente tra di loro e non partecipano attivamente all'elaborazione della tabella di inoltro. Questa è una distinzione chiave tra il controllo locale e il controllo logicamente centralizzato.

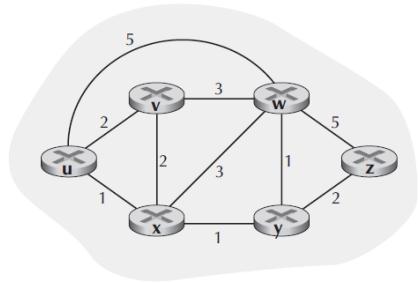
## Algoritmi di instradamento

In questo paragrafo studieremo gli **algoritmi di instradamento** (*routing algorithm*), il cui scopo è determinare i percorsi, o cammini, tra le sorgenti e i destinatari, attraverso la rete di router.

È sempre necessario avere una sequenza ben definita di router che il pacchetto attraversa viaggiando dall'host sorgente all'host destinatario, sia che il piano di controllo adotti un approccio per router sia che adotti un approccio logicamente centralizzato.

Per formulare i problemi di instradamento si utilizza un **grafo**. Ricordiamo che un grafo  $G = (N, E)$  è un insieme  $N$  di nodi e un insieme  $E$  di archi (*edge*), ove ciascun arco collega una coppia di nodi di  $N$ . Nel contesto dell'instradamento a livello di rete, i nodi del grafo rappresentano i router – che prendono decisioni sull'inoltro dei pacchetti – e gli archi che connettono tali nodi rappresentano i collegamenti fisici tra i router.

Come mostrato nella figura, a un arco è anche associato un valore che ne indica il costo. In genere, questo può riflettere la lunghezza fisica del collegamento corrispondente, la velocità del collegamento o il suo prezzo. Per ogni arco  $(x, y)$  tra i nodi  $x$  e  $y$  denotiamo con  $c(x, y)$  il suo **costo**. Se  $(x, y) \notin E$ , poniamo  $c(x, y) = +\infty$ .



Un nodo  $y$  viene detto **adiacente** o **vicino** (*neighbor*) a un nodo  $x$  se  $(x, y)$  è un arco in  $E$ .

L'obiettivo naturale di un algoritmo di instradamento è l'individuazione dei percorsi meno costosi tra sorgenti e destinazioni. Per rendere questo problema più preciso, ricordiamo che un percorso in un grafo  $G = (N, E)$  è una sequenza di nodi  $(x_1, x_2, \dots, x_p)$  tali che ciascuna delle coppie  $(x_1, x_2), (x_2, x_3), \dots, (x_{p-1}, x_p)$  sia un arco appartenente a  $E$ . Il **costo di un percorso**  $(x_1, x_2, \dots, x_p)$  è semplicemente la somma di tutti i costi degli archi lungo il percorso.

Dati due nodi qualsiasi  $x$  e  $y$ , esistono più percorsi che li congiungono, ciascuno con il proprio costo. Uno o più di tali percorsi rappresenta un **percorso a costo minimo** (*least-cost path*). Il problema è quindi chiaro: determinare un percorso tra l'origine e la destinazione che abbia il costo minimo.

Si noti che se tutti gli archi del grafo hanno lo stesso costo, il percorso a costo minimo rappresenta anche il **percorso più breve** (*shortest path*), ossia il percorso con il minor numero di collegamenti tra sorgente e destinazione.

In generale, gli algoritmi di instradamento sono classificabili come centralizzati o decentralizzati.

- Un **algoritmo di instradamento centralizzato** calcola il percorso a costo minimo tra una sorgente e una destinazione avendo una conoscenza globale e completa della rete. In altre parole, l'algoritmo riceve in ingresso tutti i collegamenti tra i nodi e i loro costi. Gli algoritmi con informazioni di stato globali sono spesso detti **algoritmi link-state** (LS, o “con stato del collegamento”), dato che l'algoritmo deve essere consci del costo di ciascun collegamento della rete.

- In un **algoritmo di instradamento decentralizzato**, il percorso a costo minimo viene calcolato in modo distribuito e iterativo. Nessun nodo possiede informazioni complete sul costo di tutti i collegamenti di rete. Inizialmente i nodi conoscono soltanto i costi dei collegamenti a loro incidenti. Poi, attraverso un processo iterativo e lo scambio di informazioni con i nodi adiacenti, un nodo gradualmente calcola il percorso a costo minimo verso una destinazione o un insieme di destinazioni. Tali algoritmi decentralizzati, che prevedono scambi interattivi tra router vicini, possono essere implementati nei piani di controllo nei quali i router interagiscono direttamente, mentre hanno poco senso nel caso di controllo centralizzato.

Un secondo criterio per classificare gli algoritmi di instradamento riguarda il fatto di essere statici o dinamici. Negli **algoritmi di instradamento statici** i percorsi cambiano molto raramente, spesso come risultato di un intervento umano (per esempio, la modifica manuale di una tabella di inoltro di un router). Gli **algoritmi di instradamento dinamici** invece determinano gli instradamenti al variare del volume di traffico o della topologia della rete. Questi ultimi rispondono meglio ai cambiamenti della rete, ma sono anche maggiormente soggetti a problemi quali l'instradamento in loop e l'oscillazione dei percorsi.

Un terzo modo per classificare gli algoritmi di instradamento è il fatto di essere più o meno sensibili (**load-sensitive** o **load-insensitive**) al carico della rete. In un **algoritmo sensibile al carico** i costi dei collegamenti variano dinamicamente per riflettere il livello corrente di congestione.

Gli attuali algoritmi di instradamento Internet (quali RIP, OSPF e BGP) sono algoritmi insensibili al carico, dato che il costo di un collegamento non riflette esplicitamente il suo attuale (o recente) livello di congestione.

### Instradamento “link-state” (LS)

In un **intradamento link-state** la topologia di rete e tutti i costi dei collegamenti sono noti, ossia disponibili in input all'algoritmo. Ciò si ottiene facendo inviare a ciascun nodo pacchetti sullo stato dei suoi collegamenti a *tutti* gli altri nodi della rete. Questi pacchetti contengono identità e costi dei collegamenti connessi al nodo che li invia.

L'algoritmo di calcolo dei percorsi che presentiamo associato a questo instradamento è noto come **algoritmo di Dijkstra**: calcola il percorso a costo minimo da un nodo (l'origine, che chiameremo  $u$ ) a tutti gli altri nodi nella rete, è iterativo e ha le seguenti proprietà: dopo la  $k$ -esima iterazione, i percorsi a costo minimo sono noti a  $k$  nodi di destinazione e, tra i percorsi a costo minimo verso tutti i nodi di destinazione, questi  $k$  percorsi hanno i  $k$  costi più bassi. Adottiamo la seguente notazione:

- $D(v)$ : costo minimo del percorso dal nodo origine alla destinazione  $v$  per quanto concerne l'iterazione corrente dell'algoritmo.
- $p(v)$ : immediato predecessore di  $v$  lungo il percorso a costo minimo dall'origine a  $v$ .
- $N'$ : sottoinsieme di nodi contenente tutti (e solo) i nodi  $v$  per cui il percorso a costo minimo dall'origine a  $v$  è definitivamente noto.

```

Algoritmo Dijkstra dal nodo origine u
1 Inizializzazione:
2   N' = {u}
3   per tutti i nodi v
4     se v è adiacente a u
5       allora D(v) = c(u,v)
6     altrimenti D(v) =
7
8 Ciclo
9   determina un w non in N' tale che D(w) sia minimo
10  aggiungi w a N'
11  aggiorna D(v) per ciascun nodo v adiacente a w e non in N':
12    D(v) = min ((D(v), D(w) + c(w,v))
13 /* il nuovo costo verso v è il vecchio costo verso v oppure
14   il costo del percorso minimo noto verso w più il costo da w a v */
15  ripeti in ciclo finché non si verifica che N' = N

```

| Passo | $N'$   | $D(v), p(v)$ | $D(w), p(w)$ | $D(x), p(x)$ | $D(y), p(y)$ | $D(z), p(z)$ |
|-------|--------|--------------|--------------|--------------|--------------|--------------|
| 0     | u      | 2,u          | 5,u          | 1,u          | $\infty$     | $\infty$     |
| 1     | ux     | 2,u          | 4,x          |              | 2,x          | $\infty$     |
| 2     | uxy    | 2,u          | 3,y          |              |              | 4,y          |
| 3     | uxyv   |              | 3,y          |              |              | 4,y          |
| 4     | uxyvw  |              |              |              |              | 4,y          |
| 5     | uxyvwz |              |              |              |              |              |

Consideriamo per esempio la rete vista in precedenza e calcoliamo i percorsi a costo minimo da  $u$  a tutte le destinazioni.

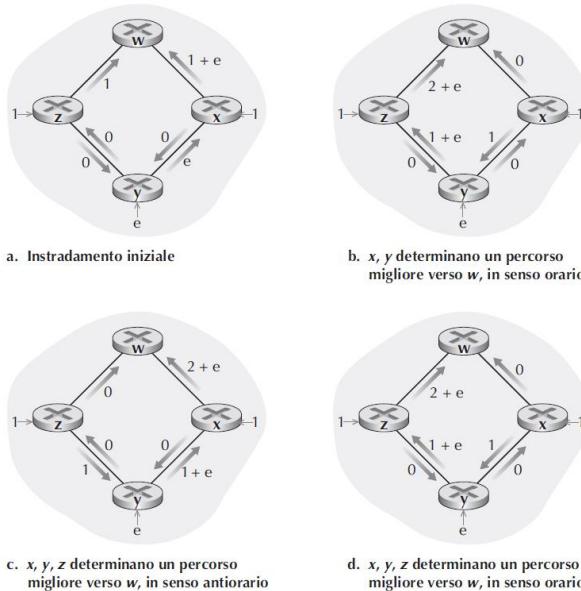
1. Nel passo di inizializzazione i valori dei percorsi a costo minimo noti da  $u$  ai suoi nodi adiacenti ( $v, w$  e  $x$ ) sono posti rispettivamente a 2, 5 e 1. In particolare, il costo verso  $w$  vale 5 dato che questo è il costo del collegamento diretto (un solo hop) da  $u$  a  $w$ . I costi verso  $y$  e  $z$  sono posti a infinito dato che tali nodi non sono adiacenti a  $u$ .
2. Nella prima iterazione prendiamo in considerazione i nodi non ancora aggiunti all'insieme  $N'$  e determiniamo il nodo a costo minimo come alla fine della precedente iterazione. Tale nodo è  $x$ , di costo 1, e pertanto  $x$  viene aggiunto all'insieme  $N'$ . Viene poi eseguita la riga 12 dell'algoritmo Dijkstra per aggiornare  $D(v)$  per tutti i nodi, ottenendo i risultati mostrati nella seconda riga (Passo 1) della tabella. Il costo del percorso verso  $v$  non è cambiato. Il costo del percorso verso  $w$  (che era 5 al termine dell'inizializzazione) passando per il nodo  $x$  è diventato 4. Viene quindi selezionato questo percorso a costo inferiore e il predecessore di  $w$  lungo il percorso minimo da  $u$  diventa  $x$ . Analogamente, il costo verso  $y$  (attraverso  $x$ ) viene calcolato essere 2, e la tabella viene aggiornata di conseguenza.
3. Nella seconda iterazione si trova che i nodi  $v$  e  $y$  hanno percorsi a costo minimo (2), ne sceglio arbitrariamente uno e aggiungiamo  $y$  all'insieme  $N'$  che ora conterrà  $u$ ,  $x$  e  $y$ . I costi verso i nodi rimanenti non ancora in  $N'$ , ossia  $v, w$  e  $z$ , sono aggiornati dalla riga 12 dell'algoritmo Dijkstra, ottenendo i risultati mostrati nella terza riga della tabella.
4. E così via...

Quando l'algoritmo Dijkstra termina, abbiamo per ciascun nodo il suo predecessore lungo il percorso a costo minimo dal nodo origine. Per ciascun predecessore abbiamo il rispettivo predecessore, e in questo modo riusciamo a costruire l'intero percorso dall'origine a tutte le destinazioni.

La complessità computazionale di questo algoritmo è  $O(n^2)$ . Un'implementazione più sofisticata di questo algoritmo, utilizzando una struttura dati nota come heap, riesce a determinare il minimo nella riga 9 in un tempo logaritmico anziché lineare, riducendo la complessità.

Consideriamo una situazione “patologica” che può manifestarsi. In questo esempio, i costi dei collegamenti non sono simmetrici; in altre parole,  $c(u, v)$  risulta uguale a  $c(v, u)$  solo se il carico trasportato in entrambe le direzioni del collegamento  $(u, v)$  è lo stesso. Inoltre, il nodo  $z$  e quello  $x$  danno origine a un’unità di traffico ciascuno verso  $w$ , mentre  $y$  invia una quantità di traffico pari a  $e$ , anche questo verso  $w$ .

Nella successiva iterazione dell’algoritmo, il nodo  $y$  determina che il percorso in senso orario verso  $w$  ha costo 1, mentre il percorso in senso antiorario ha costo pari a  $1 + e$ . Pertanto, il percorso a costo minimo di  $y$  verso  $w$  ora è quello in senso orario. Analogamente,  $x$  determina che il proprio nuovo percorso a costo minimo verso  $w$  è quello in senso orario. Quando l’algoritmo viene nuovamente eseguito,  $x$ ,  $y$  e  $z$  instradano tutti il proprio traffico su percorsi in senso antiorario. La volta successiva che l’algoritmo viene eseguito, tutti i nodi  $x$ ,  $y$  e  $z$  instradano il loro traffico in senso orario.



Che cosa si può fare per evitare tali oscillazioni? Una soluzione consiste nello stabilire che i costi dei collegamenti non dipendono dalla quantità di traffico trasportato: ipotesi inaccettabile dato che uno scopo dell’instradamento è evitare collegamenti troppo congestionati (per esempio, ad alto ritardo). Un’altra soluzione consiste nell’assicurarsi che non tutti i router lancino l’esecuzione dell’algoritmo nello stesso istante. Questa sembra essere una soluzione più ragionevole, dato che vorremmo che l’istanza in esecuzione dell’algoritmo non fosse la stessa su ciascun nodo anche se i router eseguissero l’algoritmo con la stessa periodicità.

## Instrandamento “distance-vector” (DV)

Mentre l'instrandamento LS usa informazioni globali, quello **distance-vector** è:

- *distribuito*: ciascun nodo riceve parte dell'informazione da uno o più dei suoi vicini direttamente connessi, a cui, dopo aver effettuato il calcolo, restituisce i risultati;
- *iterativo*: questo processo si ripete fino a quando non avviene ulteriore scambio informativo tra vicini;
- *asincrono*: non richiede che tutti i nodi operino al passo con gli altri.

Prima di presentare l'algoritmo usato dall'instrumento DV sarà bene discutere un'importante relazione esistente tra costi e percorsi a costo minimo. Sia  $d_x(y)$  il costo del percorso a costo minimo dal nodo  $x$  al nodo  $y$ . Allora i costi minimi sono correlati dalla nota **formula di Bellman-Ford**:

$$d_x(y) = \min_y \{c(x, v) + d_v(y)\}$$

Dove  $\min_y$  riguarda tutti i vicini di  $x$ .

La formula di B-F ha anche una significativa importanza pratica, in quanto:

- fornisce le righe della tabella di inoltro nel nodo  $x$ ;
- suggerisce la forma della comunicazione tra vicini che avrà luogo nell'algoritmo usato dal DV, detto appunto **algoritmo di Bellman Ford**.

Con l'algoritmo di B-F, ciascun nodo  $x$  mantiene i seguenti dati di instradamento.

- Per ciascun vicino  $v$ , il costo  $c(x, v)$  da  $x$  al vicino  $y$ .
- Il vettore delle distanze del nodo  $x$ , che è  $D_x = [D_x(y): y \text{ in } N]$ , contenente la stima presso  $x$  del costo verso tutte le destinazioni,  $y$ , in  $N$ .
- I vettori delle distanze di ciascuno dei suoi vicini, ossia  $D_y = [D_y(y): y \text{ in } N]$ , per ciascun nodo vicino  $y$  di  $x$ .

```

Algoritmo Bellman Ford
    A ciascun nodo x:
    1 Inizializzazione:
        2     per tutte le destinazioni y in N:
            3         Dx(y) = c(x,y) /* se y non è adiacente, allora c(x,y) = */
            4         per ciascun vicino w
                5             Dw(y) = ? per tutte le destinazioni y in N
                6             per ciascun vicino w
                    7                 invia il vettore delle distanze Dx = [Dx(y): y in N] a w
                    8
    9 ciclo
        10    attendi (finché vedi cambiare il costo di un collegamento verso
            11        qualche vicino w o finché ricevi un vettore delle distanze
            12        da qualche vicino w)
            13    per ogni y in N:
                14        Dx(y) = minv {c(x,v) + Dv(y)}
            15
            16    se Dx(y) è cambiato per qualche destinazione y
            17        invia il vettore delle distanze Dx = [Dx(y): y in N] a tutti i vicini
            18
    19 ripeti il ciclo indefinitamente

```

---

In questo algoritmo, di quando in quando un nodo invia una copia del proprio vettore delle distanze a ciascuno dei suoi vicini. Quando un nodo  $x$  riceve un nuovo vettore da qualcuno

dei suoi vicini  $v$ , lo salva e quindi usa la formula per aggiornare il proprio vettore come segue:

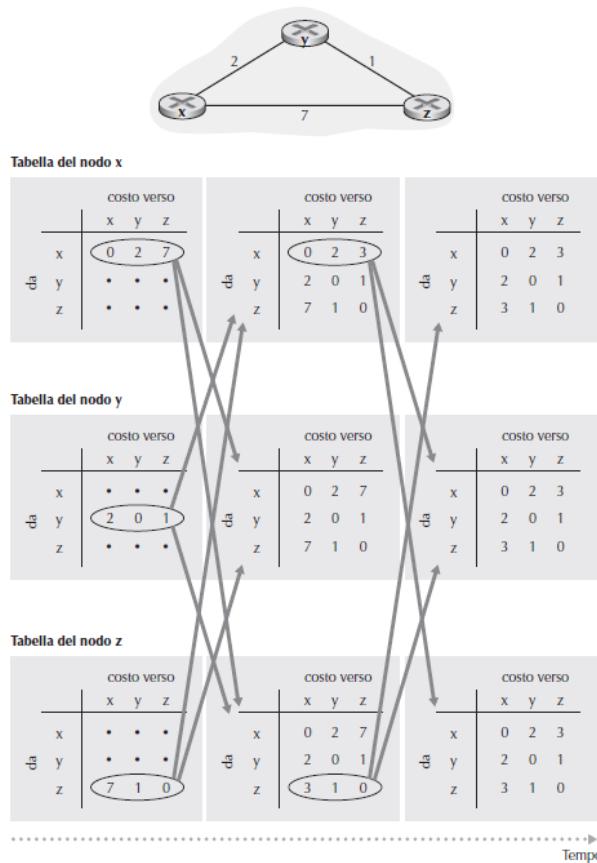
$$D_x(y) = \min_y \{c(x, v) + D_v(y)\} \text{ per ciascun nodo } y \text{ in } N.$$

Se il vettore delle distanze del nodo  $x$  è cambiato per via di tale passo di aggiornamento, il nodo  $x$  manderà il proprio vettore aggiornato a ciascuno dei suoi vicini, i quali a loro volta aggiorneranno il proprio vettore.

Tuttavia, per aggiornare la propria tabella di inoltro per una data destinazione  $y$ , ciò che il nodo  $x$  ha davvero bisogno di sapere non è la distanza sul percorso minimo verso  $y$ , bensì il nodo vicino  $v^*(y)$  che rappresenta il router successivo lungo il percorso più breve verso  $y$ . Come ci si potrebbe aspettare, il router  $v^*(y)$  è il vicino  $v$  che ottiene il valore minimo nella riga 14 dell'algoritmo. Se esistono più vicini  $v$  che ottengono il minimo, allora  $v^*(y)$  può essere uno qualunque tra questi. Pertanto, nelle righe 13 e 14, per ciascuna destinazione  $y$  il nodo  $x$  determina anche  $v^*(y)$  e aggiorna la propria tabella di inoltro per la destinazione  $y$ .

L'algoritmo Bellman-Ford è quindi **decentralizzato**; infatti, le sole informazioni detenute dal nodo sono il costo dei collegamenti verso i vicini direttamente connessi e quelle ricevute da questi vicini. Ogni nodo attende aggiornamenti dai suoi vicini (righe 10, 11 e 12), quando riceve un aggiornamento calcola il proprio nuovo vettore delle distanze (riga 14) e lo distribuisce ai suoi vicini (righe 16 e 17).

La figura a seguire illustra il funzionamento dell'instradamento DV in una semplice rete a tre nodi.



La colonna di sinistra della figura mostra tre **tabelle di instradamento** (*routing table*) iniziali per ciascuno dei tre nodi (per esempio, in alto a sinistra è indicata quella del nodo  $x$ ). All'interno di una specifica tabella di instradamento le righe rappresentano i vettori delle distanze: più precisamente, la tabella di instradamento di ciascun nodo include il proprio vettore delle distanze e quello dei suoi vicini.

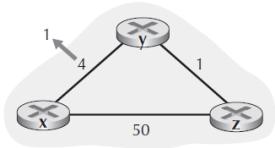
Dopo l'inizializzazione, ciascun nodo invia il proprio vettore ai suoi vicini come mostrato dalle frecce dalla prima alla seconda colonna delle tabelle. Dopo aver ricevuto gli aggiornamenti, i nodi ricalcolano il vettore delle distanze. La seconda colonna pertanto mostra, per ciascun nodo, il nuovo vettore delle distanze del nodo e i vettori delle distanze appena ricevuti dai suoi vicini.

Dopo aver ricalcolato i rispettivi vettori delle distanze, i nodi li inviano in versione aggiornata ai propri vicini (ammesso che siano cambiati), procedura indicata nella figura con le frecce dalla seconda colonna alla terza. Notiamo che solo i nodi  $x$  e  $z$  inviano aggiornamenti: il vettore delle distanze del nodo  $y$  non è cambiato e pertanto non è stato spedito. Dopo la ricezione degli aggiornamenti, i nodi ricalcolano i propri vettori delle distanze e aggiornano le tabelle di instradamento (terza colonna).

Il processo continua finché non viene più inviato alcun messaggio di aggiornamento. A questo punto, l'algoritmo entra in uno stato quiescente; in altre parole, tutti i nodi eseguono l'attesa nelle righe 10 e 11 dell'algoritmo Bellman Ford. L'algoritmo rimane in tale stato finché non cambia il costo di un collegamento.

### Intradamento DS: modifica dei costi e guasti dei collegamenti

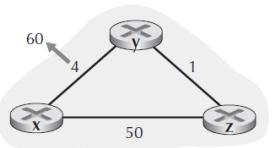
La figura mostra uno scenario in cui il costo del collegamento da  $y$  a  $x$  passa da 4 a 1. In questa sede ci concentriamo solo su  $y$  e sulle righe della tabella delle distanze di  $z$  verso la destinazione  $x$ . L'algoritmo provoca la seguente sequenza di eventi.



- All'istante  $t_0$ ,  $y$  rileva il cambiamento nel costo del collegamento (passato da 4 a 1), aggiorna il proprio vettore delle distanze e informa i vicini del cambiamento.
- All'istante  $t_1$ ,  $z$  riceve l'aggiornamento da  $y$  e aggiorna la propria tabella, calcola un nuovo costo minimo verso  $x$  (che passa da 5 a 2) e invia il nuovo vettore delle distanze ai vicini.
- All'istante  $t_2$ ,  $y$  riceve l'aggiornamento di  $z$  e aggiorna la propria tabella delle distanze. I costi minimi di  $y$  non cambiano e  $y$  non manda alcun messaggio a  $z$ . L'algoritmo entra in uno stato quiescente.

Pertanto, dopo due iterazioni l'algoritmo raggiunge uno stato di quiete. Le buone notizie sul costo diminuito tra  $x$  e  $y$  si sono propagate rapidamente nella rete.

