

Reti di Calcolatori a.a. 2019/2020

Francesco Iannelli

September 16, 2019

Indice

1	Introduzione	4
1.1	Modalità d'esame	4
1.2	Cenni di modelli stratificati	4
1.2.1	Livello applicativo	5
1.2.2	Livello di trasporto	5
1.2.3	Livello rete	5
1.2.4	Livello link	6
1.3	Introduzione alle reti	6
1.3.1	LAN	6
1.3.2	WAN	7
1.3.3	Internetwork	7
Reti a commutazione di circuito	8
Reti a commutazione di pacchetto	8
Circuiti virtuali	8
Datagram Network	9
Packet Switch e Circuit Switch a confronto	9
1.3.4	Internet	10
Servizi	11
IETF/RFC/ICANN	11
Rete di accesso	11
1.4	Metriche di riferimento	12
1.4.1	Ritardi	13
Ritardo di elaborazione del nodo	13
Ritardo di accodamento	13
Ritardo di trasmissione	13
Ritardo di propagazione	13
1.4.2	Esempi	14
2	Modelli Stratificati e Protocolli	15
2.1	OSI: Open Systems Interconnection	16
2.1.1	Modello ISO/OSI	16
2.2	Protocollo	17
2.2.1	Incapsulamento dell'informazione	18

3 Stack protocollare TCP/IP	19
3.1 Lo strato applicativo	20
3.1.1 Web	23
Sintassi della URI	24
3.1.2 HTTP	24
Sintassi della URL HTTP	25
Connessione HTTP	25
Metodo idempotente	27
Messaggi HTTP	27
Proxy	31
Cookies	31
3.1.3 TELNET	32
Network Virtual Terminal	33
3.1.4 SSH	35
TCP Port Forwarding	35
3.1.5 FTP	38
3.1.6 DNS	41
Risoluzione	44
Record DNS	46
Messaggi DNS	47
3.1.7 EMAIL - SMTP	48
Indirizzi di posta elettronica	49
Invio di messaggi	49
Ricezione dei messaggi	52
POP3	53
IMAP4	53
MIME	54
Web mail	55
3.2 Lo strato di trasporto	55
3.2.1 UDP	64
Checksum	65
DNS e UDP	65
Applicazioni UDP	65
3.2.2 TCP	66
Numerazione dei pacchetti	67
Connessione TCP	69
Controllo del flusso	73
Controllo degli errori	76
Controllo della congestione	81
Throughput	85
3.3 Esercizi	86
3.4 Lo strato di rete	95
3.4.1 Internet Protocol	98
Frammentazione	101

Indirizzi	103
Curiosità	108
Assegnazione dei blocchi di indirizzi	109
Aggregazione degli indirizzi	110
Indirizzi speciali	111
DHCP	111
Inoltro dei datagrammi IP	113
NAT	120
ICMP	122
Esercizi	125
3.4.2 Struttura di un router	127
3.4.3 Routing	130
Equazione di Bellman-Ford	131

Capitolo 1

Introduzione

1.1 Modalità d'esame

Email docente: federica.paganelli@unipi.it

Modalità d'esame: per il **modulo di teoria prova scritta** (compitini o appello) e orale (facoltativo) e per il **modulo di laboratorio** sono previsti **progetto e orale**.

N.B. Si accede alla prova orale di laboratorio solo dopo aver passato lo scritto.

1.2 Cenni di modelli stratificati

Sono architetture di comunicazione a strati.

Concetti generali:

- Stratificazione
- Information hiding
- Separation of concerns

Vantaggi della stratificazione:

- **Facilità di progettazione.**
- **Facilità di manutenzione.**
- **Possibilità di riciclo.**

Due modelli: **ISO/OSI** (approccio top-down) e **Stack TCP/IP** (approccio bottom-up), quest ultimo vincente.

Idea chiave: suddivisione in sottoproblemi.



Figura 1.1: Modello stratificato.

1.2.1 Livello applicativo

Fanno parte del livello applicativo:

- Identificativi delle risorse: **URL**, **URI** e **URN**.
- Il **web**: user agents, protocollo http.
- **Protocollo FTP**.
- **TELNET**: servizio di terminale virtuale.
- **Posta elettronica**.
- Sistema dei nomi **DNS** a dominio e la risoluzione dei nomi: iterativa e ricorsiva.
- molto altro ancora...

1.2.2 Livello di trasporto

Due tecnologie degne di nota:

1. Protocollo **TCP**: **connection-oriented**, *orientato alla connessione*.
2. Protocollo **UDP**: **connection-less**, molto più leggero, prende dati applicativi e li affida allo strato IP, NON da garanzie di consegna né di ordine.

1.2.3 Livello rete

Nel **livello di rete si ricava un percorso dall'host sorgente all'host destinatario** usando le informazioni che si trovano nell'IP.

Verrà trattato il protocollo Ipv4 e introdotto il protocollo Ipv6.

1.2.4 Livello link

Si occupa di gestire il collegamento tra due nodi **adiacenti**. La tecnologia principale è l'**ethernet**.

1.3 Introduzione alle reti

Cos'è una rete? Quante tipologie di reti ci sono? Cos'è internet?

Definizione 1.3.1. Una **rete** è un'interconnessione di dispositivi in grado di scambiarsi e interpretare le informazioni. Comprende sistemi terminali e intermedi: *e.g. router, switch e modem*.

I sistemi terminali si possono dividere in due tipi: **host** e **server**, sul-l'host girano le applicazioni utente mentre il server esegue programmi che forniscono servizi applicativi ad applicazioni.

N.B. il termine host può essere usato per indicare anche un server.

L'host, infatti, può essere sia un server sia il terminale di un utente che esegue un'applicazione client, più generalmente l'host è una macchina.

Definizione 1.3.2. Una rete è formata da **dispositivi** e da **tecnologie**.

1.3.1 LAN

Acronimo di **Local Area Network**, è una rete di area geografica limitata collegata attraverso una tecnologia ethernet **bus** o ethernet **switch**: ciascun host ha un cavo che lo collega allo switch e a ogni porta dello switch corrisponde un host. Lo **switch** possiede una tecnologia di autoapprendimento ed è una componente del **livello link**.

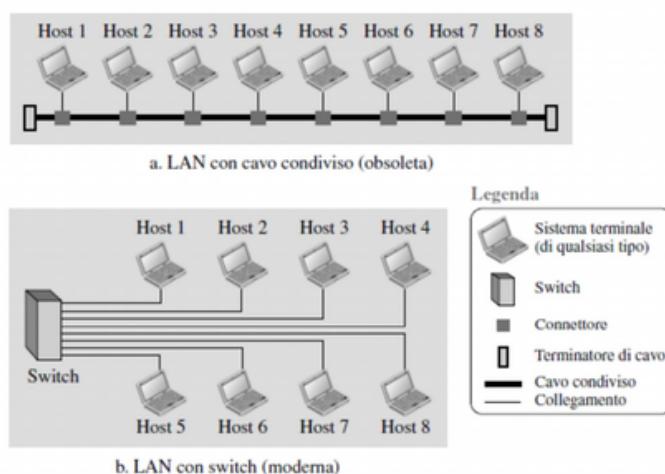


Figura 1.2: In figura due modelli diversi di LAN.

1.3.2 WAN

Acronimo di **Wide Area Network**, è una rete di area geografica estesa: è composta da due o più reti collegate tramite un mezzo di trasmissione. Le reti coinvolte potrebbero anche essere reti LAN. (*e il link potrebbe essere affittato a un'azienda da un operatore di telefonia*).

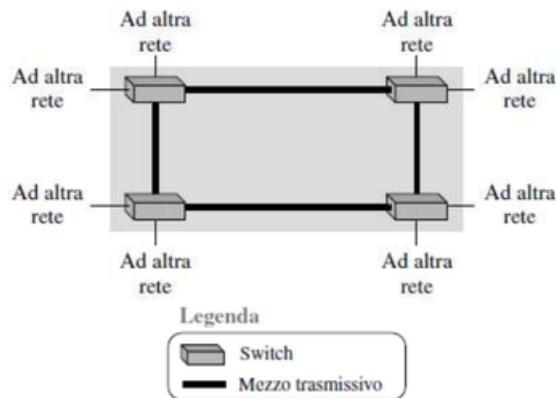


Figura 1.3: Un esempio di WAN a cavo condiviso e a commutazione.

Queste WAN permettono l'esistenza di **percorsi alternativi** e la **divisione del traffico**. Una **WAN punto a punto** invece ha solo 2 punti di terminazione.

1.3.3 Internetwork

L'internetwork è un sistema in cui ci sono più reti composte, capaci di scambiarsi informazioni e collegate. Concettualmente è una WAN ma è più complicata. I dispositivi che la compongono si distinguono in **sistemi terminali** e dispositivi come gli **switch** e i **routers** che si trovano nel percorso tra i sistemi sorgente e i sistemi destinazione.

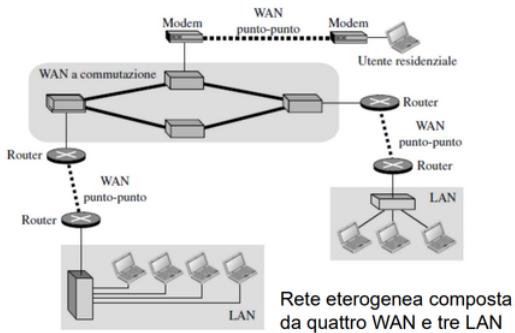


Figura 1.4: Una internetwork.

Problema: come mandare informazioni da un host a un altro?

Reti a commutazione di circuito

Nelle **reti a commutazione di circuito** le **risorse** sono **riservate end to end** per ogni connessione. Risulta quindi necessario il **setup della comunicazione** per instaurare la connessione ed elargire le risorse.

La risorsa non è tutto il link, bensì si considerano come risorse la capacità o la larghezza di banda porzionate per ogni connessione. Le risorse assegnate rimangono inattive se non utilizzate (*e.g. telefonata*). I dispositivi mantengono lo stato della connessione. **Le performance sono garantite**. La capacità delle linee (*link*) cambia a seconda della loro funzione all'interno della rete. Il **punto debole** delle reti a commutazione di circuito è la **poca flessibilità nel dispiegamento delle risorse**.

Reti a commutazione di pacchetto

Nelle **reti a commutazione di pacchetto** gli utenti inviano pacchetti che condividono le risorse del canale di comunicazione. **Non c'è** quindi **un canale dedicato** ai pacchetti di un singolo utente. La principale differenza rispetto alla commutazione di circuito risiede nell'implementazione della logica dei dispositivi di interconnessione, ovvero:

- **Commutazione di Circuito:** avviene il **setup** della connessione dove si preallocata l'utilizzo del collegamento trasmissivo con collegamenti garantiti.
- **Commutazione di Pacchetto:** non viene instaurata una connessione bensì le informazioni necessarie si trovano all'interno dei pacchetti stessi, non ci sono informazioni di connessione memorizzate nei dispositivi coinvolti.

Nelle reti a commutazione di pacchetto quindi le risorse vengono usate a seconda della necessità. Possono quindi verificarsi situazioni di contesa delle risorse e sussiste il pericolo di congestione o di perdita dei pacchetti nel caso in cui la dimensione della coda del router non fosse sufficiente a contenere il flusso dei pacchetti entranti: il commutatore (*router*) deve infatti ricevere l'intero pacchetto prima di poter cominciare a trasmetterlo sul collegamento in uscita (*store and forward*). **Non sono** quindi **garantite le prestazioni**.

Circuiti virtuali

I circuiti virtuali funzionano nel seguente modo: viene stabilito un path tra host sorgente e host destinazione e tutti i pacchetti di un certo flusso seguono lo **stesso path**.

Datagram Network

Con **datagram** si indica un'entità informativa autocontenuta che contiene le informazioni sufficienti per essere indirizzata alla destinazione senza comunicazioni aggiuntive tra sorgente e destinazione: **non è quindi detto** che pacchetti di uno stesso flusso seguano lo stesso path sulla rete.

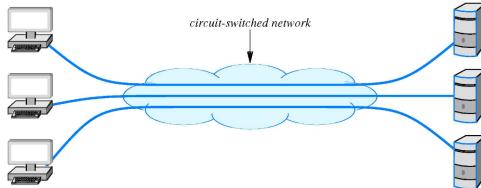


Figura 1.5: Rete a commutazione di circuito.

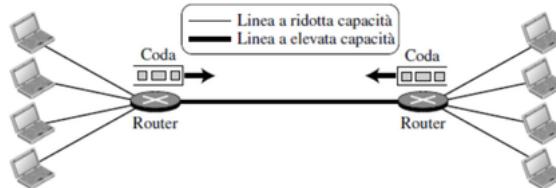


Figura 1.6: Rete a commutazione di pacchetto. Notare le diverse capacità dei link.

Packet Switch e Circuit Switch a confronto

Vi sono 35 utenti su una rete con 100 Kbit/s di connessione e un link da 1 Mbit/s. Ogni utente è attivo solo il 10% del tempo. Con una **rete a commutazione di circuito** si riescono a gestire **solo** 10 utenti. Con una **rete a commutazione di pacchetto** si hanno i seguenti casi:

1. 10 o meno utenti attivi: **nessun problema**.
2. Più di 10 utenti attivi: **ritardo**.

Tuttavia che gli utenti siano tutti e 35 attivi contemporaneamente è poco probabile (infatti $P(35) = 0.0004$). Se ne deduce che la rete a commutazione di pacchetto riesce a gestire tutti gli utenti contemporaneamente nella maggior parte dei casi. Nonostante il risultato ottenuto **non** si deve pensare che la rete a commutazione di circuito sia obsoleta. Nel corso degli anni infatti le due tecnologie sono state **integrate** in vari modi. La commutazione di circuito infatti è usata nella telefonia fissa (*PSTN: public switch telephone network*) per i servizi voce, la commutazione di pacchetto invece per i dati. Nelle reti ottiche di prima e seconda generazione si usano entrambe le tecnologie.

1.3.4 Internet

Come interconnettere reti già esistenti?

Definizione 1.3.3. Una internet (con i minuscola) è una rete costituita da due o più reti interconnesse.

La internet più famosa è chiamata **Internet** (con i maiuscola) ed è composta da migliaia di reti interconnesse. Ogni rete connessa ad Internet deve usare il protocollo IP e rispettare certe convenzioni su come vengono assegnati nomi e indirizzi. Si possono facilmente aggiungere nuove reti. Tuttavia è impensabile avere un link fisico tra ogni host, si hanno invece numerosi dispositivi di interconnessione che permettono la comunicazione da un host all'altro e da un router all'altro.

Uno scorcio delle **componenti di Internet**:

- Miliardi di **dispositivi interconnessi** (e.g. hosts, end systems).
- **Link di comunicazione** (e.g. fibre ottiche, doppini telefonici, cavi coassiali, onde radio).
- **Routers**: instradano pacchetti (*sequenze*) di dati attraverso la rete.

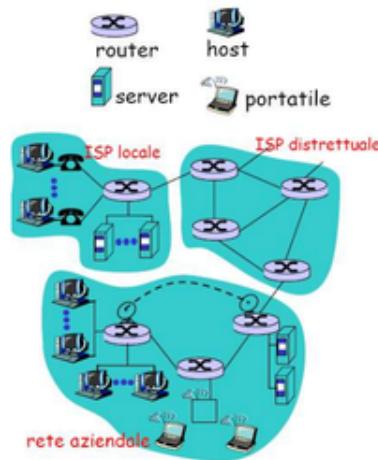


Figura 1.7: Una porzione di Internet

Uno scorcio delle **entità software** di Internet:

- Applicazioni e processi che elaborano le informazioni.
- **Protocolli** che regolamentano la trasmissione e la ricezione di informazioni e.g. TCP, IP, HTTP, FTP, PPP.
- Interfacce: verranno definite in seguito, sono le “*membrane*” che separano gli “*strati*”.

Servizi

L'**infrastruttura di comunicazione** consente il funzionamento delle applicazioni distribuite per scambio di informazioni (*e.g. WWW, email, giochi, e-commerce, database, controllo remoto, ecc.*).

Lo **stack protocollare** offre il servizio di connessione. Vi sono due approcci:

1. **Connection-less:** I dati vengono trasferiti **senza** stabilire una connessione, non c'è nessuna garanzia di ordine e consegna. *Ogni pacchetto ha una vita a sé.*
2. **Connection-oriented:** Prevede l'**instaurazione della connessione**, il trasferimento dei dati e, in seguito, la chiusura della connessione. Garantisce integrità, completezza e ordine.

IETF/RFC/ICANN

Definizione 1.3.4. L'IETF (Internet Engineering Task Force) è l'organismo che studia e sviluppa i protocolli in uso su Internet. Si basa su gruppi di lavoro a cui chiunque può accedere.

Definizione 1.3.5. RFC/STD (Request For Comments & STanDards) sono i documenti “ufficiali” che descrivono i protocolli usati su Internet. Sono pubblicamente accessibili in rete.

Definizione 1.3.6. ICANN (Internet Corporation for Assigned Names and Numbers) È un ente internazionale che coordina il sistema dei nomi di dominio (DNS), assegna i gruppi di indirizzi di rete, gli identificativi di protocollo e ha funzioni di controllo (blando) sullo sviluppo di Internet.

Rete di accesso

Internet è una internetwork che consente a qualsiasi utente di farne parte. L'utente, tuttavia, deve essere fisicamente collegato a un ISP (*internet service provider*).

Definizione 1.3.7. Il collegamento che connette l'utente al primo router di internet è detto **rete di accesso**, suddetto collegamento può essere effettuato tramite rete telefonica, rete wireless o tramite accesso diretto.

- **Accesso via rete telefonica:** servizio dial-up, ADSL o Asymmetric Digital Subscriber Line e fibra ottica.
- **Accesso tramite reti wireless.**
- **Collegamento diretto:** collegamenti WAN dedicati ad alta velocità *e.g. aziende o università.*

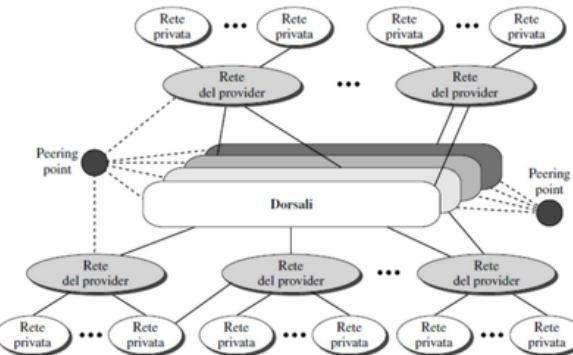


Figura 1.8: Un modello concettuale di Internet.

1.4 Metriche di riferimento

Come misurare le prestazioni di una rete?

Definizione 1.4.1. La **larghezza di banda** o **bandwidth** è la larghezza dell'intervallo di frequenze utilizzato dal sistema trasmissivo.

Definizione 1.4.2. Il **bit rate** o **trasmission rate** è la quantità di dati che possono essere trasmessi o ricevuti nell'unità di tempo. [e.g. bps = bit/s]
Il bitrate dipende dalla tecnica trasmissiva ed è proporzionale alla larghezza di banda.

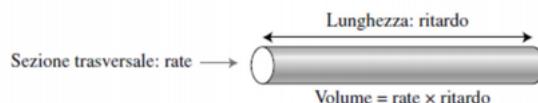
Definizione 1.4.3. Il **throughput** è la quantità di traffico che arriva realmente a destinazione nell'unità di tempo, al netto di perdite sulla rete, del funzionamento dei protocolli etc.

Definizione 1.4.4. La **latenza** o **latency** è il tempo che passa dal momento in cui il primo bit parte dalla sorgente al momento in cui l'intero messaggio arriva a destinazione.

$$L = r_{\text{propagazione}} + r_{\text{trasmissione}} + r_{\text{accodamento}} + r_{\text{elaborazione}}$$

Definizione 1.4.5. Il **volume di un link** è il numero massimo di bit che il link può contenere.

$$\text{Volume} = \text{bitrate} \times \text{ritardo}$$



1.4.1 Ritardi

Il ritardo introdotto da un nodo è la somma di questi 4 ritardi:

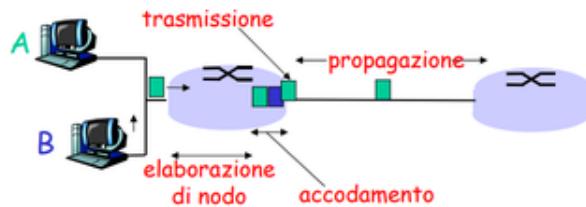


Figura 1.9: Una visione di contesto

Ritardo di elaborazione del nodo

Il ritardo di elaborazione è causato dall’elaborazione del percorso (ovvero dove inoltrare il pacchetto scegliendo il percorso *”migliore”*) e dal controllo di errori sui bit, è tipicamente piccolo e trascurabile.

Ritardo di accodamento

Il ritardo di accodamento è il tempo che un pacchetto passa nella coda del router, dipende dall’intensità e dal tipo di traffico. I pacchetti si accodano nei buffer dei router se il tasso di arrivo dei pacchetti eccede la capacità del collegamento di inoltrarli. Se non ci sono spazi liberi i pacchetti in arrivo vengono scartati.

Ritardo di trasmissione

Il ritardo di trasmissione è il tempo impiegato a trasmettere un pacchetto intero sul link.

R = rate di trasmissione del collegamento.

L = lunghezza del pacchetto.

$$r_{trasmissione} = \frac{L}{R}$$

Ritardo di propagazione

Il ritardo di propagazione è il tempo impiegato da un bit per essere propagato da un nodo (router) all’altro.

d = lunghezza del collegamento fisico

s = velocità di propagazione del collegamento fisico

$$r_{propagazione} = \frac{d}{s}$$

1.4.2 Esempi

Si consideri l'invio di un file di 1 MBit su un datalink di lunghezza 4800km:

$$d = 4800 \times 10^3 m$$

$$s = 3 \times 10^8 m/s$$

Si calcoli il ritardo di propagazione.

Soluzione:

$$r_{propagazione} = \frac{d}{s} = \frac{4800 \times 10^3 m}{3 \times 10^8 m/s} = 0.016 \text{ secondi.}$$

Sia il transmission rate pari a 64 kbps, si calcoli il ritardo di trasmissione.

$$r_{trasmissione} = \frac{L}{R} = \frac{10^6 bit}{64 \times 10^3 bps} = 15.625 \text{ secondi.}$$

Se il transmission rate fosse invece di 1 Gbps?

$$r_{trasmissione} = \frac{L}{R} = \frac{10^6 bit}{10^9 bps} = 0.001 \text{ secondi.}$$



Si calcoli il ritardo end-to-end di un pacchetto sul percorso con i router A e B. Sia trascurabile il ritardo di congestione e si suppongano uguali su tutti link il propagation delay, il transmission delay e il processing delay.

Soluzione:

Essendoci due router intermedi bisogna attraversare tre link, quindi:

$$r_{totale} = 3 \times r_{propagation} + 3 \times r_{trasmission} + 3 \times r_{processing}$$

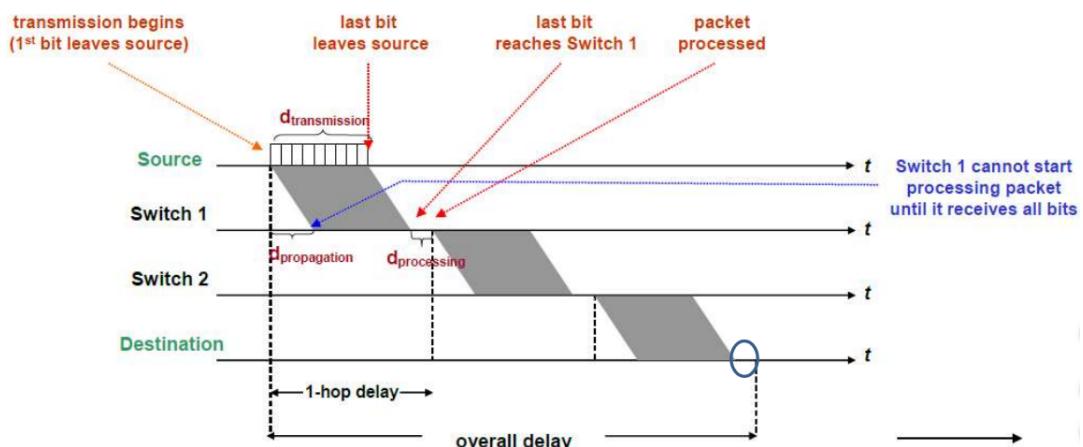


Figura 1.10: Quello che succede in dettaglio nell'esempio precedente.

Capitolo 2

Modelli Stratificati e Protocolli

Cos'è un protocollo? Cos'è uno strato?

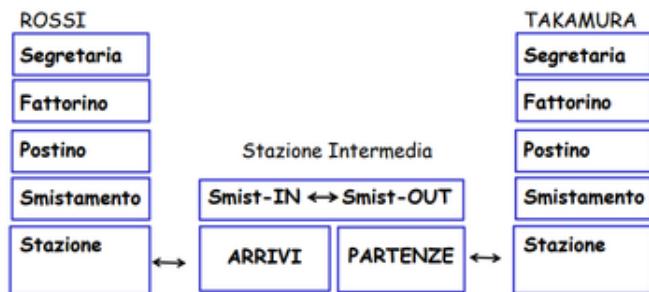


Figura 2.1: Esempio di stratificazione. Si nota come i vari strati del modello interagiscono tra di loro, dal basso verso l'alto e viceversa, per consentire al sig. Rossi e al sig. Takamura di scambiarsi lettere.

Vademecum per la lettura del seguente contenuto: N.B.:

1. Gli strati comunicano (*di solito*) **solo** con gli altri strati a loro adiacenti.
2. Uno strato **fornisce** servizi allo strato superiore e **riceve** servizi da quello inferiore.
3. La comunicazione tra strati adiacenti avviene attraverso un'**interfaccia**.
4. Tra due entità diverse comunicano fra di loro solo gli strati dello **stesso livello** e secondo un **protocollo assegnato**, queste due entità sono dette **peer**.

2.1 OSI: Open Systems Interconnection

Le **prime** reti di calcolatori nacquero come **sistemi chiusi** in cui tutti i componenti dovevano essere dello stesso costruttore. Erano quindi tecnologie chiuse e **non interoperabili** l'una con l'altra a causa di drastiche differenze (*e.g. differenza di linguaggio, modelli di stratificazione diversi e impossibilità per i programmi applicativi di riuscire ad operare in ambiente distribuito*). Alla fine degli anni '60 esistevano: ARPANET, SNA (IBM), DNA (Digital).

I **Sistemi Aperti** nascono dall'obiettivo di alcune aziende di realizzare una rete di calcolatori in cui qualsiasi terminale potesse comunicare con un qualsiasi fornitore di servizi mediante qualsiasi rete.

Per realizzare un sistema aperto è necessario stabilire delle regole comuni: **gli standards**.

Definizione 2.1.1. Un sistema che implementa **protocolli aperti** è un **sistema aperto** (open system).

Definizione 2.1.2. Un set di protocolli è **aperto** se:

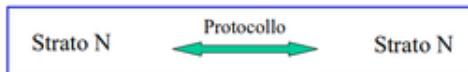
1. I dettagli (**specifiche**) dei protocolli sono disponibili pubblicamente.
2. I cambiamenti al set sono gestiti da un'organizzazione la cui partecipazione è aperta al pubblico

2.1.1 Modello ISO/OSI

L'International Organization for Standards (ISO) ha specificato uno standard per l'interconnessione di sistemi aperti: l' **Open System Interconnection Reference Model** (OSI-RM) poi diventato standard internazionale nel 1983 (ISO 7498). Si basa sul concetto di architettura a strati i cui criteri sono:

- **Divisione delle funzionalità:** il protocollo di telecomunicazione è diviso in strati o layers, ognuno dei quali svolge un compito piccolo e indipendente dagli altri.
Si cerca quindi di mantenere un minor numero di strati possibile e di far svolgere a ognuno di essi il minor numero di compiti possibile.
- **Comunicazione mediante interfacce:** i livelli comunicano mediante **chiamate standard**. Ogni livello è tenuto a rispondere alle **sole** chiamate che gli competono e che verranno invocate dal singolo livello o dai due livelli ad esso adiacenti.
- **Information hiding:** le modalità con cui le funzioni competenti ad un livello vengono svolte non è visibile dall'esterno che ne è così svincolato.

2.2 Protocollo



I protocolli definiscono il **formato** e l'**ordine** dei messaggi inviati e ricevuti tra entità della rete al livello n-esimo e le **azioni** che vengono fatte per la loro **trasmissione** e **ricezione**.

Definizione 2.2.1. Un **protocollo** è un insieme di regole che permettono a due entità uno scambio **efficace** ed **efficiente** delle informazioni. **Definisce** il **formato** e il **significato** dei frame (campi del messaggio), dei pacchetti o dei messaggi che vengono scambiati tra gli **strati paritari** di due entità diverse.

Un protocollo specifica quindi:

- La **sintassi** di un messaggio (e.g. i campi).
- La **semantica**.
- Le **azioni da compiere** (e.g. per l'invio, alla ricezione, alla trasmissione etc...).

Definizione 2.2.2. Uno **strato** o livello è un modulo interamente definito attraverso i servizi, protocolli e le interfacce che lo caratterizzano.

Definizione 2.2.3. Un' **interfaccia** è il set di regole governanti sintassi e semantica della comunicazione tra due **strati successivi** della **stessa entità**.

Definizione 2.2.4. Un **servizio** è l'insieme di **primitive** (operazioni) che uno strato fornisce ad uno strato soprastante. (*vedi sez. 3.4.1*).

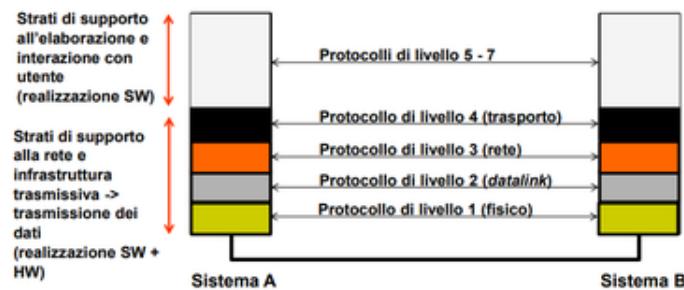


Figura 2.2: Esempio di stack protocollare.

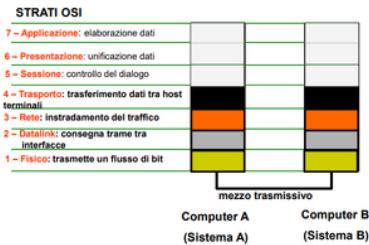


Figura 2.3: Gli strati di OSI.

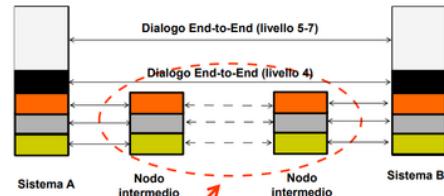


Figura 2.4: Esempio di collegamento tra end systems.

2.2.1 Incapsulamento dell'informazione

All'interno della rete l'informazione ha origine al **livello applicativo** (*livello 7 in figura*), discende quindi i vari livelli fino alla **trasmissione**, che avviene mediante il **canale fisico**. Da ogni livello attraversato viene aggiunta all'informazione una sezione informativa (o più di una) chiamata **header** che contiene informazioni pertinenti esclusivamente al livello stesso. Per i dati ricevuti invece si segue il cammino inverso. Si tratta infatti di un **processo di incapsulamento reversibile**.



- **Header:** è la qualificazione del pacchetto dati per questo livello.
- **DATA:** è il payload proveniente dal livello superiore.
- **Trailer:** è usato per rilevare e correggere gli errori.

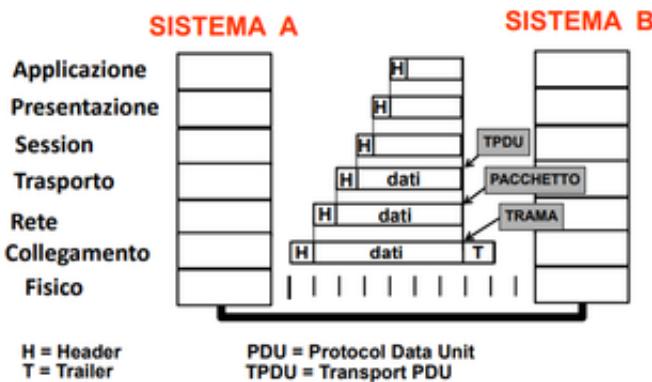


Figura 2.5: Il processo di incapsulamento. Da notare in particolare il payload.

Capitolo 3

Stack protocollare TCP/IP

TCP/IP è una **famiglia di protocolli** attualmente in uso su Internet. Si tratta di una **gerarchia di protocolli** costituita da **moduli interagenti**, ciascuno dei quali fornisce funzionalità specifiche. Il termine **gerarchia** significa che ciascun protocollo di **livello superiore** è supportato dai servizi **forniti** dai protocolli di **livello inferiore**. Definita in origine in termini di quattro livelli software soprastanti a un livello hardware, la **pila TCP/IP** è oggi intesa come **composta di cinque livelli**.

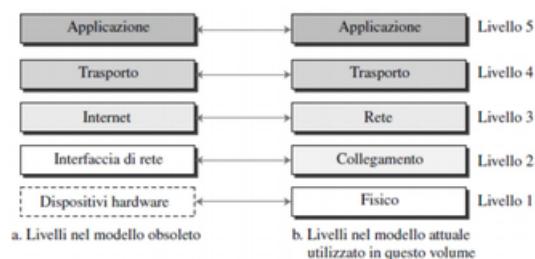


Figura 3.1: Livelli dello stack protocollare TCP/IP, si notino le differenze tra il modello originario e il modello attuale.

- Il **livello applicativo** supporta le applicazioni di rete.
- Il **livello di trasporto** supporta il trasferimento di dati da un host all'altro.
- Il **livello di rete** instrada i datagrammi dalla sorgente alla destinazione.
- Il **livello link** trasferisce dati tra elementi adiacenti della rete.
- Al **livello fisico** troviamo i bit sul link.

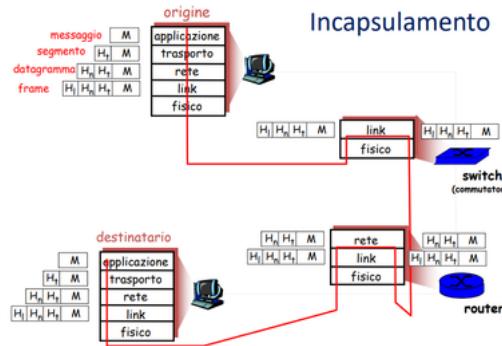


Figura 3.2: Esemplificazione del processo di incapsulamento/decapsulamento dell'informazione.

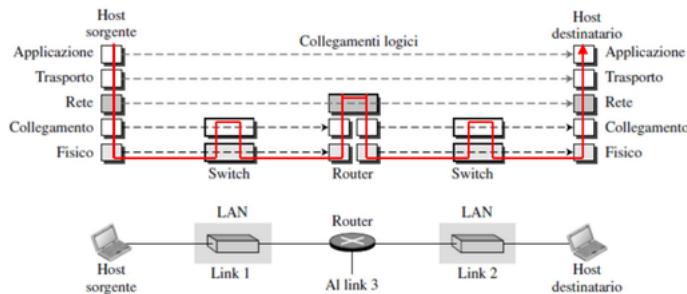


Figura 3.3: I vari collegamenti logici di comunicazione, in **rosso** invece, il **collegamento fisico**. La modularità del sistema fa in modo che gli strati paritari dei due host abbiano l'**illusione** di comunicare direttamente tra di loro. Si ricorda che queste entità situate a livelli corrispondenti su macchine (host) diverse sono dette **peer**.

3.1 Lo strato applicativo

Dello **strato applicativo** fanno parte le **applicazioni di rete**. Le applicazioni di rete sono composte da **processi distribuiti e comunicanti**, ovvero programmi eseguiti dai dispositivi terminali (host o end system) di una rete. Nella comunicazione a **livello applicativo** fra due dispositivi terminali interconnessi, due o più processi sono in esecuzione su ciascuno degli host comunicanti e si scambiano messaggi. Il protocollo dello strato applicativo definisce: (*repetita iuvant*)

- I **tipi** di messaggi scambiati al livello applicativo (e.g. richiesta e risposta).
- La **sintassi** e la **semantica** dei campi dei messaggi.
- Le **regole** di comunicazione **interprocessuale**.

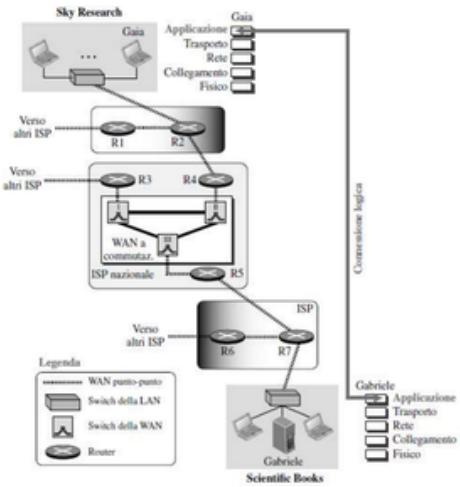


Figura 3.4: Esempio di comunicazione tra applicazioni di rete. Ancora una volta è bene ricordare che il protocollo crea l'*illusione* che i processi siano in comunicazione diretta.

Presentiamo ora i **paradigmi di comunicazione** dello strato applicativo:

- **Client-Server:** prevede un numero *limitato* di processi **server** che offrono servizi e sono **sempre** in esecuzione in attesa di ricevere richieste dai client. Un **client** è un programma che richiede un servizio. Tipicamente il client inizia il contatto con il server **invitando una richiesta** e il server risponde **offrendo il servizio** richiesto.
- **Peer-to-Peer:** comunicazione tra **peer (pari)** che possono sia **offrire** servizi che **inviare** richieste.
- **Misto.**

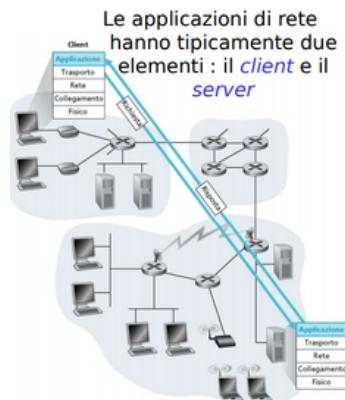


Figura 3.5: Esempio di paradigma client-server.

Abbiamo, seppur assai brevemente, esaminato la comunicazione a livello applicativo tra macchine diverse. Scendiamo ora più nel dettaglio iniziando a tracciare il percorso **reale** dei dati dallo **strato applicativo** allo **strato di trasporto**. Partiamo dalla seguente definizione:

Definizione 3.1.1. API: acronimo di Application Programming Interface, è un insieme di **procedure** e **regole** che un programmatore deve seguire per accedere a delle risorse o per realizzare l'interazione tra due entità. Facilita molto la programmazione del software client-side.

L' **API** che funge da **interfaccia** tra gli **strati di applicazione** e di **trasporto** è chiamata **socket** ed è usata dai processi dello strato applicativo per inviare e ricevere dati dallo strato di trasporto. Si tratta, ancora una volta, di una connessione **logica** poichè in realtà l'invio e la ricezione dei dati sono, nel **concreto**, responsabilità del sistema operativo e del protocollo TCP/IP.

Riportiamo un estratto dell'**API** di **TCP**:

```
connection TCPopen(IPaddress, int) //to open a conn.  
void TCPSend(connection, data) //to send data  
data TCPReceive(connection) //to receive data  
void TCPclose(connection) //to close a conn.
```

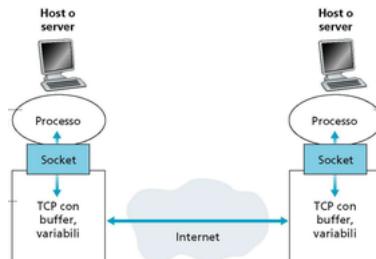


Figura 3.6: In figura due processi che comunicano tramite socket. Il processo è controllato dallo sviluppatore dell'applicazione, tutto ciò che è presente sotto la socket è controllato dal sistema operativo.

Sorge ora spontanea una domanda: **se ci fossero più processi su ogni host?**

Servirebbe un modo per **identificarli...** ci viene in aiuto il **socket address**.



Figura 3.7: Il socket address identifica sia il **processo** con il numero di porta che l'**host** con l'indirizzo IP, a ogni porta corrisponde un processo.

Riassumendo: dello strato applicativo fanno parte le **applicazioni di rete**, composte da **processi distribuiti** su macchine diverse che comunicano tra di loro mediante il protocollo proprio dello strato applicativo. Sebbene il protocollo fornisca alle applicazioni l'illusione di comunicare direttamente, la comunicazione è in realtà garantita da tutto lo stack protocollare sottostante. In particolare, lo strato applicativo dipende dai servizi offerti dallo **strato di trasporto** che gli è immediatamente sottostante. I due strati comunicano tramite una **API** che si chiama **socket**.

A seconda del **servizio di trasporto richiesto** dall'applicazione è possibile che si trovino in uso **protocolli di trasporto diversi** tra i quali: **TCP e UDP**. (*vedi sez. 2.2*)

Applicazione	Tolleranza alla perdita di dati	Throughput	Sensibilità al tempo
Trasferimento file	No	Variabile	No
Posta elettronica	No	Variabile	No
Documenti Web	No	Variabile	No
Audio/video in tempo reale	Si	Audio: da 5 Kbps a 1 Mbps Video: da 10 Kbps a 5 Mbps	Si, centinaia di ms
Audio/video memorizzati	Si	Come sopra	Si, pochi secondi
Giochi interattivi	Si	Fino a pochi Kbps	Si, centinaia di ms
Messaggistica istantanea	No	Variabile	Si e no

Figura 3.8: In figura le caratteristiche di alcune delle principali applicazioni di rete. Non tutte le applicazioni sono uguali. Tipicamente la telefonia di internet usa il protocollo UDP.

3.1.1 Web

Il **web** è formato da risorse indirizzate da **URL**, acronimo di **uniform resource locator**. Generalmente queste risorse sono **pagine web**, formate da altri oggetti referenziati (*e.g. altre pagine web, immagini ecc...*). Lo **user agent** o **client** per il web è chiamato **browser** ed il **server** è chiamato **web server**.

Definizione 3.1.2. Una **URI** o Uniform Resource Identifier è una stringa compatta di caratteri che identifica una risorsa fisica o astratta.

Le **URL** sono un **sottoinsieme** delle URI e identificano le risorse tramite la loro posizione all'interno della rete. Le **URN** acronimo di **uniform resource name** sono un altro sottoinsieme delle URI la cui funzione è di rimanere **globalmente uniche e persistenti** anche quando le risorse da loro puntate cessano di esistere o non sono più disponibili. La **sintassi** delle URI è organizzata **gerarchicamente** e i componenti sono disposti in ordine di importanza da sinistra verso destra.

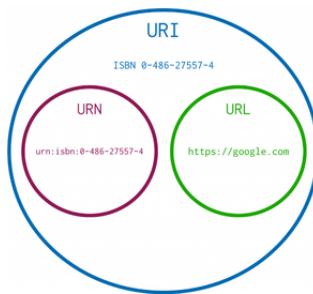


Figura 3.9: URN e URL non sono altro che specializzazioni delle URI.

Sintassi della URI

$< \text{scheme} > : // < \text{authority} > < \text{path} > ? < \text{query} >$

`http://maps.google.it/maps/place?q=largo+bruno+pontecorvo+pisa &hl=it`

- **scheme**: è **obbligatorio**, definisce lo **spazio dei nomi** della risorsa.
- **authority**: indica il **nome di dominio** di un host (*reg_name*) o il suo **indirizzo IP** in notazione decimale puntata. Tipicamente identifica un computer sulla rete.
- **path**: contiene **dati** specifici per l'authority (o per lo scheme) e **identifica** la risorsa **nel contesto** di quel namespace e di quell'authority.

Le URI possono essere **absolute** o **relative**, una **URI assoluta** identifica una risorsa **indipendentemente** dal contesto in cui è usata. Una **URI relativa** identifica una risorsa in relazione ad un'altra URL, è **priva di schema e di authority**, non viaggia sulla rete ed è interpretata dal browser in relazione al documento di partenza.

(e.g. `http://www.w3.org/pub/WWW/TheProject.html` oppure
`/pub/WWW/TheProject.html` sotto l'host: `www.w3.org`).

3.1.2 HTTP

Il protocollo **HTTP** è un protocollo di tipo **request/response**: il client inizia la connessione inviando un messaggio di **request** al server che a sua volta invia una **response**. Si tratta di un **protocollo generico**, poichè non dipende dal formato delle risorse, e **stateless** poichè **le coppie richiesta/-risposta sono indipendenti l'una dall'altra**. In **HTTP 1.0** dopo la prima coppia di richiesta/risposta la connessione viene **terminata** mentre in **HTTP 1.1** si **procede** con un'altra coppia.

Lo **schema http** è usato per accedere alle risorse attraverso il protocollo HTTP.

Sintassi della URL HTTP

http : // < host > [:< port >][< path >]

- **host:** è un **host domain di Internet** valido oppure un indirizzo IP in forma decimale puntata.
- **port:** è un **intero**, se la porta è vuota o non è indicata si usa automaticamente la porta 80.
- **path:** specifica la **request URI** (*vedi seguito*).

N.B. Il protocollo HTTP utilizza il protocollo TCP tramite la sua API.

```
//esempio client
c = TCPopen("131.115.7.24", 80);
TCPsend(c,"GET /index.html");
d = TCPreceive(c);

//esempio server
p = TCPbind(80); //where to wait for connections
d = TCPaccept(p); //waiting for connections
r = TCPreceive(d);
...
TCPsend(d,pag);
TCPclose(d);
```

Connessione HTTP

Definizione 3.1.3. Una **connessione http** è un circuito logico di livello di trasporto stabilito tra due programmi applicativi per comunicare tra loro.

Una **connessione HTTP** può essere:

- **Non persistente**(*http1.0: RFC 1945*): per ogni richiesta del **client** viene instaurata **una nuova connessione** con il **server**. Ciò aumenta il **carico** su quest ultimo e potrebbero verificarsi fenomeni di **congestione**. Infatti per visualizzare le *n* immagini di un sito il client invia di seguito *n* richieste al server.
 - **Persistente**(*http1.1: RFC 2616*): la connessione è appunto **persistente**. Nello **standard** è specificato un meccanismo che consente al server di **chiudere** la connessione TCP su richiesta del client. (*CONNECTION = CLOSE, in GENERAL HEADER, vedi seguito.*)
- N.B. Una volta che la chiusura della connessione è stata segnalata, il client **non deve** inviare altre richieste.

Esempio:

Supponiamo che l'utente digitи la seguente URL:

www.someSchool.edu/someDepartment/home.index

Con **una connessione non persistente**, in ordine temporale succedono le seguenti cose:

1. Il **client http** invia una richiesta di connessione **TCP** verso il server http al **www.someSchool.edu** (*La porta 80 è usata di default per il server http.*)
2. Il **server http** dell'host **www.someSchool.edu**, che aspetta le richieste di connessione **TCP** alla **porta 80, accetta la richiesta di connessione** e notifica il client.
3. Il **client http** invia quindi un **messaggio di richiesta**, contenente la URL.
4. Il **server http** riceve il messaggio di richiesta, compila un **messaggio di risposta** con l'oggetto richiesto indicato dalla URL: **someDepartment/home.index**, invia il messaggio e in seguito **chiude la connessione**.
5. Il **client http** riceve il messaggio di risposta che contiene il file html e lo visualizza.

Si ricorda che la ricezione e la trasmissione di tutti i messaggi elencati poco sopra avviene tramite socket.

Supponiamo ora che la URL contenga dei riferimenti a 10 immagini, in tal caso per ogni riferimento si devono ripetere tutti i passaggi definiti sopra. Salta immediatamente all'occhio la **scarsa efficienza** della procedura.

Con **una connessione persistente** invece il server lascerebbe aperta la connessione TCP dopo aver spedito la prima risposta e vi riceverebbe quindi le richieste successive. La connessione verrebbe chiusa dal server quando specificato nell'header di un messaggio **inviatogli dal client**, oppure alla mancata ricezione di richieste per un certo intervallo di tempo (*time out*).

Un ulteriore miglioramento delle prestazioni, si otterrebbe con una tecnica di **pipelining**, consistente nell'invio da parte del client di molteplici richieste senza aspettarne la ricezione da parte del server.

Il **server deve** tuttavia inviare le risposte nello stesso ordine in cui sono state ricevute le richieste e il **client non** può inviare **in pipeline** richieste che usano metodi HTTP **non idempotenti**. Ma cos'è un **metodo idempotente?**

Metodo idempotente

Definizione 3.1.4. Un **metodo idempotente** è un metodo tale che il suo effetto collaterale sulla risorsa è lo stesso per N o 1 richieste identiche che ne fanno uso. (e.g. *GET, HEAD, PUT, DELETE, OPTIONS, TRACE*)

Definizione 3.1.5. Un **metodo safe** è un metodo che non produce effetti collaterali sulle risorse. (e.g. non le modificano: *GET, HEAD, OPTIONS, TRACE...*)

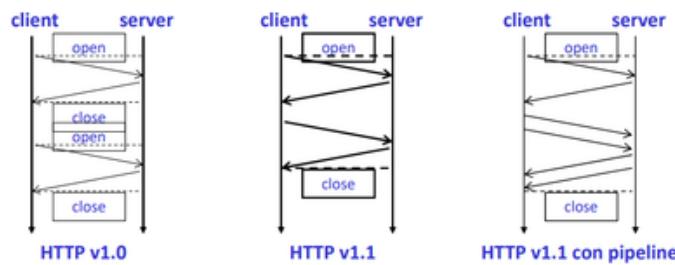


Figura 3.10: In figura sono rappresentate le differenze viste prima.

Messaggi HTTP

Riportiamo di seguito la **struttura** di un generico messaggio http:

Request = Request-Line o Response = Status-Line per i messaggi di risposta.

```
*(` general-header
| request-header o response-header
| entity-header )
CRLF
[ message-body ]
```

La prima riga o **start line** distingue i messaggi di request dai messaggi di response.

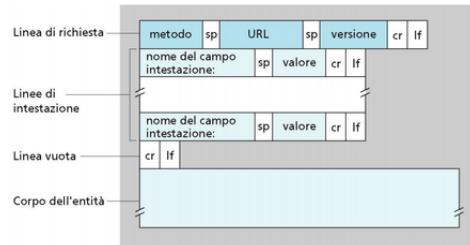


Figura 3.11: In figura un messaggio di richiesta.

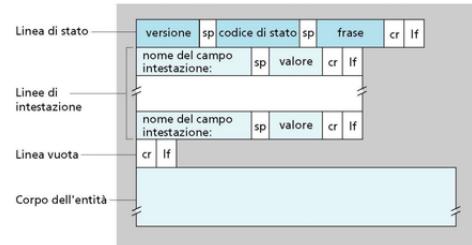


Figura 3.12: In figura un messaggio di risposta.



Figura 3.13: La struttura della **request line**. Il campo **method** è case sensitive ed indica l'operazione da eseguire sulla risorsa identificata dalla URI. Il metodo **POST** serve per inviare dal client al server le informazioni inserite nel body del messaggio, **PUT** è usato dal client per chiedere al server di creare o modificare una risorsa, **DELETE** per cancellarla.

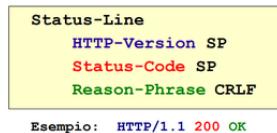


Figura 3.14: La **status-Line** è la prima riga del messaggio di risposta. Lo **status code** è un numero di 3 cifre, indica il risultato del tentativo di soddisfare la richiesta del client. La **reason-phrase** dà una breve descrizione testuale dello status-code. Lo status-code è rivolto alla macchina mentre la reason-phrase all'utente umano.

Gli **header** sono insiemi di coppie **nome : valore** che specificano alcuni parametri del messaggio trasmesso o ricevuto.

- **General Headers:** sono relativi alla **trasmissione** e si applicano a **tutto il messaggio**. (e.g. **Date**, **Connection**: usato dal mittente per specificare delle opzioni desiderate per la connessione, ad esempio **close**. **Transfer-encoding**: specifica se e quali trasformazioni sono state applicate al corpo del messaggio ad esempio **gzip**, **chunked** ecc. **Cache control**: indica quale tipologia di cache può memorizzare il messaggio, può essere **public**, **private** o **no-cache**.)
- **Entity Headers:** sono **metadati** relativi all'**entità trasmessa**. Ogni entity è costituita da un **entity body** e da una serie di **entity headers** che ne definiscono contenuto e proprietà. (e.g. **Content Length**)
- **Request Headers:** sono relativi alla **richiesta**. Supportano il **content negotiation**, il processo tramite cui il server sceglie la forma "giusta" del contenuto richiesto dal client. (*Specificano da chi è fatta la richiesta, a chi viene fatta, che tipo di risposte il client è in grado di accettare, l'autorizzazione, ecc.*)
- **Response Headers:** sono relativi al **messaggio di risposta**.

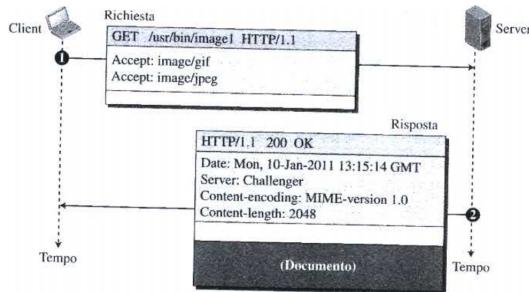


Figura 3.15: In figura un esempio di richiesta HTTP con cui il client preleva un documento, viene usato il metodo **GET** per ottenere l'immagine individuata dal percorso `usr/bin/image1`.

```

general-header = Cache-Control
| Connection
| Date
| Pragma
| Transfer-Encoding
| Upgrade
| Via
  
```

Figura 3.16: L'elenco dei general headers.

```

request-header = Accept           | Accept-Charset
| Accept-Encoding   | Accept-Language
| Authorization
| Proxy-Authorization
| From              | Host
| If-Modified-Since
| If-Unmodified-Since
| If-Match          | If-None-Match
| If-Range
| Max-Forwards      | Range
| Referer          | User-Agent
  
```

Figura 3.17: L'elenco dei request headers. **Accept** specifica quali tipi di media sono accettati in risposta dal client, **Accept-Charset**, quali caratteri e **Accept-Encoding** quali formati. È possibile specificare una **condizione**, in tal caso il server invia la risorsa solo se la condizione è soddisfatta altrimenti informa il client della motivazione del mancato invio.