Prendiamo in considerazione ora che cosa può avvenire quando il costo di un collegamento aumenta. Supponiamo che il costo del collegamento tra  $x$  e  $y$  passi da 4 a 60.



1. All'istante  $t_0$ ,  $y$  rileva che il costo del collegamento è passato da 4 a 60 e calcola il suo nuovo percorso a costo minimo verso  $x$  con la formula

$$D_y(x) = \min\{c(y, x) + D_x(x), c(y, z) + D_z(x)\} = \min\{60 + 0, 1 + 5\} = 6$$

Ovviamente, con la nostra visione globale della rete, possiamo rilevare che questo nuovo costo attraverso  $z$  è errato. Ma l'unica informazione che il nodo  $y$  possiede è che il costo diretto verso  $x$  è 60 e che  $z$  ha ultimamente detto a  $y$  di essere in grado di giungere a  $x$  con un costo di 5. Pertanto, al fine di arrivare a  $x$ ,  $y$  ora farebbe passare il percorso per  $z$ , aspettandosi che questo sia in grado di giungere a  $x$  con un costo pari a 5. All'istante  $t_1$ , abbiamo un **instradamento ciclico**: al fine di giungere a  $x$ ,  $y$  fa passare il percorso per  $z$  e  $z$  lo fa passare per  $y$ .

2. Dato che il nodo  $y$  ha calcolato un nuovo costo minimo verso  $x$ , informa  $z$  del suo nuovo vettore delle distanze all'istante  $t_1$ .
3. In un istante successivo a  $t_1$ ,  $z$  riceve il nuovo vettore delle distanze di  $y$ , che indica che il costo minimo di  $y$  verso  $x$  è 6, sa che può giungere a  $y$  a costo 1 e quindi calcola un nuovo costo minimo verso  $x$  pari a  $D_z(x) = \min\{50 + 0, 1 + 6\} = 7$ . Dato che il costo minimo di  $z$  verso  $x$  è aumentato,  $z$  informa  $y$  del suo nuovo vettore delle distanze al tempo  $t_2$ .
4. Analogamente, dopo aver ricevuto il nuovo vettore delle distanze di  $z$ ,  $y$  determina  $D_y(x) = 8$  e invia a  $z$  il suo nuovo vettore delle distanze.  $z$  allora determina  $D_z(x) = 9$  e invia a  $y$  il suo nuovo vettore delle distanze, e così via.

A causa di questi scenari, il processo che abbiamo descritto viene talvolta detto **problema di conteggio all'infinito** (*count to infinity problem*).

Lo specifico scenario con cicli appena descritto può essere evitato utilizzando una tecnica nota come **inversione avvelenata** (*poisoned reverse*). L'idea è semplice: se  $z$  instrada tramite  $y$  per giungere alla destinazione  $x$ , allora  $z$  avvertirà  $y$  che la sua distanza verso  $x$  è infinita, ossia  $z$  comunicherà a  $y$  che  $D_z(x) = \infty$ , anche se in realtà  $z$  sa che  $D_z(x) = 5$ , e continuerà a dire questa piccola bugia fintanto che instrada verso  $x$  passando per  $y$ . Dato che  $y$  crede che  $z$  non abbia un percorso verso  $x$ , non tenterà mai di instradare verso  $x$  passando per  $z$ , per tutto il tempo in cui  $z$  continua a instradare verso  $x$  passando per  $y$  (e mente a riguardo).

I cicli che non riguardano semplicemente due nodi adiacenti non verranno rilevanti dalla tecnica dell'inversione avvelenata.

### Confronto tra gli instradamenti LS e DV

I due meccanismi di instradamento studiati utilizzano approcci complementari nel calcolare i percorsi. Nel DV ciascun nodo dialoga *solo* con i vicini direttamente connessi, informandoli delle stime a costo minimo da sé stesso a *tutti* i nodi (che conosce) nella rete. Nel LS, ciascun nodo dialoga con *tutti* gli altri nodi via broadcast, ma comunica loro *solo* i costi dei collegamenti direttamente connessi.

Ricordiamo che  $N$  rappresenta l'insieme dei nodi (router) ed  $E$  l'insieme degli archi (collegamenti).

- **Complessità dei messaggi.** LS richiede che ciascun nodo conosca il costo collegamento nella rete. Ciò implica l'invio di  $O(|N| \cdot |E|)$  messaggi. A ogni iterazione l'instradamento DV richiede scambi di messaggi tra nodi adiacenti e il tempo richiesto perché l'algoritmo converga può dipendere da molti fattori.
- **Velocità di convergenza.** LS è un algoritmo  $O(|N|^2)$  che richiede  $O(|N| \cdot |E|)$  messaggi. DV può convergere lentamente e può presentare cicli di instradamento. DV presenta anche il problema del conteggio all'infinito.
- **Robustezza.** Che cosa avviene se un router si guasta, funziona male o viene sabotato? Con LS, un router può comunicare via broadcast un costo sbagliato per uno dei suoi collegamenti connessi (ma non per altri). Un nodo può anche alterare o eliminare i pacchetti ricevuti in broadcast LS. Ma i nodi LS si occupano di calcolare soltanto le proprie tabelle di inoltro, e gli altri nodi effettuano calcoli simili per quanto li riguarda. Ciò significa che i calcoli di instradamento sono in qualche modo isolati, il che fornisce un certo grado di robustezza. Con DV, un nodo può comunicare percorsi a costo minimo errati a tutte le destinazioni.

In conclusione, nessuno dei due meccanismi di instradamento è nettamente superiore all'altro ed entrambi vengono utilizzati in Internet.

## Instradamento interno ai sistemi autonomi

Nel nostro studio degli algoritmi di instradamento, ciascun router era indistinguibile dagli altri nel senso che tutti eseguivano lo stesso algoritmo per calcolare l'instradamento attraverso la rete. Tuttavia, questo modello risulta un po' semplicistico per almeno due importanti motivi.

- **Scalabilità.** Al crescere del numero di router, il tempo richiesto per calcolare, memorizzare e comunicare le informazioni di instradamento diventa proibitivo. Attualmente, Internet è costituita da centinaia di milioni di host. Archiviare le informazioni di instradamento su ciascuno di essi richiederebbe chiaramente un'enorme quantità di memoria, e il traffico generato dalla comunicazione broadcast degli aggiornamenti non lascerebbe banda per i pacchetti di dati. Si deve quindi fare qualcosa per ridurre la complessità del calcolo dei percorsi nelle reti di grandi dimensioni.
- **Autonomia amministrativa.** Internet è una rete di ISP che generalmente desiderano gestire liberamente i propri router o nascondere all'esterno aspetti dell'organizzazione interna della rete. L'ideale sarebbe che ciascuno fosse in grado di amministrare la propria rete nel modo auspicato, pur mantenendo la possibilità di connetterla alle reti esterne.

Questi problemi possono essere risolti organizzando i router in **sistemi autonomi (AS, Autonomous System)**, generalmente composti da gruppi di router posti sotto lo stesso controllo amministrativo. Un AS è identificato da un **numero di sistema autonomo (ASN)** univoco che viene assegnato da ICANN, esattamente come gli indirizzi IP.

I router di un AS eseguono lo stesso algoritmo di instradamento e gli uni hanno informazioni sugli altri. L'algoritmo di instradamento in esecuzione in un AS è detto **protocollo di instradamento interno al sistema autonomo (intra-AS routing protocol)**.

## OSPF – Open Shortest Path First

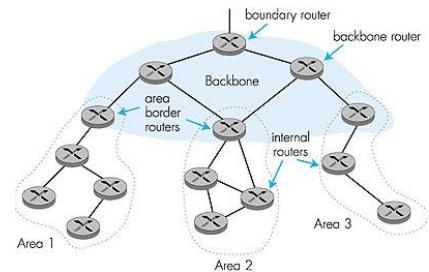
Il termine *open* nell'acronimo indica che le specifiche del protocollo sono pubblicamente disponibili. **OSPF** è un **protocollo link-state** che utilizza il **flooding** (inondazione) di informazioni riguardo lo stato dei collegamenti e l'algoritmo di Dijkstra per la determinazione del percorso a costo minimo. In OSPF, un router costruisce una mappa topologica, cioè un grafo, dell'intero sistema autonomo e manda in esecuzione (locale) l'algoritmo di Dijkstra per determinare un albero dei percorsi minimi verso tutte le sottoreti (albero in cui il router stesso rappresenta il nodo radice).

OSPF non impone una politica sulla scelta dei pesi dei collegamenti (lavoro che compete all'amministratore di rete), ma fornisce invece i meccanismi per determinare l'instradamento con percorso a costo minimo per un dato insieme di pesi dei collegamenti.

Tra i vantaggi di OSPF ricordiamo i seguenti.

- **Sicurezza.** Gli scambi tra router OSPF possono essere autenticati; di conseguenza, soltanto router fidati possono prendere parte al protocollo OSPF in un sistema autonomo, evitando che malintenzionati immettano informazioni errate nelle tabelle dei router.

- **Percorsi con lo stesso costo.** Quando più percorsi verso una destinazione hanno lo stesso costo, OSPF consente di usarli senza doverne scegliere uno per trasportare tutto il traffico.
- **Supporto integrato per l'instradamento unicast e multicast.**
- **Supporto alle gerarchie in un dominio di instradamento.** Un sistema autonomo OSPF può essere configurato in aree che eseguono diversi algoritmi di instradamento OSPF: ciascun router in un'area invia lo stato dei suoi collegamenti agli altri router nell'area. In ogni area, uno o più **router di confine d'area** (*area border router*) si fa carico dell'instradamento dei pacchetti indirizzati all'esterno. Un'area di un sistema autonomo OSPF è configurata per essere l'**area di dorsale** (*backbone area*), il cui ruolo principale è quello di instradare il traffico tra le altre aree nel sistema autonomo. La dorsale contiene sempre tutti i router di confine del sistema autonomo, ma non necessariamente soltanto quelli.



I **messaggi OSPF** sono incapsulati direttamente nel protocollo IP, ovvero non utilizzano il protocollo UDP per essere trasportati. Ogni messaggio può trasportare molteplici **Link State Advertisement (LSA)**. Il tipo di LSA principale del protocollo è il **router LSA**, costituito da:

- **Link State ID:** identifica l'LSA. Può essere l'ID del router.
- **Advertising Router:** identifica il router che ha generato l'LSA. L'ID del router può essere scelto arbitrariamente. In genere, viene scelto come ID l'indirizzo IP dell'interfaccia numericamente più piccolo.
- **Numero di link:** indica il numero di link descritti dal LSA.
- **Descrizione dei link n-esimo:** contiene le informazioni relative all'n-esimo link del router:
  - **Link ID:** identifica il link. In genere è l'indirizzo IP della relativa interfaccia.
  - **Metrica:** 16 bit per il costo del link. Può assumere  $2^{16}$  valori.

Gli LSA vengono comunicati all'intera rete tramite ***reliable flooding***:

- ***reliable*:** gli LSA ricevuti sono confermati relativo al mittente tramite messaggi di riscontro contenenti Link State ACK (**LSAck**).
- ***flooding*:** ogni router trasmette/ritrasmette **LSA-Update** ai propri vicini ogni volta che la topologia o il costo di un link vengono alterati. Per robustezza, in maniera asincrona ogni 30 minuti i routers ritrasmettono gli LSA anche se non ci sono stati aggiornamenti.

Ogni router possiede un **Link State Database (LSD)** per mantenere una collezione aggiornata di tutti i router LSA ricevuti. Il meccanismo con cui vengono scambiati gli LSA garantisce che, a regime, tutti gli LSD dei routers siano identici, ovvero che ogni router abbia le stesse conoscenze sulla topologia e sui costi dei link.

## Instradamento tra ISP: BGP

Abbiamo appena appreso come gli ISP utilizzino OSPF per determinare i percorsi ottimali per le coppie sorgente-destinazione interne a un sistema autonomo. Ma per determinare i percorsi per le coppie sorgente-destinazione che interessano più sistemi autonomi è necessario un **protocollo di instradamento inter-AS** che coordini più AS. Il **border gateway protocol**, detto **BGP**, rappresenta l'attuale standard *de facto* dei protocolli di instradamento tra sistemi autonomi in Internet. Deve accomodare esigenze commerciali di diversa natura, come:

- *Customer-Provider*: i customer devono essere raggiungibili da chiunque nella rete, ma non vogliono/devono svolgere funzioni di transito del traffico per il proprio provider.
- *Peer-Peer*: i providers devono essere connessi tra loro per permettere la comunicazione tra i loro customer. Inoltre, vogliono poter decidere con quali providers permettere il transito di traffico.

Si distinguono perciò tre livelli gerarchici di AS:

- **Transit AS** (Tier 1): ricoprono aree internazionali o nazionali di grandi dimensioni. Sono a capo della gerarchia internet e forniscono esclusivamente servizio di transito del traffico, sia tra loro che per le AS gerarchicamente sottostanti (*multihomed*).
- **Multihomed AS** (Tier 2): ricoprono aree nazionali o regionali. Forniscono servizio di transito del traffico per le AS gerarchicamente sottostanti (*stub*). Possono scambiarsi traffico sia direttamente che passando per le AS gerarchicamente superiori (*transit*).
- **Stub AS**: collegano gli host alla rete. Non forniscono alcun servizio di transito del traffico per altre AS. Utilizzano le AS gerarchicamente superiori (*multihomed*) per scambiarsi traffico.

Ogni AS viene identificato tramite un ID a 16 bit (**ASN – AS Number**), univoco per tutta la rete.

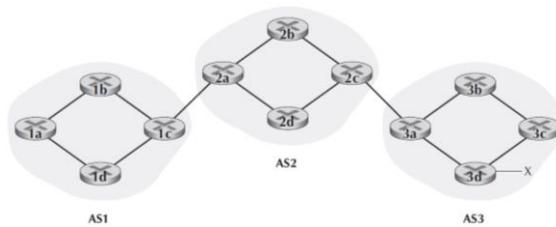
Per rispettare tali esigenze viene utilizzato un **algoritmo policy-based path-vector**:

- **Policy-based**: permette di definire tramite l'applicazione di regole cammini che rispettano le relazioni commerciali tra AS.
- **Path-vector**: estensione dell'algoritmo *distance-vector* in cui vengono utilizzati i cammini al posto delle distanze, ovvero i routers si scambiano informazioni su interi cammini verso destinatari invece che la semplice distanza. L'impiego di tale algoritmo permette di:
  - supportare instradamento basato sulle policy;
  - rilevare più velocemente la presenza di cicli nell'instradamento dei pacchetti.Se un nodo riceve un cammino in cui lui stesso è presente, sicuramente tale cammino genera un ciclo. In tal caso è sufficiente che il nodo scarti il cammino in esame.

Il protocollo BGP utilizza quindi un algoritmo policy-based path-vector per instradare i pacchetti verso **prefissi di rete**, ovvero **subnets**. Permette agli AS di:

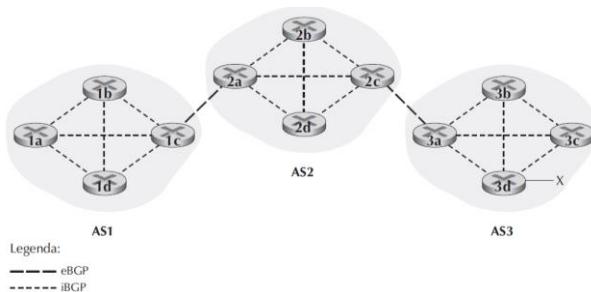
- scambiare informazioni sulla raggiungibilità dei prefissi con gli AS vicini. In particolare, permette alle subnets di comunicare la propria esistenza e posizione all'interno della rete;
- propagare le informazioni di raggiungibilità dei prefissi a tutti i router interni all'AS;
- determinare il cammino migliore verso prefissi/subnets basandosi sia sulle informazioni di raggiungibilità, sia sulle politiche descritte dalle diverse AS.

Si consideri la rete della figura a seguire con tre AS: AS1, AS2 e AS3, quest'ultimo con una sottorete di prefisso  $x$ . Ogni router, in ogni AS, funge sia da **router gateway**, vale a dire un router di bordo direttamente connesso a uno o più router in altri AS, sia da **router interno**, connesso cioè solo a host e router interni all'AS.



In BGP, coppie di router (detti **BGP peers**) si scambiano informazioni di instradamento su connessioni TCP semi-permanenti (ossia aperte e mai chiuse) usando la porta 179. Ogni connessione TCP, con tutti i messaggi BGP che vi vengono inviati, è detta **sessione BGP**. Si distinguono sessioni:

- **interne (iBGP session)**: i router della coppia BGP sono entrambi contenuti all'interno di uno stesso AS. Sono utilizzate per propagare le informazioni di instradamento ricevute dai gateway a tutti i router interni all'AS. Non sempre corrispondono ad un link fisico.
- **esterne (eBGP session)**: i router della coppia BGP fanno parte di due AS differenti. Sono utilizzate per scambiare informazioni di raggiungibilità dei prefissi tra AS. Corrispondono sempre ad un link fisico (ossia al mezzo di collegamento tra due AS).

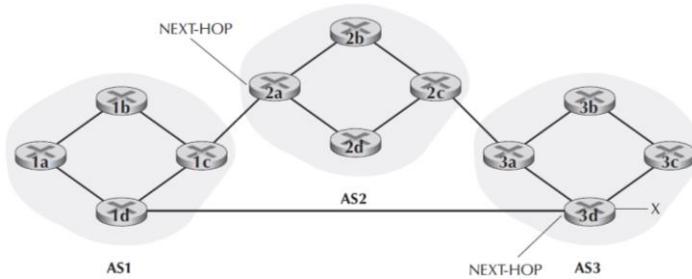


Come distribuirebbe BGP le informazioni di raggiungibilità del prefisso  $x$  ad AS1 e AS2? Il router gateway  $3a$  invia un messaggio eBGP “AS3  $x$ ” al router gateway  $2c$  che a sua volta lo invia su una sessione iBGP a tutti i router di AS2 compreso il gateway  $2a$ . Quest'ultimo quindi invia un messaggio eBGP “AS2 AS3  $x$ ” al router gateway  $1c$ , che infine usa iBGP per

inviare il messaggio “AS2 AS3 x” a tutti i router di AS1. Completato tale processo, ogni router di AS1 e AS2 è a conoscenza dell’esistenza di x e del percorso per raggiungerlo.

Le **informazioni di raggiungibilità** scambiate consistono in **cammini** (*route*) verso prefissi, costituiti da:

- **prefisso**: indica la *subnet* di destinazione del cammino. Gli AS possono aggregare in un unico prefisso più sottoreti ad esso interne al fine di ottimizzare l’instradamento.
- **attributi**: informazioni relative al cammino. Le più importanti sono:
  - **AS-PATH**: contiene una lista degli ID degli AS attraverso la quale instradare i pacchetti. Viene popolata durante la propagazione delle informazioni di instradamento, memorizzando progressivamente gli AS traversati. In particolare, ogni nuovo AS incontrato viene aggiunto all’inizio della lista così che dal punto di vista del ricevente gli AS siano in ordine.
  - **NEXT-HOP**: indirizzo IP dell’interfaccia del router dalla quale inizia il cammino in termini di AS-PATH, ovvero la prima AS in AS-PATH. Corrisponde all’ultimo gateway esterno incontrato durante la propagazione.



L’attributo NEXT-HOP per la rotta “AS2 AS3 x” da AS1 a x passando per AS2 è l’indirizzo IP dell’interfaccia a sinistra del router 2a; l’attributo NEXT-HOP per la rotta “AS3 x” da AS1 a x che salta AS2 è l’indirizzo IP dell’interfaccia più a sinistra del router 3d.

Ogni router mantiene una **tabella di instradamento BGP** contenente tutti i cammini noti. La tabella viene popolata utilizzando una strategia di aggiornamento incrementale, nella quale BGP peers si inviano due tipi di notifiche:

1. **annunci (announcement)**: il router viene a conoscenza di una strada attiva, in quanto una nuova subnet viene connessa o riceve un annuncio da un vicino. Il router aggiorna la propria tabella ed eventualmente propaga l’annuncio agli altri vicini aggiornando se necessario l’AS-PATH ed il NEXT-HOP del cammino.
2. **revoche (withdrawal)**: il router viene a conoscenza di una strada non più attiva, in quanto una subnet viene sconnessa o riceve una revoca da un vicino. Il router aggiorna la propria tabella ed eventualmente propaga la revoca agli altri vicini.

Per realizzare tutto ciò vengono scambiati quattro tipi di **messaggi BGP**:

- **Open**: usato per aprire la connessione TCP verso il router accoppiato e autenticarsi.
- **Update**: usato per inviare nuovi cammini o per invalidare vecchi cammini.
- **Keepalive**: usato per mantenere la connessione TCP aperta in assenza di aggiornamenti e per confermare la corretta apertura della sessione (ACK in risposta al messaggio di Open).

- **Notification:** usato per riferire al mittente errori sui messaggi precedentemente ricevuti o per chiudere la sessione BGP, e quindi la connessione TCP.

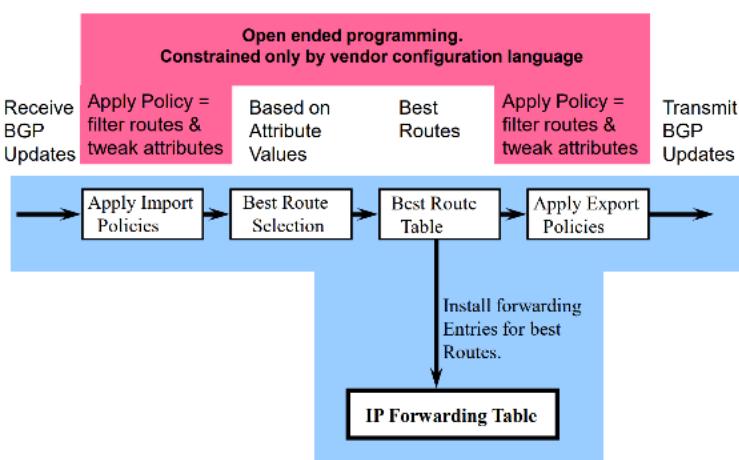
### Selezione del cammino in BGP

I router possono imparare più strade verso uno stesso prefisso. La **selezione del cammino** viene effettuata tramite **regole ad eliminazione**. Vengono applicate nel seguente ordine:

1. **Prefix destination:** si selezionano dalla tabella tutti i cammini diretti alla *subnet* di destinazione.
2. **Policy decision:** dei cammini verso la *subnet* si selezionano solo quelli che rispettano le regole definite dal gestore del router. Servono a soddisfare le esigenze commerciali ed organizzative dell'AS.
3. **Shortest AS-PATH:** dai cammini idonei si selezionano quelli che attraversano il minor numero di AS.
4. **Closest NEXT-HOP (hot-potato routing):** dai cammini a minor numero di AS, si seleziona quello che possiede costo minimo per raggiungere la prossima AS, ovvero il suo gateway (NEXT-HOP) più vicino.
5. Criteri addizionali usati nel caso in cui dalle eliminazioni precedenti siano rimasti molteplici cammini.

La popolazione della tabella di instradamento può essere direttamente influenzata dalle policy definite dal gestore, in modo da ottimizzare il processo di selezione dei cammini. L'applicazione delle policy avviene sia all'ingresso che all'uscita di informazioni di raggiungibilità:

- **Import policy:** vengono filtrati i cammini provenienti dai vicini, in quanto non rispettano le politiche del gestore. Possono anche essere manipolati degli attributi per influenzare le preferenze sulla selezione dei cammini.
- **Export policy:** vengono filtrati i cammini da inviare ai vicini, in quanto il gestore ha interesse a non far utilizzare determinati cammini ad altri AS. Possono anche essere manipolati degli attributi per influenzare la selezione dei cammini dei router vicini.

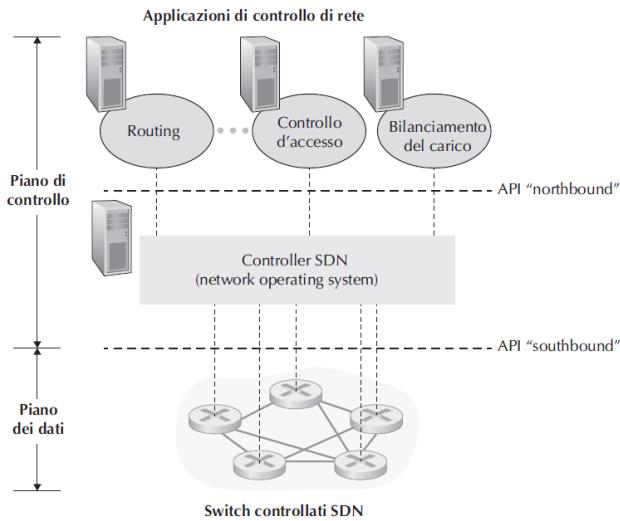


Facendo un breve riepilogo, possiamo dire che complessivamente le operazioni effettuate per instradare un pacchetto all'interno della rete sono le seguenti:

1. utilizzando il **protocollo inter-AS BGP** si determinano i gateway dai quali è possibile raggiungere la subnet verso la quale è diretto il pacchetto;
2. utilizzando le **informazioni intra-AS** si determinano i cammini a minor costo verso ogni gateway. Viene quindi scelto il gateway raggiunto dal cammino meno costoso (*hot-potato routing*);
3. dalla **tavella di inoltro** del router si determina l'interfaccia dalla quale è possibile raggiungere il gateway scelto, e quindi verso cui inoltrare il pacchetto.

## Il piano di controllo SDN

In questo paragrafo approfondiremo il **piano di controllo SDN**, la logica globale di rete che controlla l'inoltro dei pacchetti tra i dispositivi di una rete SDN, così come la configurazione e la gestione di tali dispositivi e dei loro servizi.



In un'architettura SDN possono essere identificate quattro caratteristiche fondamentali.

- *Inoltro basato sui flussi.* L'inoltro dei pacchetti può essere effettuato da uno switch SDN sulla base del valore dei campi dell'intestazione a livello di trasporto, di rete e di collegamento.  
Si ricordi che le regole di inoltro dei pacchetti sono specificate nella tabella dei flussi degli switch; è compito del piano di controllo SDN calcolare, gestire e installare le occorrenze della tabella dei flussi in tutti gli switch della rete.
- *Separazione del piano dei dati e del piano di controllo.* Il **piano dei dati** consiste di switch di rete, dispositivi relativamente semplici, ma veloci, che eseguono le regole "match-action" nelle loro tabelle dei flussi. Il **piano di controllo** consiste di server e software che determinano e gestiscono le tabelle dei flussi degli switch.
- *Funzioni di controllo di rete: esterne agli switch del piano dei dati.* Come mostrato nella figura, lo stesso piano di controllo consiste di due componenti, ossia un controller SDN e un insieme di applicazioni di controllo di rete. Il controller mantiene informazioni di stato della rete quali per esempio lo stato dei collegamenti, degli switch e degli host remoti; fornisce tali informazioni alle applicazioni di controllo di rete eseguite nel piano di controllo e fornisce il mezzo attraverso il quale tali applicazioni possono monitorare, programmare e controllare i sottostanti dispositivi di rete.
- *Una rete programmabile.* La rete è programmabile attraverso le applicazioni di controllo di rete che vengono eseguite nel piano di controllo.

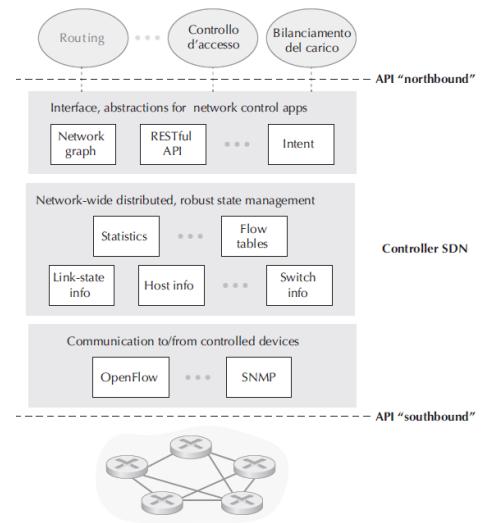
Dalla nostra trattazione si evince che le SDN rappresentano una separazione delle funzionalità di rete: gli switch del piano dei dati, i controller SDN e le applicazioni di

controllo di rete sono entità distinte che possono essere fornite da differenti fornitori e organizzazioni. Tutto ciò contrasta con il modello precedente SDN.

Cominciamo a trattare il piano di controllo SDN considerando le capacità generiche che esso deve fornire. Come già visto, il piano di controllo SDN si divide a grandi linee in due componenti: il controller SDN e le applicazioni di controllo SDN.

Le funzionalità del controller possono essere divise in tre livelli.

1. *Livello di comunicazione.* Effettua le comunicazioni tra il controller SDN e i dispositivi di rete controllati. Tale protocollo costituisce il livello più basso dell'architettura del controller. La comunicazione tra il controller e i dispositivi controllati passa attraverso un'interfaccia del controller nota come **interfaccia “southbound”**.
2. *Livello di gestione dello stato globale della rete.* Le decisioni di controllo finali prese dal piano di controllo SDN richiedono che il controller abbia informazioni aggiornate sullo stato di host, link, switch e altri dispositivi SDN controllati della rete.
3. *L'interfaccia con il livello di applicazione di controllo della rete.* Il controller interagisce con le applicazioni di controllo di rete attraverso la sua **interfaccia “northbound”**.



### Il protocollo OpenFlow in SDN

Il **protocollo OpenFlow** opera (su TCP con numero di porta 6653) tra un controller SDN e uno switch o un altro dispositivo che implementi le API OpenFlow.

Alcuni dei più importanti messaggi del protocollo *inviai dal controller allo switch controllato* sono i seguenti:

- **Configuration:** permette al controller di interrogare e impostare i parametri di configurazione di uno switch.
- **Modify-state:** usato dal controller per aggiungere/cancellare o modificare le occorrenze della tabella dei flussi dello switch e per impostare le proprietà delle porte dello switch.
- **Read-state:** usato dal controller per raccogliere le statistiche e i valori dei contatori nelle tabelle dei flussi e nelle porte dello switch.
- **Send-packet:** usato dal controller per inviare un pacchetto specifico fuori da una specifica porta dello switch; il messaggio stesso contiene il pacchetto da inviare nel suo carico.

Alcuni dei più importanti messaggi del protocollo *inviai dallo switch controllato al controller* sono i seguenti:

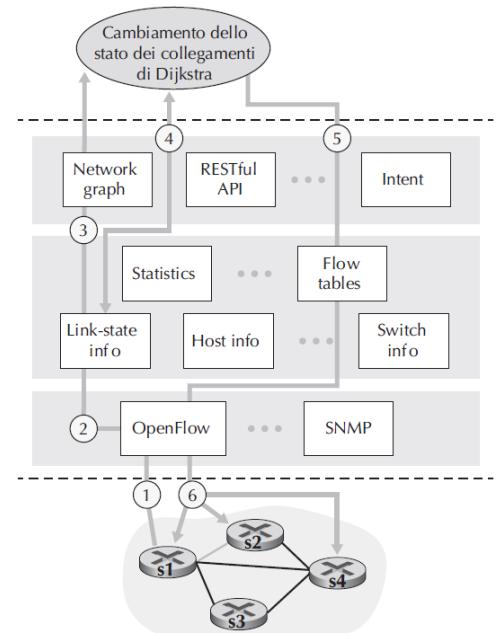
- **Flow-removed**: informa il controller che un'occorrenza della tabella dei flussi è stata cancellata, per esempio per un evento di timeout o per un messaggio modify-state ricevuto.
- **Port-status**: usato dallo switch per informare il controller di un cambiamento nello stato di una porta.
- **Packed-in**: come visto, un pacchetto in arrivo su una porta di uno switch che non abbia corrispondenza alcuna nella tabella dei flussi viene inviato al controller per ulteriori elaborazioni. Inoltre, anche i pacchetti che abbiano trovato una corrispondenza possono essere inviati al controller in seguito a un'azione che deve essere intrapresa a seguito della corrispondenza. Il messaggio packed-in è usato per inviare tali pacchetti al controller.

Vediamo ora un **eSEMPIO DI ITERAZIONE TRA PIANO DEI DATI E PIANO DI CONTROLLO**.

Nell'esempio:

- l'algoritmo di Dijkstra viene eseguito come un'applicazione separata, esterna agli switch;
- gli switch inviano gli aggiornamenti sui collegamenti solo al controller SDN e non li scambiano tra di loro.

Si assume che il collegamento tra lo switch s1 e lo switch s2 cada; che si implementi l'instradamento basato sul cammino minimo e che le regole di inoltro dei flussi in entrata e in uscita in s1, s3 e s4 ne siano condizionati, mentre le operazioni in s2 rimangono inalterate. Si assume inoltre che OpenFlow venga utilizzato come protocollo a livello di comunicazione e che il piano di controllo effettui unicamente la funzione di instradamento sulla base dello stato dei collegamenti.



1. Lo switch s1 osserva la caduta del collegamento con s2 e notifica tale cambiamento nello stato del collegamento al controller SDN utilizzando un messaggio Open-Flow port-status.
2. Il controller SDN riceve il messaggio OpenFlow con il cambiamento dello stato del collegamento, lo notifica al gestore dello stato del collegamento che aggiorna il database degli stati dei collegamenti.
3. L'applicazione di controllo di rete che implementa l'algoritmo di Dijkstra riceve la notifica del cambiamento dello stato del collegamento.

4. L'applicazione di instradamento interagisce con il gestore dello stato del collegamento per ottenere lo stato aggiornato del collegamento; potrebbe anche consultare altre componenti del livello di gestione dello stato. Quindi computa i nuovi percorsi minimi.
5. L'applicazione di instradamento interagisce quindi con il gestore della tabella dei flussi che determina quali tabelle debbano essere aggiornate.
6. Il gestore della tabella dei flussi usa quindi il protocollo OpenFlow per aggiornare le occorrenze della tabella dei flussi negli switch coinvolti: s1, che adesso instrada i pacchetti destinati a s2 attraverso s4, s2 che adesso inizia a ricevere i pacchetti da s1 attraverso s4, e s4 che adesso deve inoltrare i pacchetti da s1 destinati a s2.

## Link Layer e LAN

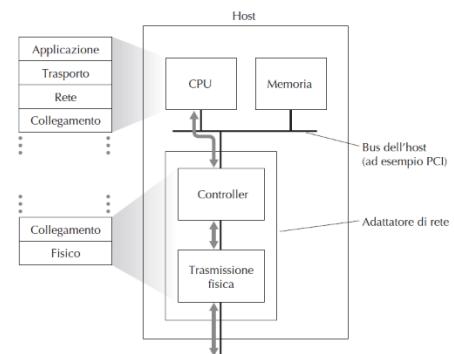
Faremo riferimento a qualunque dispositivo che opera a livello di collegamento (livello 2) indicandolo come **nodo**. Ci riferiremo inoltre ai canali di comunicazione, che collegano nodi adiacenti lungo un cammino, come a **collegamenti** (*link*).

Su ogni collegamento, un nodo trasmittente incapsula il datagramma in un **frame del livello di collegamento** (*link-layer frame*) e lo trasmette lungo il collegamento stesso.

Sebbene il servizio di base del livello di collegamento sia il trasporto di datagrammi da un nodo a quello adiacente lungo un singolo canale di comunicazione, i dettagli dei servizi forniti possono variare da un protocollo all'altro. I possibili servizi che possono essere offerti includono i seguenti.

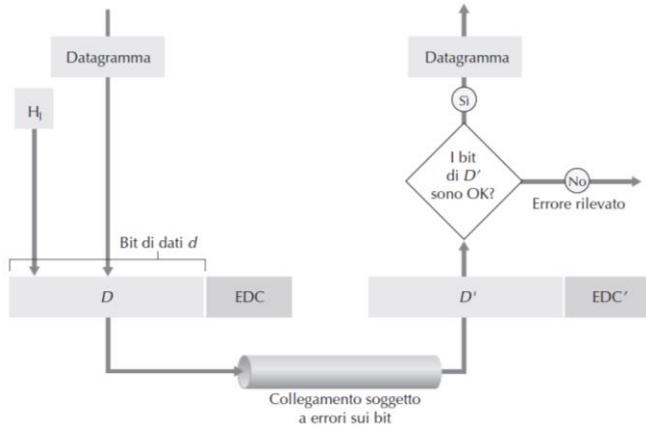
- **Framing.** Quasi tutti i protocolli incapsulano i datagrammi del livello di rete all'interno di un frame a livello di collegamento, prima di trasmetterlo. I frame sono costituiti da un campo dati, nel quale è inserito il datagramma, e da vari campi di intestazione. La struttura del frame è specificata dal protocollo.
- **Accesso al collegamento.** Un protocollo che controlla l'accesso al mezzo trasmisivo (**MAC**, *medium access control*) specifica le regole con cui immettere i frame nel collegamento. Nei *collegamenti punto a punto*, con un solo mittente e un solo destinatario, in cui il protocollo MAC è semplice (o non esiste), il mittente può inviare il frame quando il canale risulta libero. Un caso più interessante è quello in cui vari nodi condividono un singolo canale broadcast (il cosiddetto problema degli accessi multipli). In questo caso il protocollo MAC aiuta a coordinare la trasmissione dei frame da parte dei nodi.
- **Consegna affidabile.** I protocolli a livello di collegamento che forniscono un servizio di consegna affidabile garantiscono il trasporto senza errori di ciascun datagramma. Questo servizio può essere realizzato attraverso ACK e ritrasmissioni.
- **Rilevazione e correzione degli errori.** Siccome non è utile inoltrare i datagrammi contenenti errori, molti protocolli del livello di collegamento forniscono un meccanismo per rilevarne la presenza. Ciò è possibile grazie all'inserimento, da parte del nodo trasmittente, di bit di controllo all'interno del frame e all'esecuzione di un semplice controllo da parte del nodo ricevente. È implementato in hardware.

Per un dato collegamento, il protocollo del livello di collegamento è sostanzialmente realizzato da un **adattatore di rete** (*network adapter*), noto anche come **scheda di rete** (**NIC**, *Network Interface Card*). Il cuore della scheda di rete è il controller a livello di collegamento (*link layer controller*), che è di solito un chip dedicato, che implementa molti dei servizi a livello di collegamento sopracitati. La figura mostra una scheda di rete collegata al bus dell'host, dove viene considerato dagli altri componenti dell'host come un qualsiasi altro dispositivo di I/O.



## Error detection

Nel nodo trasmittente, ai dati  $D$  che devono essere protetti da errori vengono aggiunti dei bit detti **EDC** (**error detection and correction**). I dati  $D$  e i bit  $EDC$  sono inviati in un frame al nodo ricevente. Questo legge una sequenza di bit  $D'$  ed  $EDC'$  che può essere diversa dall'originale, come risultato della modifica dei bit in transito.



Il nodo ricevente deve determinare se  $D'$  coincide con  $D$ , potendo contare soltanto su  $D'$  e su  $EDC'$ . Le attuali tecniche non sempre consentono al nodo ricevente di rilevare se si sono verificati errori nei bit. Anche con l'utilizzo dei bit di rilevazione degli errori è possibile che ci siano degli **errori non rilevati**.

Dovremo quindi optare per uno schema per la rilevazione degli errori che riduca la probabilità di questo evento. Generalmente, le tecniche più sofisticate comportano un'elevata ridondanza, nel senso che sono necessari calcoli più complessi e la trasmissione di molti bit aggiuntivi.

## Multiple access link

Il **collegamento punto a punto** è costituito da un trasmittente a un'estremità del collegamento e da un unico ricevente all'altra. Molti protocolli del livello di collegamento sono stati progettati per questi collegamenti, tra cui *PPP* (*point-to-point protocol*) e *HDLC* (*high-level data link control*).

Il **collegamento broadcast** può avere più nodi trasmittenti e riceventi connessi allo stesso canale broadcast condiviso.

Esamineremo il **problema dell'accesso multiplo**, ossia del problema di coordinare l'accesso di più nodi trasmittenti e riceventi in un canale broadcast condiviso. I cosiddetti **protocolli di accesso multiplo** fissano le modalità con cui i nodi regolano le loro trasmissioni sul canale condiviso.

Dato che tutti i nodi sono in grado di trasmettere frame, è possibile che due o più lo facciano nello stesso istante, per cui tutti i nodi riceveranno contemporaneamente più frame. Tra questi si genera una **collisione** a causa della quale nessuno dei nodi riceventi riuscirà a interpretare i frame che, in un certo senso, risultano ingarbugliati tra loro. Di conseguenza con la collisione si verifica una perdita di frame, mentre il canale rimane inutilizzato. È chiaro che, se una situazione di questo genere dovesse ripetersi con eccessiva frequenza, gran parte della banda del canale risulterebbe sprecata.

Per far sì che il canale broadcast venga utilizzato in modo efficiente, occorre coordinare le trasmissioni dei nodi attivi. In generale, possiamo classificare praticamente tutti i protocolli di accesso multiplo in una di queste categorie:

- protocolli **a suddivisione del canale** (*channel partitioning protocol*);
- protocolli **ad accesso casuale** (*random access protocol*);
- protocolli **a rotazione** (*taking-turn protocol*).

Idealmente, un protocollo di accesso multiplo per un canale broadcast con velocità di  $R$  bit al secondo dovrebbe avere le seguenti caratteristiche:

1. se un solo nodo deve inviare i dati, questo dispone di un throughput pari a  $R$  bps;
2. se  $M$  nodi devono inviare dati, questi dispongono di un throughput pari a  $R/M$  bps;
3. il protocollo è decentralizzato; in altre parole, non ci sono nodi principali che qualora non funzionassero correttamente potrebbero rendere inattivo l'intero sistema;
4. il protocollo è semplice, in modo che risulti economico da implementare.

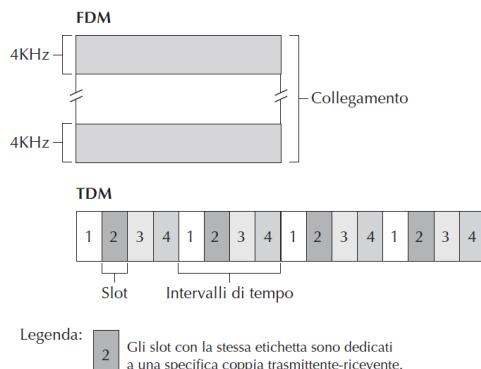
### Protocolli a suddivisione del canale

Supponiamo che il canale supporti  $N$  nodi e che la sua velocità di trasmissione sia di  $R$  bps.

Il **protocollo TDM** (*Time Division Multiplexing*) suddivide il tempo in **intervalli di tempo** (*time frame*) e poi divide ciascun intervallo di tempo in  $N$  **slot temporali** (*time slot*). Ogni slot è quindi assegnato a uno degli  $N$  nodi. Ogni volta che un nodo ha un pacchetto da inviare, trasmette i bit del pacchetto durante lo slot di tempo assegnatogli. Generalmente, le dimensioni dello slot sono stabilite in modo da consentire la trasmissione di un singolo pacchetto.

TDM viene utilizzato in quanto riesce ad evitare le collisioni ed è perfettamente imparziale: ciascun nodo ottiene, durante ciascun intervallo di tempo, un tasso trasmittivo di  $R/N$  bps. Tuttavia, presenta due specifici inconvenienti: anche quando non vi sono altri nodi che devono inviare un pacchetto, quello che vuole trasmettere è vincolato ad attendere il suo turno nella sequenza di trasmissione e a non superare il limite medio di  $R/N$  bps.

Il **protocollo FDM** (*Frequency Division Multiplexing*) divide il canale in frequenze differenti (ciascuna con una larghezza di banda di  $R/N$ ) e assegna ciascuna frequenza a un nodo. Tuttavia, anche con FDM la larghezza di banda è limitata a  $R/N$ , pure quando vi è un solo nodo che ha un pacchetto da spedire.



## Protocolli ad accesso casuale

I **protocolli MAC ad accesso casuale** utilizzano il canale alla massima capacità ogni volta che un nodo deve trasmettere. Quando si verifica una collisione, i nodi coinvolti ritrasmettono ripetutamente i loro frame (cioè i pacchetti) fino a quando raggiungono la destinazione, senza collisioni. La ritrasmissione del frame non è immediata, ma il nodo attende per un periodo di tempo casuale (*random delay*); ogni nodo coinvolto in una collisione seleziona un ritardo casuale indipendente da quello degli altri nodi.

I principali protocolli sono:

- ALOHA – L’ascolto del canale avviene dopo l’invio del frame
  - **unslotted**: versione ALOHA in cui il tempo è continuo;
  - **slotted**: versione ALOHA in cui il tempo viene diviso in slot.
- CSMA (*Carrier Sense Multiple Access*) – L’ascolto del canale avviene prima dell’invio.

## ALOHA

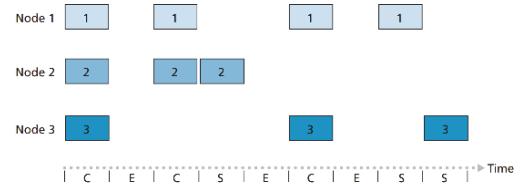
Nei **protocolli ALOHA** la trasmissione di un frame da parte di un nodo avviene indipendentemente da quello che stanno facendo gli altri nodi sul canale.

Consideriamo la **versione slotted** assumendo che:

- tutti i frame hanno la stessa dimensione;
- il tempo è diviso in slot di dimensione fissa sufficiente a trasmettere un unico frame;
- tutti i nodi sono sincronizzati in maniera da conoscere l’inizio degli slot;
- i nodi trasmettono solamente all’inizio di uno slot;
- se più nodi trasmettono all’interno di uno slot, la collisione è rilevata da tutti i nodi.

Ogni nodo trasmette nello slot successivo alla generazione del frame. Si effettuano le seguenti **operazioni**:

- se non avviene collisione, il nodo può trasmettere nel prossimo slot se ci sono frame da inviare;
- se avviene collisione, il nodo ritrasmette il frame con **probabilità  $p$**  negli slot successivi fino al corretto invio.



Tale strategia di trasmissione possiede i seguenti **vantaggi e svantaggi**:

- [PRO] nel caso in cui sia attivo un solo nodo, questo può trasmettere alla massima capacità del link;
- [PRO] è altamente decentralizzato, in quanto occorre solo sincronizzare gli slot temporali;
- [CON] con molteplici nodi attivi l’efficienza è ridotta, in quanto avvengono collisioni nella trasmissione e ci sono slot temporali che non vengono utilizzati;
- [CON] è necessario sincronizzare tutti i nodi connessi al canale.

Supponendo di avere  $N$  nodi con molti frame da inviare, ad ogni slot di tempo si ha che:

$$P(\text{successo nodo}) = p \cdot (1 - p)^{N-1} \Rightarrow P(\text{successo}) = N \cdot p \cdot (1 - p)^{N-1} \rightarrow_{N \rightarrow \infty} 1/e$$

Ne deriva quindi che con un numero elevato di nodi attivi tutti in trasmissione l'efficienza massima di utilizzo del canale utilizzando il protocollo slotted ALOHA è pari a  $1/e \approx 36,8\%$ .

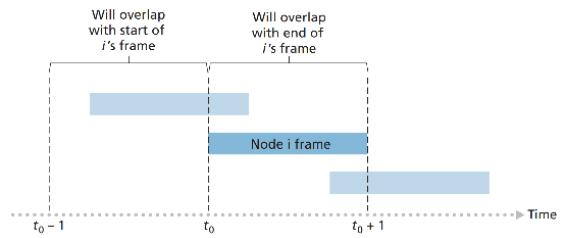
Consideriamo ora la **versione *unslotted***. Ogni nodo trasmette non appena un frame viene generato e si effettuano le seguenti operazioni:

- se non avviene la collisione, il nodo può subito trasmettere se ci sono altri frame da inviare;
- se avviene collisione, il nodo ritrasmette il frame con **probabilità  $p$** , altrimenti attende per un tempo pari all'invio di un frame prima di ripetere tale logica.