<pre> GET http://www.commonServer.com/information/file1 HTTP/1.1 If-Modified-Since: Thu, Sept 04 00:00:00 GMT </pre>	Riga di richiesta Riga di intestazione Riga vuota
--	--

Figura 3.18: In figura un esempio di richiesta condizionale.

HTTP/1.1 304 Not Modified Date: Sat, Sept 06 08 16:22:46 GMT Server: commonServer.com (Corpo vuoto)	Riga di stato Prima riga di intestazione Seconda riga di intestazione Riga vuota Corpo vuoto
--	---

Figura 3.19: In figura la risposta alla richiesta condizionale di prima. La riga di stato indica che il file non è stato modificato successivamente alla data indicata dal client, nulla viene inviato e il corpo del messaggio è vuoto.

```

response-header = Age
| Location
| Proxy-Authenticate
| Public
| Retry-After
| Server
| Vary
| Warning
| WWW-Authenticate

```

Figura 3.20: I campi del **response-header** consentono al server di inviare informazioni aggiuntive che non possono essere poste nella status-line. Ad esempio: **Age**: indica quanto tempo è trascorso, in secondi, tra l'invio della risposta da parte del mittente e la generazione della stessa all'origine. **Location** è usato per reindirizzare il client verso un'altra URI per completare la sua richiesta o per fornire una nuova risorsa. Il campo **Server** contiene informazioni sul software usato dal server di origine per portare a compimento la richiesta del client.

"100" - Continue "200" - OK "202" - Accepted "204" - No Content "206" - Partial Content "300" - Multiple Choices "302" - Moved Temporarily "304" - Not Modified "400" - Bad Request "402" - Payment Required "404" - Not Found "406" - Not Acceptable "408" - Request Time-out "410" - Gone "412" - Precondition Failed "414" - Request-URI Too Large "500" - Internal Server Error "502" - Bad Gateway "504" - Gateway Time-out	"101" - Switching Protocols "201" - Created "203" - Non-Authoritative Information "205" - Reset Content "301" - Moved Permanently "303" - See Other "305" - Use Proxy "401" - Unauthorized "403" - Forbidden "405" - Method Not Allowed "407" - Proxy Authentication Required "409" - Conflict "411" - Length Required "413" - Request Entity Too Large "415" - Unsupported Media Type "501" - Not Implemented "503" - Service Unavailable "505" - HTTP Version not supported
--	--

Figura 3.21: In figura l'elenco degli status-codes e il loro significato. Descrivono lo stato della risposta. I codici dal 100 al 199 indicano informazioni, dal 200 al 299 che una richiesta è stata eseguita con successo, quelli tra 300 e 399 reindirizzano il client verso un'altra URL, quelli tra 400 e 499 indicano che si è verificato un errore lato client e quelli dal 500 al 599 dal lato server.

Fin'ora abbiamo mostrato come il client accede alle risorse del web inviando richieste al server. Cosa succederebbe se, ad esempio, un client richiedesse continuamente la **stessa** risorsa? Potremmo evitare di inoltrare richieste tutte uguali al server?

Sì, ci viene in aiuto il **web caching**. L'obbiettivo del web caching è di soddisfare le richieste del client **senza** contattare il server. Funziona memorizzando **copie temporanee** di risorse web, servendole poi al client così da **ridurre** l'uso di **banda**, limitare il **workload** sul server e di conseguenza **diminuire tempo di risposta** verso gli altri clients.

Consideriamo due modelli possibili di web caching:

- **User Agent Cache**: lo **user agent** (il browser) mantiene una **copia** delle risorse visitate dall'utente.
- **Proxy Cache**: il **proxy intercetta il traffico** e mette in cache le risposte. Successive richieste alla stessa URI possono essere servite direttamente dal proxy senza inoltrare la richiesta al server. Sta poi all'utente configurare il browser per consentire gli accessi Web via proxy.

Proxy

Definizione 3.1.6. Un **proxy** è un **programma intermediario** che agisce sia da **server** che da **client**, **invia**ndo e **servendo** **richieste** per altri clients. Le richieste sono gestite internamente o inoltrate a server terzi.

Resta ora da chiarire come è possibile che, sebbene abbiamo definito il protocollo HTTP come **stateless**, alcuni siti, *riconoscano* gli utenti. Sebbene infatti il protocollo HTTP sia a tutti gli effetti stateless, le applicazioni web **non** lo sono. Come fare quindi a conciliare queste due realtà?

Cookies

Definizione 3.1.7. I **cookies** sono stringhe di testo contenenti informazioni relative all'utente.

I **cookies** funzionano nel seguente modo:

1. Un client invia al server una richiesta HTTP.
2. Il server invia al client la risposta HTTP e in più una linea **set-cookie: 1678453. (esempio fittizio)**
3. Il client **memorizza** il cookie in un file e lo associa al server. Lo aggiungerà con la seguente linea: **cookie: 1678453** a tutte le sue successive richieste.
4. Alla successiva richiesta da parte del client, il server **risalirà** tramite il cookie alle informazioni ad esso **associate**.

3.1.3 TELNET

TELNET, acronimo per **TERminal NETwork**, è un **protocollo client-server** che fornisce una comunicazione **interattiva ed orientata al testo** tra due macchine, è basato sul trasporto **connection-oriented** ed usa il protocollo **TCP**. Consente all'utente di effettuare una sessione di **login** in una macchina remota e quindi di utilizzarne il terminale. L'utente che si autentica tramite login remoto ha accesso infatti a tutti i **comandi** e ai **programmi** disponibili su di essa. I comandi vengono eseguiti come se l'utente li digitasse dalla tastiera stessa della macchina. Per estensione, **telnet** è anche il nome di un programma usato per avviare una sessione **TELNET** verso un host remoto. **TELNET** infatti include due programmi: un programma **TELNET client** e un programma **TELNET server**. **TELNET client** (**telnet**) interagisce con l'utente sulla macchina locale e scambia messaggi con **TELNET server**. Riportiamo di seguito una semplificazione del funzionamento:

1. L'utente, dalla **propria** macchina **locale** (**TELNET client**), stabilisce una connessione **TCP**, persistente per tutta la durata della sessione, con una **macchina remota** (**TELNET server**) alla **porta 23** e vi si **autentifica**.
2. Tutte le **battute** dei **tasti** della macchina locale vengono **trasmesse** dal client alla **macchina remota**.
3. La **macchina remota** accetta la connessione **TCP** e il **TELNET server** trasmette i dati al sistema operativo locale.
4. L'**output** della **macchina remota** viene quindi **ricevuto** e **trasmesso** sul terminale dell'utente.

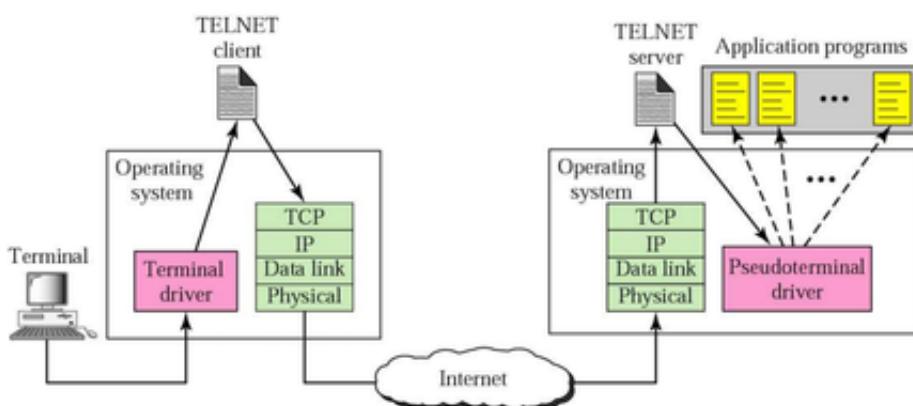


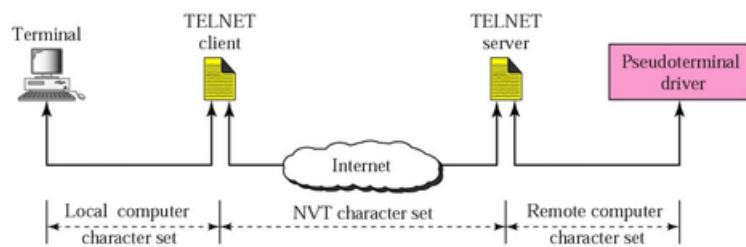
Figura 3.22: In figura il funzionamento di TELNET.

N.B. TELNET è un protocollo STATEFUL.

Network Virtual Terminal

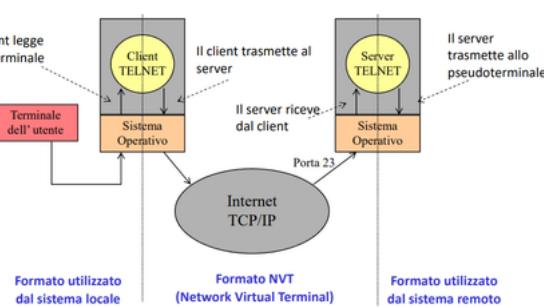
I terminali non sono tutti uguali, possono differire gli uni dagli altri per il **set** e la **codifica** dei caratteri, per la **lunghezza** della **linea** e della **pagina** e per i **tasti funzione** individuati da diverse sequenze di caratteri.

Una **soluzione** è stata trovata mediante la definizione di un **Network Virtual Terminal (NVT)** che **definisce un set di caratteri universali**. L'NVT è quindi un dispositivo “immaginario” che fornisce una rappresentazione astratta di un terminale. Gli host, sia client che server, **traducono** le loro caratteristiche **locali tramite il set universale** così da apparire **in rete** come un NVT e assumono che l'host remoto sia anch'esso un NVT. Questa operazione di traduzione è ovviamente **reversibile**.



Il funzionamento di TELNET, con l'aggiunta della trasformazioni intermedie sopra descritte diventa:

1. ...
2. Tutte le **battute** dei **tasti** della macchina locale vengono **trasformate** in NVT e successivamente **trasmesse** dal **TELNET client** alla **macchina remota**.
3. La **macchina remota** accetta la connessione TCP, il **TELNET server** traduce da NVT allo standard del sistema operativo remoto e infine gli trasmette i dati ricevuti.
4. ...



N.B. NVT invia i caratteri di controllo prioritariamente con TCP URGENT.

Concludiamo con qualche tecnicismo:

I terminali NVT si scambiano dati in formato **7-bit US-ASCII** e adottano inoltre un approccio **in-band signaling**, ovvero dati e comandi viaggiano sullo **stesso canale**. Per distinguere i due tipi di informazione si usa la seguente convenzione: ogni carattere è inviato come un **ottetto di bit** con **il primo bit settato a zero**. I caratteri, in notazione decimale, vanno dal numero 0 al 127. I comandi invece sono identificati tramite ottetti speciali di 1, in n. d. vanno dal numero 240 al numero 254, per distinguerli dai dati sono sempre preceduti da un carattere speciale: **IAC o Interpret As Command** identificato, sempre in n. d., dal numero 255. Essenzialmente quindi si usa un canale di 8 bit per scambiare dati di tipo 7 bit ASCII. I messaggi scambiati durante la fase iniziale della comunicazione, ovvero prima del login, sono **messaggi di controllo** e costituiscono la **Telnet Option Negotiation**, in sostanza sono usati per scambiare **informazioni** sulle **caratteristiche** degli host.

Comando	Codifica decimale	Significato
IAC	255	Interpret as command
EL	248	Erase line
EC	247	Erase character
IP	244	Interrupt process
EOR	239	End of record

Ma quindi NVT conviene?

Rispondiamo alla domanda mostrando dei semplici calcoli:

Supponiamo che N sia il numero di sistemi distinti che si vogliono far interoperate:

- **Senza** l'uso di NVT, si necessita la scrittura di **N-1** client per ogni sistema ($N-1$ TELNET-clients che traducano negli $N-1$ standards dei sistemi), e 1 TELNET-server per ogni sistema:
In totale si avranno $N \times (N - 1) + N$ applicativi.
- **Usando** NVT, bisogna scrivere 1 TELNET-server per ogni sistema e N TELNET-client (1 per ogni sistema che traduca dallo standard di sistema allo standard di NVT):
Avremo quindi $N + N = 2N$ applicativi.

Quindi per $N > 2$ **conviene** usare NVT!

3.1.4 SSH

TELNET non possiede **alcuna** misura di sicurezza poichè è stato progettato per l'uso su **reti private**, trasmette tutto in chiaro, anche le **password!** Con l'avvento delle reti pubbliche però si è reso necessario prendere delle contromisure.

SSH o Secure SHell è un'applicazione nata per sostituire TELNET.

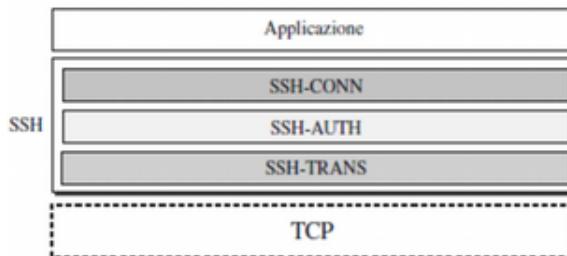


Figura 3.23: Le componenti di SSH

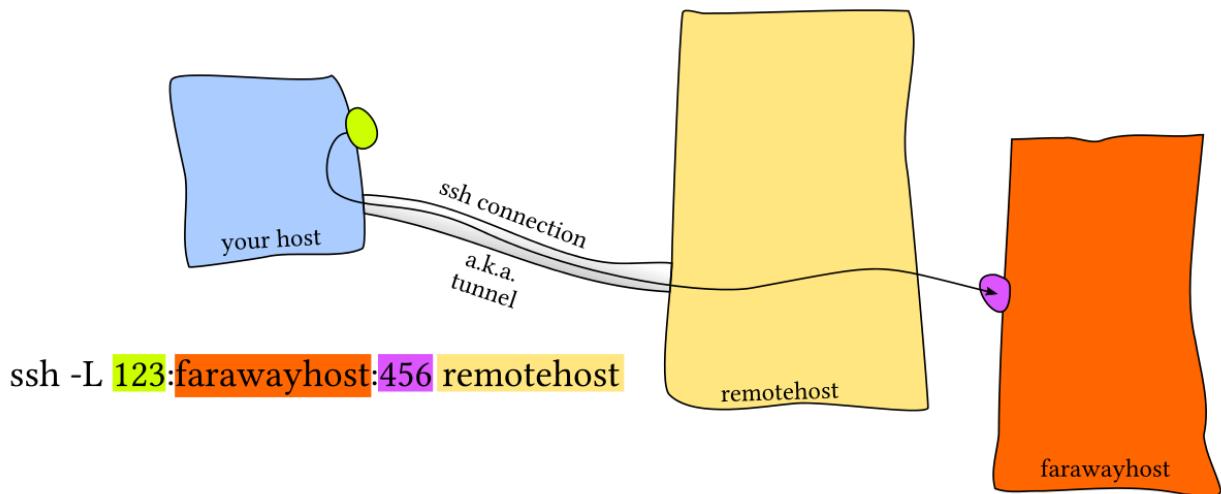
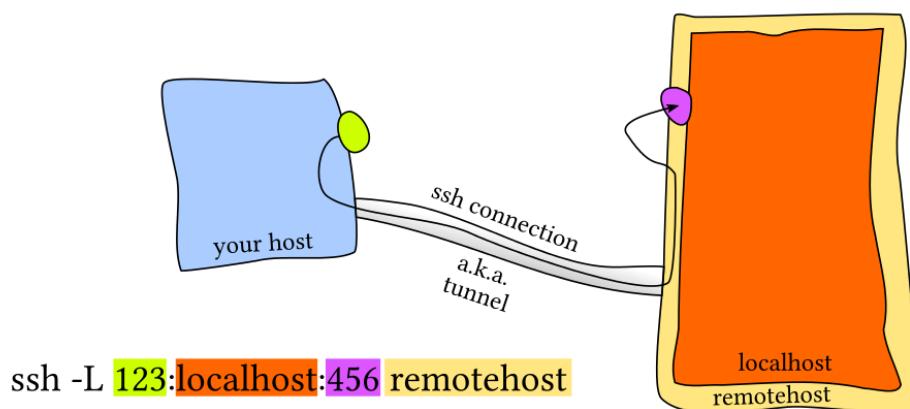
Il **protocollo** di livello applicazione **SSH** è composto da **tre** diverse componenti: **SSH-TRANS**, **SSH-AUTH** e **SSH-CONN**. La componente **SSH-TRANS** o **SSH-Transport Layer Protocol** costruisce un canale di comunicazione sicuro, tramite tecniche crittografiche, sfruttando la connessione offerta da TCP. Il **protocollo TCP** trasmette tutte le informazioni in chiaro, **non** è quindi in grado **da solo** di garantire **privacy** e **confidenza**. *SSH-TRANS è anche in grado di riconoscere se il server a cui ci stiamo connettendo è quello autentico o meno.* La componente **SSH-AUTH** si occupa di autenticare il client. *Sono disponibili altre tecniche di autenticazione oltre alla classica con username e password tra le quali l'accesso basato su coppie di chiavi crittografiche.* Infine la componente **SSH-CONN** sfruttando i servizi offerti dalle altre due componenti offre i servizi di **terminale, trasferimento file, creazione di tunnel** ecc...
SSH offre quindi molti più servizi di **TELNET**.

TCP Port Forwarding

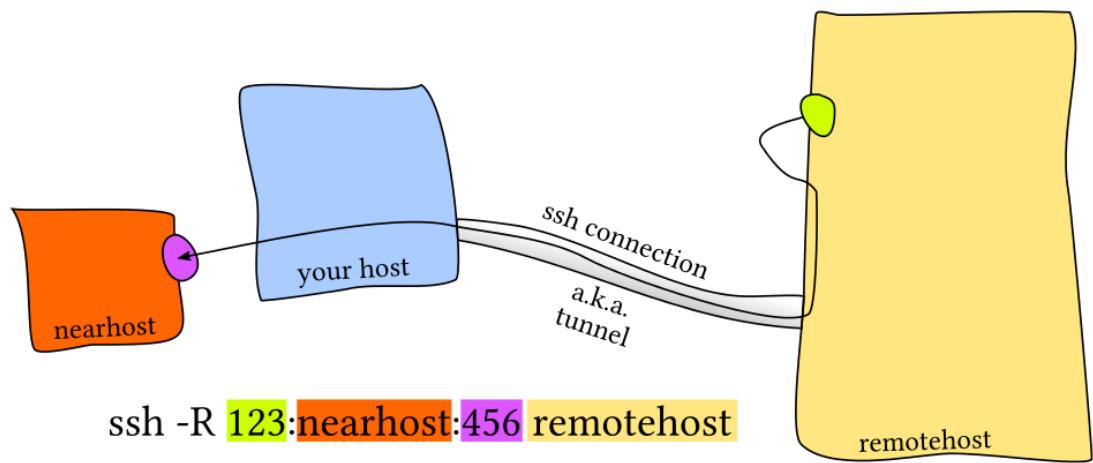
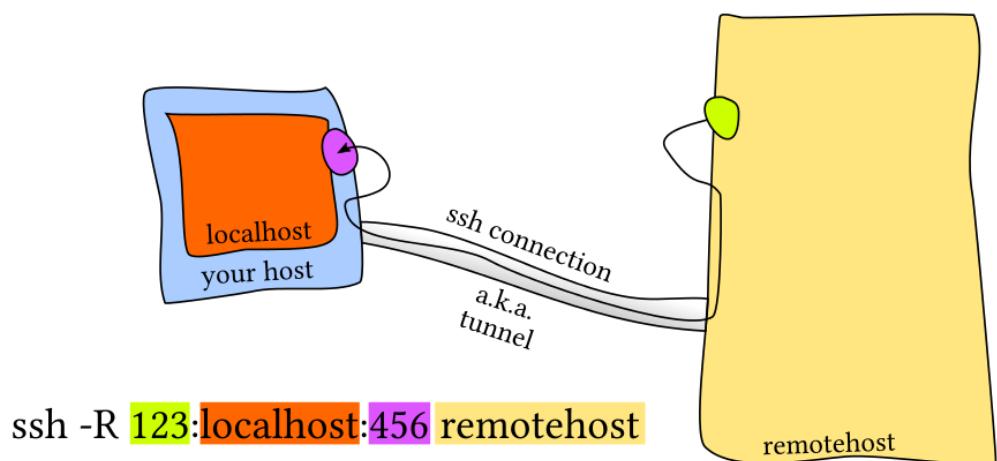
Il **TCP Port Forwarding** è un meccanismo che permette di **creare un canale di comunicazione sicuro** attraverso il quale **veicolare qualsiasi tipo di connessione TCP**. Opera creando un canale di comunicazione **cifrato** tra la **porta all'indirizzo remoto** a cui ci si vuole collegare e una **porta locale** libera. Le applicazioni punteranno il collegamento alla porta locale e la connessione verrà inoltrata automaticamente all'host remoto tramite un canale sicuro.

Segue una spiegazione delle differenze tra **local port forwarding** e **remote port forwarding**.

- **Local:** `ssh -L` specifica che il traffico sulla porta indicata della macchina locale deve essere reindirizzato verso la porta indicata della macchina remota.
e.g. `ssh -L sourcePort:forwardToHost:onPort connectToHost` significa: connettiti via ssh a `connectToHost`, e inoltra tutti i tentativi di connessione che arrivano alla porta locale `sourcePort` verso la porta `onPort` della macchina chiamata `forwardToHost`, che si raggiunge tramite la macchina chiamata a sua volta `connectToHost`.



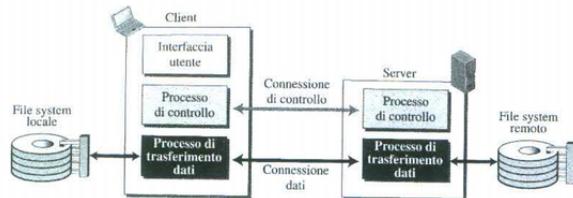
- **Remote:** ssh -R specifica che il traffico sulla porta indicata della macchina remota deve essere reindirizzato verso la porta indicata della macchina locale.
e.g. ssh -R sourcePort:forwardToHost:onPort connectToHost significa: connettiti via ssh a **connectToHost**, e inoltra tutti i tentativi di connessione che arrivano alla porta remota **sourcePort** verso la porta **onPort** della macchina **forwardToHost**, che si raggiunge tramite la tua macchina locale.



N.B. SSH è un protocollo STATEFUL.

3.1.5 FTP

FTP acronimo per **F**ile **T**ransfer **P**rotocol, è un **protocollo** per il **trasferimento** di **dati** tra due host di una rete, è lo **standard** per il **trasferimento** di file **offerto da TCP/IP**. Adotta il modello **client-server**: il **client** richiede il trasferimento di un file che può consistere sia nell'**acquisizione** di una copia locale modificabile sia nell'eventuale **trasferimento** della copia modificata sull'host remoto (**server**).



Il **client** ha **tre** componenti: **interfaccia utente**, **processo di controllo** e **processo di trasferimento dati** mentre il **server** remoto ne ha solo **due**: **processo di controllo** e **processo di trasferimento dati**. La **separazione** del trasferimento dei dati da quello dei comandi rende il protocollo FTP **efficiente**. Infatti la **connessione di controllo** usa **regole semplici** così da ridurre lo scambio delle informazioni a **una riga di comando** e **una di risposta per ogni interazione**. Mentre la **connessione dati** usa **regole più complicate** a causa della varietà delle informazioni che vi transitano. **FTP** offre funzionalità **aggiuntive** oltre al semplice trasferimento di dati, infatti **mette a disposizione**:

- **Accesso interattivo:** l'utente può **navigare**, **cambiare** e **modificare** l'albero di directory nel file system dell'host remoto.
- **Specificazione del formato dei dati da trasferire** (e.g. file di testo o file binari)
- **Autenticazione:** il client può **autenticarsi** con username e password.

Poco sopra abbiamo accennato al fatto che **FTP** prevede l'instaurazione di **due tipi di connessione tra il client e il server** ovvero:

1. **Control connection:** prevede uno scambio di comandi e **codici di ritorno** e.g. **200 OK**, tra client e server. Segue il protocollo TELNET e rimane aperta per l'intera durata della sessione interattiva. Si usa la **porta 21** del **server** e la codifica standard **NVT ASCII**.
2. **Data connection:** prevede il trasferimento di dati mediante procedure e la **specifica dei tipi**. I dati trasferiti possono essere parte di un file, un file o un set di file. **Viene aperta e chiusa per ogni singolo scambio**. Per lo scambio di dati il **server** usa la **porta 20**.

La **Data Connection** non segue il protocollo TELNET e la sua apertura avviene secondo uno schema completamente diverso, ovvero:

1. Il **client**, non il server, effettua un'**apertura passiva** usando una **porta effimera** e resta in attesa di connessione, viene fatto dal client poichè è tale processo che invierà i comandi, tramite la connessione di controllo, per il trasferimento dei file.
2. Il client **invia** questo **numero di porta** al **server** per mezzo del comando PORT.
3. Il **server**, ricevuto il numero di porta, **effettua un'apertura attiva**, ovvero apre la connessione, usando la propria **porta** nota **20** e quella effimera offerta dal client.

Per effettuare il trasferimento dei file inoltre, il client deve **definire il tipo** di file, la **struttura dati** e la **modalità di trasmissione** al fine di risolvere i problemi di eterogeneità tra client e server, va infatti ricordato che programma client e programma server sono su macchine **diverse**. Questo scambio di informazioni avviene mediante la **connessione di controllo**.

FTP è quindi un protocollo **STATEFUL** poichè il **server deve tener traccia dello stato dell'utente**: bisogna tenere conto infatti, tra le altre cose, della directory del file system remoto in cui si trova l'utente!