Dato che il tempo è considerato continuo, non sono necessari meccanismi di sincronizzazione, rendendolo un **protocollo completamente decentralizzato**. Tuttavia, ciò aumenta la probabilità di collisione in quanto aumenta l'intervallo di tempo nel quale un solo nodo deve trasmettere per inviare con successo un frame.

Infatti, si consideri un nodo che vuole trasmettere un frame all'istante di tempo  $t_0$ . Per non avere collisione si deve avere che nessun altro nodo:

- deve trasmettere nell'intervallo di tempo  $[t_0 - t_{frame}, t_0]$ , in quanto genererebbe collisione con la parte iniziale del frame;
- deve trasmettere nell'intervallo di tempo  $[t_0, t_0 + t_{frame}]$ , in quanto genererebbe collisione con la parte finale del frame.



Supponendo di avere  $N$  nodi con molti frame da inviare, si ha quindi che:

$$P(\text{successo nodo}) = p \cdot (1 - p)^{2(N-1)} \Rightarrow P(\text{successo}) = N \cdot p \cdot (1 - p)^{2(N-1)} \xrightarrow{N \rightarrow \infty} 1/2e$$

Ne deriva che con un numero elevato di nodi attivi tutti in trasmissione l'efficienza massima di utilizzo del canale utilizzando il protocollo *unslotted ALOHA* è pari a  $1/2e = 18,4\%$ .

### CSMA

Nel **protocollo CSMA** (*Carried Sense Multiple Access*) la trasmissione di un frame da parte di un nodo avviene in funzione a quello che stanno facendo gli altri nodi sul canale. In particolare:

- un nodo **ascolta il canale prima di trasmettere** (*carrier sensing*, rilevamento della portante). Se il canale sta già trasmettendo un frame, il nodo aspetta finché rileva che il canale è libero per un intervallo di tempo e quindi inizia la trasmissione.
- il nodo che sta trasmettendo **rimane contemporaneamente in ascolto del canale** (*collision detection*, rilevamento della collisione). Se osserva che un altro nodo sta trasmettendo un frame che interferisce col suo, arresta la propria trasmissione, aspetta un intervallo di tempo casuale e poi ripete il processo.

Il ritardo di propagazione da un estremo all'altro di un canale broadcast (il tempo richiesto da un segnale per propagarsi da un nodo all'altro) avrà un ruolo cruciale nel determinare le sue prestazioni. Maggiore è questo ritardo, maggiore sarà la possibilità che il nodo, pur

attento a rilevare la portante, non si accorga che è già cominciata la trasmissione da parte di un altro nodo, causando quindi **collisioni**.

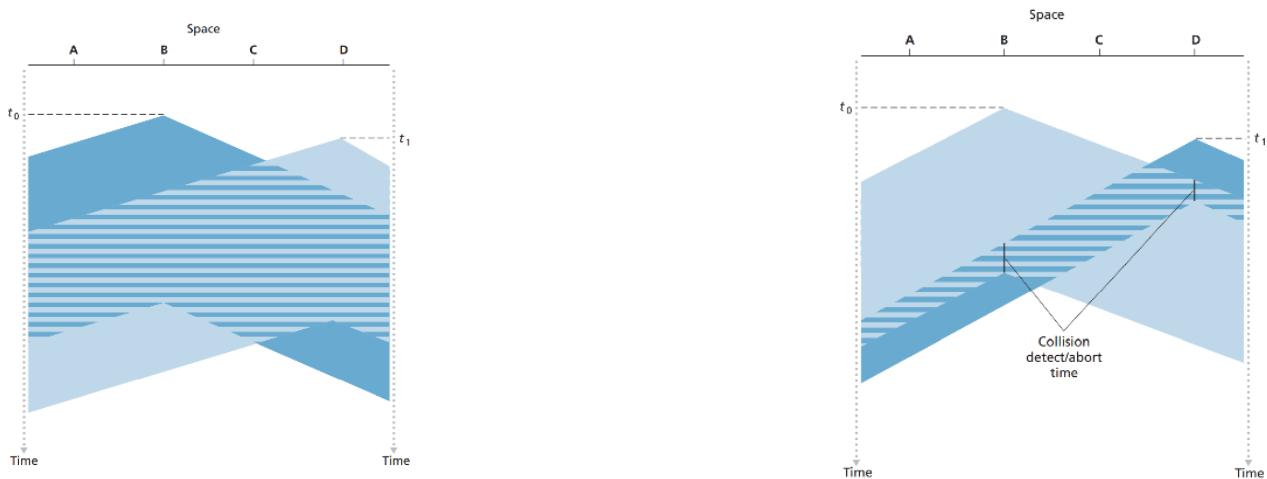
L'introduzione della rilevazione di collisione in un protocollo di accesso multiplo ne migliorerà le prestazioni, evitando l'inutile trasmissione dell'intero frame danneggiato dall'interferenza con un altro frame.

Riassumiamo le operazioni del **protocollo CSMA/CD** (CSMA con *collision detection*) dal punto di vista di una scheda di rete collegata a un canale broadcast.

1. La scheda ottiene direttamente un datagramma dal livello di rete, prepara un frame a livello di collegamento e lo sistema in un suo buffer.
2. Quando riscontra che il canale è libero, inizia la trasmissione del frame. Se il canale risulta occupato, resta in attesa sino al momento in cui non rileva più il segnale e solo allora inizia l'invio del frame.
3. Durante la trasmissione verifica la presenza di eventuali segnali provenienti da altre schede di rete sul canale broadcast.
4. La scheda di rete, se trasmette l'intero frame senza rilevare energia di segnale proveniente da altre schede, ha finito il suo lavoro; altrimenti, se riscontra energia di segnale durante la trasmissione, interrompe immediatamente la trasmissione del frame.
5. Dopo aver annullato la trasmissione, la scheda di rete aspetta per un tempo casuale e poi ritorna al passo 2.

L'algoritmo di ***binary exponential backoff*** (attesa binaria esponenziale) si occupa della determinazione del **tempo di backoff**: quando il trasmittitore riscontra l' $n$ -esima collisione durante la trasmissione di un dato frame, stabilisce casualmente un valore  $K$  nell'insieme  $\{0, 1, 2, \dots, 2n - 1\}$ ; quindi, più alto è il numero di collisioni, più grande sarà l'intervallo da cui  $K$  viene estratto.

A seguire, a sinistra abbiamo il diagramma spazio-temporale di due nodi CSMA con trasmissioni in collisione, mentre a destra abbiamo quello di due nodi CSMA con rilevamento della collisione.



## Protocolli a rotazione

I **protocolli MAC a rotazione** riservano completamente il canale ad un nodo per volta. I principali metodi con la quale viene effettuata la turnazione sono:

- **Polling**: al canale è connesso un *nodo master* che invita i *nodi slave* a trasmettere. Elimina le collisioni e gli slot vuoti che si riscontrano nei protocolli ad accesso casuale e quindi può avere un'efficienza più elevata, ma presenta alcuni svantaggi:
  - introduzione del *ritardo di polling*, ossia il tempo richiesto per notificare a un nodo il permesso di trasmettere;
  - un malfunzionamento del nodo master comporta la rottura del protocollo.
- **Token-passing**: i nodi connessi al canale si passano sequenzialmente un *token*. Se il nodo che riceve il token non ha pacchetti da inviare, lo inoltra immediatamente al successivo; altrimenti, procede a trasmettere il numero massimo di frame consentito, prima di inoltrare il token al nodo successivo. Un malfunzionamento del nodo con il token comporta la rottura del protocollo.

## Indirizzi a livello di collegamento e ARP

Analizzeremo il protocollo per la risoluzione degli indirizzi (**ARP**, *Address Resolution Protocol*), che fornisce ai nodi un meccanismo per trasformare indirizzi IP in indirizzi a livello di collegamento.

In realtà, non sono i nodi (cioè, gli host e i router) ad avere indirizzi a livello di collegamento, ma sono i loro adattatori (schede di rete) a possederli. Un host o un router con più interfacce di rete avrà più indirizzi a livello di collegamento associati a esse, esattamente come accadeva per gli indirizzi IP. Gli indirizzi a livello di collegamento sono indicati con varie terminologie, come **indirizzo fisico**, **indirizzo LAN** o **indirizzo MAC**.

Per molte LAN (come Ethernet e 802.11), l'indirizzo MAC è lungo sei byte, il che consente di avere  $2^{48}$  possibili indirizzi, che sono generalmente espressi in notazione esadecimale, scrivendo due cifre esadecimali per ogni byte.

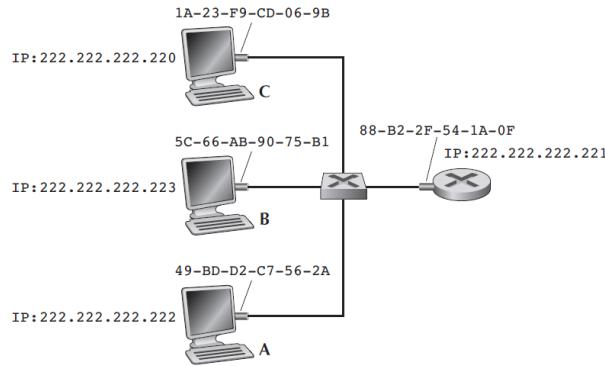
Un'interessante proprietà degli indirizzi MAC è che non esistono due schede di rete con lo stesso indirizzo. La IEEE sovrintende alla gestione degli indirizzi MAC. L'indirizzo MAC di una scheda di rete ha una struttura piatta (cioè non gerarchica) e non cambia mai. Esso è quindi analogo al numero di codice fiscale di una persona, che ha anch'esso struttura orizzontale e non varia a seconda del luogo in cui la persona si trasferisce. Gli indirizzi IP invece sono analoghi all'indirizzo postale di una persona. Questi indirizzi hanno una struttura gerarchica e devono essere aggiornati quando la persona cambia residenza.

Dato che esistono sia indirizzi del livello di rete (come gli indirizzi IP di Internet) sia indirizzi del livello di collegamento (gli indirizzi MAC), si presenta la necessità della loro conversione. Internet affida questo compito al **protocollo di risoluzione degli indirizzi** (ARP, *Address Resolution Protocol*).

Dunque, un modulo ARP traduce un indirizzo IP in un indirizzo MAC. Per molti aspetti è simile al DNS, che traduce i nomi degli host in indirizzi IP. Un'importante differenza fra i

due traduttori consiste nel fatto che DNS esegue l'operazione per host localizzati in qualunque punto di Internet, mentre ARP risolve soltanto gli indirizzi IP per i nodi nella stessa sottorete.

Ma come funziona ARP? Nella RAM dei nodi vi è una **tabella ARP** che contiene la corrispondenza tra indirizzi IP e MAC, oltre a un valore relativo al TTL che indica quando bisognerà eliminare una data voce dalla tabella.

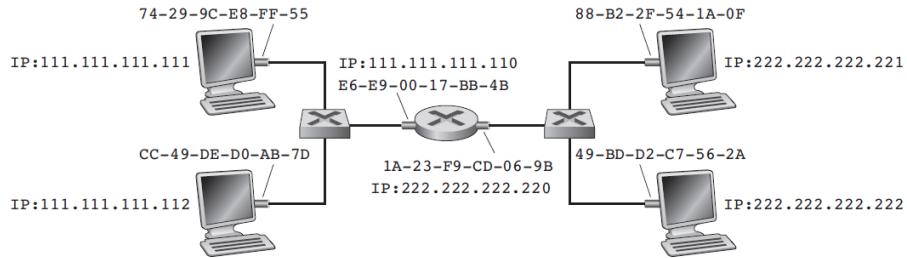


Supponiamo che il nodo 222.222.222.220 voglia inviare un datagramma a un altro nodo della sottorete. Il nodo trasmittente ha la necessità di ottenere l'indirizzo MAC del nodo di destinazione partendo dall'indirizzo IP di quel nodo. Ciò è semplice se la tabella ARP del nodo trasmittente ha una voce per il nodo di destinazione, ma che cosa accade se la tabella al momento non la possiede?

In particolare, supponiamo che il nodo 222.222.222.220 voglia inviare un datagramma al nodo 222.222.222.222. Il nodo trasmittente passa alla scheda di rete un **pacchetto di richiesta ARP** (pacchetto che ha lo scopo di interrogare tutti gli altri nodi della sottorete riguardo all'indirizzo MAC corrispondente all'indirizzo IP da risolvere) insieme all'indicazione di inviare il pacchetto all'indirizzo broadcast della rete, cioè FF-FF-FF-FF-FF-FF. La scheda di rete incapsula il pacchetto ARP in un frame, utilizza l'indirizzo broadcast per la destinazione del frame e lo trasmette nella sottorete.

Il frame contenente la richiesta ARP è ricevuto da tutte le altre schede di rete sulla sottorete. Ciascuna di queste, poiché l'indirizzo è quello broadcast, trasferisce il pacchetto ARP al proprio nodo che controlla se il proprio indirizzo IP corrisponde a quello di destinazione indicato nel pacchetto ARP. L'unico nodo (se esiste) che ha l'indirizzo corrispondente invia al nodo richiedente un frame di risposta ARP con la corrispondenza desiderata. Il nodo richiedente 222.222.222.220 può quindi aggiornare la sua tabella ARP e inviare il datagramma IP, incapsulato in un frame, il cui MAC di destinazione è quello del nodo che ha risposto alla richiesta ARP precedente.

Diamo ora uno sguardo alla situazione più complicata in cui un nodo voglia inviare un datagramma a un host esterno alla propria sottorete (cioè, attraverso un router).



Supponiamo che l'host 111.111.111.111 voglia inviare un datagramma IP all'host 222.222.222.222. Come di consueto, l'host trasmittente passa il datagramma alla sua scheda di rete, ma deve anche comunicargli un appropriato indirizzo MAC di destinazione.

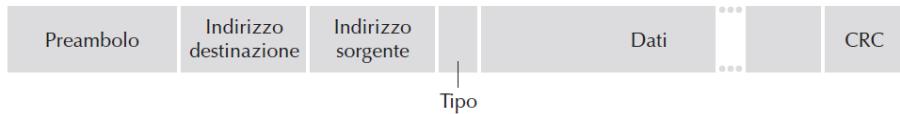
Affinché un datagramma possa passare da 111.111.111.111 a un nodo sulla Sottorete 2, deve prima essere inviato all'interfaccia del router 111.111.111.110. Quindi, l'indirizzo MAC appropriato è quello della scheda di rete del router 111.111.111.110, cioè E6-E9-00-17-BB-4B. Ma come può l'host trasmittente acquisire l'indirizzo MAC di 111.111.111.110? Ovviamente, tramite l'utilizzo di ARP.

Dopo aver ottenuto l'indirizzo MAC, la scheda di rete trasmittente crea un frame e lo trasferisce nella Sottorete 1. La scheda di rete del router sulla Sottorete 1 riconosce che il pacchetto è indirizzato a lui e allora lo passa al livello di rete del router. A questo punto, il datagramma IP è stato trasportato con successo dall'host sorgente al router. Il router consulta quindi la propria tabella e individua la scheda cui va inoltrato il datagramma, ossia l'interfaccia 222.222.222.220, che a sua volta lo trasferisce alla sua scheda di rete; questa incapsula il datagramma in un nuovo frame e lo spedisce nella Sottorete 2. Adesso, l'indirizzo MAC di destinazione del frame è davvero quello finale, che il router ha ottenuto tramite ARP.

## Ethernet

**Ethernet** è una tecnologia per la realizzazione di LAN cablate. Come protocollo MAC viene utilizzato *unslotted CSMA/CD*. In genere, i nodi vengono collegati tra loro mediante uno **switch** secondo una **topologia a stella**.

Adesso, esaminiamo i sei campi del pacchetto di Ethernet.



- **Preambolo** (8 byte). I primi sette byte del preambolo servono per “risvegliare” le schede di rete dei riceventi e sincronizzare i loro clock con quello del trasmittente. Gli ultimi due bit degli otto byte avvisano la scheda di rete del destinatario che “le cose importanti” stanno per arrivare.
- **Indirizzo di destinazione** (6 byte). Campo che contiene l’indirizzo MAC della scheda di rete di destinazione.
- **Indirizzo sorgente** (6 byte). Campo che include l’indirizzo MAC della scheda che trasmette il pacchetto.
- **Tipo** (2 byte). Consente a Ethernet di supportare vari protocolli di rete. È assimilabile al campo “protocollo” nel datagramma del livello di rete e ai campi “numero di porta” nel segmento del livello di trasporto; tutti questi campi servono a far comunicare un protocollo di un livello con quello del livello superiore.
- **Campo dati** (da 46 a 1500 byte). Contiene il datagramma IP. Dato che l’MTU per Ethernet è di 1500 byte, se il datagramma supera questo valore allora l’host deve frammentare il datagramma. Altrimenti, se il datagramma IP è inferiore alla dimensione minima del campo dati, equivalente a 46 byte, il campo dovrà essere “riempito” (*stuffed*) fino a raggiungere quel dato valore.
- **Controllo a ridondanza ciclica – CRC** (4 byte). Consente alla scheda di rete ricevente di rilevare la presenza di un errore nei bit del frame.

A livello di rete, tutte le tecnologie Ethernet forniscono un **servizio senza connessione**, nel senso che quando una scheda di rete vuole inviare un datagramma a un host della rete, non fa altro che incapsularlo in un frame Ethernet e immetterlo nella LAN, senza alcuna forma di handshake preventivo con il destinatario.

Inoltre, forniscono un **servizio non affidabile** a livello di rete. In particolare, quando la scheda di rete B riceve un frame da A, lo sottopone a un controllo CRC, ma non invia un acknowledgement né se il frame supera il controllo né in caso contrario: semplicemente lo scarta. Quindi, A non sa se il frame trasmesso abbia superato il controllo CRC.

## Switch a livello di collegamento

Una LAN Ethernet può essere partizionata in diversi segmenti interconnessi dai seguenti tipi di dispositivi:

- **Hub** – dispositivo di *livello 1* (fisico) che permette di collegare insieme diversi canali fisici, rendendoli di fatto un unico canale fisico dal punto di vista dei sistemi connessi all'hub. Si noti che permette di estendere la massima distanza tra i nodi connessi, in quanto tra due nodi connessi allo stesso hub sono presenti due differenti canali fisici. Tutti i sistemi ed i canali fisici interconnessi da hub costituiscono un **unico segmento LAN**; inoltre,
  - tutti i canali fisici devono essere caratterizzati dalla stessa velocità di trasmissione;
  - i sistemi connessi tramite hub condividono lo stesso **dominio di collisione**.
- **Bridge** – dispositivo di *livello 2* (collegamento) che permette di connettere due segmenti LAN, ognuno avente il proprio dominio di collisione. Ciò è ottenuto mediante i meccanismi di *filtering*, *forwarding* e *flooding*.
- **Switch** – un bridge multi-porta, ovvero che permette di connettere tra loro molteplici segmenti LAN.

Il ruolo dello **switch** è ricevere i frame in ingresso e inoltrarli sui collegamenti in uscita. Lo switch stesso è *trasparente* ai nodi; cioè, un nodo indirizza un frame a un altro nodo, piuttosto che indirizzarlo allo switch e invia il frame nella LAN, senza sapere che uno switch riceverà il frame e lo inoltrerà agli altri nodi.

Alla ricezione di un frame vengono applicati i seguenti principi:

- **Filtering** (filtraggio): se il frame è diretto verso un destinatario MAC noto nello stesso segmento LAN dalla quale proviene, viene **scartato**;
- **Forwarding** (inoltro): se il frame è diretto verso un destinatario MAC noto in un *altro* segmento LAN rispetto a quello da cui proviene, viene **inoltrato sulla relativa interfaccia**;
- **Flooding**: se il frame è diretto verso un destinatario MAC *ignoto*, viene **inoltrato su tutte le interfacce** ad eccezione di quella di provenienza.

L'unione di tali meccanismi permette di ottenere l'**isolamento del traffico** tra i segmenti LAN interconnessi tramite switch e, quindi, di separare i loro domini di collisione, abbassando la frequenza delle collisioni e aumentando così le prestazioni della rete.

Le operazioni di uno switch sono eseguite mediante una **tabella di commutazione** (*switch table*) composta da voci che contengono: (1) l'indirizzo MAC del nodo, (2) l'interfaccia dello switch che conduce al nodo, (3) il momento in cui la voce per quel nodo è stata inserita nella tabella.

Uno switch ha la pregevole proprietà di costruire automaticamente, dinamicamente e in modo autonomo le proprie tabelle, ovvero senza l'intervento di un operatore o di un

protocollo di configurazione. In altre parole, si potrebbe dire che gli switch **auto-apprendono**:

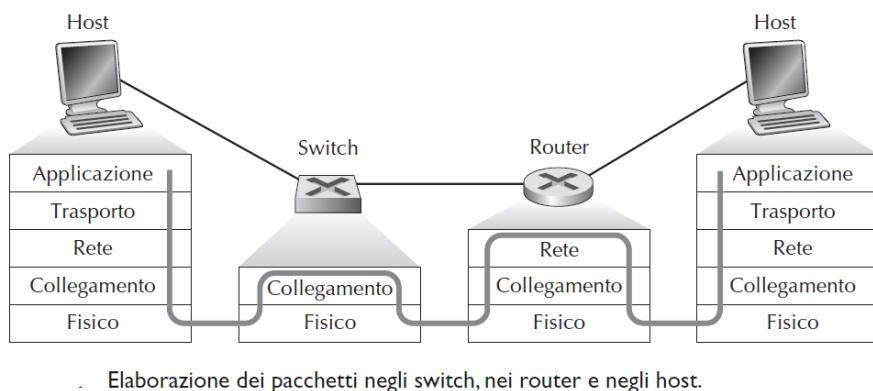
1. La tabella è inizialmente vuota.
2. Di ogni frame che riceve, lo switch archivia nella sua tabella (1) l'indirizzo MAC del campo indirizzo sorgente del frame, (2) l'interfaccia da cui arriva il frame, (3) il momento di arrivo, registrando in tal modo il segmento LAN su cui risiede il nodo trasmittente. Quando tutti i nodi nella LAN avranno inviato un frame, allora la tabella sarà completa.
3. Quando, dopo un dato periodo di tempo detto ***aging time*** (tempo di invecchiamento), lo switch non riceve frame da un determinato indirizzo sorgente, lo cancella dalla tabella.

Inoltre, gli switch sono **dispositivi plug-and-play**, in quanto non richiedono interventi dell'amministratore di rete o dell'utente. Basta semplicemente collegare i segmenti di LAN alle sue interfacce, senza dover configurare le tabelle al momento dell'installazione o quando un host è rimosso da un segmento LAN.

L'uso di soli switch nella realizzazione di una rete LAN permette di:

- **eliminare le collisioni**: ogni nodo della LAN è connesso ad altri tramite un canale full-duplex dedicato. Risulta perciò che ogni singolo canale corrisponde ad un segmento LAN con il proprio dominio di collisione. Dato che gli switch inoltrano un frame alla volta sui canali tramite *store-and-forward*, le collisioni non possono più verificarsi.
- **utilizzare canali eterogenei**: ogni canale può operare con le proprie modalità e ad una velocità arbitraria, in quanto isolato da tutti gli altri. Ciò permette di soddisfare in maniera più efficiente le necessità prestazionali della rete LAN.
- **semplificare la gestione**: è possibile prevedere ridondanza per fronteggiare guasti e, nel caso non sia possibile, scollegare dalla LAN solamente l'host che riscontra problemi di connessione. Inoltre, gli switch raccolgono statistiche sull'uso dei canali, permettendo di diagnosticare e valutare in maniera accurata le prestazioni della rete LAN.

### Confronto switch vs. router



Confrontiamo il funzionamento di uno switch/bridge con quello di un router. Per il **router**:

- Dispositivo di livello 3, ossia del livello di rete.
- Mantiene una tabella di inoltro ed implementa algoritmi di instradamento.
- [PRO] Supporta topologie arbitrarie e limita i cicli tramite uso di TTL e degli algoritmi di instradamento.
- [PRO] Fornisce protezione contro tempeste broadcast.
- [CON] Le operazioni sono complesse e richiedono maggiore processamento dei pacchetti.
- [CON] Richiede la configurazione dell'indirizzo IP e apprendono l'instradamento dai router vicini.