Concludiamo menzionando il fatto che esistono server che supportano connessioni FTP **senza autenticazione** (**Anonymous FTP**). Tipicamente consentono di accedere **solo** ad una parte del file system e permettono **solo** un subset di operazioni (e.g. la PUT **non** è permessa). Di solito **si usa** un **username comune** (solitamente "ftp" or "anonymous") e una **password qualsiasi** (e.g. l'indirizzo email dell'utente). *Erano usate per distribuire a un pubblico dei file senza dover generare numerosi username e password.*

Comandi di controllo	Significato
USER username	username d'autenticazione
PASS password	password d'autenticazione
LIST dirname	elenca i file della directory corrente
NLST dirname	richiede elenco file e directory (ls)
RETR filename	recupera (get) un file dalla directory corrente
STOR filename	memorizza (put) un file nell'host remoto
ABOR	interrompe ultimo comando e trasferimenti in corso
PORT portnumber	indirizzo e numero di porta del client
SYST	il server restituisce il tipo di sistema
QUIT	(quit) chiude la connessione

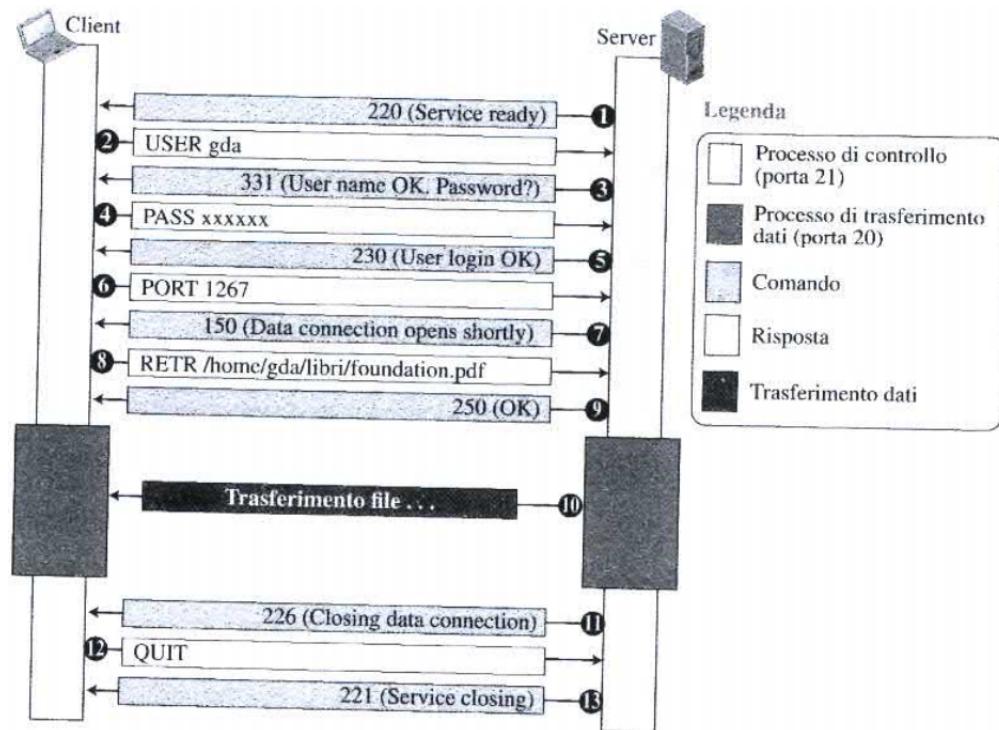


Figura 3.24: Esempio di trasferimento di un file, notare la connessione dati.

Ogni comando FTP **genera** almeno **una risposta**, le risposte sono composte da due parti: **un numero di tre cifre e un testo**.

Risposte FTP	Descrizione
125	connessione dati aperta
150	stato del file OK
200	comando OK
220	servizio pronto
221	servizio in chiusura
225	connessione dati aperta
226	connessione dati in chiusura
230	login dell'utente OK
250	azione sul file OK
331	nome utente OK, attesa PW
425	non è possibile aprire la connessione dati
450	azione sul file non eseguita, file non disponibile
452	azione interrotta, spazio insufficiente
500	errore di sintassi, comando non riconosciuto
501	errore di sintassi nei parametri o negli argomenti
530	login dell'utente fallito

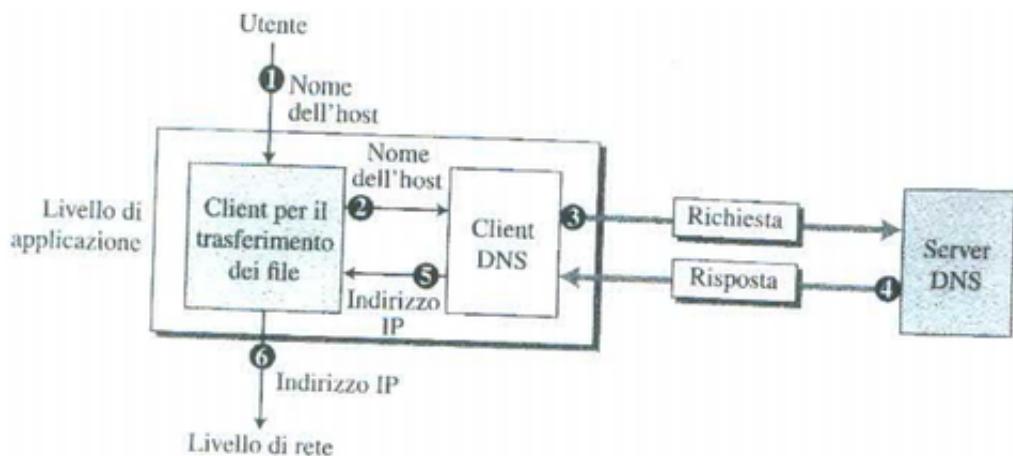
3.1.6 DNS

Sappiamo oramai che i dispositivi connessi in rete vengono individuati dai protocolli TCP/IP mediante i loro **indirizzi IP**, gli utenti, d'altro canto, preferiscono usare dei **nomi** invece che degli indirizzi numerici. Un **nome** identifica un **oggetto** mentre un **indirizzo** specifica **dove** l'oggetto è situato.

Come fare per associare i nomi agli indirizzi?

Agli albori di Internet l'associazione tra nomi logici e indirizzi IP era statica! Tutti i nomi *logici* e i relativi indirizzi IP erano contenuti in un file chiamato **host file** e periodicamente tutti gli host ne prelevavano una versione aggiornata, chiamata a sua volta **master host file**, da un server ufficiale. A noi utenti moderni però dovrebbe immediatamente saltare all'occhio la seguente **problematica**: le **dimensioni attuali di Internet** rendono questo approccio **impraticabile**. Non sarebbe infatti possibile che ogni host possegga una **copia aggiornata** di un elenco del genere, in più la dimensione di questo sarebbe sterminata, per non parlare del **volume di traffico** sul server ufficiale, il pericolo dovuto alla presenza di un **unico punto di fallimento** e l'**impossibilità di scalare** di questo sistema.

Fu così che all'inizio degli anni '80 venne ideato il **DNS** o **Domain Name System**. L'**idea centrale** è di **suddividere** la sconfinata mole di associazioni nome-indirizzo e **distribuirne** le varie parti su **calcolatori sparsi per il mondo**. *Come funziona? Così:*



Supponiamo che un utente utilizzi un client di trasferimento file per accedere a un file su un server. L'utente conoscerà solo il **nome** del server e.g. *cheneso.com*, lo **stack TCP/IP** invece ha bisogno dell'**indirizzo IP** del server per stabilire una connessione. Alla pagina seguente troviamo i sei passi necessari per **associare l'indirizzo IP** al **nome** del server.

1. L'**utente** comunica il nome del server al client di trasferimento file.
2. Il **client di trasferimento file** trasmette il nome del server al **client DNS**.
3. Ogni computer una volta avviato conosce l'indirizzo IP di un **server DNS**, il **client DNS** invia dunque, usando la **porta 53**, al **server DNS** la richiesta di traduzione del nome simbolico del server.
4. Il **server DNS** risponde con l'indirizzo IP del server desiderato.
5. Il **client DNS** comunica al **client di trasferimento file** l'indirizzo IP del server.
6. Il **client di trasferimento file** utilizza così l'indirizzo IP ricevuto per accedere al server.

Per far sì che questo meccanismo funzioni bisogna **eliminare le ambiguità sui nomi** e per far ciò si è definito uno **spazio dei nomi**. Lo **spazio dei nomi** ha una **struttura gerarchica** per una ragione principale: se tutti i nomi fossero composti da una sola stringa alfanumerica servirebbe un'**autorità centrale** che controllasse l'unicità di ogni singolo nome. Una **struttura gerarchica** consente invece di avere nomi composti da diverse parti e di **delegare il controllo** su ciascuna parte a enti o società diverse, **decentralizzando** in tal modo il processo di controllo.

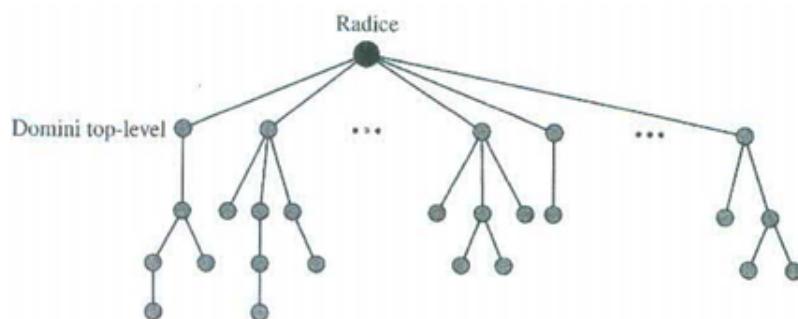


Figura 3.25: In figura lo spazio dei nomi di dominio.

Nello **spazio dei nomi di dominio** i nomi hanno una struttura ad **albero** con la radice in cima e un numero di livelli compreso tra 0 e 127. Ogni nodo è individuato da un'**etichetta** costituita da **massimo 63 caratteri** (*la radice ha l'etichetta vuota*), tutti i nodi collegati a uno stesso nodo da rami diversi hanno etichette **diverse**, ciò garantisce l'**univocità** dei nomi. Ogni nodo dell'albero ha inoltre un **nome di dominio**, che, se letto da sinistra verso destra, è costituito da tutte le etichette, separate da punti, di tutti i nodi a partire dal nodo stesso fino alla radice, la cui etichetta è la stringa **nulla**. Immediatamente sotto la radice si trovano i **domini top-level**.

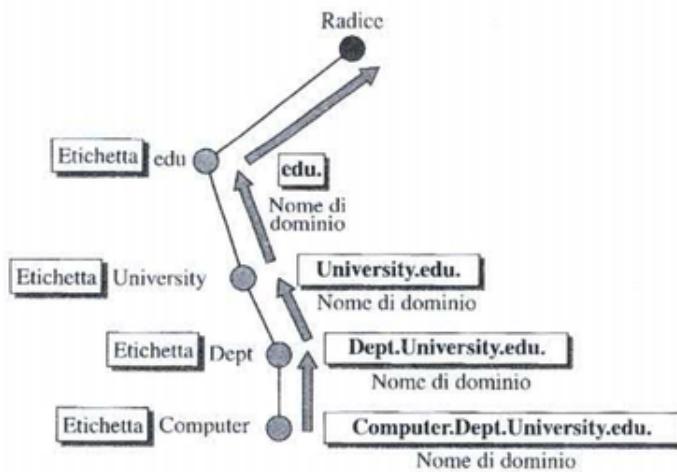


Figura 3.26: In figura le etichette e i nomi di dominio.

Definizione 3.1.8. Un **dominio** è un **sottoalbero** dello spazio dei nomi che viene identificato dal **nome di dominio** della sua radice.

Ma dove sono contenute le informazioni relative allo spazio dei nomi di dominio?

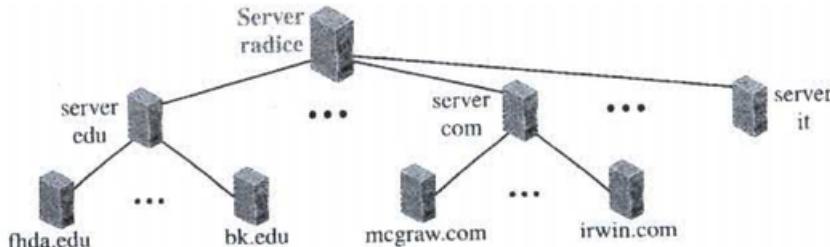


Figura 3.27: In figura la gerarchia dei name server.

All'interno dei **DNS-Servers** o **name servers**. L'intero spazio dei nomi è stato **diviso** in diversi domini, differenziati al **top-level**, e i domini così ottenuti sono stati divisi a loro volta ottenendo dei sottodomini. Ogni **name server** è responsabile di un dominio o di un sottodominio. Vi è quindi una **gerarchia di server**. Una **zona** è tutto ciò di cui è responsabile un **server**, se un server è responsabile di un dominio e non effettua suddivisioni in sottodomini, allora la sua zona e il suo dominio **coincidono**. Se invece il server suddivide il proprio dominio in sottodomini la sua zona e il suo dominio **differiscono**. Ogni name-server ha un database chiamato **file di zona** contenente le informazioni su tutti i nodi che ricadono nella **sua zona** di competenza.

Un **root server** è un server che ha per zona l'intero albero dello spazio dei nomi.

Soltanmente i **root server** si limitano a immagazzinare riferimenti relativi ad altri server e **delegano** loro tutte le responsabilità.

Esempio di top-level domains:

Dominio	Uso
com	organizzazioni commerciali
edu	istituti di istruzione
mil	gruppi militari
gov	istituzioni governative americane
net	principali centri di supporto alla rete
org	organizzazioni diverse dalle precedenti
it, uk, us, fr, ecc.	codice geografico per nazioni

I server DNS possono essere **primari** o **secondari**.

- Un server **primario** possiede sul disco e aggiorna il **file di zona** relativo alla zona sotto la sua responsabilità.
- Un server **secondario** riceve le informazioni relative a una zona da un **server primario**.

I server **primari** e **secondari** hanno la medesima autorità sulla loro zona di competenza, è bene specificare che **un server può essere primario per una zona e secondario per un'altra**, **primario** e **secondario** sono quindi aggettivi **relativi alla zona**. Si noti inoltre come l'introduzione di un server secondario in una zona porti a una duplicazione del **file di zona** che può risultare utile per eventuali guasti al **server primario**.

Risoluzione

Il processo con cui si associa l'indirizzo IP da un nome è detto **processo di risoluzione**. Il **protocollo DNS** è progettato come protocollo **client-server**. Un host che voglia ricavare un indirizzo IP da un nome si rivolge al **programma client** detto anche **resolver** che invierà un'opportuna richiesta al **server DNS più vicino** il quale, se dispone della risposta, invierà l'indirizzo o il nome (*è infatti possibile fare il processo al contrario*), oppure inoltrerà la richiesta a un altro server o comunicherà al resolver l'indirizzo di un altro server a cui fare riferimento. Il **resolver**, ricevuta la risposta, la esaminerà per verificarne la presenza di errori e la trasmetterà al processo che l'ha effettuata. La **risoluzione** può essere **ricorsiva** o **iterativa**.

- Con la **risoluzione ricorsiva**, la **query DNS** viaggia dall'host su cui è in esecuzione il processo applicativo che ne ha fatto richiesta fino a un host che conosce l'indirizzo IP richiesto, eventualmente scalando, poi discendendo e infine percorrendo a all'indietro l'**intera gerarchia dei server DNS**. Al server locale che ha richiesto la query viene inviato solo l'indirizzo IP risultante, ha richiesto quindi una **conversione completa**.

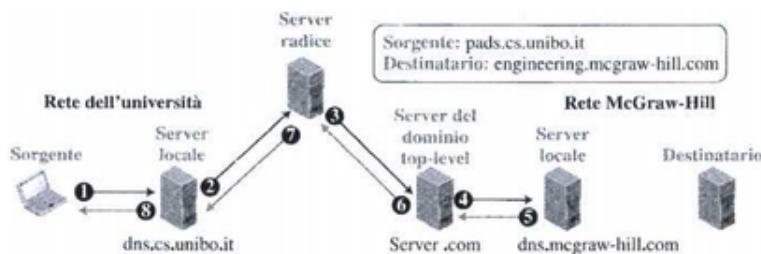


Figura 3.28: In figura un esempio di risoluzione ricorsiva.

- Con la **risoluzione iterativa** ogni **server-DNS** che non è in grado di risolvere la **query-DNS** dell'host risponde, direttamente al **server locale**, con l'indirizzo di un altro **server** in grado di risolverla. Al server locale che ha richiesto la query non viene quindi inviato solo l'indirizzo IP risultante. La **risoluzione iterativa** è supportata da **tutti i name server**, l'host può richiedere che venga usata la **risoluzione ricorsiva** ma potrebbe non essere disponibile.

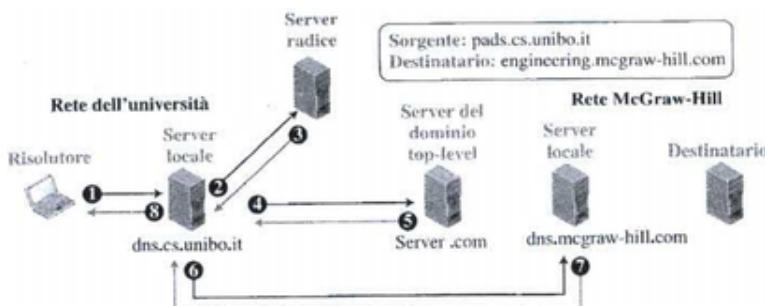


Figura 3.29: In figura un esempio di risoluzione iterativa.

Ogni volta che a un **server** arriva una richiesta di risoluzione di un nome, che non fa parte del suo dominio, deve cercare, all'interno del suo database, un altro server a cui inoltrare la richiesta. Ridurre questo tempo di ricerca significa ridurre il tempo di attesa della risposta e aumentare l'efficienza. Ancora una volta ci viene in aiuto il **caching**: una volta che un **server** ha appreso un' **associazione**, la inserisce in **cache**. Il server marca le risposte prese dalla propria cache come **unauthoritative** o non autorevoli.

Dopo un certo lasso di tempo chiamato **TTL** o **time to live** il server cancella dalla cache l'associazione per evitare l'invio di risposte obsolete.

Record DNS

Il **database di un server DNS** non è altro che una collezione di **records** strutturati nel seguente modo:

(Nome di dominio, Tipo, Classe, TTL, Valore)

A ogni **nome di dominio**, quindi a ogni **nodo dell'albero dello spazio dei nomi di dominio**, è associato un record composto da 5 campi.

- **Nome di dominio** identifica il record della risorsa.
- **Tipo** definisce come interpretare il campo **Valore**.
- **Classe** definisce il tipo di rete, IN sta per Internet.
- **TTL** indica il numero di secondi per cui l'informazione deve essere ritenuta valida
- **Valore** contiene l'informazione memorizzata relativa al **nome di dominio**.

Di seguito i **tipi** dei **record**:

Tipo	Interpretazione
A	indirizzo Ipv4 a 32 bit
NS	identifica i server autoritativi di una zona
CNAME	il nome di dominio è un alias per quello ufficiale
SOA	informazioni autoritative riguardanti una zona
MX	server di posta del dominio corrente
AAAA	indirizzo Ipv6

Che protocollo di livello trasporto è usato?

Il sistema DNS può usare sia il **protocollo TCP** che l'**UDP**. Il **protocollo UDP** viene usato quando la dimensione del messaggio di risposta è **inferiore a 512 byte**, molto spesso infatti i **datagrammi utente UDP** non possono superare i 512 byte come dimensione massima. In caso contrario si usa il **protocollo TCP**.

Come si aggiungono nuovi domini al DNS?

Pagando i **registrar**, ovvero aziende commerciali accreditate dall'ICANN.

Messaggi DNS

I messaggi DNS posso essere di **due tipi**: di **interrogazione** o di **risposta** e i due tipi hanno lo **stesso** formato.

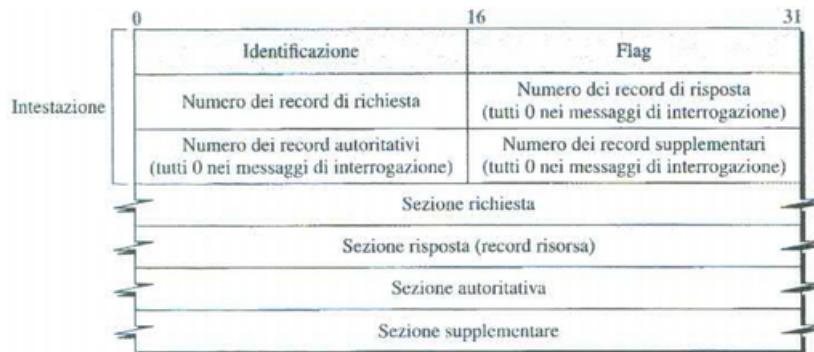


Figura 3.30: La struttura dei messaggi DNS, il messaggio interrogazione contiene solo la sezione richiesta mentre il messaggio di risposta contiene la sezione richiesta, la sezione risposta ed eventualmente le altre due.

Descriviamo ora brevemente i vari campi situati prima dell'**intestazione** lunga **12 bytes**:

- **Identificazione** è il campo usato dal **client** per associare la risposta all'interrogazione.
- Il campo **Flag** indica se si tratta di un messaggio di richiesta o di risposta e segnala inoltre la presenza di eventuali errori.
- I **quattro** campi successivi dell'**intestazione** specificano il numero di ciascun tipo di record presente nel messaggio.

Poi del **messaggio**:

- La **sezione di richiesta** che è inclusa nell'interrogazione e poi ripetuta anche nel messaggio di risposta è **formata** da **uno o più record di richiesta**.
- La **sezione di risposta** è presente **esclusivamente** nei messaggi di risposta consiste in **uno o più record di risorsa**
- La **sezione autoritativa** fornisce le informazioni di uno o più server autorevoli per l'interrogazione.
- La **sezione supplementare** contiene **informazioni aggiuntive** che potrebbero essere **utili** al **client DNS**.

In UNIX si può utilizzare il comando nslookup per ottenere associazioni nome simbolico : indirizzo numerico.

3.1.7 EMAIL - SMTP

La **posta elettronica** è uno dei primi servizi applicativi di Internet, la sua nascita risale infatti al 1971, quando un tale Ray Tomlinson installò su ARPANET un sistema in grado di scambiare messaggi fra le varie università. La **posta elettronica** consente agli utenti di scambiarsi **messaggi**, sebbene si siano trattate **altre applicazioni** fornenti questo servizio e.g. **HTTP** e **TCP**, il funzionamento della posta elettronica è tale da distinguersi da queste due applicazioni citate poco prima. Infatti in applicazioni come **FTP** e **HTTP** il programma **server** è **sempre attivo** e in attesa di richieste, che quando arrivano, vengono processate. Vi è quindi una richiesta e una risposta. Nella **posta elettronica** il **funzionamento** è **differente**: infatti l'invio di un messaggio è considerato una **transazione unidirezionale**, la risposta può anche non arrivare e se arriva è considerata un'altra transazione unidirezionale. Secondariamente **non avrebbe molto senso tenere in esecuzione continuata un programma server** in attesa che qualcuno ci invii un messaggio, potremmo ragionevolmente voler spegnere il computer nell'attesa. Ciò implica che l'idea di modello client-server debba essere realizzata in un altro modo, magari utilizzando dei **server intermedi** e **di saccoppiando** le funzionalità di **ricezione** da quelle di **invio**. Di seguito illustriamo concisamente l'architettura della posta elettronica.

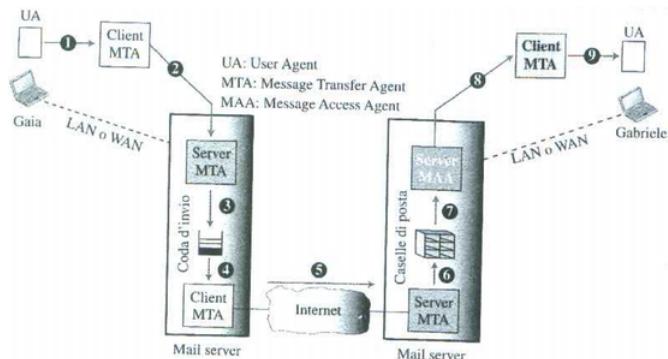


Figura 3.31: Un esempio di architettura di posta elettronica.

Tipicamente due utenti, mittente e destinatario, sono connessi a due **server di posta**. Ogni utente possiede una **mailbox** sul server, ovvero una porzione sulla memoria del server a cui solo lui può accedere e ogni server ha una **coda di invio**, o **spool**, dove memorizza i messaggi in attesa di essere inviati. Più:

- **UA o user agent**, prepara il messaggio e lo invia al server.
- **MTA o message trasfer agent**, è un'applicazione **push**.
- **MAA o message access agent**, è un'applicazione **pull**.

Illustriamo ora **come avviene l'invio di un messaggio** da parte di *Gaia* a *Gabriele*.

1. Gaia, utilizzando un programma **UA**, prepara il messaggio e lo invia al proprio server di posta.
2. Il **server di posta memorizza** il messaggio in una coda, e lo **invia** tramite un programma **MTA** al server di posta di Gabriele. **N.B. sono necessari due programmi MTA per ogni server di posta, un MTA server sempre attivo, in attesa di messaggi, e un MTA client, attivo all'evenienza, che contatta il server di posta a cui deve essere inviato un messaggio ed effettua l'invio.**
3. Il server di posta **riceverà** il messaggio e lo **memorizzerà** nella **casella postale di Gabriele** che, a sua volta usando un programma **MAA client**, contatterà il programma **MAA server** del **server** e riceverà infine il messaggio.

Lo **user agent** è il primo componente di un sistema di posta elettronica, **facilita all'utente l'invio e la ricezione dei messaggi**. Può avere o un'interfaccia a **riga di comando**, ma è abbastanza desueto, o di tipo **grafico**. Se l'utente decide di leggere i messaggi nella sua casella di posta elettronica lo **user agent** mostra un elenco dei messaggi ricevuti. Ogni riga dell'elenco offre un breve resoconto del messaggio, solitamente: **indirizzo del mittente, ora e oggetto del messaggio**.

Indirizzi di posta elettronica

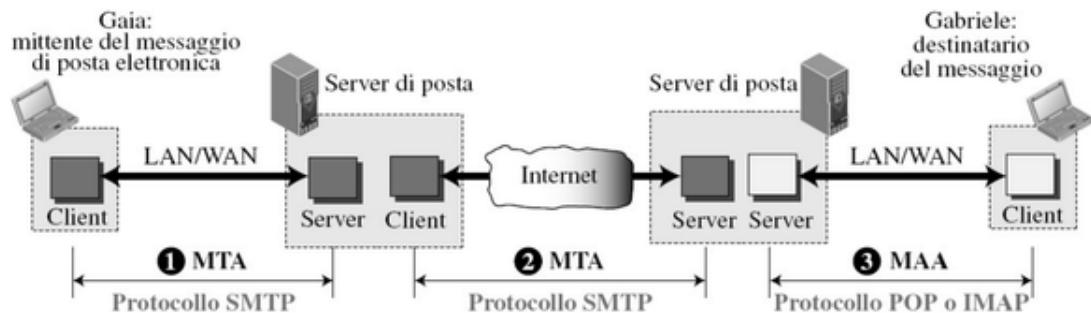
Gli indirizzi di posta elettronica individuano gli utenti in modo **univoco**. Su **Internet** consistono di **due parti**:



La **parte locale** identifica la **casella di posta del destinatario** sul server e il nome di dominio identifica il **server**.

Invio di messaggi

Con il protocollo **SMTP** o **simple mail transfer protocol**. Il protocollo **SMTP** definisce in maniera formale l'**interazione tra il client MTA e il server MTA**. Nell'operazione di invio di un messaggio SMTP è utilizzato **due volte**: tra il **mittente** e il **suo server di posta** e tra il **server di posta del mittente** e **quello del destinatario**. Di seguito un'immagine per schiarire le idee.



Il **protocollo SMTP** definisce come deve avvenire l'interazione, per mezzo di **comandi e risposte**, tra **client e server MTA**. I comandi sono inviati dal **client MTA** al **server MTA** e viceversa le risposte. I **comandi** e le **risposte** terminano tutti con la medesima **coppia di caratteri**: ritorno a capo e fine linea. La connessione tra client MTA e server MTA è una sola ed è bidirezionale.

- I **comandi** sono composti da una **keyword** e da **uno o più argomenti**.