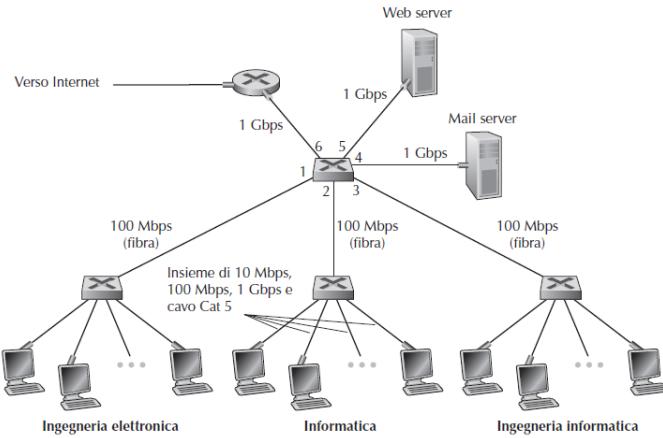
Per lo **switch/bridge**:

- Dispositivo di livello 2, ossia del livello di collegamento.
- Mantiene una tabella di switching e svolge solamente funzioni di inoltro.
- [PRO] Le operazioni sono semplici e richiedono meno processamento dei frame.
- [PRO] Popolano automaticamente la tabella di switching.
- [CON] Tutto il traffico passa per l'albero ricoprente della LAN, anche se altri canali sono disponibili.
- [CON] Non fornisce protezione contro tempeste broadcast.

|                             | <b>Hub</b> | <b>Bridge</b>    | <b>Switch</b>    | <b>Router</b> |
|-----------------------------|------------|------------------|------------------|---------------|
| <b>Livello</b>              | 1 - Fisico | 2 - Collegamento | 2 - Collegamento | 3 - Rete      |
| <b>Isolamento traffico</b>  | No         | Si               | Si               | Si            |
| <b>Trasparente</b>          | Si         | Si               | Si               | No            |
| <b>Plug-and-Play</b>        | Si         | Si               | Si               | No            |
| <b>Instradamento ottimo</b> | No         | No               | No               | Si            |

## LAN Virtuali (VLAN)

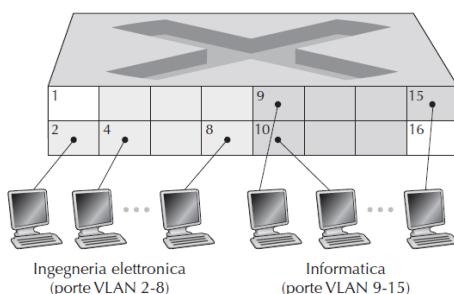
Possiamo identificare tre inconvenienti nella configurazione rappresentata dalla figura a seguire.



- *Mancanza di isolamento del traffico.* Sebbene la gerarchia localizzi il traffico di un gruppo in un solo switch, il traffico broadcast deve ancora attraversare l'intera rete istituzionale. Le prestazioni della LAN aumenterebbero se si potesse limitare il campo di tale traffico broadcast e forse, ancora più importante, sarebbe limitarlo per ragioni di sicurezza e riservatezza. Questo tipo di isolamento potrebbe essere fornito sostituendo lo switch centrale della figura con un router.
- *Uso inefficiente degli switch.* Se invece di 3 gruppi, l'istituzione ne avesse 10, sarebbero necessari 10 switch di primo livello (quelli che raccolgono gruppi di host). Se ogni gruppo fosse piccolo, meno di 10 persone, un singolo switch a 96 porte sarebbe probabilmente sufficiente, ma non fornirebbe isolamento di traffico.
- *Gestione degli utenti.* Se un dipendente si muovesse tra più gruppi, sarebbe necessario cambiare la posatura della rete per connetterlo a un altro switch.

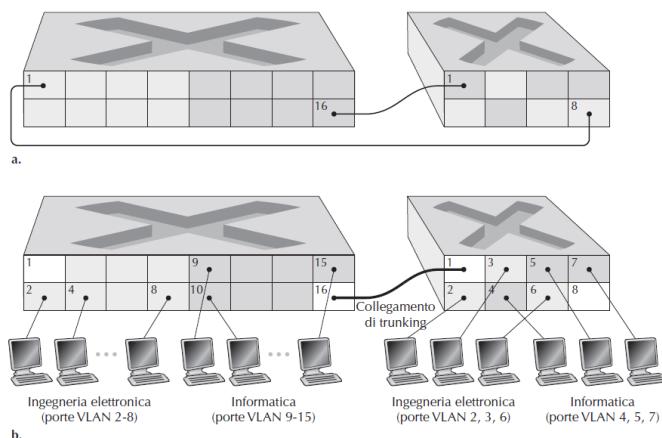
Queste difficoltà possono essere superate utilizzando uno switch che supporti una **virtual local area network (VLAN o virtual LAN)**, che permette di definire più reti locali virtuali su una singola infrastruttura fisica di rete locale. Gli host all'interno di una VLAN comunicano tra loro come se fossero tutti (e nessun altro) connessi allo switch.

In una VLAN basata sulle porte, le porte (interfacce) dello switch vengono divise dal gestore di rete in gruppi, ognuno dei quali costituisce una VLAN le cui porte formano un dominio broadcast. La figura mostra un singolo switch con 16 porte.



È facile immaginarsi come lo switch sia configurato e funzioni: il gestore di rete dichiara che una porta appartiene a una determinata VLAN (le porte non dichiarate appartengono a una VLAN di default) utilizzando un software per la gestione dello switch. Una tabella di associazione tra porte e VLAN viene mantenuta all'interno dello switch; l'hardware dello switch si limita a consegnare frame tra porte appartenenti alla stessa VLAN.

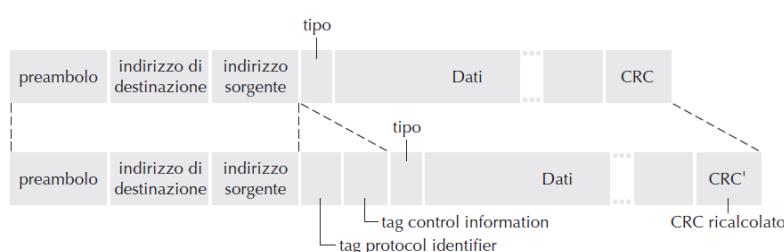
Supponiamo che i dipartimenti siano ospitati in molti edifici separati dove debba essere fornito accesso alla rete e che quest'ultimo faccia parte della VLAN del dipartimento. La figura mostra un secondo switch a 8 porte, ognuna definita come appartenente o alla VLAN di Ingegneria elettronica o a quella di Informatica. Ma come dovrebbero interconnettersi questi due switch?



Una soluzione semplice consisterebbe nel definire che una porta appartenga alla VLAN di Informatica su ogni switch (e in modo simile per la VLAN di Ingegneria) e nel connettere tra di loro queste porte (soluzione (a)). Tuttavia, tale soluzione *non è scalabile*, in quanto  $N$  VLAN richiederebbero  $N$  porte per ogni switch, semplicemente per interconnettere i due switch.

Un approccio più scalabile per l'interconnessione tra switch VLAN è noto come **VLAN trunking**. In questo approccio (soluzione (b)), una porta speciale per ogni switch viene configurata come porta di trunking per interconnettere i due switch VLAN. La porta di trunking appartiene a tutte le VLAN e i frame inviati a qualunque VLAN vengono inoltrati attraverso il collegamento di trunking all'altro switch. Ma come fa uno switch a sapere che un frame che arriva a una porta di trunking appartiene a una VLAN particolare?

IEEE ha definito un formato esteso di frame Ethernet nello standard 802.1Q, per frame che attraversano un trunk VLAN. Esso consiste nel frame standard Ethernet con aggiunta una **etichetta VLAN** (o *tag VLAN*) di quattro byte nell'intestazione che trasporta l'identità della VLAN a cui il frame appartiene.



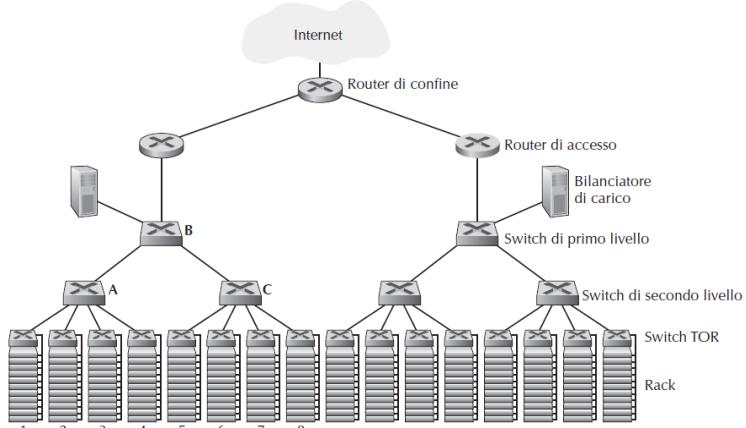
L'etichetta VLAN è aggiunta al frame dallo switch sul lato di trasmissione del trunk, mentre viene analizzata e rimossa dallo switch sul lato ricevente. L'etichetta VLAN consiste di un campo TPID (*tag protocol identifier*) di due byte (con un valore fisso esadecimale di 81-00) e un campo *tag control information* di due byte, contenente il campo di identificazione della VLAN a 12 bit e un campo di priorità a 3 bit, simile al campo TOS del datagramma IP.

### La rete dei data center

I **data center** sono strutture che ospitano da decine a centinaia di migliaia di host. Ognuno di essi ha una sua rete, detta ***data center network***, che interconnette tra loro gli host e interconnette il data center con Internet.

In un data center le api operaie sono gli host: forniscono contenuti, memorizzano e-mail e documenti e collettivamente eseguono calcoli massivi distribuiti. Gli host nei data center sono generalmente calcolatori dotati solo di CPI, memoria, e spazio disco (**blade**). Sono riposti in speciali armadi indicati generalmente con il termine **rack**, ognuno avente tipicamente dalle 20 alle 40 blade. Associato a ogni rack c'è uno switch (che, per abitudine, viene posto in cima al rack), chiamato ***top of rack (TOR) switch***, che interconnette gli host del rack tra di loro e con gli altri switch del data center. Inoltre, a ogni host viene assegnato un indirizzo IP interno al data center.

Per gestire i flussi di traffico tra client esterni e host interni la rete del data center ha uno o più **router di confine** (*border router*), che connettono la rete del data center a Internet. La figura mostra un esempio di una rete di un data center.

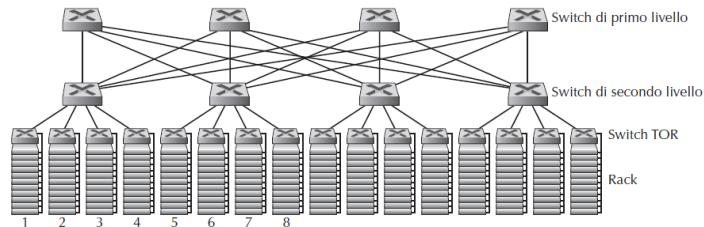


All'interno del data center le richieste esterne vengono prima dirette a un **load balancer** (bilanciatore di carico) il cui compito è distribuire le richieste agli host, bilanciando il lavoro in funzione del loro carico corrente.

Quando il load balancer riceve una richiesta per una particolare applicazione, la inoltra a uno degli host che gestisce tale applicazione. Un host può quindi richiedere i servizi di altri host che lo aiutino a processare la richiesta. L'host, quando finisce di elaborare la richiesta, invia la risposta al load balancer, che a sua volta la ritrasmette al client esterno.

Per estendersi fino a decine e centinaia di migliaia di host, un data center deve spesso impiegare una **gerarchia di router e switch** come quella mostra sopra.

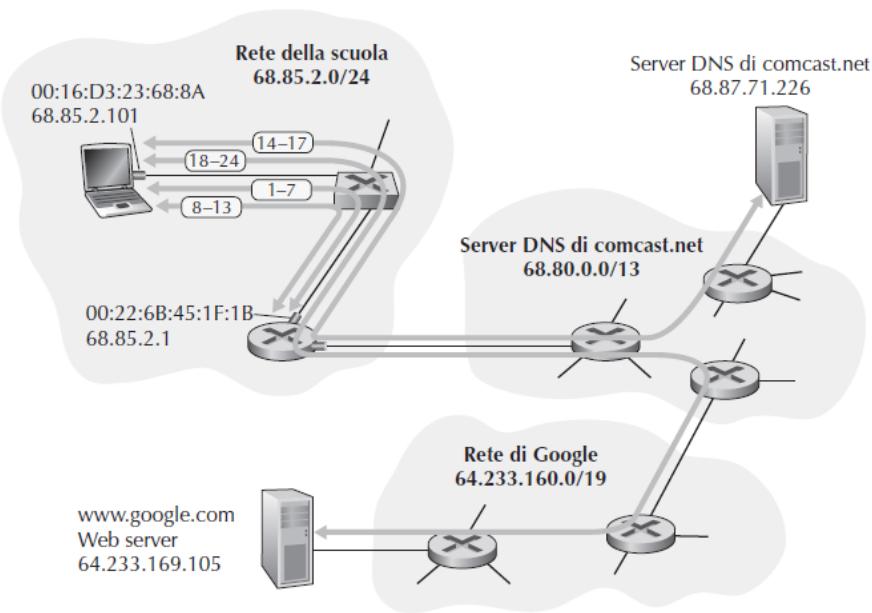
Per ridurre i costi dei data center e allo stesso tempo migliorare le prestazioni di throughput e ritardo, i giganti del cloud computing su Internet sono continuamente alla ricerca di nuove strutture per la rete dei data center. Un approccio possibile è sostituire la gerarchia di switch e router con una **topologia completamente connessa** come quella di seguito.



In questo progetto ogni switch di primo livello si connette a tutti gli switch di secondo livello in modo che il traffico da host a host non debba mai risalire i livelli di switch e che con  $n$  switch di primo livello esistano  $n$  percorsi disgiunti tra qualsiasi coppia di switch di secondo livello.

## Cronaca di una richiesta di una pagina web

Il nostro viaggio attraverso la pila di protocolli è completo. Ricapitoliamolo considerando una versione integrata e totale di quello che abbiamo imparato finora. Un modo per ottenere questa visione globale è identificare i tanti protocolli coinvolti quando si soddisfa anche la più semplice delle richieste: **scaricare una pagina web**. La figura mostra il nostro esempio: uno studente, Bob, connette il proprio laptop a uno switch Ethernet della sua scuola e scarica una pagina web (e.g. la home page di [www.google.com](http://www.google.com)).



### DHCP, UDP, IP e Ethernet

Supponiamo che Bob accenda il suo laptop e lo colleghi a un **cavo Ethernet** connesso allo **switch Ethernet della scuola**, che a sua volta è connesso al **router della scuola**, come mostrato in figura. Il router della scuola è connesso a un **ISP**, che fornisce il **servizio DNS** alla scuola. Quindi, il server DNS risiede nella **rete Comcast** piuttosto che nella rete della scuola. Assumiamo che il **server DHCP** sia eseguito nel router.

Quando Bob connette il suo laptop alla rete, non può fare niente, nemmeno scaricare una pagina web, senza un **indirizzo IP**. Quindi, la prima azione riguardante la rete intrapresa dal laptop di Bob è quella di eseguire il **protocollo DHCP** per ottenere un indirizzo IP, insieme ad altre informazioni, dal server DHCP locale.

1. Il sistema operativo del laptop di Bob crea un **messaggio DHCP request** e inserisce questo messaggio in un **segmento UDP** con porta di destinazione 67 (server DHCP) e porta sorgente 68 (client DHCP). Il segmento UDP viene quindi inserito all'interno di un **datagramma IP** con indirizzo IP di destinazione broadcast (255.255.255.255) e indirizzo IP sorgente 0.0.0.0, in quanto il laptop di Bob non ha ancora un indirizzo IP.
2. Il datagramma IP contenente il messaggio di richiesta DHCP è quindi posto in un **frame Ethernet**, che ha indirizzo MAC di destinazione FF:FF:FF:FF:FF:FF in modo che il frame venga inviato in broadcast a tutti i dispositivi connessi allo **switch**, tra

i quali si spera ci sia un server DHCP; l'indirizzo MAC della sorgente del frame è quello del laptop di Bob **00:16:D3:23:68:8A**.

3. Il frame Ethernet broadcast contenente la richiesta DHCP è il **primo frame inviato dal laptop di Bob allo switch Ethernet**. Lo switch invia in broadcast il frame in entrata a tutte le porte in uscita, compresa la porta che lo connette al **router**.
4. Il router riceve il frame broadcast Ethernet che contiene la richiesta DHCP sulla sua interfaccia con indirizzo MAC **00:22:6B:45:1F:1B** e il datagramma IP è estratto dal frame Ethernet. L'indirizzo IP di destinazione del datagramma broadcast indica che tale datagramma IP dovrebbe essere elaborato dai protocolli di livello superiore su questo nodo; quindi, sul **payload del datagramma** (un **segmento UDP**) viene effettuato un **demultiplexing** verso UDP e il messaggio di richiesta DHCP viene estratto dal segmento UDP. Ora, il server DHCP ha il messaggio di richiesta DHCP.
5. Supponiamo ora che il server DHCP in esecuzione sul router possa allocare indirizzi IP nel blocco **CIDR 65.85.2.0/24**. Supponiamo che il server DHCP allochi l'indirizzo **68.85.2.101** al laptop di Bob. Il server DHCP crea un **messaggio DHCP ACK** contenente tale indirizzo IP, insieme all'indirizzo IP del server DNS (68.87.71.226), l'indirizzo IP del gateway di default (68.85.2.1) e il blocco della sottorete (68.85.2.0/24). Il messaggio DHCP è posto all'interno di un segmento UDP inserito in un datagramma IP a sua volta posto in un frame Ethernet. Il frame Ethernet ha come indirizzo MAC sorgente quello dell'interfaccia del router sulla rete domestica (**00:22:6B:45:1F:1B**) e come indirizzo MAC di destinazione quello del laptop di Bob (**00:16:D3:23:68:8A**).
6. Il frame Ethernet contenente il messaggio DHCP ACK è inviato (in unicast) dal router allo switch. Lo switch, poiché è in grado di *auto-apprendere* e ha precedentemente ricevuto dal laptop di Bob un frame Ethernet contenente la richiesta DHCP, sa di dover inoltrare un frame indirizzato a **00:16:D3:23:68:8A** solo alla porta di uscita verso il laptop di Bob.
7. Il laptop di Bob riceve il frame Ethernet contenente il messaggio DHCP ACK, estraе dal frame Ethernet il datagramma IP, estraе dal datagramma IP il segmento UDP ed estrarre dal segmento UDP il messaggio DHCL ACK. Il client DHCP di Bob memorizza il suo indirizzo IP e quello del server DNS. Inoltre, installa l'indirizzo del gateway di default nella sua **tabella di inoltro**. A questo punto, il laptop di bob ha inizializzato le sue componenti di rete ed è pronto a iniziare a elaborare la lettura della pagina web.

## DNS e ARP

Quando Bob scrive l'URL di `www.google.com` nel suo browser web inizia la lunga catena di eventi che alla fine porterà a visualizzare la home page di Google. Il web browser di Bob inizia il processo creando una **socket TCP** che verrà usata per inviare una **richiesta HTTP** a `www.google.com`. Il laptop di Bob, per creare la socket, deve conoscere l'indirizzo IP di `www.google.com`. Il **protocollo DNS** viene usato per fornire il servizio di traduzione da nome a indirizzo IP.

8. Il sistema operativo del laptop di Bob crea un **messaggio di DNS query** inserendo la stringa “`www.google.com`” nella sezione riguardante la richiesta del messaggio DNS. Tale messaggio DNS è quindi posto all'interno di un **segmento UDP** con porta di destinazione 53 (server DNS). Il segmento UDP è quindi posto all'interno di un **datagramma IP** con indirizzo IP di destinazione `68.87.71.226` (l'indirizzo del server DNS restituito nel messaggio DHCP ACK al passo 5) e indirizzo IP sorgente `68.85.2.101`.
9. Il laptop di Bob, quindi, inserisce il datagramma contenente il messaggio di richiesta DNS in un **frame Ethernet** che verrà inviato, con indirizzo a livello di collegamento, al **gateway della rete della scuola** di Bob. Tuttavia, il laptop di Bob non ne conosce l'indirizzo MAC, per ottenere il quale deve usare il **protocollo ARP**.
10. Il laptop di Bob crea un messaggio di **ARP query** con indirizzo IP di destinazione `68.85.2.1` (il gateway di default), pone il messaggio ARP all'interno di un frame Ethernet con indirizzo di destinazione broadcast `FF:FF:FF:FF:FF` e invia il frame Ethernet allo switch, che lo consegna a tutti i dispositivi connessi compreso il gateway.
11. Il router gateway riceve il frame e scopre che l'indirizzo IP `68.85.2.1` nel messaggio ARP corrisponde all'indirizzo IP della sua interfaccia. Il gateway prepara quindi un messaggio **ARP reply** che indica che il suo indirizzo MAC `00:22:6B:45:1F:1B` corrisponde all'indirizzo IP `68.85.2.1`. Pone il **messaggio di risposta ARP** in un frame Ethernet con l'indirizzo di destinazione `00:16:D3:23:68:8A` (il laptop di Bob) e invia il frame allo switch che lo consegna al laptop di Bob.
12. Il laptop di Bob riceve il frame contenente il messaggio di risposta ARP ed estrarre l'indirizzo MAC del gateway (`00:22:6B:45:1F:1B`) dal messaggio di risposta ARP.
13. Il **laptop di Bob può ora indirizzare il frame Ethernet contenente l'interrogazione DNS all'indirizzo MAC del gateway**. Il laptop di Bob invia questo frame allo switch che lo **consegna al gateway**.

## Instrandamento intra-dominio al server DNS

14. **Il gateway riceve il frame** ed estrae il datagramma IP contenente l'interrogazione DNS. Il router ricerca l'indirizzo di destinazione di tale datagramma (68.87.71.226) e determina sulla base della sua **tabella di inoltro** che il datagramma dovrebbe essere inviato al router più a sinistra nella rete Comcast mostrata nella figura. Il datagramma IP è posto all'interno di un frame appropriato per il collegamento che connette il router della scuola al router Comcast più a sinistra; il frame viene trasmesso.
15. **Il router riceve il frame**, estrae il datagramma IP, esamina l'indirizzo di destinazione del datagramma (68.87.71.226) e determina l'interfaccia di uscita sulla quale inoltrare il datagramma al server DNS sulla base della sua tabella di inoltro, che è stata riempita dal protocollo intra-dominio di Comcast (come **OSPF**) e dal **protocollo inter-dominio di Internet, BGP**.
16. Infine, il datagramma IP contenente l'interrogazione DNS **arriva al server DNS**, che estrae il messaggio di interrogazione DNS, ricerca il nome **www.google.com** nel suo database DNS e trova il **record di risorsa DNS** che contiene l'indirizzo IP (64.233.169.105) di **www.google.com** (assumendo che sia nella cache del server DNS). Il server DNS scrive un messaggio **DNS reply** contenente la corrispondenza tra il nome dell'host e l'indirizzo IP e lo pone in un **segmento UDP**; infine, pone il segmento all'interno di un **datagramma IP indirizzato al laptop di Bob** (68.85.2.101). Questo datagramma verrà restituito al router della scuola attraverso la rete Comcast e da qua al laptop di Bob tramite lo switch Ethernet.
17. Il laptop di bob estrae l'indirizzo IP del server **www.google.com** dal messaggio DNS. Finalmente, **il laptop di Bob è pronto a contattare il server www.google.com**.