Nome	Argomenti	Significato
HELO	nome host mittente	host mittente si identifica
MAIL FROM	mittente del messaggio	identifica mittente messaggio
RCPT TO	destinatario	identifica destinatario messaggio
DATA	corpo del messaggio	il messaggio
QUIT		termina sessione SMTP corrente
RSET		interrompe la transazione in atto
VRFY	nome destinatario	verifica validità nome destinatario

- Le **risposte** sono costituite da un **codice a tre cifre** seguite eventualmente da testo.

Codice	Descrizione
220	servizio pronto
221	servizio in chiusura canale trasmissione
250	comando richiesto completato
354	corpo del messaggio
421	servizio non disponibile
450	mailbox non disponibile
502	comando non disponibile
550	user unknown

N.B. Il protocollo **SMTP** usa il protocollo di trasporto **TCP** per consegnare in maniera **affidabile** i messaggi.

La consegna di un messaggio tramite SMTP prevede **tre fasi**:

1. **Apertura della connessione** o **handshaking** con cui il client SMTP stabilisce una connessione TCP alla porta nota 25 con il server SMTP. Consiste di altre **tre sotto fasi**:
 - 1.1. Il **server** invia al **client** il codice **220** per indicare che è **pronto alla ricezione di messaggi** o il codice **421** in caso **contrario**.
 - 1.2. Il **client** si **identifica** con il comando **HELO** seguito dal suo **nome di dominio** in modo tale da informare il server del proprio nome di dominio.
 - 1.3. Il **server** invia il codice **250** o altri codici a seconda della situazione particolare.
2. **Invio del messaggio**: se l'apertura della connessione tra il client SMTP e il server SMTP è avvenuta con **successo**, il client può inviare **un singolo messaggio** a uno o più destinatari. Riportiamo di seguito gli **otto** passi necessari per portare a termine questa operazione:
 - 2.1. Il **client** invia al server il comando MAIL FROM con argomento l'indirizzo mail del mittente, in modo tale che, nel caso in cui si verifichino degli errori, il server **sappia a chi inviare i messaggi di errore**.
 - 2.2. Il **server** risponde con il codice 250.
 - 2.3. Il **client** invia al server il comando RCPT TO con argomento l'indirizzo mail del destinatario
 - 2.4. Il server risponde con il codice 250.
 - 2.5. Il **client** invia il comando DATA per iniziare il trasferimento del messaggio.
 - 2.6. Il **server** risponde con il codice 354.
 - 2.7. Il **client** invia il messaggio come sequenza di righe, ognuna terminante con la coppia di caratteri ritorno a capo e fine linea. Il messaggio termina con una riga **contenente solo un punto**.
 - 2.8. Il server risponde con il codice 250.
- Se ci sono **più destinatari** i passi 2.3 e 2.4 sono **ripetuti**.
3. **Chiusura della connessione**: Il **client**, trasferito il messaggio, **chiude la connessione**. Questa operazione avviene in **due fasi**:
 - 3.1 Il **client** invia al server il comando QUIT.
 - 3.2 Il **server** risponde con il codice 221.

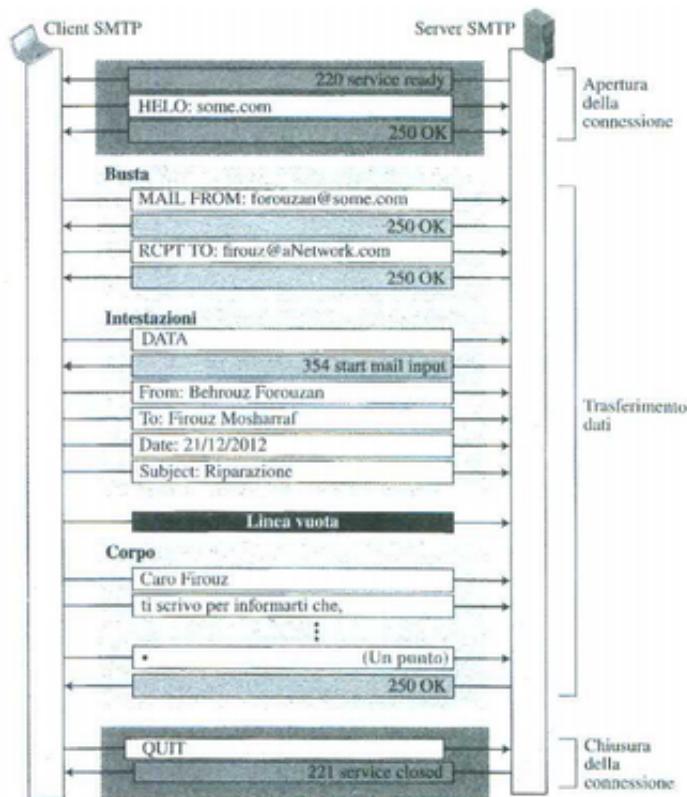


Figura 3.32: In figura l'invio di un messaggio. L'RFC 2822 definisce lo standard per il formato del messaggio. **I campi sono diversi dai comandi SMTP.** Il corpo del messaggio contiene solamente caratteri ASCII a 7 bit.

Ricezione dei messaggi

Poco sopra abbiamo descritto l'invio di un messaggio di posta elettronica come la **composizione di tre fasi**: una di **preparazione** del messaggio, una di **invio** e una di **ricezione**. Le prime due utilizzano i **protocollo SMTP**, che è un protocollo **push**, il messaggio viene *spinto* dal client mittente verso il server. L'ultima fase, quella di ricezione, usa invece un protocollo **pull**, il client destinatario *tira* i messaggi dal server. Attualmente sono in uso **due** protocolli di tipo **pull**: **POP3 o Post Office Protocol v.3** e **IMAP4 o Internet Mail Access Protocol v.4**. **POP3** è molto semplice ma ha funzionalità limitate. Il software **client POP3** è installato sul computer del destinatario mentre il software **server POP3** sul suo server di posta. Il **client POP3** apre una connessione TCP sulla porta **110** del **server POP3** e invia il proprio nome utente e la propria password per accedere alla casella postale. L'utente richiede poi la lista dei messaggi presenti e li preleva una alla volta.

POP3

Il protocollo **POP3** prevede due modalità: **delete** e **keep**, con la modalità **delete** i messaggi vengono automaticamente eliminati dalla mailbox dopo il prelievo, con la modalità **keep** vengono tenuti per uso futuro.

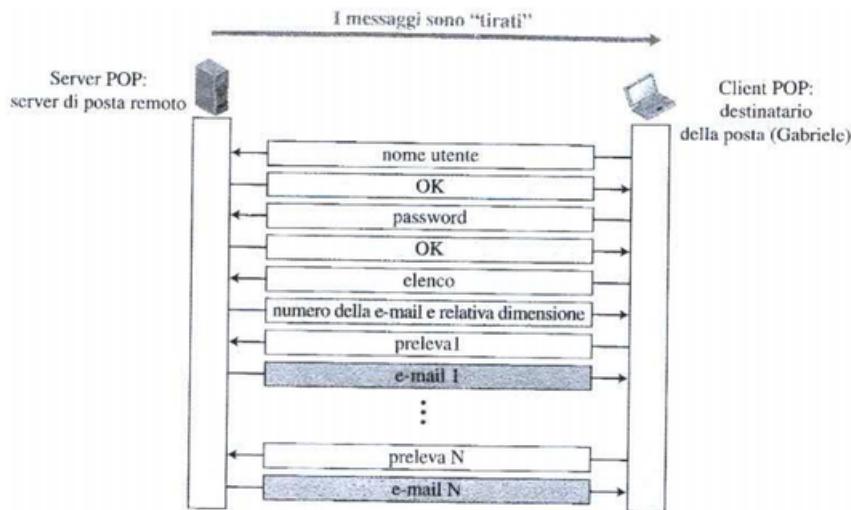


Figura 3.33: In figura il protocollo POP3.

IMAP4

Il protocollo **IMAP4** è simile al POP3 ma è più **potente** e **complesso**. POP3 non consente di **gestire più caselle** di posta sul server, di **organizzare** la posta e di **controllare una parte del messaggio** prima di **prelevarlo** nella sua interezza. IMAP4 invece fornisce varie funzionalità aggiuntive, tra le quali:

- **Controllare le intestazioni** dei messaggi prima di **prelevarli**.
- **Ricercare una stringa** specifica **nei messaggi** prima di **prelevarli**.
- **Prelevare i messaggi in modo parziale**. Utile per quando ci sono limitazioni di larghezza di banda.
- **Creare, cancellare e rinominare** le **mailbox** sul server di posta.
- **Creare una gerarchia di cartelle** all'interno della mailbox a scopo di archiviazione.

I protocolli visti fin'ora supportano **soltanto** messaggi nel formato standard **NVT ASCII a 7 bit**.

MIME

Il **MIME** o **Multipurpose External Mail Extension** è un protocollo **supplementare** che permette l'invio di messaggi in formato **diverso** dall'**ASCII**. Il **MIME** agisce **sia** dal lato **mittente** **che** dal lato **destinatario**. Traduce tutti i dati in formato **non ASCII** in **ASCII** **prima** di inviare il messaggio al MTA e opera poi a ritroso quest'operazione di traduzione **presso** il destinatario dopo che ha ricevuto il messaggio tramite il MAA.

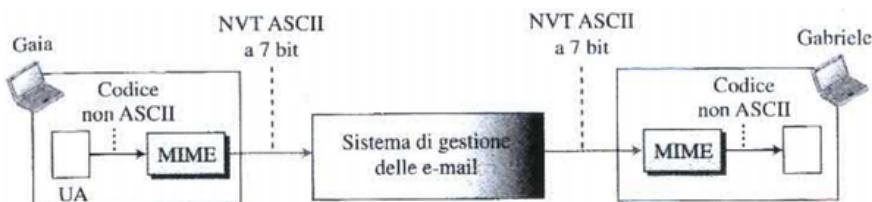


Figura 3.34: In figura il funzionamento del protocollo MIME.

Il **protocollo MIME** definisce **cinque** tipi di intestazioni specifiche che si aggiungono a quelle originali previste dal protocollo di posta elettronica:

- **MIME - version:** vi è dichiarata la versione di MIME usata.
- **Content Type:** vi sono dichiarati i tipi di dato contenuti nel corpo del messaggio. Il tipo del contenuto e il sottotipo sono separati da una barra. Il protocollo MIME usa sette tipi di dato diversi:
 - **Text:** plain o html, formato del testo.
 - **Multipart:** mixed, parallel o alternative, dà informazioni circa le parti da cui è composto il messaggio.
 - **Message:** RFC 882, partial, external-body, indicano rispettivamente: se il messaggio è incapsulato, se è una parte di un altro messaggio o se è un riferimento ad un altro messaggio.
 - **Image:** jpeg o gif.
 - **Video:** mpeg.
 - **Audio:** basic.
 - **Application:** PostScript o octet-stream.
- **Content-Transfer-Encoding:** definisce la codifica utilizzata per il messaggio.
- **Content-ID:** individua univocamente una parte nei messaggi che sono composti da più parti.
- **Content-Description:** indica se il corpo del messaggio contiene un'immagine, un file audio o video.

Web mail

Concludiamo questa sezione offrendo una breve panoramica del servizio chiamato **web mail**. Data la larga diffusione della posta elettronica molti siti web ne offrono il servizio. I **webmail servers** a seconda del **client** interessato, che può essere **SMTP** o **HTTP**, ricevono ed evadono i messaggi tramite o il **protocollo HTTP** o il **protocollo SMTP** come illustrato nelle seguenti figure:

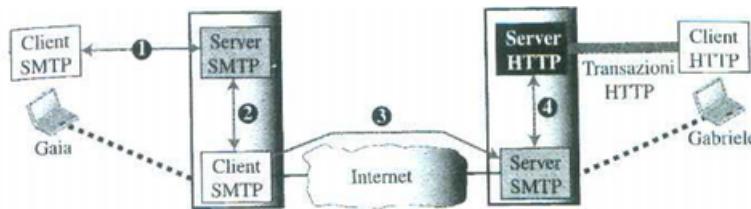


Figura 3.35: In figura uno scenario in cui **solo** il ricevente utilizza **HTTP**.

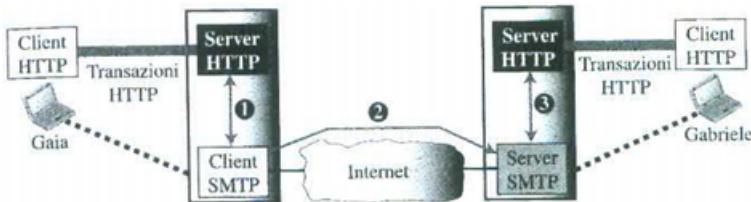


Figura 3.36: In figura uno scenario in cui **sia** che il ricevente che il destinatario utilizzano **HTTP**.

3.2 Lo strato di trasporto



Figura 3.37: Livelli dello stack protocollare TCP/IP.

Nello **stack protocollare TCP/IP** il **livello di trasporto** è posizionato tra il livello applicazione e il livello rete, **fornisce** servizi al **livello applicazione** e ne **riceve** dal **livello rete**. Realizza una connessione logica fra **processi applicativi** in esecuzione su host system diversi.

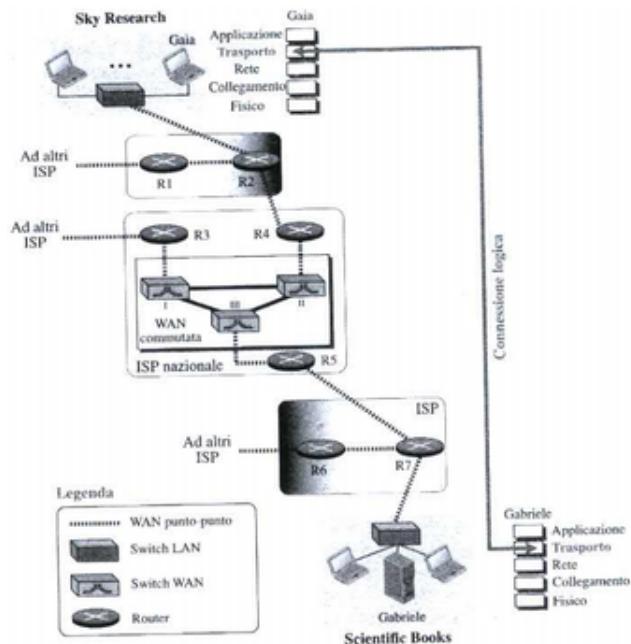


Figura 3.38: In figura una rappresentazione della connessione logica tra i livelli trasporto di due host.

Come anticipato poco prima, uno dei **servizi offerti** dal **livello di trasporto** è di **supportare la comunicazione** tra **processi applicativi**. Tuttavia le **informazioni, una volta inviate, prima** di arrivare ai singoli processi devono arrivare all'**host** su cui essi sono in esecuzione.

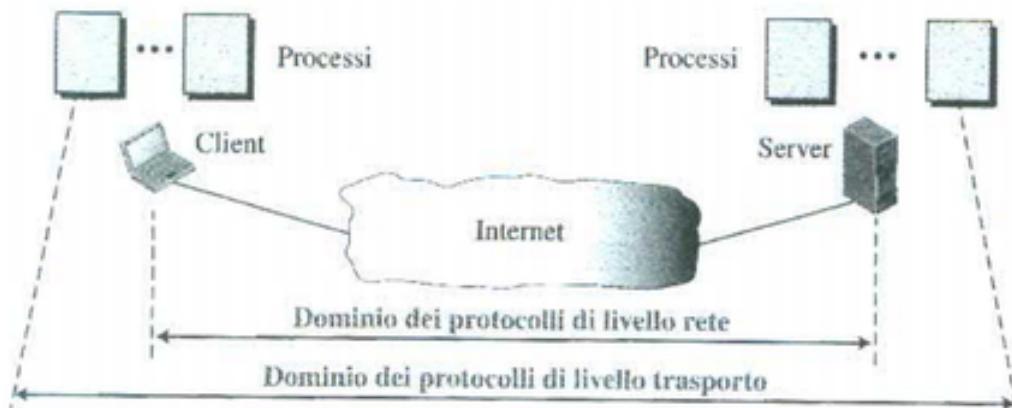


Figura 3.39: In figura un esempio di comunicazione tra dispositivi. Si notino i ruoli dei protocolli di rete e dei protocolli di trasporto.

I **protocolli di rete** si occupano di trasferire informazioni tra **macchine** e i **protocolli di trasporto** di indirizzarle ai **processi**.

Come indirizzare le informazioni verso i processi?

Come osservato nella sezione precedente, uno dei principali modelli di comunicazione interprocessuale è il **client-server**. Per garantire quindi la **comunicazione** tra il **processo client** e il **processo server** c'è bisogno di:

1. **Identificare l'host locale e l'host remoto.** Spesso infatti, se non quasi sempre, il processo client e il processo server sono in esecuzione su host **diversi**. Gli host, in **rete**, vengono identificati mediante il loro **indirizzo IP**.
2. **Identificare il processo.** I sistemi operativi moderni sono **multiutente** e **multiprocesso**. I processi necessitano quindi di un identificatore detto **numero di porta**. I protocolli TCP/IP usano numeri di porta compresi tra **0** e **65535**, l'intervallo è codificabile con **16 bit**.

Al **client** viene **assegnato** un **numero di porta** detto **effimero**, ovvero di breve durata, dato il **breve** tempo di vita di un processo client. I numeri di porta **effimeri** sono superiori al **1023**.

Anche al **server** verrà **associato** un **numero di porta** che dovrà però essere **noto** al **client**. Nei **protocolli TCP/IP**, per i server, si usano **numeri di porta universali** detti anche **numeri di porta noti**. Ogni **client applicativo** conosce il numero di porta noto del **server applicativo** corrispondente.

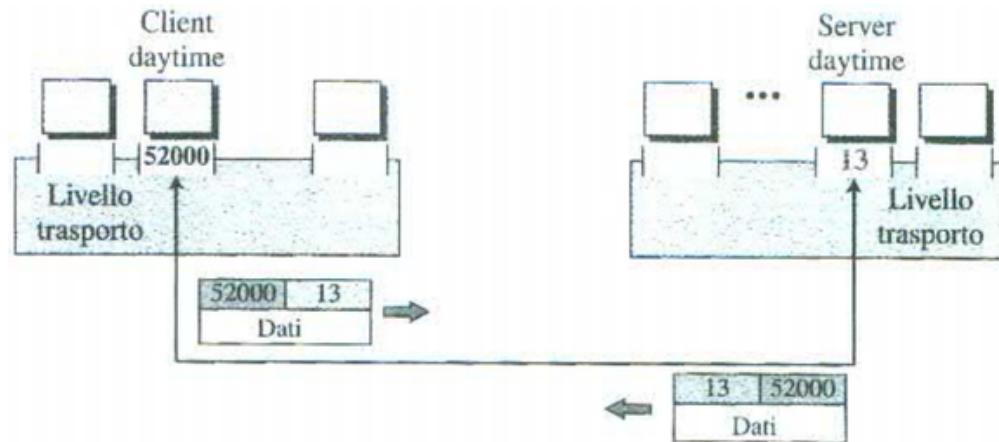


Figura 3.40: In figura i numeri di porta di un processo client e del corrispondente processo server.

Ricapitolando: l'**indirizzo IP** individua **un host** tra i miliardi di host dell'intera rete mondiale ed è un numero di 32 bit. Il **numero di porta** individua **un processo** in esecuzione su tale host.

L'ICANN, ovvero **Internet Corporation for Assigned Names and Numbers** ha suddiviso i numeri di porta in **tre** categorie:

1. **Numeri di porta noti**: sono compresi tra **0** e **1023** e sono **assegnati** dall'authority **ICANN**.
2. **Numeri di porta registrati**: sono compresi tra **1024** e **49151**, non sono **controllati né assegnati** dall'autorithy ICANN ma è **possibile registrarli** per evitare duplicazioni.
3. **Numeri di porta dinamici**: vanno da **49152** a **65535**, non sono **controllati né registrati** e possono essere usati come numeri di porta **privati o temporanei**.

In ambiente UNIX nel file /etc/services sono memorizzate le associazioni nome server : porta nota.

Definizione 3.2.1. La combinazione di **indirizzo IP** e **numero di porta** è detta **socket address**.

Quando un processo applicativo deve inviare un messaggio a un altro processo applicativo, lo passa al livello trasporto insieme a una **coppia** di **socket address**, uno che identifica il **processo stesso** e uno che identifica il **processo destinatario**. Il **protocollo di trasporto** riceve questi dati e vi aggiunge la propria **intestazione**, **incapsulando** il tutto in un **pacchetto**. Questi pacchetti sono chiamati **segmenti** nel **protocollo TCP** e **datagrammi utente** nel **protocollo UDP**. L'**incapsulamento** avviene dal lato del **mittente** e il **decapsulamento** avviene dal lato del **ricevente**.

Il metodo con cui un entità riceve informazioni da più di una sorgente è chiamato **multiplexing**, viceversa con **demultiplexing** si fa riferimento a un'entità che trasmette informazioni a più di un destinatario. Il **livello di trasporto** effettua **multiplexing** sul versante **mittente**, raccogliendo i messaggi da varie socket e incapsulandoli, e **demultiplexing** su quello del **destinatario**, consegnando i vari pacchetti in ingresso al socket appropriato.

Come garantire un controllo del flusso?

Il **livello di trasporto** supporta la comunicazione tra **processi applicativi**. Il **processo mittente produce informazioni** e le passa poi allo **strato di trasporto mittente** che le **consuma** e produce **pacchetti**. Lo **strato di trasporto destinatario** a sua volta **consuma pacchetti** e **produce informazioni** che verranno in seguito inviate al **processo destinatario**. Se i pacchetti vengono consumati con una velocità **inferiore** o **superiore** a quella con cui si è in grado di crearli o viceversa si va in contro a situazioni di **grave inefficienza** o addirittura di **perdita di dati**. Il controllo di flusso si occupa di **quest'ultima problematica**.

Una soluzione tipica per evitare la **perdita di dati** consiste nell'utilizzare **due buffer**. Uno dal **lato trasporto del destinatario** e uno dal **lato trasporto del mittente**. Se il **destinatario** ha il buffer **saturo** lo segnalerà al **mittente** che a sua volta **tratterrà** i messaggi nel **proprio** buffer e li rinvierà quando il **destinatario** segnalerà che il suo buffer non è più saturo.

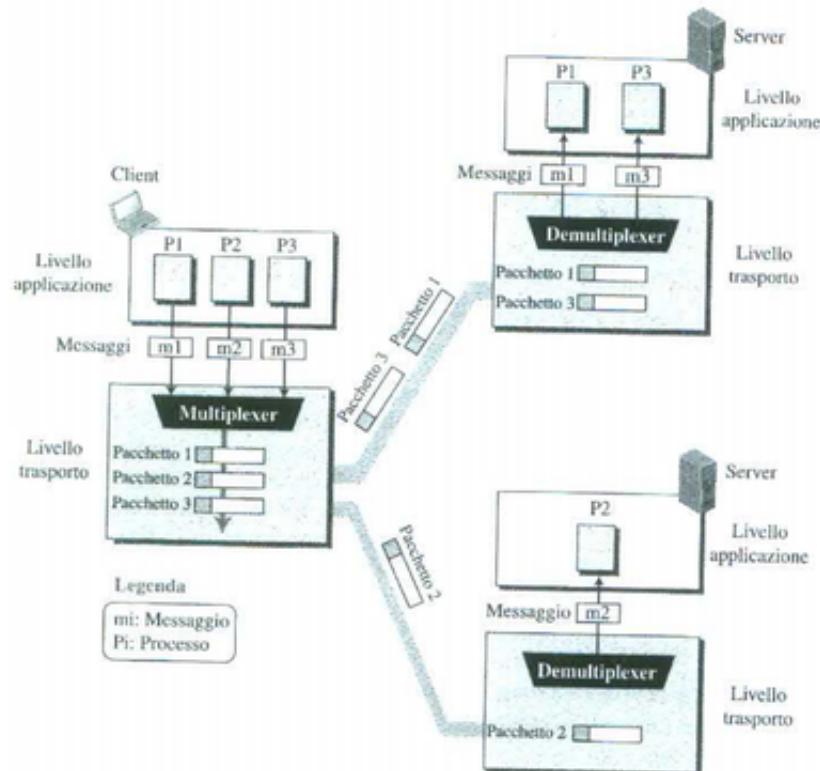


Figura 3.41: In figura multiplexing e demultiplexing.



Figura 3.42: In figura il controllo di flusso.

Come garantire l'affidabilità?

Aggiungendo i **servizi di controllo degli errori** al livello di trasporto. Il **livello di rete** infatti, come si vedrà più avanti, è **fallace**, ciò comporta che si debba **garantire l'affidabilità**, se richiesta dalle applicazioni, **al livello di trasporto**. Nel **controllo degli errori** sono coinvolti solo i **livelli trasporto del mittente e del destinatario**, in particolare, è quest'ultimo che gestisce il **controllo degli errori**:

- Rileva e scarta i pacchetti corrotti, notificando il problema al livello trasporto del mittente.
- Tiene traccia dei pacchetti persi e scartati e ne gestisce la **rispedizione**.
- Riconosce i pacchetti duplicati e li elimina.
- Bufferizza i pacchetti fuori sequenza finché non arrivano quelli mancanti.

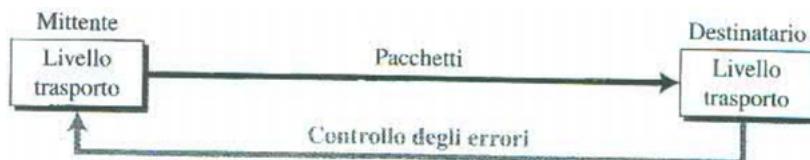


Figura 3.43: Il controllo degli errori.

Gli ultimi tre punti implicano che **il livello di trasporto del destinatario sappia riconoscere i pacchetti persi, duplicati o fuori sequenza**. Ciò è possibile grazie alla presenza di un campo, detto **numero di sequenza**, posto all'interno dei pacchetti. La **rispedizione** è gestita identificando il pacchetto mancante o corrotto mediante il suo numero di sequenza. I **pacchetti duplicati o mancanti** sono riconosciuti ordinando la sequenza di pacchetti entranti e verificando tutti i numeri di sequenza.

N.B. i numeri di sequenza vanno da 0 a $2^n - 1$, dove **n** è il numero di bit allocati per i numeri di sequenza specificato nell'intestazione del pacchetto. Se la sequenza di pacchetti è più lunga di 2^n pacchetti si ricomincia ad enumerarli da 0, di fatto i **numeri di sequenza** sono in **modulo** $2^n - 1$.

e.g. per n = 3 si ha la sequenza: 0 1 2 3 4 5 6 7 0 1 2 3 4 ...

Per **notificare** al mittente la corretta ricezione di uno o più pacchetti viene utilizzato il **numero di riscontro** o **ACK**. Il mittente identifica i pacchetti persi utilizzando un **timer** e **attivandolo dopo ogni invio**, se non riceve un **ACK prima della scadenza** allora rispedisce il pacchetto.

È possibile combinare controllo degli errori e controllo del flusso?

Sì, è **possibile**. Per farlo si usa una coppia di **buffer numerati**, combinando l'esigenza di avere due buffer a quella di numerare i pacchetti. Dal **lato mittente**, quando si **costruisce un pacchetto** per la spedizione, si usa come **numero di sequenza** la successiva posizione libera all'interno del buffer e lo si tiene in memoria finché non arriva l'**ACK** corrispondente. Dal **lato destinatario** invece, si tiene in memoria nel buffer il pacchetto ricevuto finché l'applicazione non è pronta a riceverlo, **solo** a quel punto viene inviato l'**ACK** e il pacchetto viene **eliminato**. L'intervallo dei numeri di sequenza: da 0 a $2^n - 1$, fa sì che possano essere rappresentati come un **cerchio** e il buffer, a seconda della sua **dimensione**, come una **sezione** di esso, chiamata anche **sliding window**. Quando all'interno del **buffer (sliding window)** si libera una sezione **contigua** di posizioni **a partire dall'inizio**, la **sliding window** scorre.

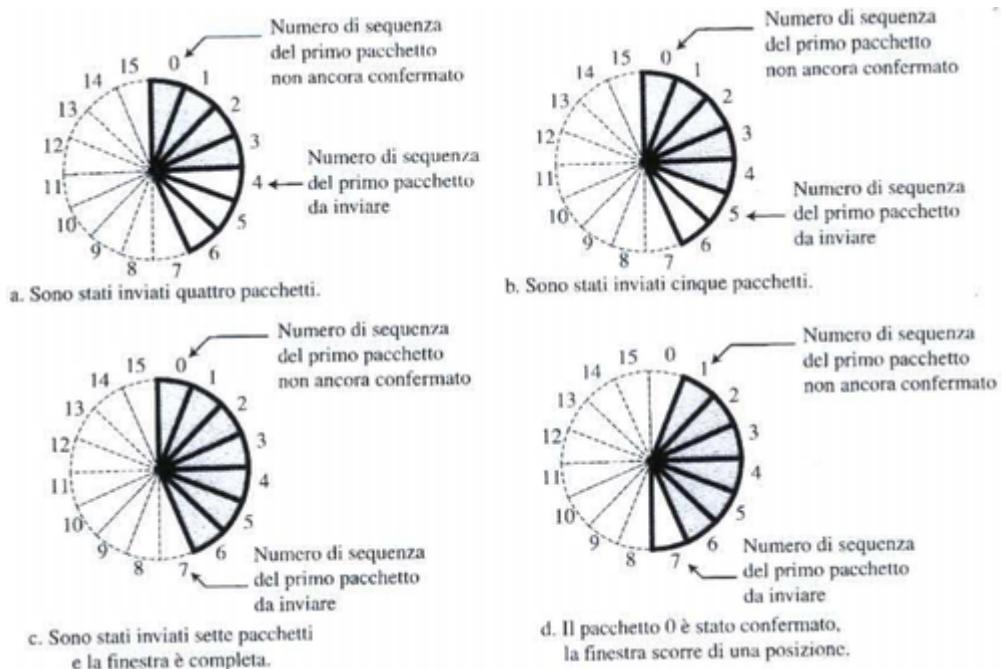


Figura 3.44: La sliding window. Si ricorda che è solo un'astrazione per far capire il concetto di fondo, in realtà si usano delle variabili per tenere traccia delle posizioni.

Esiste un controllo della congestione?

Sì, **esiste**, sebbene sia un **problema del livello rete**, il **protocollo TCP** implementa un **proprio** meccanismo di controllo della congestione.

Che differenza c'è tra un servizio privo di connessione e un servizio orientato alla connessione?

Un **protocollo di livello trasporto**, così come un protocollo di livello rete, può fornire **due differenti tipi di servizi**: **connection-oriented** o **connection-less**. A livello di rete un servizio **connection-less** può comportare che datagrammi facenti parte dello stesso messaggio transitino su percorsi fisici **diversi**. A **livello di trasporto** invece, il percorso dei pacchetti **non è rilevante**, la presenza o l'assenza di **connessione** implicano rispettivamente **la dipendenza** o **l'indipendenza fra i pacchetti**.

- In un servizio privo di connessione il processo applicativo mittente deve suddividere i suoi messaggi in porzioni accettabili dal **protocollo di livello trasporto connection-less**, che li tratterà come se fossero unità indipendenti e scorrelate. Le varie porzioni sono **consegnate** al livello trasporto **in ordine**, ma **non verranno ricevute** al processo applicativo destinatario **in tal modo**. Inoltre i pacchetti **non sono numerati**. Non è possibile implementare efficacemente **il controllo di flusso**, **il controllo degli errori** e **il controllo della congestione**.

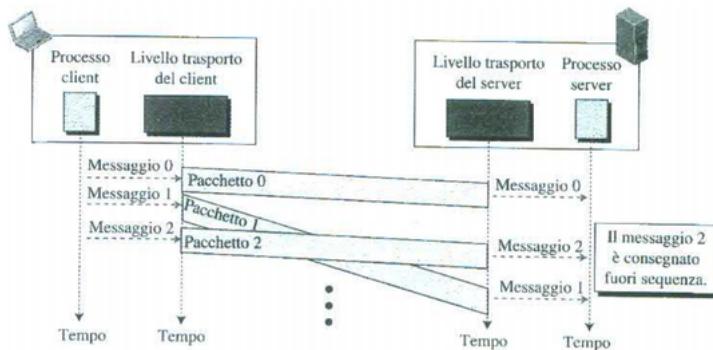


Figura 3.45: Un servizio privo di connessione.

- In un servizio **orientato alla connessione** i due processi comunicanti devono per prima cosa stabilire **una connessione logica** e poi possono iniziare a scambiarsi dati. Una volta **terminato lo scambio** la connessione viene **chiusa**. Come menzionato in precedenza ancora una volta l'approccio **connection-oriented** è **diverso** tra il livello di trasporto e il livello di rete. Nel livello di rete si presuppone una coordinazione totale fra gli host finali e tutti i router della rete, al livello di trasporto solo tra i due host finali. Ciò significa che **è possibile utilizzare un protocollo di trasporto connection-oriented indipendentemente dal protocollo di rete utilizzato**.

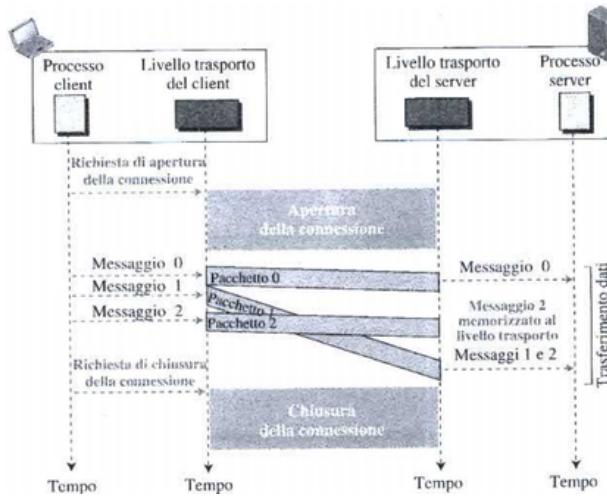


Figura 3.46: Un servizio orientato alla connessione.

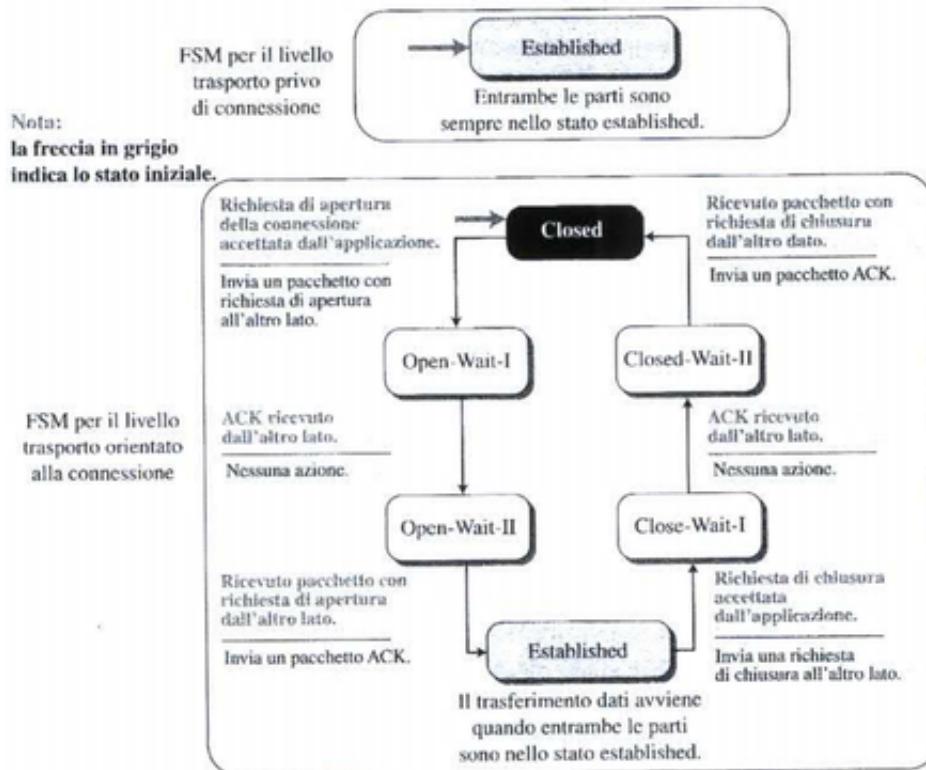


Figura 3.47: Differenze tra i due approcci descritti sopra rappresentati come automi a stati finiti. La rappresentazione dell'approccio connection-oriented in esempio prevede l'apertura di una connessione bidirezionale.

3.2.1 UDP

Il protocollo **UDP** o **User Datagram Protocol** è un protocollo di livello trasporto **inaffidabile e privo di connessione**. Non aggiunge niente ai **servizi IP** se non la comunicazione tra processi che avviene mediante l'utilizzo di **due code**, una in **ingresso** e una in **uscita**. Essendo **moltissime** ha un **overhead minimo** ed è per ciò usato da processi che vogliono inviare **messaggi contenuti** senza preoccuparsi troppo dell'affidabilità che può comunque essere aggiunta al livello applicazione. **Al livello trasporto non è effettuato nessun controllo di flusso né di congestione.**

I pacchetti UDP sono chiamati **datagrammi utente** e data l'assenza di una connessione vengono **trasmessi in modo indipendente** gli uni dagli altri. La loro **intestazione** è costituita da **4 campi di 2 byte**, per un totale di 8 byte. **N.B.** Solo i processi che inviano messaggi di dimensione **inferiore a 65507**, i.e. $65535 - 8$ byte di intestazione UDP = $65527 - 20$ di intestazione IP = 65507, **possono usare il protocollo UDP**.

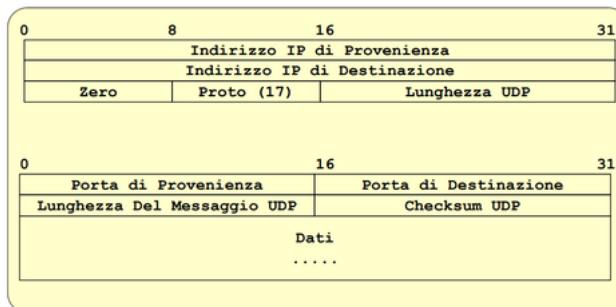


Figura 3.48: La struttura di un datagramma utente. La prima parte in figura è uno **pseudo header** e non viene trasmesso. Il campo proto serve a distinguere i pacchetti UDP (17) da quelli TCP.

I primi due campi definiscono i numeri di porta rispettivamente del **mittente** e del **destinatario**. Il terzo campo definisce la **lunghezza totale del datagramma** compresa l'intestazione. I 16 bit possono definire una lunghezza totale da 0 a 65535 ma in realtà la dimensione è **inferiore**, il datagramma utente viene infatti inserito in un datagramma IP di lunghezza totale di 65535. Lo **pseudoheader** è una parte dell'intestazione del **pacchetto IP** in cui viene incapsulato il datagramma utente. L'ultimo campo dell'intestazione può contenere la **checksum**, se il campo è settato con tutti 0, il mittente ha deciso di non calcolarla. La **checksum** è l'**unico** controllo degli errori che UDP mette a disposizione, usa tutti i campi del datagramma **compreso lo pseudoheader**, se quest ultimo non fosse incluso infatti, si rischierebbe che il datagramma, seppur privo di errori in invio, finisca a un host sbagliato per via di errori nel pacchetto IP che lo trasporta.

Checksum

La **checksum** viene calcolata nel seguente modo:

- **Mittente**

1. Il messaggio viene diviso in parole da 16 bit.
2. Il valore della checksum viene inizialmente impostato a 0.
3. Tutte le parole del messaggio vengono sommate usando l'addizione complemento a 1.
4. Viene fatto il complemento a 1 della somma e il risultato è la **checksum**.

- **Destinatario**

1. Il messaggio che comprende la checksum viene ricevuto.
2. Vengono ripetuti i passaggi 3 e 4 descritti in precedenza.
3. Se il valore della checksum è 0 allora il messaggio viene accettato altrimenti viene scartato.

```

1110011001100110 +
1101010101010101 =
11011101110111011 troncamento
1011101110111011 somma
0100010001000100 checksum

```

DNS e UDP

Il **DNS** usa **UDP** poiché le richieste che il **client DNS** invia al **server DNS** sono **brevi** e possono essere **contenute in un singolo datagramma UDP**. Inoltre le **risposte** devono arrivare **velocemente** e soprattutto viaggia un **solo messaggio** per volta: uno di richiesta e uno di risposta, quindi **non c'è un problema di ordine di sequenza**.

Applicazioni UDP

UDP potrebbe essere usato da applicazioni i cui processi:

- **Richiedono uno scambio di dati di volume limitato, con scarso interesse verso il controllo del flusso e degli errori.** Non è quindi adatto ad applicazioni come **SMTP** e **FTP**.
- **Possiedono meccanismi interni di controllo di flusso e di errore.**
- **Trasmettono in multicast.**
- **Non tollerano latenza** e.g. applicazioni interattive o real time.

3.2.2 TCP

Il protocollo **TCP** è un protocollo **orientato alla connessione e affidabile**, è il protocollo di livello trasporto **più utilizzato** su Internet. Quando un processo sull'host A decide di inviare e ricevere dati da un processo sull'host B avvengono le seguenti **tre cose**:

1. I due processi stabiliscono una connessione **logica**.
2. Vengono scambiati dati in **entrambe** le direzioni.
3. La connessione viene terminata.

Oltre a essere **orientato alla connessione**, il protocollo TCP è anche **orientato al flusso dati**. Al contrario di UDP quindi due processi che comunicano via TCP hanno l'*illusione* di essere connessi con un **tubo**. Dimostra che il processo in trasmissione **produce** il flusso mentre il processo in ricezione lo **consuma**.

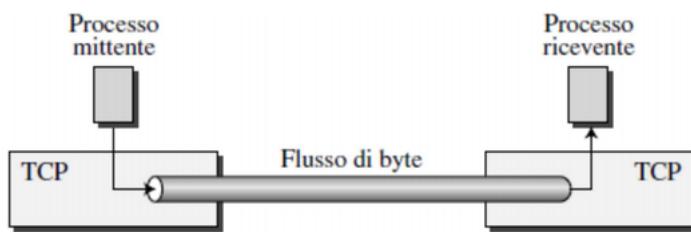


Figura 3.49: Rappresentazione del flusso dati.

Poiché il processo produttore e il processo consumatore non scrivono e leggono dati necessariamente alla stessa velocità è necessario un **controllo del flusso** che consiste in **due buffer uno di trasmissione**, in cui si memorizzano i segmenti **inviai**, e **uno di ricezione** in cui si memorizzano quelli **ricevuti**.

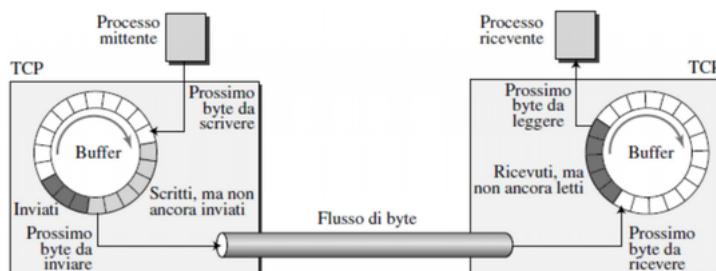


Figura 3.50: I due buffer.

Tipicamente viene usato un **buffer circolare**, poiché consente più agilmente di mettere a disposizione per la scrittura le celle che contengono byte inviati e la cui ricezione è stata confermata.

Sebbene la **bufferizzazione** gestisca il **controllo del flusso**, per inviare dati in uno **stream** bisogna ovviare a un altro problema. Il **livello IP infatti** invia i dati in **pacchetti** e non in un flusso di byte. Per far fronte a ciò **TCP raggruppa un certo numero di byte in unità chiamate segmenti, indipendenti** dal programma applicativo, a cui aggiunge un'intestazione prima di affidarli al livello rete. I **segmenti** vengono poi incapsulati in datagrammi IP e trasmessi. La **bufferizzazione** consente una **riduzione del traffico** sulla rete ottimizzando in un certo modo il numero di segmenti da trasmettere.

N.B. non è detto che tutti i segmenti abbiano la **stessa dimensione**.

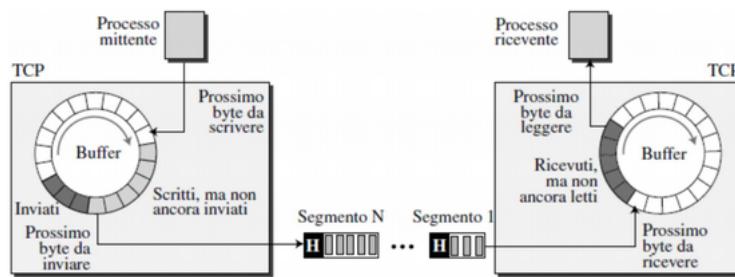


Figura 3.51: Il flusso di dati viene partizionato in segmenti, ognuno con il suo header.

Il **protocollo TCP** inoltre fornisce un servizio **full duplex**, nel quale, instaurata la connessione, le due entità possono **scambiarsi e ricevere** i dati **contemporaneamente**. Ciascuna entità TCP ha infatti i propri buffer di **invio** e **ricezione**.

Numerazione dei pacchetti

L'affidabilità è garantita con un **meccanismo di numerazione** che prevede due campi: **il numero di sequenza** e **il numero di riscontro o ACK**. Questi due campi sono contenuti nell'**intestazione** dei segmenti TCP e fanno riferimento al **numero dei byte**, non al numero di segmento. **TCP infatti numera i byte che vengono trasmessi nella connessione**. Questa **numerazione** avviene all'interno del **buffer di trasmissione**, dopo che i byte sono stati generati dal processo applicativo mittente. I numeri di sequenza e di riscontro sono **diversi** nelle due **direzioni** della comunicazione. Il numero di sequenza del primo byte non necessariamente è 0 ma è **generato arbitrariamente** nell'intervallo tra 0 e $2^{32} - 1$. Dopo aver numerato i byte, **TCP assegna a ogni segmento un numero di sequenza corrispondente al numero del primo byte del segmento**. Il **primo numero di sequenza o ISN** è un numero casuale tra 0 e $2^{32} - 1$. **Tutti gli altri numeri** sono il numero di sequenza del segmento precedente a cui viene sommato il numero di byte, **reali o fintizi**, contenuti dal successivo.

Oltre al **numero di sequenza** nell'intestazione di un **segmento TCP** è contenuto anche un **numero di riscontro**, più brevemente detto **ACK**. L' **ACK** è utilizzato dalle due entità comunicanti per **confermare i byte ricevuti**, indica infatti il **numero del prossimo byte** che l'entità si aspetta di ricevere. *e.g. se un'entità utilizza 5643 come ACK significa che ha ricevuto tutti i byte fino al 5642.* **N.B.:** non necessariamente vuol dire che l'entità ha ricevuto 5642 byte.

Riportiamo di seguito la **struttura** di un **segmento TCP**:

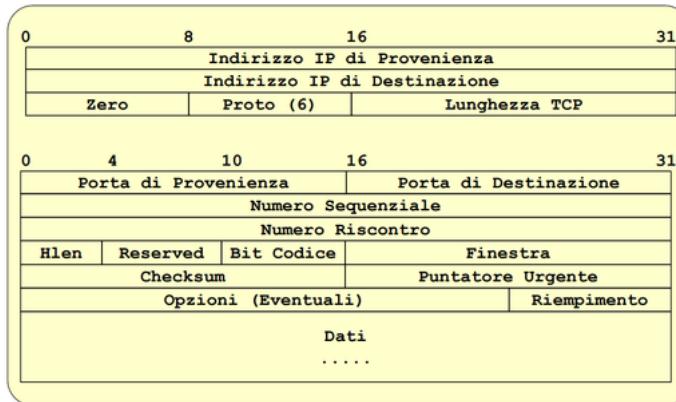


Figura 3.52: La struttura di un segmento TCP.

Il **segmento** consiste di un'**intestazione** di dimensione compresa tra 20 byte, in assenza di opzioni, e 60 byte altrimenti, seguita dai **dati** provenienti dal processo applicativo.

- **Numero di porta sorgente:** contiene il numero di porta del processo mittente sull'host che invia il segmento.
- **Numero di porta destinazione:** contiene il numero di porta del processo destinatario sull'host che riceve il segmento.
- **Numero di sequenza:** contiene il numero di sequenza associato al primo byte di dati contenuto nel segmento.
- **Numero di riscontro:** contiene il numero di sequenza del byte che l'entità si aspetta di ricevere.
- **HLEN o lunghezza dell'intestazione:** indica il numero di parole di 4 byte presenti nell'intestazione, varia da 5 ($\frac{20}{4}$) a 15 ($\frac{60}{4}$).
- **Flags di controllo:** contiene i 6 bit di controllo, più di un bit può essere attivo contemporaneamente. Questi bit intervengono nel **controllo del flusso, nell'apertura e nella chiusura della connessione e nella determinazione della modalità di trasferimento dei dati.**



Figura 3.53: I flags di controllo e il loro significato.

- **Dimensione della finestra:** contiene la dimensione della finestra di ricezione di cui di cui dispone l'entità, la dimensione massima è di 65535 byte, legata ai 16 bit che compongono il campo. È solitamente indicato con RWN o receiving window ed è determinato dal ricevente, il mittente deve rispettare le sue indicazioni.
- **Checksum:** contiene il **checksum**, è calcolato da TCP con le stesse modalità descritte per UDP, tuttavia è **obbligatoriamente richiesto** per i datagrammi TCP. Lo pseudoheader ha lo stesso scopo descritto per UDP, però il **campo proto vale 6**.
- **Puntatore urgente:** è rilevante se il flag **URG** è attivato. Contiene la cifra da sommare al numero di sequenza del segmento per ottenere il numero dell'ultimo bit urgente della sezione dati.
- **Opzioni:** contiene da 0 a 40 byte di informazioni opzionali.

Connessione TCP

Il **protocollo TCP** è **connection-oriented** il che vuol dire che stabilisce una connessione logica tra il processo mittente e il processo destinatario e tutti i segmenti TCP vengono spediti lungo questo percorso. L'utilizzo di una connessione virtuale **semplifica i processi di conferma, di controllo degli errori e di rispedizione**. La trasmissione orientata alla connessione del protocollo TCP richiede **tre fasi: apertura della connessione, trasmissione dei dati e chiusura della connessione**.

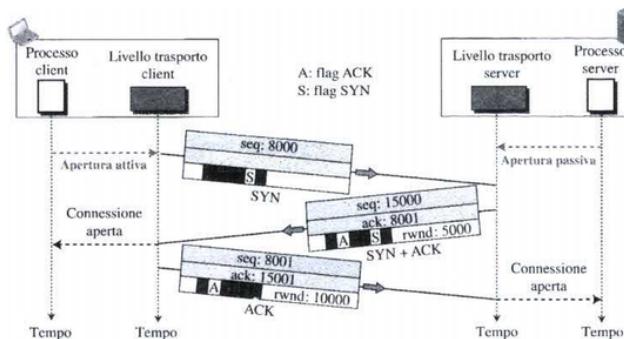


Figura 3.54: L'apertura della connessione TCP.

1. **Apertura della connessione:** nel TCP, essendo un protocollo **full duplex**, le due entità comunicanti devono inizializzare la connessione e ciascuna deve ottenerne l'*approvazione* da parte dell'altra. L'apertura della connessione TCP è detta **three-way handshake** e avviene nel seguente modo:
 - 1.1. Il **client** invia il primo segmento: si tratta di un **segmento SYN**, **senza dati utente** e **contenente solo il flag SYN** che specifica l'**ISN** della connessione client-server.
 - 1.2. Il **server** invia il secondo segmento, **senza dati utente**, con i flag **ACK** e **SYN**, l'**ACK** viene usato dal server per comunicare al client la corretta ricezione del primo segmento e il **SYN** per specificare l'**ISN** della connessione server-client. Dato che il segmento contiene un **riscontro**, conterrà anche la dimensione della finestra di ricezione a cui il client dovrà attenersi.
 - 1.3. A questo punto il **client** invia il terzo segmento e con il flag **ACK** comunica al server la corretta ricezione del secondo segmento. Il terzo segmento **può** contenere dati utente e se li contiene possiede anche un **numero di sequenza**.

Dopo aver instaurato la connessione è possibile trasferire i dati. Come detto sopra, il **protocollo TCP mittente memorizza i dati** generati dal processo mittente in un **buffer**, la quantità di dati che verranno poi inseriti nel segmento e inviati è a discrezione del TCP stesso. **Anche** il protocollo **TCP destinatario memorizza i dati in un buffer** e li invia al processo destinatario quando questo è pronto alla ricezione o quando il TCP destinatario lo ritiene opportuno.

Cosa succede se non possono essere tollerati ritardi nella consegna e nella ricezione dei dati?

Il protocollo TCP gestisce quest'eventualità con la **funzione push** che può essere **richiesta dal processo mittente**. In questo caso il **TCP mittente invia immediatamente i dati al TCP destinatario** e tramite il flag **PSH** specifica che i dati devono essere **immediatamente consegnati al processo destinatario**.

Cosa succede se un processo necessita che il TCP elabori dei dati per primi indipendentemente dalla loro posizione nel segmento?

Il processo invierà i dati urgenti in un segmento con il flag **URG** attivato, li posizionerà all'**inizio** del segmento stesso e indicherà poi al TCP mittente, mediante il **puntatore urgente**, il **numero dell'ultimo byte di dati urgenti** contenuti nel segmento. e.g. se il numero di sequenza del segmento vale 15000 e il puntatore urgente vale 200, l'ultimo byte urgente sarà il 15200.

Come avviene la chiusura della connessione TCP?

Entrambe le entità coinvolte possono chiudere la connessione, solitamente però la chiusura della connessione è iniziata dal processo client. Sono offerte due vie per la chiusura della connessione: **handshake a tre vie** e **handshake a quattro vie con half close**.

- **Handshake a tre vie:** inizia con l'invio da parte del TCP client di un **segmento FIN** all'interno del quale viene settato il **flag FIN**. Questo segmento può contenere o meno dati utente, se non li contiene consuma solo un numero di sequenza. Il **server TCP**, dopo aver ricevuto il segmento FIN, notifica il client TCP della ricezione e annuncia la chiusura della connessione server-client inviando un **segmento FIN + ACK**. Questo segmento può contenere o meno dati da parte del server, se non li contiene consuma solo un numero di sequenza. Il **client TCP** finalizza la chiusura della connessione inviando un segmento ACK per notificare il server della ricezione del segmento precedente. Questo segmento **non contiene dati utente e non consuma numeri di sequenza**.