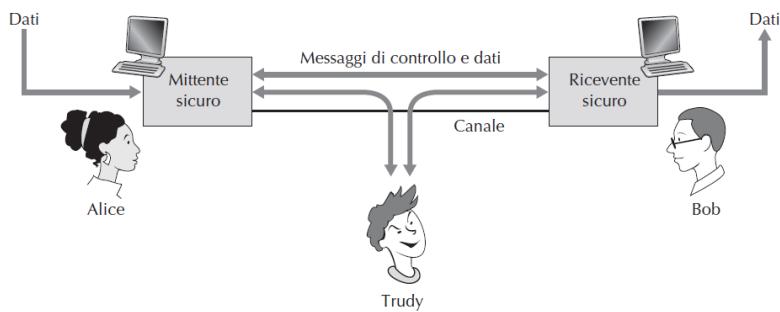
## Interazione client-server: TCP e HTTP

18. Il laptop di Bob ora può creare la **socket TCP** che verrà usata per inviare un messaggio di **HTTP GET** a **www.google.com**. Quando Bob crea la socket TCP, per prima cosa il laptop di Bob effettua **l'handshake a tre vie** con TCP su **www.google.com**. Quindi, il laptop di Bob crea un **segmento TCP SYN** con porta di destinazione 80 (per HTTP), pone il segmento TCP all'interno di un datagramma IP di destinazione **64.233.169.105** (**www.google.com**), pone il datagramma all'interno di un frame con indirizzo MAC di destinazione **00:22:6B:45:1F:1B** (il gateway) e invia il frame allo switch.
19. I **router inoltrano il datagramma contenente TCP SYN a www.google.com**, usando ognuno la propria tabella di inoltro.
20. Infine, il **datagramma contenente il TCP SYN arriva a www.google.com**. Il messaggio **TCP SYN viene estratto** dal datagramma e viene effettuato un **demultiplexing** verso la socket di benvenuto associata alla porta 80. Viene creata una **socket di connessione** per la connessione TCP tra il server HTTP di Google e il laptop di Bob. Un **segmento TCP SYNACK** viene generato, posto all'interno di un datagramma indirizzato al laptop di Bob e infine inserito in un frame appropriato al collegamento tra **www.google.com** e il primo router.
21. Il datagramma contenente il segmento **TCP SYNACK viene inoltrato** attraverso le reti di Google, Comcast e della scuola per arrivare infine alla scheda Ethernet del laptop di Bob. Viene effettuato un **demultiplexing del datagramma** all'interno del sistema operativo alla socket TCP creata al passo 18, che entra nello stato di connessione.
22. Con la **socket** sul laptop di Bob finalmente **pronta per trasmettere byte** a **www.google.com**, il browser di Bob crea il messaggio HTTP GET contenente l'URL da leggere. Quindi, il **messaggio HTTP GET viene scritto nella socket** e diventa il payload di un **segmento TCP**. Il segmento TCP viene posto in un datagramma, inviato e consegnato a **www.google.com** come descritto nei passi da 18 a 20.
23. Il server HTTP di **www.google.com** legge dalla socket TCP il messaggio HTTP GET, crea un messaggio di **risposta HTTP**, pone il contenuto della pagina web richiesta nel corpo del messaggio di risposta HTTP che invia alla socket TCP.
24. Il datagramma contenente il **messaggio di risposta HTTP viene inoltrato** attraverso le reti di Google, Comcast e della scuola **al laptop di Bob**. Il browser di **Bob legge dalla socket la risposta HTTP**, estrae il codice HTML della pagina web dal corpo della risposta HTTP e finalmente **visualizza la pagina web**.

# Network Security

Possiamo identificare le proprietà auspicabili per la **comunicazione sicura**.

- *Riservatezza.* Solo mittente e destinatario dovrebbero essere in grado di comprendere il contenuto del messaggio trasmesso. Dato che questo può essere intercettato da qualche “spione” (*eavesdropper*) è necessario **cifrarlo** in modo da renderlo incomprensibile a chi lo intercetta.
- *Integrità del messaggio.* Occorre che il contenuto della comunicazione non subisca, durante la trasmissione, alterazioni dovute a cause fortuite o a manipolazioni. Per garantire l'integrità dei messaggi si possono utilizzare estensioni delle tecniche di checksum.
- *Autenticazione.* Mittente e destinatario devono essere reciprocamente sicuri della loro identità, cioè devono poter confermare che l'altra parte sia effettivamente chi dichiara di essere.
- *Sicurezza operativa.* Dispositivi operativi, come firewall e sistemi di rilevamento delle intrusioni, sono usati per contrastare gli attacchi contro le reti delle istituzioni. Un firewall si pone tra la rete dell'istituzione e quella pubblica e controlla i pacchetti in transito. Un sistema di rilevamento delle intrusioni esegue un controllo approfondito dei pacchetti, avvisando gli amministratori di rete in caso di attività sospette.



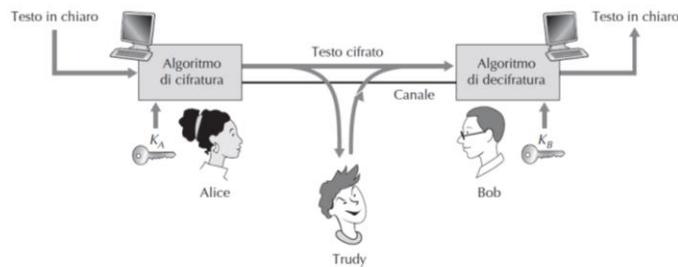
La figura illustra uno scenario in cui Alice vuole inviare un'e-mail a Bob. Per comunicare in modo sicuro, rispettando i requisiti di riservatezza, autenticazione e integrità di messaggio, i due si scambieranno pacchetti di controllo e di dati, in modo similare a quanto avviene con TCP, e tutti o alcuni di questi messaggi saranno tipicamente cifrati. A questo punto un malintenzionato (Trudy) potrebbe procedere con una delle seguenti azioni.

- *Ascoltare e registrare i messaggi di controllo e dati in transito sul canale* (intercettazione o *eavesdropping*).
- *Rimuovere, aggiungere o modificare i messaggi o il loro contenuto.*

Se non si adottano appropriate contromisure, l'intruso ha quindi la possibilità di mettere in atto un'ampia gamma di attacchi: spiare la conversazione (eventualmente rubando password e dati), impersonare un altro soggetto, dirottare (*hijack*) una sessione in corso, negare il servizio agli utenti legittimi, sovraccaricando le risorse del sistema, e così via.

## Principi di crittografia

In pratica, le tecniche di crittografia consentono al trasmittente di mascherare i dati in modo che un intruso non possa comprenderne il contenuto. Il ricevente, naturalmente, deve invece essere in grado di recuperare i dati originali.



Supponiamo che Alice voglia inviare un messaggio a Bob. Il messaggio originario scritto da Alice è detto **testo in chiaro** (*plaintext* o *cleartext*); questo poi viene trasformato da un **algoritmo di cifratura** in un **messaggio cifrato** (*ciphertext*) che risulta illeggibile a qualsiasi intruso. È importante notare che in molti casi le tecniche di crittografia, comprese quelle utilizzate in Internet, sono di *dominio pubblico*, standardizzate e disponibili a chiunque, anche a un potenziale intruso. È evidente che, se tutti conoscono il metodo per cifrare i dati, occorrono informazioni segrete che impediscono ai curiosi malevoli di decifrare i dati trasmessi. Qui entrano in gioco le chiavi.

Nella figura Alice fornisce all'algoritmo di cifratura una **chiave**,  $K_A$ , cioè una stringa alfanumerica che genera il testo cifrato. La notazione  $K_A(m)$  denota la forma cifrata utilizzando la chiave  $K_A$  del messaggio in chiaro,  $m$ . Lo specifico algoritmo di cifratura che utilizza la chiave  $K_A$  risulterà evidente dal contesto. Analogamente, Bob fornisce una chiave,  $K_B$ , all'**algoritmo di decifratura** che legge il testo cifrato e restituisce quello originale in chiaro. Cioè, quando Bob riceve un messaggio cifrato  $K_A(m)$ , lo può decifrare calcolando  $K_B(K_A(m)) = m$ .

Nei cosiddetti **sistemi a chiave simmetrica** le chiavi di Alice e Bob sono identiche e segrete. Nei **sistemi a chiave pubblica** si utilizzano due chiavi. Una è di pubblico dominio, l'altra è conosciuta o da Bob o da Alice, ma non da entrambi.

## Crittografia a chiave simmetrica

A titolo esemplificativo soffermiamoci su un antico algoritmo a chiave simmetrica conosciuto come **cifrario di Cesare**: si procede alla sostituzione di ciascuna lettera del messaggio in chiaro con un'altra sfasata, rispetto alla prima, di un numero  $k$  di posti nell'alfabeto.

Uno sviluppo rispetto al metodo ancora esposto è costituito dal cosiddetto **cifrario monoalfabetico**. Anche in questo caso si procede allo scambio di una lettera dell'alfabeto con un'altra, la sostituzione non avviene però seguendo uno schema regolare, ma cambiando tutte le occorrenze di una data lettera con un'altra – sempre la stessa – scelta in modo arbitrario. Questo cifrario appare più efficiente di quello di Cesare, in quanto usando un alfabeto di 26 lettere (come quello inglese) ci troviamo di fronte a ben 26 fattoriale differenti possibilità di accoppiamento (un valore dell'ordine di  $10^{26}$ ) anziché le 25 di prima. Tuttavia, basandosi su informazioni statistiche, come il fatto che le lettere “e” e “t” sono quelle che ricorrono con maggiore frequenza nelle parole inglesi (valgono il 13% e il 9% delle

occorrenze) o che alcuni gruppi di lettere appaiono spesso insieme (“che”, “zione”, “mente”, ...) è possibile rendere più agevole la violazione del codice.

In generale, si possono immaginare tre differenti scenari di decodifica, in funzione delle informazioni possedute dall’intruso.

- *Attacco al testo cifrato.* È il caso in cui l’intruso ha solamente accesso al testo intercettato senza alcuna indicazione sul contenuto del messaggio e può avere come unico supporto l’analisi statistica.
- *Attacco con testo in chiaro noto.* Si verifica quando l’intruso conosce alcuni possibili accoppiamenti (testo in chiaro/testo cifrato).
- *Attacco con testo in chiaro scelto.* Costituisce il caso in cui l’intruso è in grado di ottenere la forma cifrata di un messaggio a lui noto. Se Trudy riesce a intercettare un messaggio di Alice che contiene tutte le lettere dell’alfabeto, allora può decifrare lo schema crittografico.

La **cifratura polialfabetica** rappresenta un’ulteriore evoluzione delle precedenti forme di crittografia attraverso l’impiego di molteplici sostituzioni monoalfabetiche. In sostanza, le varie occorrenze di una stessa lettera vengono codificate in modo diverso a seconda della posizione in cui appaiono nel messaggio in chiaro. La figura mostra uno schema di cifratura polialfabetica con due diversi cifrari di Cesare,  $C_1$  con  $k = 5$  e  $C_2$  con  $k = 19$  che potremmo scegliere di utilizzare seguendo la sequenza  $C_1, C_2, C_2, C_1, C_2$ . Cioè, la prima lettera del testo in chiaro deve essere sostituita con  $C_1$ , la seconda e la terza con  $C_2$ , la quarta con  $C_1$  e la quinta con  $C_2$ . Lo schema si ripete in modo ciclico.

|                    |   |
|--------------------|---|
| Lettere in chiaro: | a b c d e f g h i j k l m n o p q r s t u v w x y z |
| $C_1(k=5)$ :       | f g h i j k l m n o p q r s t u v w x y z a b c d e |
| $C_2(k=19)$ :      | t u v w x y z a b c d e f g h i j k l m n o p q r s |

### Cifrari a blocchi

Esaminiamo ora come viene effettuata la crittografia a chiave simmetrica. Esistono due grandi classi di tecniche di cifratura: i **cifrari a flusso** e i **cifrari a blocchi**.

In un cifrario a blocchi, il messaggio da cifrare è elaborato in blocchi da  $k$  bit. Per esempio, se  $k = 64$ , allora il messaggio viene suddiviso in blocchi di 64 bit e ciascun blocco viene cifrato in modo indipendente. Per codificare un blocco, il cifrario usa una corrispondenza uno a uno per far corrispondere il blocco di  $k$  bit di testo in chiaro ai blocchi di  $k$  bit di testo cifrato.

Supponiamo che  $k = 3$  in modo che il cifrario a blocchi faccia corrispondere 3 bit in ingresso (testo in chiaro) a tre bit in uscita (testo cifrato). Una possibile corrispondenza è data dalla tabella.

| Ingresso | Uscita | Ingresso | Uscita |
|----------|--------|----------|--------|
| 000      | 110    | 100      | 011    |
| 001      | 111    | 101      | 010    |
| 010      | 101    | 110      | 000    |
| 011      | 100    | 111      | 001    |

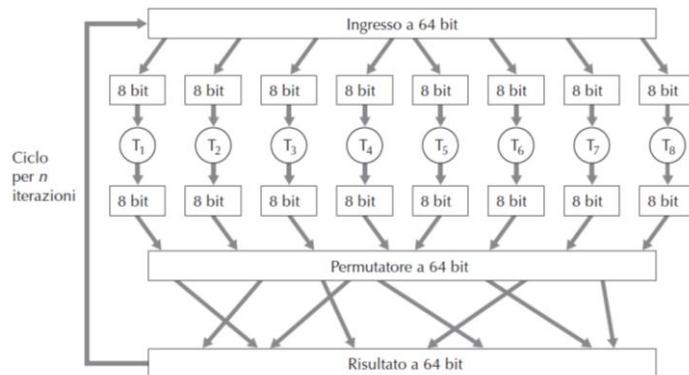
Notiamo che vi è una corrispondenza uno-a-uno: cioè vi è un'uscita diversa per ciascun ingresso.

Quante possibili corrispondenze ci sono? Per rispondere a questa domanda, osserviamo che una corrispondenza non è nient'altro che una permutazione di tutti i possibili ingressi. Ci sono  $2^3$  possibili ingressi che possono essere permutati in  $8! = 40.320$  possibili corrispondenze. Possiamo vedere ciascuna corrispondenza come una chiave.

L'attacco a forza bruta per questo cifrario consiste nel provare a decifrare il testo cifrato usando tutte le possibili corrispondenze. I cifrari a blocchi tipicamente usano blocchi molto più grandi con  $k = 64$  o maggiore.

Sfortunatamente, sono difficili da implementare. Per  $k = 64$  e una certa corrispondenza, Alice e Bob dovrebbero mantenere una tabella con  $2^{64}$  valori in ingresso: un compito improponibile. Inoltre, se Alice e Bob devono cambiare la chiave, dovrebbero entrambi rigenerare la tabella.

Invece, i cifrari a blocchi usano tipicamente **funzioni che simulano in modo casuale tabelle permutate**. Un esempio di una funzione di questo tipo, per  $k = 64$ , è illustrato in figura.



La funzione prima suddivide il blocco di 64 bit in 8 parti di 8 bit ciascuna. Ciascuna parte viene elaborata da una tabella  $8 \times 8$  bit, che ha quindi una dimensione più maneggevole. Per esempio, il primo pezzo è elaborato dalla tabella indicata con  $T_1$ . Successivamente, le otto parti in uscita vengono riassembrate nel blocco a 64 bit. Le posizioni dei 64 bit nel blocco vengono poi mescolate (permutate) per produrre l'uscita a 64 bit. Questo risultato viene rinviato all'ingresso a 64 bit, dove inizia un'altra iterazione. Dopo  $n$  di queste iterazioni, la funzione fornisce il testo cifrato del blocco a 64 bit.

La chiave di questo algoritmo di cifratura a blocchi sarà costituita dalle 8 tabelle di permutazione, supponendo che sia fissa la funzione di mescolamento.

Oggi ci sono parecchi cifrari a blocchi comuni, compreso **DES** (*Data Encryption Standard*), **3DES** e **AES** (*Advanced Encryption Standard*). Ciascuno di questi standard usa funzioni e ciascuno di questi algoritmi usa anche una stringa di bit come chiave. Per esempio, DES usa dei blocchi a 64 bit con una chiave a 56 bit. AES usa blocchi di 128 bit e può funzionare con chiavi di 128, 192 o 256 bit. Con una chiave di lunghezza  $n$ , ci sono  $2^n$  possibili chiavi.

## Cifrari a blocchi concatenati

Due o più blocchi di testo in chiaro possono essere identici e, per questi blocchi identici, un cifrario a blocchi produrrebbe, ovviamente, lo stesso testo cifrato. Un attaccante, conoscendo forse il protocollo di rete usato tra Alice e Bob, potrebbe, potenzialmente, indovinare il testo in chiaro, quando vede blocchi di testo cifrato uguali. L'attaccante potrebbe anche essere in grado di decifrare l'intero messaggio, identificando i blocchi di testo cifrato identici e usando le conoscenze sulla struttura dei protocolli sottostanti.

Per risolvere questo problema i cifrari a blocchi usano una tecnica chiamata **cifrari a blocchi concatenati** (*cipher block chaining*, CBC). Per spiegare come funziona, siano:

- $m(i)$  l' $i$ -esimo blocco di testo in chiaro
- $c(i)$  l' $i$ -esimo blocco di testo cifrato
- $a \oplus b$  lo XOR di due stringhe di bit  $a$  e  $b$ ,
- $K_s$  l'algoritmo di cifratura a blocchi con chiave  $S$ .

Il mittente genera una stringa di  $k$  bit,  $r(i)$  per l' $i$ -esimo blocco e calcola  $c(i) = K_s(m(i) \oplus r(i))$ . Il mittente invia quindi  $c(1), r(1), c(2), r(2)$ , e così via. Il ricevente, avendo ricevuto  $c(i)$  e  $r(i)$ , può ricostruire ogni blocco calcolando  $m(i) = K_s(c(i) \oplus r(i))$ . È importante notare che Trudy, sebbene possa spiare  $r(i)$  perché inviato in chiaro, non può ottenere  $m(i)$ , perché non conosce la chiave  $K_s$ . Si noti inoltre che anche se due blocchi di testo  $m(i)$  e  $m(j)$  fossero uguali, i corrispondenti blocchi cifrati sarebbero differenti, in quanto con alta probabilità sarebbero diversi i numeri casuali  $r(i)$  e  $r(j)$ .

L'introduzione di una componente casuale risolve un problema, ma ne introduce un altro. Infatti, Alice deve inviare il doppio dei bit, uno cifrato e il corrispondente casuale, raddoppiando la richiesta di banda. Per risolvere questo problema, i cifrari a blocchi usano la già citata tecnica chiamata **cipher block chaining** (CBC). L'idea di base è di inviare un solo numero casuale con il primo messaggio e quindi usare i blocchi calcolati come numero casuale successivo. CBC opera nel seguente modo.

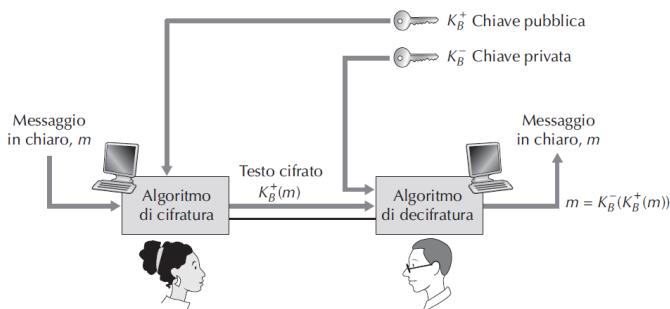
1. Prima di cifrare il messaggio o il flusso di dati, il mittente genera una stringa di  $k$  bit, chiamata **vettore di inizializzazione** (IV, *initialization vector*), che indicheremo con  $c(0)$ . Il mittente manda IV in chiaro al destinatario.
2. Per il primo blocco, il mittente calcola  $m(1) \oplus c(0)$ , cioè lo XOR tra il primo blocco di testo in chiaro e IV. Poi usa il risultato come input per l'algoritmo di cifratura a blocchi e ottiene il corrispondente blocco di testo cifrato, cioè  $c(1) = K_s(m(1) \oplus c(0))$ . Il mittente manda il blocco di testo cifrato  $c(1)$  al ricevente.
3. Per l' $i$ -esimo blocco, il mittente genera l' $i$ -esimo blocco di testo cifrato come  $c(i) = K_s(m(i) \oplus c(i - 1))$ .

Quando il ricevente riceve  $c(i)$ , lo decifra con  $K_s$  per ottenere  $s(i) = m(i) \oplus c(i - 1)$ , dato che il destinatario conosce anche  $c(i - 1)$ , otterrà il blocco di testo in chiaro  $m(i) = s(i) \oplus c(i - 1)$ . Anche se due blocchi di testo in chiaro sono uguali, i corrispondenti cifrati saranno, quasi sempre, diversi.

## Crittografia a chiave pubblica

Ma è possibile inviare messaggi cifrati senza essersi precedentemente accordati su una chiave segretamente condivisa? La soluzione a questo problema fu un algoritmo (ora conosciuto come **scambio di chiave di Diffie-Hellman**) che ha portato allo sviluppo degli attuali sistemi di crittografia a chiave pubblica, particolarmente utili anche per l'autenticazione e le firme digitali.

L'utilizzo della **crittografia a chiave pubblica** è concettualmente molto semplice. Supponiamo che Alice voglia comunicare con Bob. Come mostra la figura, quest'ultimo (il destinatario dei messaggi) non possiede un'unica chiave segreta (come nel caso dei sistemi a chiave simmetrica), ma due: una **pubblica**  $K_B^+$  (disponibile a chiunque, compresa Trudy) e una **privata**  $K_B^-$  (che soltanto lui conosce).



Per comunicare con Bob, Alice prima di tutto si procura la chiave pubblica di Bob e codifica quindi il suo testo in chiaro ( $m$ ), utilizzando la chiave di pubblica di Bob e un dato (per esempio standardizzato) algoritmo di cifratura, e genera un messaggio criptato che indicheremo con  $K_B^+(m)$ . Quando Bob riceve il messaggio cifrato, utilizza la sua chiave privata e un algoritmo per decodificarlo. In altri termini, Bob calcola  $K_B^-(K_B^+(m))$ .

Alice può utilizzare la chiave pubblica di Bob per inviargli un messaggio segreto senza che nessuno di loro debba distribuire chiavi segrete. Invertendo la cifratura con chiave pubblica e quella con chiave privata si ottiene lo stesso risultato, cioè  $K_B^-(K_B^+(m)) = K_B^+(K_B^-(m)) = m$ .

Vengono subito in mente **due obiezioni**. La prima è che l'intruso che intercetta il messaggio potrà venire a conoscenza sia della chiave pubblica del destinatario sia dell'algoritmo che il mittente ha usato per la codifica. La seconda è che, dato che la chiave di Bob è pubblica, chiunque potrebbe inviargli un messaggio cifrato, magari fingendo di essere Alice. Dunque, per associare un trasmittente a un messaggio è necessaria la **firma digitale**.

## RSA

Sebbene esistano numerosi algoritmi che soddisfano i requisiti esposti, l'**algoritmo RSA** (acronimo derivato dal nome dei suoi autori) è diventato praticamente sinonimo di crittografia a chiave pubblica.

RSA fa largo uso delle operazioni in modulo  $n$ . Ricordiamo che le operazioni di addizione e moltiplicazione sono facilitate dalle seguenti formule:

$$[(a \text{ mod } n) \pm (\cdot)(b \text{ mod } n)] \text{ mod } n = (a \pm (\cdot)b)$$

Un'altra identità molto utile che può essere derivata dalla terza delle uguaglianze precedenti è  $(a \text{ mod } n)^b \text{ mod } n = a^b \text{ mod } n$ .

*Cifrare un messaggio con RSA è equivalente a cifrare un unico numero intero che lo rappresenta.*

Due sono i punti principali su cui si basa RSA: (1) la scelta della chiave pubblica e di quella privata, (2) gli algoritmi di cifratura e decifratura.

Per generare la chiave pubblica e quella privata, Bob deve seguire i seguenti passi.

1. Scegliere **due numeri primi  $p$  e  $q$** : tanto più grandi sarà il loro valore tanto più difficile risulterà violare RSA anche se, ovviamente, cifratura e decifratura richiederanno più tempo. Si raccomanda che il prodotto di  $p$  e  $q$  sia dell'ordine di 1024 bit.
2. Calcolare  $n = pq$  e  $z = (p - 1)(q - 1)$ .
3. Scegliere un numero  **$e$  (*encryption*)** minore di  $n$ , diverso da 1 e relativamente primo a  $z$  (ovvero che non abbia divisori in comune con  $z$ ).
4. Trovare un numero  **$d$  (*decryption*)** tale che  $ed \text{ mod } z = 1$ .
5. La chiave pubblica di Bob,  $K_B^+$ , è la sequenza dei bit concatenati della **coppia  $(n, e)$** ; quella privata,  $K_B^-$  è la **coppia  $(n, d)$** .

Nel nostro esempio, la cifratura di Alice e la decifratura di Bob sono eseguite come segue.

- Supponiamo che Alice voglia inviare a Bob una stringa di bit, o un numero  $m < n$ . **Per la codifica**, Alice calcola  $m^e$  e poi il resto intero di  $m^e/n$ . Allora, il **messaggio cifrato  $c$**  inviato risulta  $c = m^e \text{ mod } n$ . La sequenza di bit corrispondente al testo cifrato  $c$  viene inviata a Bob.
- **Per decifrare il messaggio ricevuto**, Bob calcola  $m = c^d \text{ mod } n$ , che richiede l'utilizzo della sua chiave segreta  $(n, d)$ .

### Chiavi di sessione

L'elevamento a potenza richiesto da RSA è un processo che impiega molto tempo. Per contro, DES è almeno 100 volte più veloce di RSA, e a livello hardware il divario è ancora maggiore, essendo 1000 o 10.000 volte più veloce.

Come risultato, nella prassi comune, RSA è sovente utilizzato congiuntamente alla crittografia a chiave simmetrica. Così, se Alice vuole inviare a Bob un gran numero di dati cifrati, per codificarli può innanzitutto scegliere la chiave che sarà usata per codificare i dati stessi, detta **chiave di sessione  $K_S$** .

Alice deve informare Bob della chiave di sessione, poiché è la chiave simmetrica condivisa che sarà usata nel cifrario a chiave simmetrica. Quindi, ne codifica il valore con la chiave pubblica RSA di Bob: cioè, calcola  $c = (K_S)^e \text{ mod } n$ . Quando questi riceve la chiave di sessione cifrata con RSA,  $c$ , la decifra e ottiene la chiave di sessione  $K_S$ , utilizzata da Alice per il trasferimento di dati cifrati.

## Integrità dei messaggi e firma digitale

Rivolgiamo ora l'attenzione all'altro tema egualmente importante che riguarda l'uso della crittografia per garantire l'**integrità dei messaggi**, tema noto anche come **autenticazione dei messaggi**.

Supponiamo che Bob riceva un messaggio, che può essere cifrato o in chiaro, e che creda che sia stato inviato da Alice. Per autenticare questo messaggio, Bob deve verificare:

1. che il messaggio sia stato effettivamente originato da Alice;
2. che non sia stato alterato lungo il percorso verso Bob.

## Funzioni hash crittografiche

Una **funzione hash** prende un ingresso,  $m$ , e calcola una stringa di lunghezza fissata, detta hash. Una **funzione hash crittografica** deve però soddisfare un'ulteriore proprietà: deve essere computazionalmente impossibile trovare due messaggi  $x$  e  $y$  diversi, tali che  $H(x) = H(y)$ . In altre parole, data la coppia messaggio-hash  $(m, H(m))$ , creata dal trasmittente, un intruso non può falsificare il contenuto di un altro messaggio,  $y$ , che abbia lo stesso valore hash dell'originale.

Un algoritmo molto utilizzato per l'hash dei messaggi è **MD5**, in grado di calcolare una hash di 128 bit con un processo a quattro fasi. Si inizia con la normalizzazione attraverso l'aggiunta del valore 1 seguito da una serie di 0, in un numero tale da soddisfare determinate condizioni. Le successive fasi prevedono: l'aggiunta in coda di una rappresentazione a 64 bit della lunghezza del messaggio originale, l'inizializzazione di un accumulatore e, per ultimo, un passaggio in cui i blocchi composti da 16 gruppi di 32 bit (word) del messaggio vengono "triturati" attraverso un processo che prevede quattro cicli di elaborazione.

Un altro importante algoritmo hash, attualmente in uso, è **hash sicuro** (SHA-1, *secure hash algorithm*), che produce una sintesi del messaggio di 160 bit. La maggior lunghezza del risultato rende SHA-1 più sicuro.

## Codice di autenticazione dei messaggi

Ora che abbiamo compreso le funzioni hash, facciamo un primo tentativo di come dovremmo realizzare l'integrità di un messaggio.

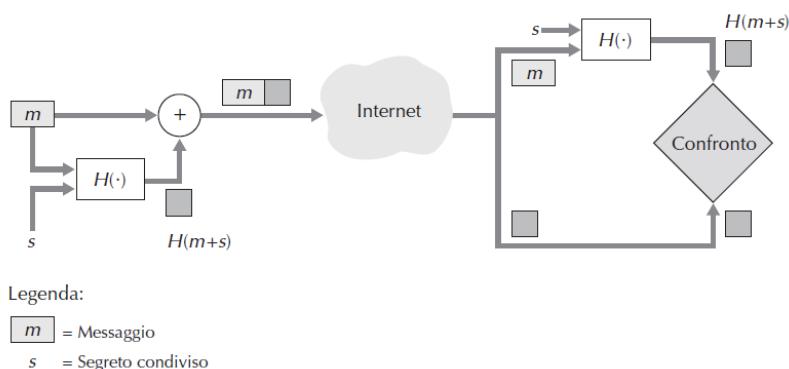
1. Alice crea un messaggio  $m$  e calcola la stringa hash  $h = H(m)$ , per esempio con SHA-1.
2. Alice aggiunge  $h$  al messaggio  $m$ , creando il messaggio esteso  $(m, h)$  e lo manda a Bob.
3. Bob riceve il messaggio esteso  $(m, h)$  e calcola  $H(m)$ . Se  $H(m) = h$ , Bob conclude che va tutto bene.

Questo approccio presenta ovviamente un difetto: Trudy può creare un messaggio falso,  $m'$ , nel quale dice di essere Alice, calcola e manda a Bob  $(m', H(m'))$ . Quando Bob riceve il messaggio, tutto va a buon fine nel passo 3 e Bob non sospetta alcuna attività insolita.

Per garantire l'integrità dei messaggi, Alice e Bob hanno bisogno di un segreto condiviso. Questo segreto condiviso, che non è altro che una stringa di bit, è chiamato **chiave di autenticazione**.

1. Alice crea un messaggio  $m$ , concatena  $s$  con  $m$  per creare  $m + s$ , calcola la stringa hash  $H(m + s)$ , per esempio con SHA-1.  $H(m + s)$  è chiamato **codice di autenticazione del messaggio (MAC, message authentication code)**.
2. Alice aggiunge il *MAC* al messaggio  $m$ , creando il messaggio esteso  $(m, H(m + s))$  e lo manda a Bob.
3. Bob riceve il messaggio esteso  $(m, h)$  e, avendo ricevuto  $m$  e conoscendo  $s$ , calcola il MAC  $H(m + s)$ . Se  $H(m + s) = h$ , Bob conclude che va tutto bene.

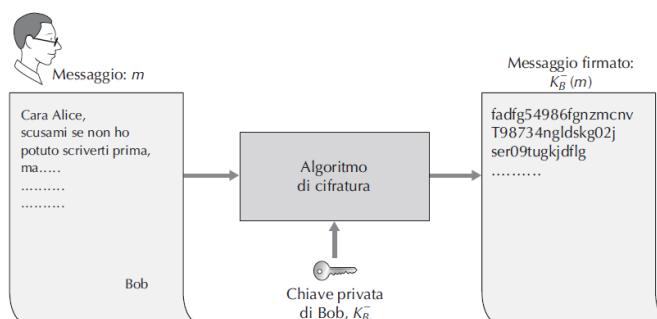
Un riassunto della procedura è illustrato nella figura seguente.



### Firme digitali

Nel mondo digitale, per indicare il titolare o il creatore di un documento, o dichiarare di essere d'accordo con il suo contenuto, si ricorre alla **firma digitale**, una tecnica di crittografia che consente di raggiungere svariati obiettivi.

Supponiamo che Bob voglia firmare (e inviare) digitalmente un documento  $m$ . Come illustrato nella figura, per firmare il documento utilizza la sua chiave privata,  $K_B^-$ , per calcolare  $K_B^-(m)$ .



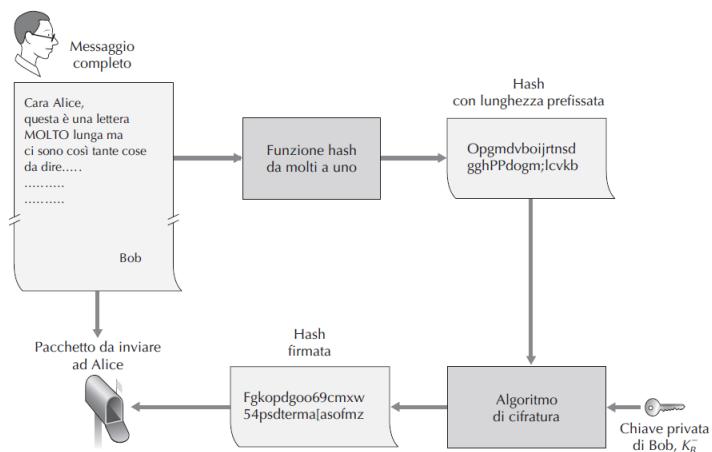
Supponiamo che Alice ottenga  $m$  e  $K_B^-(m)$  e voglia convincere un giudice che il documento è stato effettivamente firmato da Bob e che lui era la sola persona che poteva farlo. Alice applica la chiave pubblica di Bob e calcola  $K_B^+(K_B^-(m))$  e riproduce  $m$ , che corrisponde

esattamente al documento originale. Il giudice deduce che solo Bob può aver apposto la firma, per i seguenti motivi.

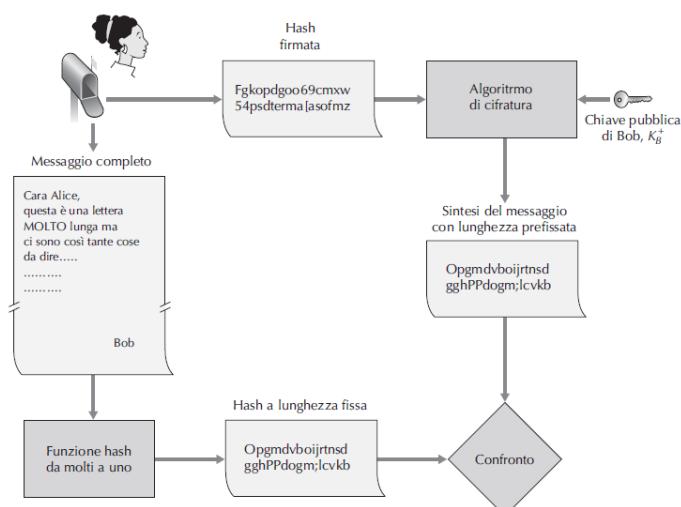
- Chiunque abbia firmato il messaggio deve aver utilizzato la chiave di cifratura privata,  $K_B^-$ , nel calcolo della firma  $K_B^-(m)$ , in modo che  $K_B^+(K_B^-(m)) = m$ .
- Presupponendo che Bob non abbia dato a nessuno la propria chiave e che questa non sia stata rubata, allora solo lui poteva conoscere  $K_B^-$ .

È anche importante notare che la firma creata da Bob per  $m$  non sarà valida per nessun altro messaggio, in quanto se  $m'$  è un qualunque altro messaggio,  $K_B^+(K_B^-(m'))$  non sarà uguale a  $m'$ . Di conseguenza, le tecniche di crittografia a chiave pubblica forniscono anche l'integrità del messaggio, consentendo al ricevente di verificare che il messaggio era inalterato come alla sorgente.

La figura a seguire fornisce un riassunto della **procedura per creare la firma digitale**.



**Il processo di verifica dell'integrità del messaggio** vede Alice per prima cosa applicare la chiave pubblica del trasmittente al messaggio per ottenerne una hash e poi impiegare la funzione hash al messaggio con il testo in chiaro, in modo da procurarsi la seconda hash. Se i due risultati coincidono, Alice può essere tranquilla dell'integrità del messaggio e sull'identità del suo autore.



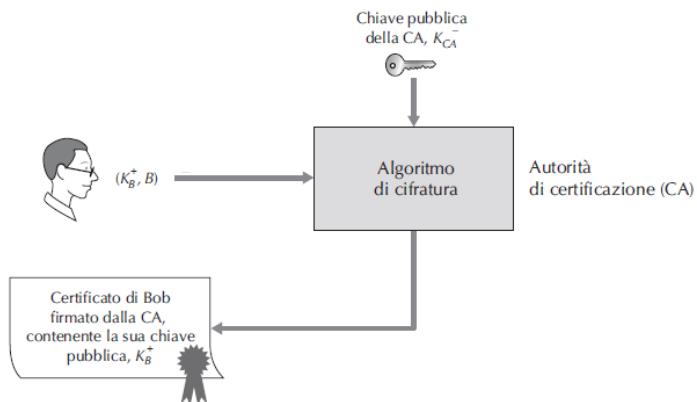
**Confrontiamo** brevemente **firma digitale e MAC**, in quanto presentano alcune corrispondenze, ma anche delle differenze sottili e importanti. Entrambi partono da un messaggio o da un documento. Per creare un MAC dal messaggio, gli aggiungiamo la chiave di autenticazione e prendiamo l'hash del risultato. Si noti che non sono coinvolte né la crittografia a chiave simmetrica né quella a chiave pubblica. Per creare la firma digitale, prima prendiamo l'hash del messaggio e poi la cifriamo con la nostra chiave privata (usando la crittografia a chiave pubblica). Una firma digitale, quindi, è una tecnica più gravosa, in quanto richiede l'infrastruttura di chiave pubblica (**PKI**, *public key infrastructure*) sottostante, con le relative autorità di certificazione.

### Certificazione della chiave pubblica

Un'importante applicazione della firma digitale è la **certificazione della chiave pubblica**, cioè la certificazione che una chiave pubblica appartenga a una specifica entità.

Generalmente, la relazione tra una data chiave pubblica e una determinata entità è stabilita da un'**autorità di certificazione (CA)**, *Certification Authority*, il cui compito è di validare l'identità ed emettere certificati e ha i seguenti ruoli.

1. Verifica che un'entità sia veramente chi afferma di essere.
2. Una volta verificata l'identità, la CA rilascia un **certificato** che autentica la corrispondenza fra chiave pubblica ed entità. Il certificato contiene la chiave pubblica e le informazioni di identificazione globali e uniche del suo proprietario ed è firmato digitalmente dalla CA.



### Autenticazione di un punto terminale

Con **autenticazione di un punto terminale** (o *end-point authentication*) si intende il processo attraverso il quale una entità prova la sua identità a un'altra entità su una rete di calcolatori.

Nell'autenticazione in rete, le parti comunicanti non possono scambiarsi informazioni biometriche, come l'aspetto del viso o il timbro della voce, in quanto le parti possono essere apparecchiature quali router o processi client e server. Quindi, l'autenticazione deve essere basata unicamente sullo scambio di messaggi o di dati come parte di un **protocollo di autenticazione**.

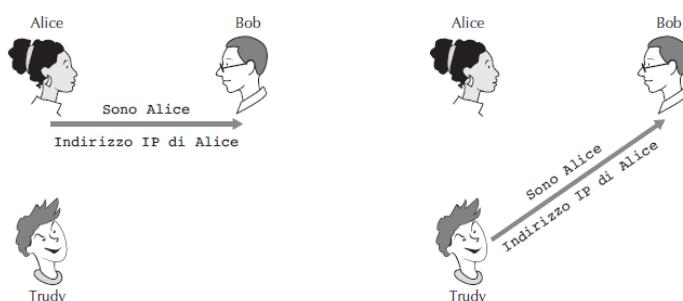
Analizzeremo varie versioni di un protocollo di autenticazione, che chiameremo **ap** (*authentication protocol*), elencando pregi e difetti di ciascuna. Cominciamo supponendo che Alice debba autenticarsi a Bob.

### Protocollo di autenticazione *ap1.0*

Alice invia un messaggio a Bob dicendo semplicemente di essere Alice.

Ovviamente, il destinatario non può assolutamente essere sicuro che la persona che ha inviato il messaggio “Sono Alice” sia davvero Alice, perché anche Trudy potrebbe averlo fatto.

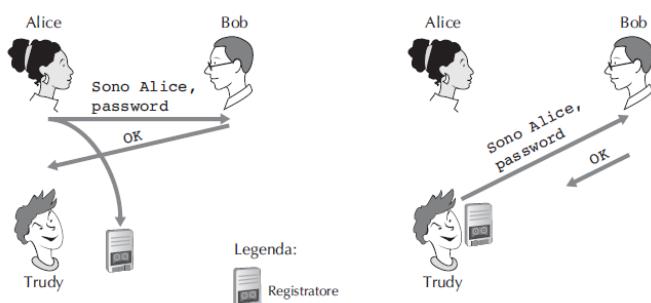
### Protocollo di autenticazione *ap2.0*



Se Alice avesse un indirizzo di rete conosciuto (per esempio, l’indirizzo IP) che utilizza abitualmente, Bob potrebbe autenticarla verificando la corrispondenza fra l’indirizzo sorgente sul datagramma IP che trasporta il messaggio e quello di Alice.

Sappiamo che non è particolarmente difficile creare un datagramma IP contenente un indirizzo sorgente artificiale e poi inviarlo attraverso il protocollo a livello di collegamento al primo router. Da quel momento, il datagramma con il falso indirizzo sorgente sarà coscientemente instradato verso il destinatario. Questo approccio è una forma di **spoofing** di IP.

### Protocollo di autenticazione *ap3.0*



Alice invia la sua password a Bob.

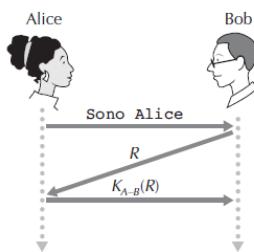
Se Trudy intercetta la comunicazione di Alice, può scoprirne la password. Per quanto improbabile possa sembrare, va ricordato che quando si è in collegamento Telnet con un’altra macchina e ci si registra, la password viene di login è inviata al server in chiaro. Chiunque sia collegato alle LAN di client o server può intercettare (leggere e archiviare) tutti i pacchetti trasmessi sulla LAN e rubare la password di login.

### Protocollo di autenticazione *ap3.1*

Un modo per superare tutti i limiti di *ap3.0* consiste nel cifrare la password in modo da impedire all'intruso di appropriarsene. Se Alice condivide una chiave simmetrica segreta  $K_{A-B}$  con Bob, potrebbe utilizzarla per cifrare il suo messaggio di identificazione e la password. Bob provvederebbe quindi a decodificarla e a controllarne la veridicità. Se il responso è positivo, il destinatario si sente sicuro in quanto il mittente, oltre alla password, conosce anche la chiave condivisa con cui è stata cifrata.

Bob è ancora soggetto al cosiddetto **attacco di replica** (*playback attack*). Trudy può infatti inserirsi nella comunicazione, registrare la versione cifrata della chiave e successivamente riprodurla per inviarla a Bob sostenendo di essere Alice.

### Protocollo di autenticazione *ap4.0*



Un **nonce** (*number used once*) è un numero che il protocollo userà soltanto una volta nel seguente modo.

1. Alice invia il messaggio, “Sono Alice”, a Bob.
2. Bob sceglie un *nonce*,  $R$ , e lo trasmette ad Alice.
3. Alice utilizza  $K_{A-B}$  per codificare il *nonce*, e gli re-invia il valore risultante,  $K_{A-B}(R)$ .
4. Bob decifra il messaggio ricevuto: se il *nonce* è quello da lui inviato, Alice è autenticata.

Attraverso l'utilizzo del *nonce*,  $R$ , e con il controllo del valore di ritorno  $K_{A-B}(R)$ , Bob può essere sicuro che si tratta veramente di Alice (perché conosce la chiave segreta necessaria per cifrare  $R$ ) e che si trova in quel momento all'altro capo del collegamento (in quanto ha codificato il *nonce*,  $R$ , che Bob ha appena creato).

È stato impiegato un *nonce* e la crittografia a chiave simmetrica. In *ap5.0* si utilizza la crittografia a chiave pubblica, ovviando così al problema del primo scambio della chiave segreta condivisa.

## E-mail sicure

Immaginiamo che Alice voglia inviare una e-mail a Bob e che, ancora una volta, Trudy voglia intromettersi. Iniziamo col definire le caratteristiche di sicurezza cui deve rispondere il nostro progetto. Prima, e più importante, è la **riservatezza**: ovviamente né Alice né Bob vogliono che Trudy legga le loro e-mail. La seconda è l'**autenticazione del mittente**: se Bob riceve il messaggio “Non ti amo più. Non voglio rivederti. La tua ex, Alice”, vorrebbe, naturalmente, essere sicuro che il messaggio provenga da Alice e non da Trudy. Un’ulteriore caratteristica è l'**integrità**, cioè la certezza che i messaggi non siano modificati durante il trasporto. Infine, il sistema di posta elettronica dovrebbe fornire l'**autenticazione del ricevente**, vale a dire: Alice vuole essere sicura di aver inviato la lettera proprio a Bob e non a qualcun altro che lo sta impersonando (per esempio Trudy).

Per ottenere la riservatezza si utilizza la crittografia a chiave pubblica, usando per esempio RSA. Purtroppo, è una metodologia relativamente inefficiente, in modo particolare per messaggi molto lunghi.

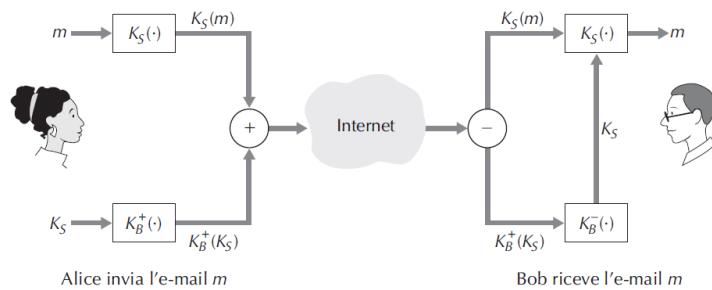
Per risolvere il problema dell’efficienza, utilizziamo una chiave di sessione. In particolare:

1. Alice sceglie arbitrariamente una chiave simmetrica,  $K_S$ ;
2. Cifra il suo messaggio,  $m$ , con  $K_S$ ;
3. Codifica la chiave simmetrica con la chiave pubblica di Bob,  $K_B^+$ ;
4. Concatena messaggio cifrato e chiave simmetrica cifrata per formare un “pacco”;
5. Invia il “pacco” all’indirizzo e-mail di Bob.

Quando Bob riceve il “pacco”:

1. Utilizza la sua chiave privata  $K_B^-$  per ottenere la chiave simmetrica  $K_S$ ;
2. Utilizza la chiave simmetrica  $K_S$  per decodificare il messaggio  $m$ .

Nelle figure, il “+” e il “-“ cerchiati rappresentano rispettivamente la concatenazione e la separazione di informazioni.

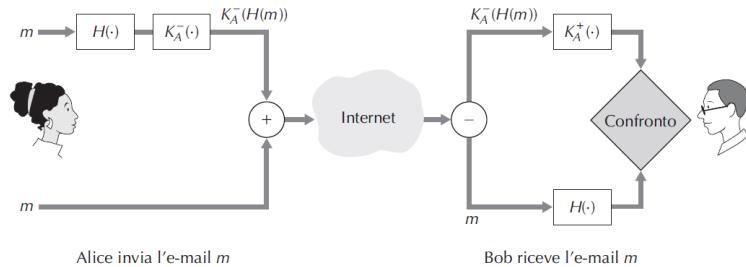


Supponiamo, ora, che la principale preoccupazione di Alice e Bob siano l'autenticazione del mittente e l'integrità dei messaggi. Vengono utilizzate le firme digitali e una funzione di hash. Nello specifico:

1. Alice applica una funzione hash  $H$  (come MD5) al suo messaggio,  $m$ ;
2. Cifra il risultato con la sua chiave privata,  $K_A^-$ , per creare la firma digitale;
3. Concatena messaggio in chiaro e firma;
4. Invia la concatenazione, come un unico “pacco”, all’indirizzo e-mail di Bob.