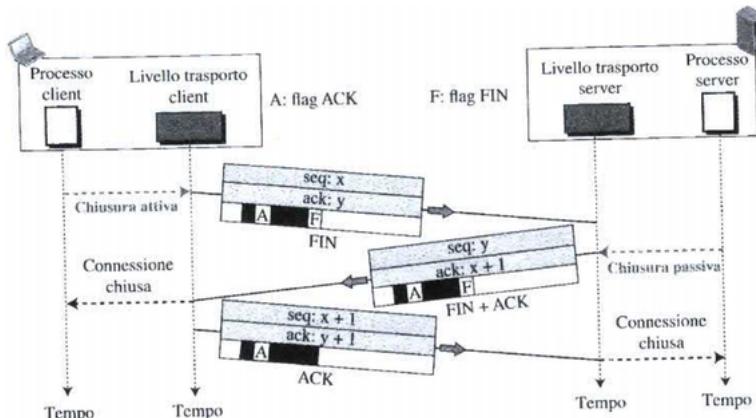


Figura 3.55: Chiusura della connessione tramite handshake a tre vie.

- Nella **chiusura handshake a quattro vie con halfclose** un **processo smette di inviare dati mentre ne sta ancora ricevendo dall'altro**. *e.g. l'ordinamento di una grossa quantità di dati inviati da un client a un server*. Il **client TCP** richiede la halfclose inviando un **segmento FIN**, il server accetta la richiesta di chiusura inviando un **segmento ACK**. Il trasferimento dei dati client-server **termina** ma il **server può ancora inviare dati al client**. Quando a sua volta il **server termina** l'invio dei dati, invia un **segmento FIN** al client che risponde con un **segmento ACK**.

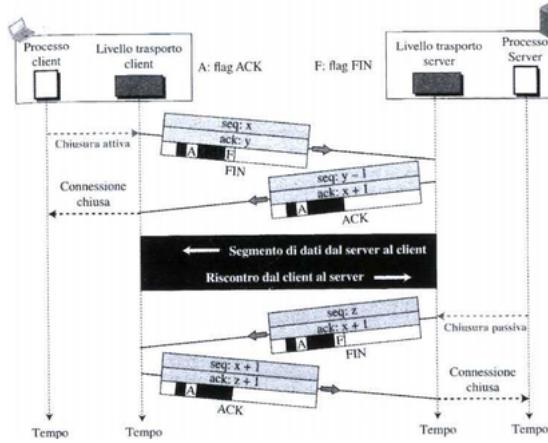


Figura 3.56: Chiusura della connessione tramite handshake a quattro vie con halfclose.

Una visione generale con un **diagramma delle transizioni di stato del protocollo TCP**:

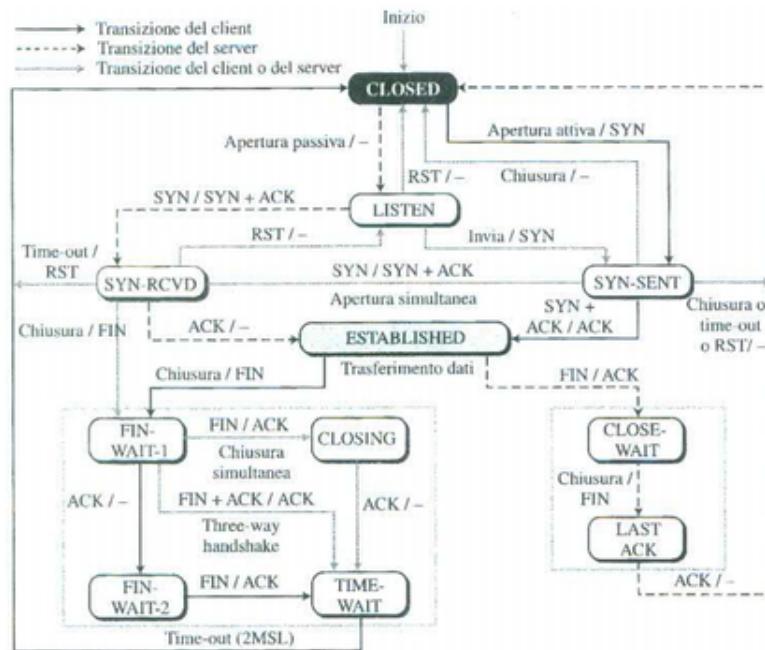


Figura 3.57: Il diagramma delle transizioni di stato.

TIME_WAIT è lo stato finale in cui un'entità che esegue la chiusura attiva si trova prima della chiusura definitiva della connessione. Vi resta per due volte la MSL (Maximum Segment Lifetime). **Garantisce una terminazione affidabile della connessione in caso di perdita dell'ultimo ACK e consente l'eliminazione dalla rete dei segmenti duplicati.**

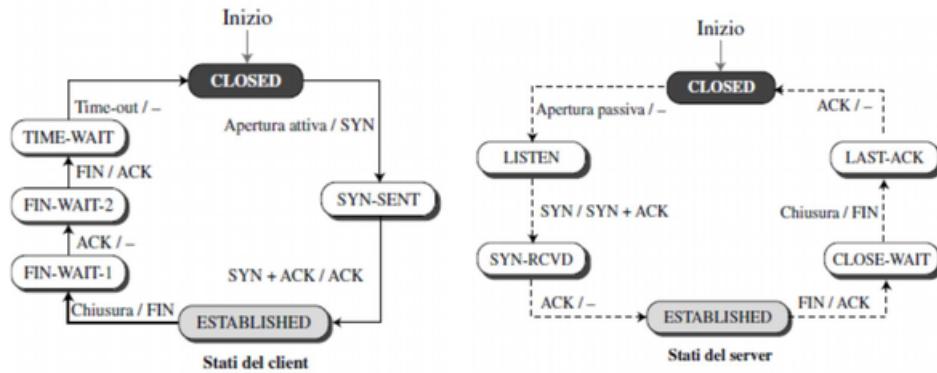


Figura 3.58: Il diagramma delle transizioni di stato con chiusura four-way halfclose.

Riportiamo di seguito i **significati degli stati**.

Stato	Significato
LISTEN	server in attesa di SYN
SYN-SENT	SYN inviato e attesa ACK
SYN-RECEIVED	SYN e ACK inviati e attesa di ACK
ESTABLISHED	connessione aperta
FIN-WAIT-1	primo FIN inviato attesa di ACK
FIN-WAIT-2	ricevuto ACK primo FIN attesa di secondo FIN
CLOSE-WAIT	ricevuto primo FIN e ACK inviato
TIME-WAIT	ricevuto secondo FIN e ACK inviato, attesa 2MSL
CLOSING	attesa di ACK chiusura
LAST-ACK	secondo FIN inviato attesa di ACK
CLOSED	assenza di connessione

Controllo del flusso

Descriviamo il **ciclo di vita dei dati** scambiati all'interno della comunicazione TCP. I dati sono:

1. **Generati** dal processo applicativo mittente.
2. Consegnati al **TCP mittente**.
3. Consegnati al **TCP destinatario**.
4. **Consumati** dal processo applicativo destinatario.

Il **controllo del flusso** è gestito con il seguente meccanismo: il **TCP destinatario** invia un feedback al **TCP mittente** che a sua volta invia un feedback al **processo applicativo mittente**. Quando la finestra del TCP mittente è **piena** i dati, provenienti dal processo, vengono **rifiutati**.

Il protocollo TCP prevede due finestre, una di **ricezione** e una di **trasmissione**, per ognuna delle due direzioni di comunicazione, in totale sono quindi presenti **4 finestre**. Ricordiamo che TCP instaura **una sola connessione**. La dimensione della finestra di trasmissione è *inizialmente* determinata all'apertura della connessione. Si **apre** quando la dimensione della finestra di ricezione segnalata dal TCP destinatario lo consente e si **chiude** quando viene confermata la ricezione dei byte inviati. **Apertura, chiusura e ridimensionamento sono controllate dal TCP destinatario.**

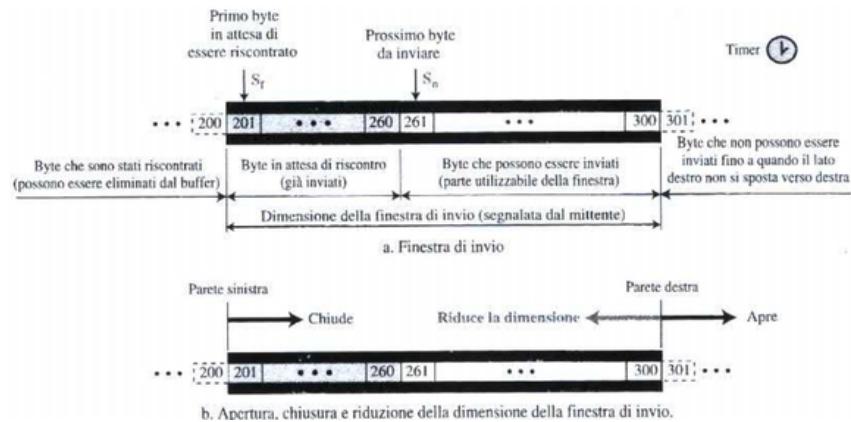


Figura 3.59: La finestra di trasmissione o di invio TCP.

Invece la **finestra di ricezione** si **chiude** quando giungono altri byte dal **TCP mittente** e si **apre** quando il processo applicativo destinatario richiede altri byte.

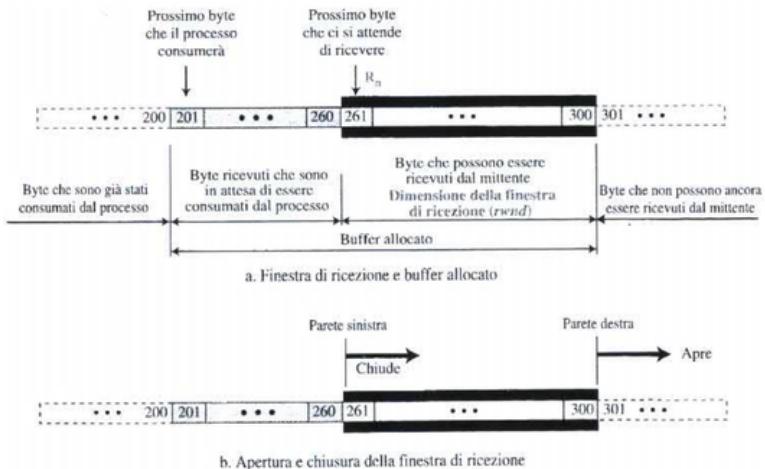


Figura 3.60: La finestra di ricezione TCP. Ipotizzeremo che non si riduca mai.

Alleghiamo di seguito un'immagine in grado di spiegare efficacemente ciò che succede in uno scenario **dinamico**.

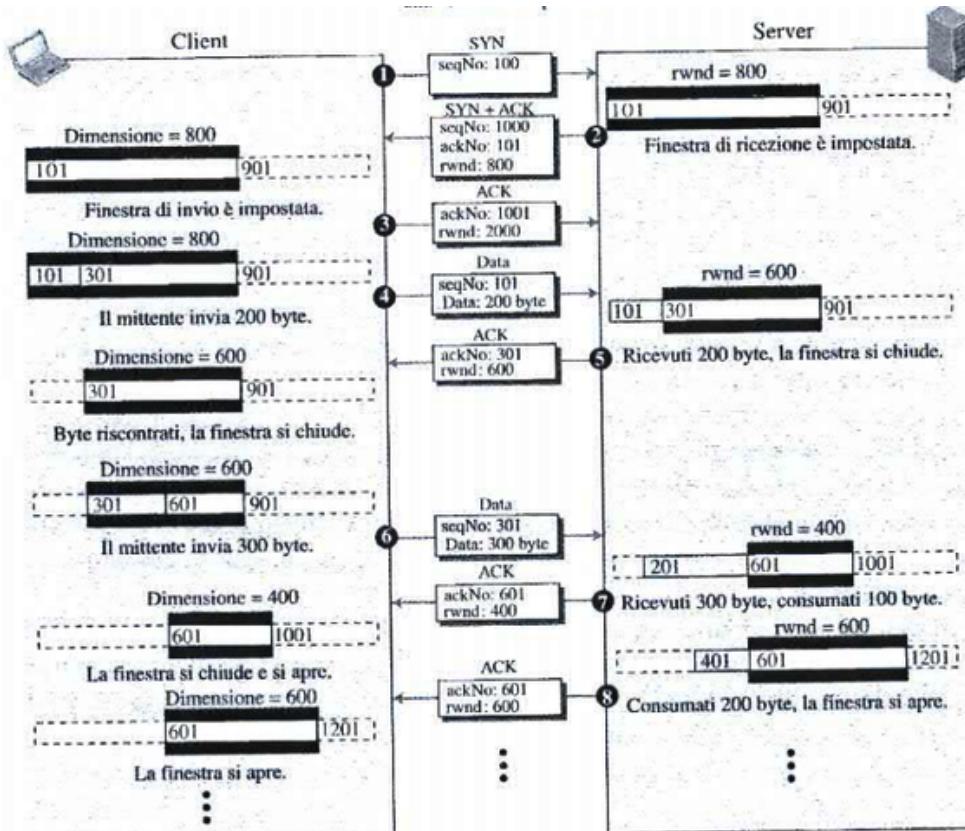


Figura 3.61: Il controllo di flusso in una ipotetica connessione TCP unidirezionale.

$$\text{rwindow} = \text{RecBuff} - (\text{LastByteReceived} - \text{LastByteRead})$$

Il **mittente** si assicura che:

$$\text{LastByteSent} - \text{LastByteAcked} < \text{rwindow}$$

Ovvero che la quantità di dati trasmessi e non ancora riscontrati sia **minore della finestra di ricezione**.

N.B. rwindow può essere uguale a 0, qualora ad esempio il TCP destinatario non volesse ricevere dati per un certo periodo di tempo. Il mittente in questo caso **non** riduce realmente la dimensione della sua finestra ma continua a mandare **segmenti sonda** di 1 byte per ricevere l'**aggiornamento** sulla dimensione di rwindow. Questa tecnica si definisce **probing** ed è usata per **prevenire lo stallo**.

Controllo degli errori

Come è ormai più che noto **TCP è un protocollo di trasporto affidabile** e garantisce al processo applicativo che i dati vengano consegnati in **sequenza, senza errori, smarrimenti o duplicazioni**.

Gli strumenti utilizzati da TCP per individuare gli errori di trasmissione sono **tre**:

1. **La checksum:** ciascun **segmento TCP** contiene un campo **checksum** di 16 bit **obbligatorio** e utilizzato per identificare i **segmenti corrotti**. Se un segmento viene rilevato essere tale viene **scartato** e considerato **smarrito**.
2. **I messaggi di riscontro o ACK:** TCP usa gli ACK per riscontrare la **ricezione dei segmenti** che contengono un **numero di sequenza**, possono essere segmenti **dati** o di **controllo**. I segmenti ACK **non** usano numeri di sequenza e perciò **non** vengono **riscontrati**. Nella sua **implementazione originale** TCP è stato progettato per riscontrare i segmenti in modo **cumulativo**, ovvero notificando il numero del byte che si attende e ignorando i segmenti duplicati e fuori sequenza. Nelle **versioni più recenti** è implementato il **selective ack o SACK** il quale prevede che i **pacchetti** ricevuti **fuori sequenza** vengano ugualmente **memorizzati** e inoltre sia presente un riscontro per i pacchetti fuori sequenza e quelli duplicati contenuto nel campo **OPTIONS**. Un'entità TCP genera **riscontri** quando:
 - **Invia un segmento dati** a un'altra entità e **contemporaneamente invia l'ACK** contenente numero del prossimo byte che si aspetta di ricevere, così da **ridurre il traffico**. Se destinatario **non** ha dati da inviare e riceve un segmento in ordine ritarda l'invio dell'ACK di **500ms** a meno che non riceva un nuovo segmento sempre per una ragione di riduzione del traffico.
 - Vengono ricevuti **due segmenti nel giusto ordine e nessuno dei due è ancora stato riscontrato**. Viene inviato immediatamente un segmento ACK, dal destinatario al mittente, per **evitare la ritrasmissione inutile dei segmenti**.
 - Arriva un **segmento fuori sequenza** il destinatario invia immediatamente un ACK per **consentire la ritrasmissione rapida dei segmenti**.
 - **Arriva un segmento mancante o un segmento duplicato**.
3. **I timeout:** il **TCP** mittente inizializza un **timer di ritrasmissione**, o **RTO** per ogni segmento inviato. Il **primo** segmento inviato e in attesa di riscontro, che ricordiamo essere memorizzato nel buffer del TCP mittente, allo scadere del timer, è **ritrasmesso**.

N.B. Un segmento viene ritrasmesso o alla **scadenza del timer** o quando vengono ricevuti **tre ACK duplicati** per quello precedente, questa funzione è chiamata **ritrasmissione veloce**.

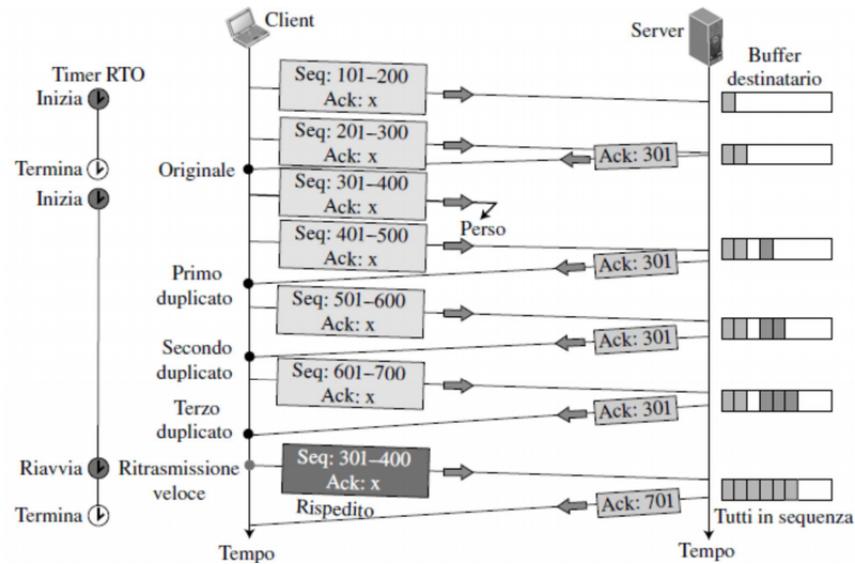


Figura 3.62: Un'esempio di scambio di dati tra due entità con un esempio di ritrasmissione veloce. Si supponga in uso una versione moderna di TCP dove i pacchetti fuori sequenza vengono memorizzati e non scartati.

L'RTO ha un **valore dinamico** calcolato in base al tempo di andata e di ritorno dei segmenti trasmessi detto anche **RTT** o **Round Trip Time**.

Presentiamo ora una collezione di vari scenari di operatività del protocollo TCP.

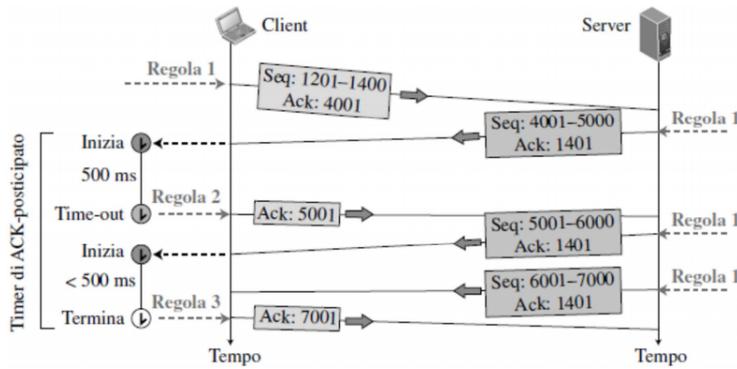


Figura 3.63: Uno scenario di operatività normale. Notare, come descritto in precedenza, il ritardo nella trasmissione dell'ACK dovuto alla ricezione di un solo segmento.

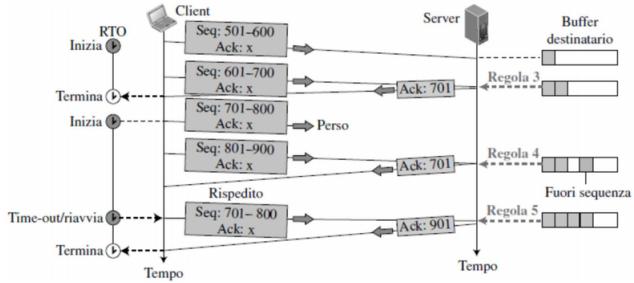


Figura 3.64: Uno scenario di perdita di un segmento. Notare, come descritto in precedenza, che alla ricezione del secondo segmento in ordine, alla ricezione del terzo segmento in *disordine* e alla ricezione del quarto segmento mancante viene immediatamente inviato un ACK.

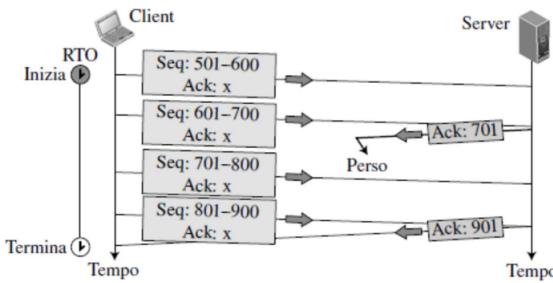


Figura 3.65: Uno scenario di perdita di un ACK. Notare come lo scenario venga ripristinato automaticamente in una condizione di normalità dalla spedizione dell'ACK successivo.

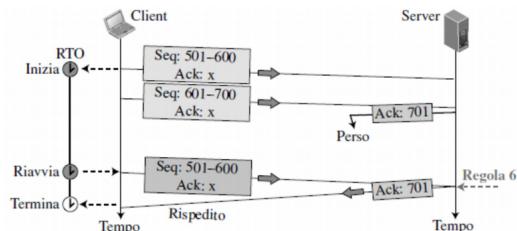


Figura 3.66: Uno scenario di perdita di un ACK. Stavolta si torna alla normalità tramite la ricezione di un segmento duplicato e il conseguente ed immediato invio dell'ACK da parte del server..

N.B. Lo smarrimento di un riscontro può provocare lo stallo ed è nel caso in cui, a seguito della ricezione da parte del TCP mittente di un segmento ACK con il campo **rwnd** settato a **0**, si perda il successivo ACK con il campo rwnd non nullo. Il **TCP destinatario** supporrà che il TCP mittente lo abbia ricevuto e **resterà in attesa** dei dati.

Riassumiamo gli argomenti fin qui trattati presentando degli **ASF** per il **TCP mittente** e il **TCP destinatario**.

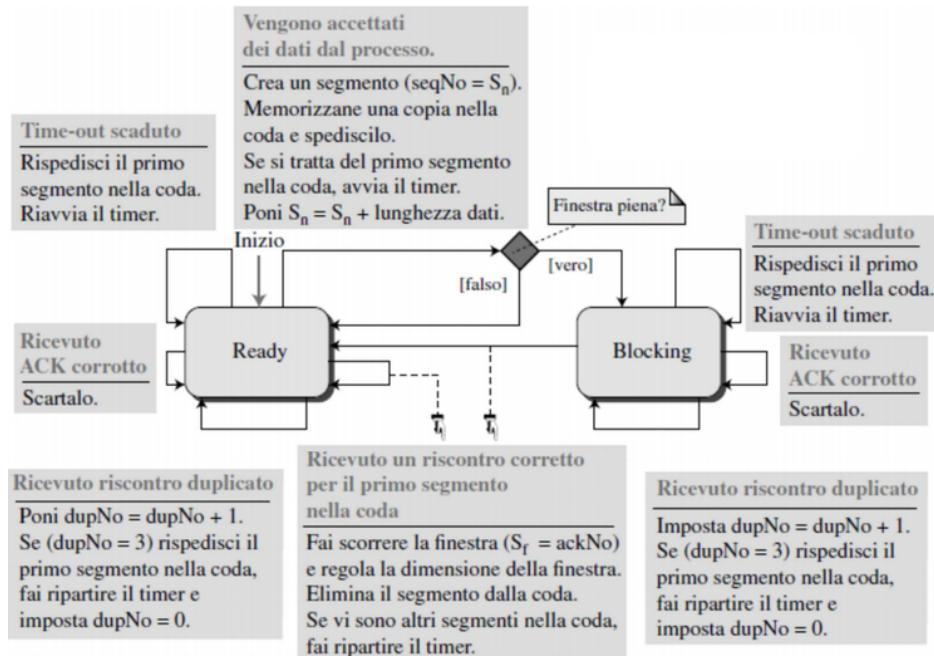


Figura 3.67: ASF del TCP mittente.

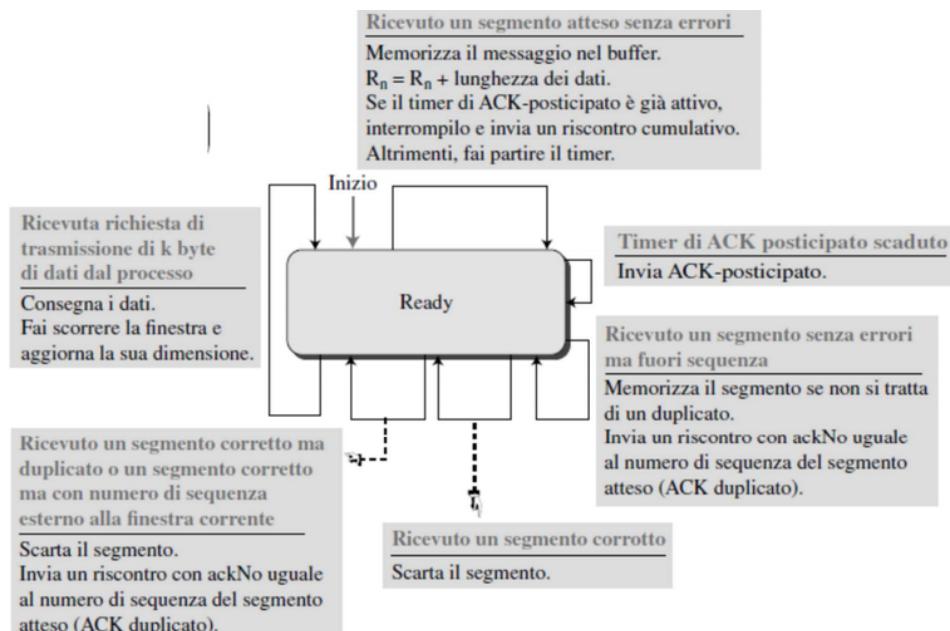


Figura 3.68: ASF del TCP destinatario.

Come si calcola l'RTO o Retransmission Time-Out?