Quando Bob lo riceve:

1. Applica la chiave pubblica di Alice,  $K_A^+$ , all'hash del messaggio che è stato firmato;
2. Confronta il risultato con quanto ottenuto con la sua funzione di hash  $H$ . Se coincidono, può essere sufficientemente sicuro che il messaggio provenga da Alice e che non sia stato alterato durante il trasferimento.



In conclusione, lo schema riportato in seguito fornisce un grado di sicurezza soddisfacente per la maggior parte degli utilizzatori di e-mail. Ma rimane ancora da valutare un argomento importante: occorre che Alice ottenga la chiave pubblica di Bob e questi quella di Alice. Ancora una volta si pone in modo centrale il problema della distribuzione. Per esempio, Trudy potrebbe spacciarsi per Bob e dare ad Alice la propria chiave pubblica, riuscendo così a carpire i messaggi inviati a Bob. Un modo sicuro, molto diffuso nella pratica, per affrontare il problema consiste nella certificazione della chiave pubblica per mezzo di una CA.

## PGP

**Pretty Good Privacy (PGP)** è uno schema di cifratura per la posta elettronica diventato uno standard *de facto*. PGP crea una coppia di chiavi: quella pubblica, che può essere collocata sul sito web dell'utente o su un server di chiavi pubbliche, e quella privata, protetta da una password che deve essere inserita ogni volta che viene utilizzata la chiave. Inoltre, il software offre all'utente l'opzione di poter firmare digitalmente il messaggio, di codificarlo o di effettuare entrambe le operazioni.

PGP fornisce anche un meccanismo di certificazione della chiave pubblica, diverso da quello più convenzionale fornito da una CA. Le chiavi pubbliche PGP sono certificate attraverso una **rete di fiducia**. Alice stessa può certificare qualsiasi coppia chiave/nome utente quando non ha dubbi sui suoi componenti. Inoltre, PGP permette ad Alice di affermare che si fida di un altro utente per attestare l'autenticità di ulteriori chiavi. Alcuni utenti PGP raccolgono e si scambiano le chiavi pubbliche che certificano vicendevolmente firmandole con le proprie chiavi private.

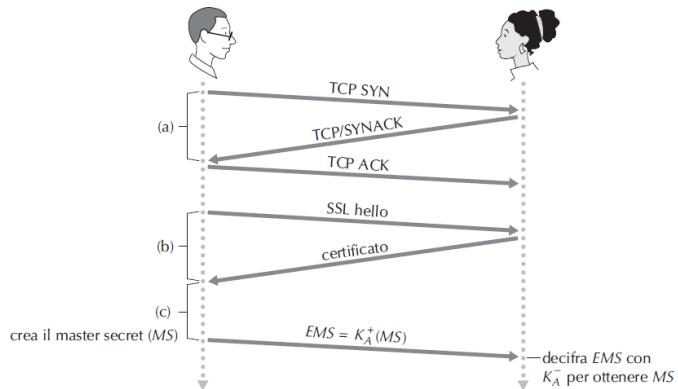
## Rendere sicure le connessioni TCP: TLS

Esamineremo come la crittografia può essere usata per arricchire TCP con servizi di sicurezza, comprese la riservatezza, l'integrità dei dati e l'autenticazione end-point. Questa versione arricchita di TCP è comunemente nota come **secure sockets layer** (SSL). Una versione leggermente modificata di SSLv3 è invece chiamata **transport layer security** (TLS).

TLS fornisce una semplice API (*application programmer interface*) verso le socket, analoga a quella fornita da TCP: quando un'applicazione vuole usare TLS, include le classi/librerie TLS che forniscono l'interfaccia socket TLS allo sviluppatore dell'applicazione.

Iniziamo col descrivere una versione semplificata di TLS, che ci consenta di comprenderne in modo generale il funzionamento. Distinguiamo quattro fasi.

- **Handshake.** Durante questa fase, Bob ha bisogno di (a) stabilire una connessione TCP con Alice, (b) verificare che Alice sia realmente Alice e (c) inviarle una chiave segreta principale che verrà utilizzata da entrambi per generare tutte le chiavi simmetriche di cui hanno bisogno per la sessione TLS.



Bob genera un **master secret (MS)**, ossia un valore segreto che verrà usato solo per questa sessione TLS e dal quale verranno derivate altre chiavi. Il master secret viene cifrato con la chiave pubblica di Alice, per creare un **encrypted master secret (EMS)** che viene inviato ad Alice, la quale decifra l'EMS con la sua chiave privata e ottiene MS. Dopo questa fase Bob ha autenticato Alice, ed entrambi (ma nessun'altro) conoscono il master secret per questa sessione TLS.

- **Derivazione delle chiavi.** Alice e Bob usano MS per generare 4 chiavi
  - $E_B$ , chiave di cifratura di sessione per i dati inviati da Bob ad Alice
  - $M_B$ , chiave MAC di sessione per i dati inviati da Bob ad Alice
  - $E_A$ , chiave di cifratura di sessione per i dati inviati da Alice a Bob
  - $M_A$ , chiave MAC di sessione per i dati inviati da Alice a Bob

Le due chiavi di cifratura verranno usate per cifrare i dati, e le due chiavi MAC verranno usate per verificarne l'integrità.

- **Trasferimento dati.** TLS suddivide il flusso di dati in record a cui aggiunge un MAC per la verifica dell'integrità e che poi cifra. Per creare il MAC, Bob passa i dati del record assieme alla chiave  $M_B$  come a una funzione hash. Per cifrare il pacchetto composto da record e MAC, Bob usa la sua chiave di cifratura di sessione  $E_B$ . Il pacchetto cifrato viene poi passato a TCP per essere trasportato in Internet.
- **Chiusura della connessione.** Bob invia un segmento TCP FIN ad Alice. Ma un modello così semplice prepara il campo a un attacco di tipo *truncation*, dove Trudy si pone nel mezzo di una sessione TLS e la termina prematuramente con un messaggio TCP FIN. Se Trudy facesse questo, Alice penserebbe di aver ricevuto tutti i dati da Bob, quando effettivamente ne ha ricevuto solo una parte. La soluzione a questo problema è indicare nel campo tipo se il record serve a terminare la sessione TLS.

Sebbene questo approccio contribuisca a fornire l'integrità dei dati per l'intero flusso del messaggio, non è ancora a prova di bomba. In particolare, supponiamo che Trudy stia usando un attacco di tipo “man-in-the-middle” e abbia la capacità di inserire, cancellare e sostituire segmenti nel flusso di segmenti TCP inviati da Alice a Bob.

La soluzione a questo problema consiste nell'usare dei numeri di sequenza oltre che quelli di TCP. TLS si comporta come segue: Bob mantiene un contatore di numeri di sequenza che inizia da zero e viene incrementato per ciascun record che TLS invia. Bob non include effettivamente il numero di sequenza nel record vero e proprio, ma lo include nel MAC, quando ne effettua il calcolo. Quindi, il MAC è ora un hash di dati, cui si aggiunge la chiave MAC  $M_B$  e il numero di sequenza corretto. Alice conserva traccia dei numeri di sequenza di Bob, per verificare l'integrità dei dati di un record, includendo nel calcolo del MAC il numero di sequenza appropriato.

## Sicurezza a livello di rete: IPsec e reti private virtuali

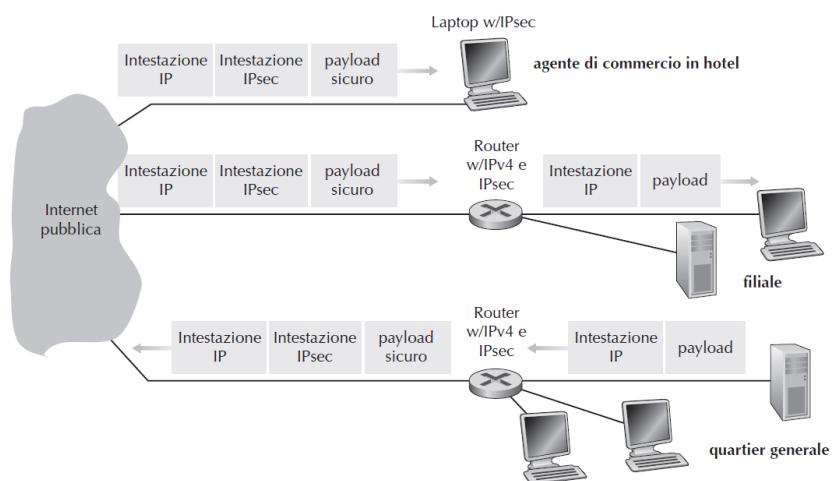
**IPsec** è il protocollo di sicurezza di IP che fornisce **sicurezza a livello di rete**. Molti istituti usano IPsec per creare **reti private virtuali** (VPN, *virtual private network*) che fanno uso della Internet pubblica.

Per ottenere riservatezza tra due entità a livello di rete, l'entità che invia cifra il campo dati di tutti i pacchetti in partenza verso l'entità che riceve. Se fosse disponibile un simile servizio, allora tutti i dati inviati da una entità all'altra sarebbero occultati a eventuali terze parti che stanno intercettando traffico di rete.

Oltre alla riservatezza, un protocollo di sicurezza a livello di rete potrebbe potenzialmente fornire altri livelli di sicurezza, quali autenticazione della sorgente, integrità dei dati e prevenzione degli attacchi di replay.

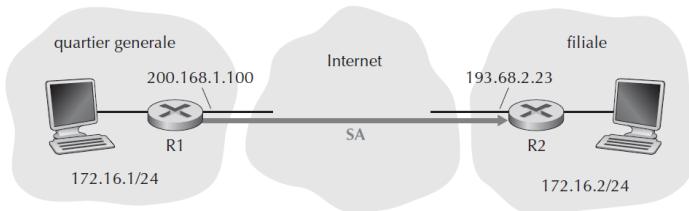
Un istituto con sedi in più aree geografiche distinte desidera spesso una propria rete IP (**rete privata**), in modo che i suoi host e server possano comunicare in modo sicuro e riservato.

Oggiorno molti istituti, invece di installare e mantenere una rete privata, creano una VPN sulla Internet pubblica, nel senso che il traffico tra uffici viene inviato su questa piuttosto che su una rete fisica indipendente. Per fornire riservatezza, il traffico tra uffici viene cifrato prima di essere inviato.



Due sono i principali protocolli della suite IPsec: il protocollo di **authentication header** (AH, intestazione per l'autenticazione) e l'**encapsulation security payload** (ESP, incapsulamento sicuro del payload). Un'entità sorgente IPsec, quando invia datagrammi sicuri a un'entità di destinazione usa il protocollo AH o il protocollo ESP. Il protocollo AH fornisce solo l'autenticazione della sorgente e l'integrità dei dati, mentre ESP fornisce anche la riservatezza.

Prima di procedere all'invio di datagrammi sicuri, l'host sorgente e quello di destinazione creano un canale logico (unidirezionale) a livello di rete, chiamato **associazione di sicurezza** (SA, *Security Association*). Per guardare com'è fatta all'interno una SA riferiamoci a quella tra il router R1 e il router R2 della figura a seguire.



Il router R1 mantiene informazioni di stato sulla SA, tra cui:

- un identificatore a 32 bit della SA, chiamato **indice dei parametri di sicurezza (SPI, Security Parameter Index)**
- l’interfaccia di origine della SA (in questo caso **200.168.1.100**) e l’interfaccia di destinazione della SA (in questo caso **193.68.2.23**)
- il tipo di codifica utilizzato (e.g. 3DES con CBC)
- la chiave di codifica
- il tipo di controllo di integrità (e.g. HMAC con MD5)
- la chiave di autenticazione.

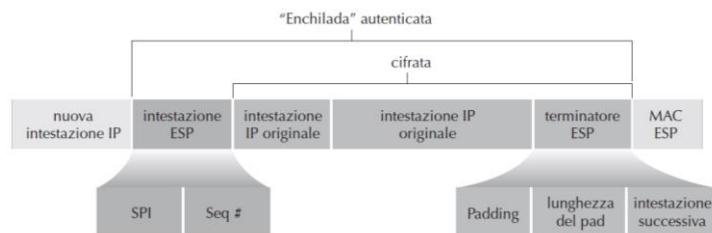
Il router R1, ogni volta che costruisce un datagramma IPsec da inoltrare su questa SA, accede a queste informazioni di stato per determinare come autenticare e cifrare il datagramma. Analogamente, il router R2 mantiene le stesse informazioni di stato per questa SA per autenticare e decifrare tutti i datagrammi IPsec in arrivo da questa SA.

Le entità IPsec memorizzano le informazioni di stato di tutte le loro SA nel loro **database di associazione di sicurezza (SAD, Security Association Database)**: una struttura dati che risiede nel kernel del sistema operativo.

### Il datagramma IPsec

IPsec prevede due forme diverse di pacchetto: quella chiamata **modalità tunnel** e quella chiamata **modalità trasporto**, di cui la prima, essendo più appropriata per le VPN, è più utilizzata.

Il **formato del datagramma IPsec** è mostrato di seguito.



Il router R1, per convertire il datagramma IPv4 in un datagramma IPsec, usa la seguente ricetta:

- appende in fondo al datagramma IPv4 (che comprende i campi di intestazione originari) un campo “coda ESP” cifra il risultato usando l’algoritmo e la chiave specificati dalla SA

- appende all'inizio dei dati cifrati un campo chiamato “intestazione ESP” (il pacchetto risultante è chiamato **enchilada**), crea un'autenticazione MAC sull'enchilada usando l'algoritmo e la chiave specificati nella SA
- appende il MAC alla fine dell'enchilada e ottiene il payload
- infine, crea una nuova intestazione IP con tutti i campi di intestazione classici di IPv4, lunghi normalmente 20 byte, e la appende prima del payload calcolato al punto precedente.

Il datagramma IPsec, dopo essere stato inviato da R1 sulla Internet pubblica, passerà attraverso molti router prima di giungere a R2. Tutti questi router elaboreranno il datagramma come se fosse un datagramma ordinario, in quanto non sono consapevoli che esso trasporti dati cifrati con IPsec.

C'è un'altra importante sottigliezza da affrontare: *R1, quando riceve un datagramma non sicuro da un host nel quartier generale destinato a un indirizzo IP di destinazione esterno, come fa a sapere se debba essere convertito in un datagramma IPsec? E se deve essere elaborato da IPsec, come fa R1 a sapere quale, o quali, SA debbano essere usate per costruire il datagramma IPsec?* Il problema viene risolto facendo in modo che le entità IPsec, insieme a un SAD, mantenga un'altra struttura dati chiamata **database delle regole di sicurezza (SPD, Security Policy Database)**. L'SPD indica che tipi di datagrammi debbano essere elaborati con IPsec in funzione degli indirizzi IP di sorgente e destinazione e del tipo di protocollo; per quelli da elaborare, indica quale SA debba essere utilizzata. In un certo senso, le informazioni contenute nell'SPD indicano che cosa fare quando un datagramma arriva, quelle nel SAD indicano come farlo.

*Infine, quali servizi fornisce esattamente IPsec?* Esaminiamoli dal punto di vista di un attaccante (Trudy) che sta sul percorso tra R1 e R2 della figura vista in precedenza.

- Non può vedere il datagramma originale, così come il numero di protocollo, l'indirizzo IP sorgente e l'indirizzo IP di destinazione.
- Non sa che trasportano dati TCP, UDP o ICMP e nemmeno se trasportano dati http, SMTP o di altro tipo.
- Supponiamo che Trudy tenti di falsificare un datagramma nella SA invertendone alcuni bit. Quando il datagramma falsificato arriva a R2, il controllo di integrità tramite MAC fallirà, frustrando i tentativi di Trudy.

Supponiamo che Trudy tenti di mascherarsi da R1. Tale attacco sarà inutile, in quanto il datagramma arrivato a R2 fallirà il controllo di integrità. Infine, includendo IPsec i numeri di sequenza, Trudy non sarà in grado di sferrare con successo un attacco basato sulla ripetizione.

## IKE: gestione delle chiavi in IPsec

Nel **protocollo IKE** (*Internet Key Exchange*), ogni entità IPsec ha un certificato che include la chiave pubblica dell'entità. Il protocollo IKE ha due certificati, negozia gli algoritmi di autenticazione e cifratura e scambia in modo sicuro il materiale per creare chiavi di sessione nelle SA IPsec.

IKE impiega due fasi per eseguire questi compiti. La **prima fase** consiste nei seguenti due scambi di coppie di messaggi tra R1 e R2.

- Durante il primo scambio di messaggi, i due lati usano Diffie-Hellman per creare una **SA IKE** bidirezionale tra i router, la quale fornisce un canale di autenticazione e cifratura tra i router. Durante il primo scambio di messaggi, vengono stabilite le chiavi per la cifratura e autenticazione della SA IKE e un master secret che verrà utilizzato nella fase 2 per calcolare le chiavi SA IPsec.
- Durante il secondo scambio di messaggi, i due lati si rivelano le loro identità firmando i loro messaggi. Tuttavia, poiché i messaggi sono inviati su un canale SA IKE sicuro, le identità non vengono rivelate a una spia passiva. Inoltre, durante questa fase, i due lati negoziano gli algoritmi di cifratura e di autenticazione IPsec che devono essere utilizzati dalla SA IPsec.

Nella **seconda fase** di IKE i due lati creano una SA per ogni direzione. Alla fine della fase le chiavi di sessione per l'autenticazione e la cifratura sono stabilite da entrambe le parti per le due SA. Le due parti possono quindi utilizzare l'SA per inviare datagrammi sicuri.

## Firewall

Il **firewall** è una combinazione di hardware e software che separa una rete privata dal resto di Internet e consente all'amministratore di controllare e gestire il flusso di traffico tra il mondo esterno e le risorse interne, stabilendo quali pacchetti lasciare transitare e quali bloccare. Un firewall ha tre obiettivi.

- *Tutto il traffico dall'esterno verso l'interno e viceversa deve passare attraverso il firewall.*
- *Solo al traffico autorizzato, secondo la definizione nella politica di sicurezza locale, sarà consentito passare.*
- *Il firewall stesso deve essere immune dalla penetrazione.*

I firewall possono essere classificati in tre categorie: i tradizionali **filtrati di pacchetti** (*packet filter*), i **filtrati con memoria di stato** (*stateful filter*) e i **gateway a livello applicativo** (*application level gateway*).

Il **filtraggio dei pacchetti** prevede in primo luogo l'analisi dell'intestazione dei datagrammi e l'applicazione a questi delle regole di filtraggio stabilite dall'amministratore di rete per determinare quali devono essere bloccati e quali possono passare. Nella tabella a seguire sono elencate alcune politiche che un'organizzazione può stabilire e la loro realizzazione da parte di un filtro di pacchetti.

| Politica   | Configurazione del firewall   |
|--|---|
| Nessun accesso web all'esterno   | Bloccare tutti i pacchetti uscenti con qualsiasi indirizzo IP di destinazione e porta di destinazione 80  |
| Nessuna connessione TCP entrante, eccetto quelle dirette al solo server web pubblico | Bloccare tutti i pacchetti TCP SYN entranti verso qualsiasi indirizzo IP tranne quelli verso l'indirizzo IP 130.207.244.203 e con porta destinazione 80 |
| Evitare che le radio web consumino tutta la banda disponibile                        | Bloccare tutti i pacchetti UDP entranti, eccetto i pacchetti DNS  |
| Evitare che la rete possa essere usata per un attacco DoS                            | Bloccare tutti i pacchetti ICMP ping diretti a un indirizzo broadcast (per esempio 130.207.255.255)   |
| Evitare che la rete possa essere rilevata tramite Traceroute                         | Bloccare tutti i messaggi ICMP uscenti per TTL scaduto  |

Le **regole del firewall** sono implementate nei router tramite **liste di controllo degli accessi** presenti su ciascuna interfaccia del router. Un esempio di lista di controllo degli accessi per l'organizzazione 222.22/16 è mostrata nella tabella a seguire e viene usata sull'interfaccia che collega il router dell'organizzazione a un ISP esterno.

| Azione   | Indirizzo sorgente       | Indirizzo destinazione   | Protocollo | Porta sorgente | Porta destinazione | Bit di flag |
|----------|--------------------------|--------------------------|------------|----------------|--------------------|-------------|
| consenti | 222.22/16                | al di fuori di 222.22/16 | TCP        | >1023          | 80                 | qualsiasi   |
| consenti | al di fuori di 222.22/16 | 222.22/16                | TCP        | 80             | >1023              | ACK         |
| consenti | 222.22/16                | al di fuori di 222.22/16 | UDP        | >1023          | 53                 | -           |
| consenti | al di fuori di 222.22/16 | 222.22/16                | UDP        | 53             | >1023              | -           |
| blocca   | qualsiasi                | qualsiasi                | tutti      | qualsiasi      | qualsiasi          | tutti       |

La prima regola consente a qualsiasi pacchetto TCP con porta destinazione 80 di lasciare la rete dell'organizzazione, mentre la seconda regola consente a qualsiasi pacchetto TCP con porta sorgente 80 e bit ACK impostato a 1 di entrare nella rete. Le seconde due regole, insieme, consentono ai pacchetti DNS di entrare e uscire dalla rete.

In un **filtro di pacchetti tradizionale** le decisioni di filtraggio vengono prese su ciascun pacchetto in modo indipendente. I filtri con **memoria di stato**, in realtà, tengono traccia delle connessioni TCP in una tabella di connessione e usano questa conoscenza per prendere le decisioni di filtraggio. Questo è possibile perché il firewall può osservare l'inizio di una nuova connessione, osservando l'handshake a tre vie (SYN, SYNACK e ACK), e la fine della stessa, quando vede un pacchetto FIN per quella connessione. Il firewall può anche, in modo conservativo, assumere che la connessione è terminata quando non vede alcuna attività su di essa, per esempio per 60 secondi.

Le informazioni sull'identità degli utenti interni non sono comprese nelle intestazioni IP/TCP/UDP, ma si trovano nei dati a livello di applicazione. Per ottenere un più elevato livello di sicurezza, i firewall devono combinare il filtraggio dei pacchetti con un **gateway a livello applicativo**: un server specifico attraverso il quale tutti i dati delle applicazioni (in ingresso e in uscita) sono vincolati a passare. Un gateway a livello applicativo guarda oltre le intestazioni IP/TCP/UDP e prende decisioni sulle politiche in base ai dati applicativi. Un host può avere diversi gateway, ma ciascuno costituisce un server separato, con propri processi.

Questa tecnica non è però priva di svantaggi. Innanzitutto, occorre un gateway diverso per ciascuna applicazione e, inoltre, c'è un costo in termini di prestazioni in quanto tutti i dati devono passare attraverso il gateway.

### Sistemi di rilevamento delle intrusioni

Per rilevare molti tipi di attacchi, abbiamo bisogno di eseguire un **controllo approfondito del pacchetto**, cioè guardare, oltre ai campi dell'intestazione, anche gli effettivi dati applicativi che il pacchetto trasporta.

Un dispositivo che genera allarmi quando osserva traffico potenzialmente malevolo viene chiamato **sistema di rilevamento delle intrusioni (IDS, *intrusion detection system*)**, mentre un dispositivo che filtra il traffico sospetto viene chiamato **sistema di prevenzione delle intrusioni (IPS, *intrusion prevention system*)**.

Un'organizzazione può impiegare uno o più sistemi IDS nella propria rete istituzionale, come in figura. Quando vengono impiegati più punti di rilevamento (sensori), questi tipicamente lavorano in maniera coordinata, mandando informazioni sulle attività di traffico sospette a un processore IDS centrale, che raccoglie, integra le informazioni e manda segnalazioni agli amministratori di rete, quando lo ritiene opportuno.