TCP fa partire un **timer** quando invia un segmento all'**inizio** della coda di invio, quando il **timer scade** TCP **rispedisce il primo** segmento all'inizio della coda e fa ripartire il timer. Per definire l'**RTO** bisogna prima definire l'**RTT** o **Round Trip Timer**. Infatti l'**RTO** deve essere maggiore de l'**RTT**. Ricordiamo che l'**RTT** è il **tempo trascorso da quando si invia un segmento a quando ne si riceve il riscontro**. Calcolare l'**RTT** è un procedimento abbastanza complicato a causa della sua **variabilità** nel percorso.

Inizialmente l'**RTT** vale 0 e l'**RTO** è **preimpostato**. Dopo la prima misurazione, ottenuta inviando un segmento e contando il tempo che passa tra l'invio e il riscontro, avremo un *RTT_{ESTIMATED}*.

N.B. non si ha **nessuna** garanzia che il prossimo segmento impiegherà lo **stesso** tempo per essere inviato e riscontrato!

Perciò dalla **seconda misurazione** in avanti RTT viene calcolato utilizzando l'**RTT** dei segmenti precedenti con la seguente formula:

$$RTT_{ESTIMATED} = (1-\alpha) \times RTT_{ESTIMATED} + \alpha \times RTT_{SAMPLE} \quad (3.1)$$

L'**RTT_{SAMPLE}** è riferito all'**ultimo** segmento inviato. **RTT_{ESTIMATED}** è quindi la **combinazione dei suoi precedenti valori e il nuovo valore RTT_{SAMPLE}**. Il valore di α viene posto a 1/8 in modo da rendere via via meno importanti gli RTT dei pacchetti più vecchi. Si ha quindi:

$$RTT_{ESTIMATED} = 0,875 \times RTT_{ESTIMATED} + 0,125 \times RTT_{SAMPLE} \quad (3.2)$$

Oltre al valore RTT stimato è necessaria anche una stima della sua **variabilità** data dalla **seguente formula**:

$$RTT_{DEV} = (1 - b) \times RTT_{DEV} + b \times | RTT_{SAMPLE} - RTT_{ESTIMATED} | \quad (3.3)$$

Inizialmente $RTT_{DEV} = \frac{RTT_{sample}}{2}$.

RTT_{DEV} rappresenta una **stima di quanto RTT_{SAMPLE} si discosta da RTT_{ESTIMATED}**. Il valore di b viene posto a 1/4. Una volta ottenuti questi valori, **il timeout viene normalmente calcolato come**:

$$RTO = RTT_{ESTIMATED} + 4 \times RTT_{DEV} \quad (3.4)$$

In altre parole si prende il valore più recente di **RTT_{ESTIMATED}** e gli si somma **quattro volte RTT_{DEV}**, solitamente piccolo, per **equilibrarlo**.

Controllo della congestione

Ricordiamo che il fenomeno della **congestione** è originato dal tentativo delle sorgenti di **richiedere più banda di quella disponibile** sul percorso fino alle destinazioni.

Il **traffico eccessivo** nella rete **può provocare**:

- **Lunghi ritardi** causati dall'accodamento dei pacchetti nei buffer dei router.
- **Perdita di pacchetti** causata dall'overflow nei buffer dei router.

Il **protocollo TCP** utilizza la **finestra di congestione** come principale strategia per evitare la congestione.

TCP usa le **finestre di ricezione** e le **finestre di trasmissione** per far sì che i due rispettivi buffer non vengano sovraccaricati. Fondamentalmente però la congestione è un **problema del livello di rete** e sebbene il protocollo IP non lo gestisca, la soluzione delle due finestre assicura l'assenza di congestione **agli estremi** della comunicazione ma **non nel mezzo**. La politica di rispedizione dei segmenti persi di TCP in combinazione con la congestione di rete potrebbe portare al **collasso dell'intera rete**. TCP non può quindi ignorare la congestione della rete ma non può nemmeno essere troppo conservativo più di quanto che è già nell'inoltrare i segmenti. Si utilizza perciò una **seconda variabile** chiamata **cwnd o congestion window** il cui valore dipende dal **livello di congestione della rete**. Con l'aggiunta di questa seconda variabile il valore della finestra d'invio è determinato anche dalla congestione presente in rete.

$$\text{Dimensione della finestra} = \min(\text{rwnd}, \text{cwnd})$$

Ma come si determina il valore di cwnd?

Prima di rispondere a questa domanda è bene spiegare **come fa il protocollo TCP ad accorgersi della presenza di congestione sulla rete**. Il **TCP mittente** interpreta come sinonimi di congestione:

1. Il **timeout**.
2. La ricezione di **tre riscontri duplicati**.

Il secondo caso è **probabilmente meno critico del primo** poiché significa che un segmento è stato smarrito ma tre sono stati ricevuti. Può indicare o che la **rete è al limite della congestione o vi che è appena uscita**. L'**assenza** di una ricezione rapida e regolare dei riscontri è invece **evidenziata** dalla scadenza del **timeout**.

La strategia principale usata da TCP per controllare la congestione prevede **tre fasi**:

1. **Slow Start**.
2. **Congestion Avoidance**.
3. **Fast Recovery**.

A ogni fase corrisponde un algoritmo diverso.

- **Slow Start:** si basa sull'inizializzare la dimensione della finestra di congestione rendendola pari a quella della **MSS** o **Maximum Segment Size**. Aumentandola poi di MSS ogni volta che **un** segmento viene riscontrato. La MSS è stabilita all'inizio della connessione. Così facendo l'algoritmo parte lentamente ma procede con velocità via via crescente. La dimensione di cwnd nella fase di **slow start** può essere espressa come 2^n dove n sono i **riscontri ricevuti**. Se **due segmenti vengono confermati con lo stesso riscontro** cwnd **aumenta solo di 1**. La crescita è comunque **esponenziale** nell'unità di tempo RTT, ma di esponente diverso. La fase continua **fino** al raggiungimento da parte di cwnd di una soglia chiamata **ssthresh** o **slow start threshold**, mantenuta dal **mittente**. Al raggiungimento di questa soglia si entra nella fase di **congestion avoidance**.
- **Congestion Avoidance:** una volta che **cwnd** ha raggiunto una dimensione **relativamente grande**, per evitare che la crescita esponenziale finisca per causare problemi di congestione, il **protocollo TCP** usa un altro algoritmo che incrementa in modo **lineare** anziché **esponenziale** il valore di cwnd. La fase di **congestion avoidance** prevede che ogni volta che **l'intera finestra di ricezione viene riscontrata** la sua dimensione aumenti **di un'unità**.

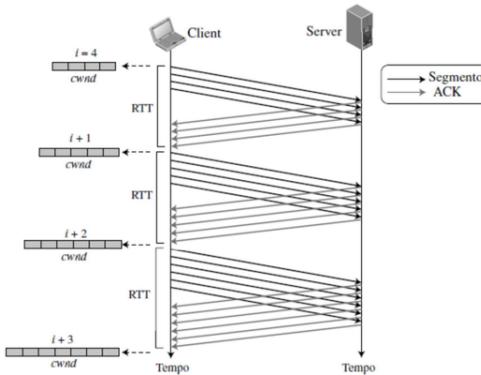


Figura 3.69: In figura la crescita della cwnd nella fase di congestione avoidance.

- La terza fase chiamata **Fast Recovery** è **opzionale** in TCP. La **versione originale non ne faceva uso**. Inizia quando arrivano **tre riscontri duplicati** interpretati come inizio di leggera congestione della rete. Aumenta la dimensione di cwnd in modo **lineare**, incrementandola di un'unità ogni volta che riceve **due riscontri duplicati**.

Analizziamo ora come, attraverso l'affermarsi di modelli sempre più recenti del protocollo TCP siano **cambiate le modalità di transizione da una fase all'altra**. La **prima versione** di TCP chiamata **TCP Tahoe** utilizzava **solo le prime due fasi**. Se durante la fase di **slow start** rileva la presenza di congestione allora **resetta** la dimensione di **cwnd** a 1 e **assegna** a **ssthresh** il vecchio valore di **cwnd dimezzato**. Se non la rileva entro il raggiungimento di **ssthresh** passa alla fase di **congestion avoidance**. In questa fase **non vi è un limite sulla dimensione di rwnd** e se non è rilevata congestione **aumenta linearmente fino alla chiusura della connessione**, se invece viene rilevata si riparte dalla fase di slow start con **cwnd pari a 1** e **ssthresh pari al vecchio valore di cwnd dimezzato**.

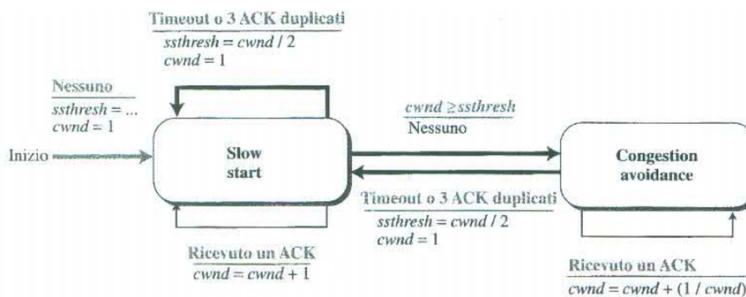


Figura 3.70: TCP Tahoe rappresentato come ASF.

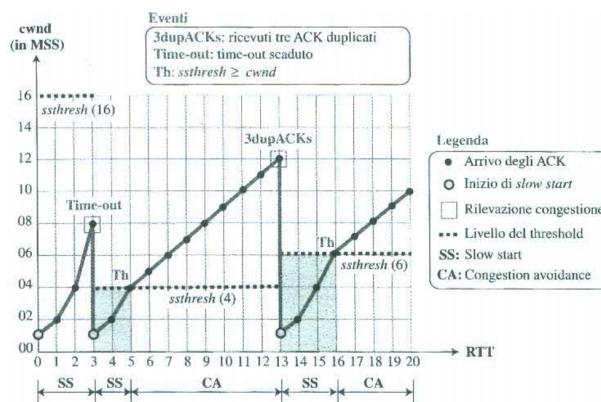


Figura 3.71: In figura l'evoluzione della dimensione di cwnd con TCP Tahoe.

In una versione più recente del protocollo TCP chiamata **TCP Reno**, è prevista anche la terza fase di **fast recovery**. TCP Reno inoltre **tratta i due sintomi della congestione in modo differente**. La mancata ricezione di un riscontro e il conseguente **timeout** portano TCP Reno a tornare nella fase di **slow start**, mentre la ricezione di **tre ACK duplicati**, sia che avvenga in **slowstart** che in **congestion avoidance**, portano TCP Reno ad **assegnare a ssthresh il valore di cwnd dimezzato**, a **cwnd il nuovo valore di ssthresh aumentato di tre unità** e a spostarsi nella fase di **fast recovery**, che può essere visto come uno **stato intermedio tra slow start e congestion avoidance**. In **fast recovery** **cwnd continua a crescere esponenzialmente**, se arriva un **ACK non duplicato** si giunge nello stato di **congestion avoidance** se invece giunge un altro **ACK duplicato** rimane in **fast recovery** e si **aumenta la cwnd di un'unità**, infine se scatta un timeout si torna in **slow start** con le modalità descritte per TCP Tahoe.

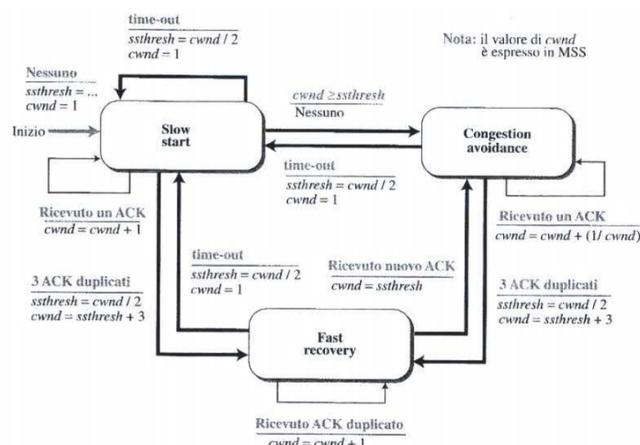


Figura 3.72: TCP Reno rappresentato come ASF.

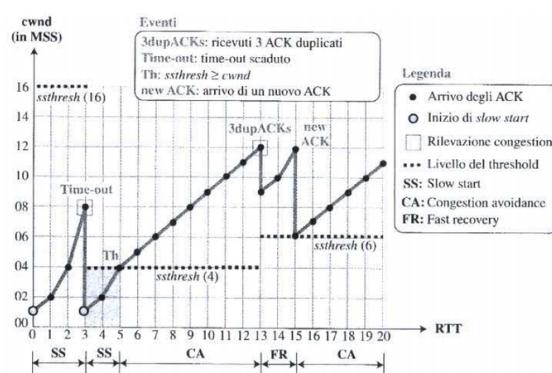


Figura 3.73: In figura l'evoluzione della dimensione di cwnd con TCP Reno.

Throughput

Il throughput di TCP dipende dal comportamento della finestra di congestione e non è quindi costante, se riportato su un grafico avrebbe un profilo simile a quello dei denti di una sega detto anche **AIMD** o **addictive increase multiplicative decrease**. Ciò è dovuto a come è determinata, dopo la prima fase di **slow start**, la dimensione della finestra di congestione. Se i denti fossero tutti uguali il throughput sarebbe uguale a:

$$\frac{\frac{|max+min|}{2}}{RTT}$$

Sapendo che ogni rilevazione di congestione cwnd è impostato alla metà del suo valore precedente, **il throughput può essere calcolato come:**

$$\frac{0.75 \times W_{max}}{RTT}$$

Dove W_{MAX} è la dimensione media di cwnd in presenza di congestione.

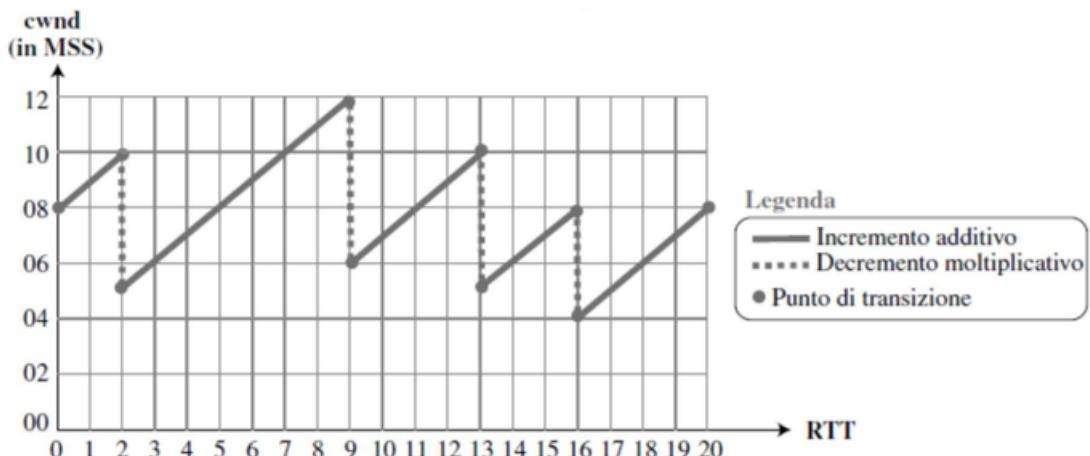


Figura 3.74: In figura il profilo AIMD.

Proviamo a **calcolare il throughput** avendo come dati il **grafico**, l' **MSS** pari a 10 KB (kilobyte) e un **RTT** di 100 ms. Otteniamo

$$W_{MAX} = \frac{10 + 12 + 10 + 8 + 8}{5} = 9.6 \text{ MSS}$$

da cui

$$throughput = \frac{0,75 \times W_{MAX}}{RTT} = \frac{0,75 \times (96 \times 8)_{Kbit}}{100_{ms}} = 5,76 Mbit/s$$

3.3 Esercizi

1. L'utente

mickey@disney.com

invia dal suo PC una email a

donald@disney.com

. Indicare la sequenza di comandi SMTP inviati e ricevuti dal PC di **mickey@disney.com** se:

- (a) Il mailserver di disney.com non è raggiungibile.
- (b) Il mailserver di disney.com è raggiungibile.

Soluzione:

- (a) Il PC di mickey@disney.com non riesce a stabilire una connessione TCP con il mailserver di disney.com, quindi **nessun** messaggio SMTP viene inviato o ricevuto.
- (b) Il PC di mickey@disney.com stabilisce una connessione TCP con il mailserver di disney.com su cui scambia i seguenti comandi SMTP:
R: 220 service ready
I: HELO ...
R: 250 OK
I: MAIL FROM: mickey@disney.com
R: 250 OK
I: RCPT TO: donnald@disney.com
R: 250 OK
I: DATA
I: ...
I: ...
R: 250 OK
I: QUIT
R: 221 service closed

È bene ricordarsi che il corpo del messaggio termina **sempre** con ritorno a capo e fine linea.

Per chiarimenti è consigliato il ripasso del capitolo sull'**SMTP**.

2. Un host deve risolvere il nome simbolico:

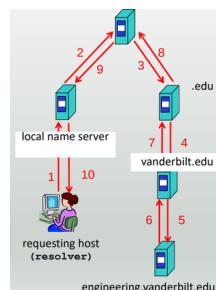
host.engineering.vanderbilt.edu

il cui indirizzo IP **non** è noto al suo resolver (i.e. servizio DNS dell'host). Supponendo che la gerarchia dei name server abbia 4 livelli, indicare, giustificando la risposta, il numero di messaggi DNS che, nel caso peggiore, circoleranno in Internet per risolvere tale nome simbolico, se:

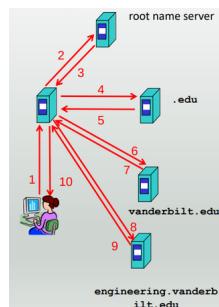
- (a) Si utilizza ad ogni livello una risoluzione ricorsiva.
- (b) Si utilizza ad ogni livello una risoluzione iterativa.

Soluzione:

- (a) Bisogna percorrere tutto l'albero dei name server, dal name server locale fino ad un root server, e poi da questo al name server autoritativo, che nel caso peggiore è il name server locale per engineering.vanderbilt.edu, per poi tornare indietro: in totale sono $4 + 4$ messaggi a cui vanno aggiunti i 2 messaggi dal resolver al name server locale del client, e viceversa: quindi **10 messaggi**.



- (b) Anche ora i messaggi saranno **10**, perchè i name server coinvolti saranno 4, nel caso pessimo, più i 2 messaggi dal resolver al name server locale del client, e viceversa.



3. Un client C chiede la pagina web

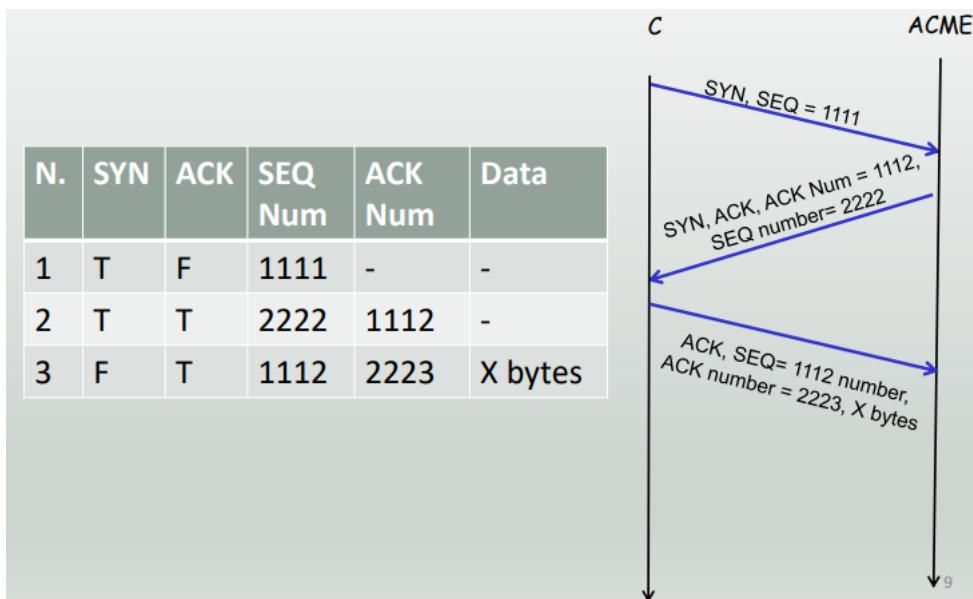
www.acme.com/home/products.html

al server B di

www.acme.com

con una GET che è contenuta in un segmento TCP il cui payload (campo Dati) è lungo X byte. Indicare, giustificando la risposta, i valori dei campi sequence number, ack number, e dei flags ACK ed SYN e lunghezza del campo DATA, in ciascuno dei segmenti che C e B si scambiano per aprire la connessione nell'ipotesi che l'ack finale dell'apertura della connessione sia inviato in piggybacking assieme alla GET. Si supponga che in B ed in C rwnd sia molto grande, che non scada alcun timeout, che non ci siano errori di trasmissione, che nessun segmento vada perduto, e che il numero di sequenza iniziale di C sia 1111 e quello di B sia 2222.

Soluzione:



Si ricorda che per **piggybacking** si indica la tecnica mediante la quale l'invio dell'ACK è unito all'invio dei dati. La tecnica migliora l'efficienza dei protocolli bidirezionali.

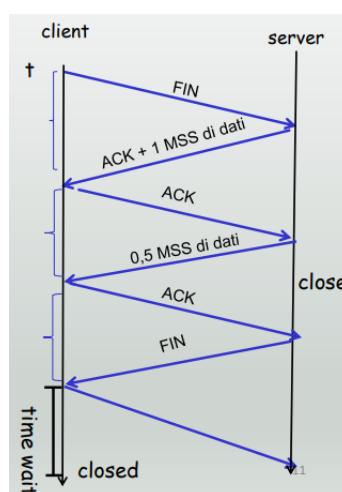
4. Un client C ha stabilito una connessione TCP con un server web S per scaricare una pagina web che consiste di tre oggetti. Al tempo t, subito dopo avere inviato la richiesta per il terzo oggetto, l'host di C invia a S un segmento con il flag FIN a true. Indicare, giustificando la risposta, il tempo minimo necessario al TCP di C per chiudere definitivamente la connessione supponendo che:

- La dimensione del terzo oggetto sia 1,5 MSS.
- RTT sia costantemente 700 msec e il maximum segment lifetime sia 1100 msec.
- Tutti i segmenti vengano ricevuti corretti ed in ordine, e che il valore di cwnd del TCP di S sia 1MSS quando esso riceve il FIN inviato dall'host di C. Trascurare i tempi di preparazione e di trasmissione dei segmenti e assumere che S abbia già a disposizione tutti i dati da inviare.

Soluzione:

$$3RTT + 2MSL = 4300\text{msec}$$

Il TCP di C riceve al tempo $t + RTT$ il riscontro S1 del segmento FIN da lui inviato. Se S1 trasporta in piggybacking il primo MSS dei dati del terzo oggetto e il TCP di C invierà un riscontro C2 per tali dati, quindi S potrà inviare l'ultima porzione di dati e attenderne il riscontro, infine il TCP di S invierà un segmento con il flag FIN a true. Il TCP di C invierà un riscontro del FIN e considererà chiusa la connessione dopo avere atteso 2MSL, ovvero al tempo $t + 3RTT + 2MSL = t + 4300$ msec.



5. Si descriva il meccanismo di controllo di flusso in TCP.

Una possibile soluzione:

Con controllo di flusso si intende la capacità del mittente di evitare la possibilità di saturare il buffer del ricevitore. Infatti, a livello TCP ogni host imposta un buffer di invio e uno di ricezione. Il processo applicativo destinatario legge i dati dal buffer di ricezione (non necessariamente nell'istante in cui arrivano). Il controllo di flusso ha lo scopo di regolare la frequenza di invio del mittente in base alla frequenza di lettura dell'applicazione ricevente allo scopo di non saturare il buffer del ricevente. TCP implementa questa funzione tramite una variabile detta **receive window** mantenuta nel mittente: questa variabile fornisce un'idea di quanto spazio è ancora a disposizione nel buffer del ricevitore. Tale valore è comunicato nel **campo window** dell'header TCP dall'host destinatario.

- Il valore di **receive window** è pari a:

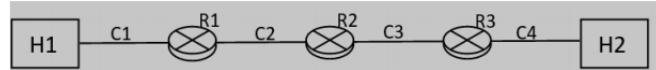
$$\text{rwnd} = \text{RcvBuffer} - (\text{LastByteReceived} - \text{LastByteRead})$$
- L'host destinatario comunica la dimensione di RcvWindow al mittente.
- Il mittente si assicura che $\text{LastByteSent} - \text{LastByteAcked} < \text{rwnd}$ pari ovvero alla quantità di dati trasmessi e non ancora riscontrati.

Nelle situazioni in cui il buffer risulta pieno ($\text{rwnd} = 0$), il mittente continua a mandare **segmenti sonda** di 1 byte per ricevere l'aggiornamento sulla dimensione di rwnd ed evitare lo stallo.

Per approfondire l'argomento recarsi al **capitolo sul TCP**.

6. Due host H1 e H2 comunicano tramite un canale che attraversa tre router R1, R2 e R3 e 4 link di capacità C1, C2, C3 e C4, rispettivamente come mostrato in figura. La comunicazione avviene tramite commutazione di pacchetto con trasmissione di tipo store and forward. Assumendo che il ritardo di propagazione sia trascurabile, che i ritardi di accodamento nei router R1, R2 e R3 sia rispettivamente a1, a2 e a3, e che il ritardo di elaborazione nei tre router sia uguale a 1 ms, dire quanto tempo è necessario per la trasmissione da H1 a H2 di un pacchetto di dimensione L nel seguente caso:

- $L = 10 \text{ KBytes}$.
- $C_1 = C_2 = C_3 = C_4 = 2 \text{ Mbps}$.
- $a_1 = a_2 = a_3 = 0,01 \text{ s}$.



Soluzione: Il ritardo introdotto da ogni router è dato dalla somma tra: ritardo di trasmissione, ritardo di propagazione, ritardo di accodamento e tempo di elaborazione. In particolare:

- il tempo di trasmissione di H1 è pari a $\frac{L}{C_1} = 40$ ms.
- il tempo di trasmissione di R1 è pari a $\frac{L}{C_2} = 40$ ms.
- il tempo di trasmissione di R2 è pari a $\frac{L}{C_3} = 40$ ms.
- il tempo di trasmissione di R3 è pari a $\frac{L}{C_4} = 40$ ms.

Quindi il ritardo introdotto dai router è: ritardo introdotto dai router R1,R2,R3 = $40 + 10 + 1 = 51$ ms.

Il ritardo dovuto alla trasmissione da H1 è pari a $\frac{L}{R} = 40$ ms. Quindi il tempo complessivo per la trasmissione del pacchetto da H1 a H2 è:

$$51 \times 3 + 40 = 193 \text{ ms}$$

7. Tizio manda dal suo account di email

tizio@libero.it

un messaggio di posta elettronica con testo di 256 caratteri a Caio

caio@occupato.it

- Specificare il contenuto dei primi due messaggi TCP inviati dal mailserver di libero.it, ms.libero.it, per ricevere tale messaggio.
- Specificare, per ciascun segmento, payload, numero di sequenza, ack number, flags posti ad 1, porta origine e porta destinazione.

Soluzione:

- Il primo segmento avrà il payload vuoto e il payload del secondo conterrà 220 service ready.
- Il primo segmento sarà SYNACK, e quindi: come già detto, payload vuoto, flags SYN e ACK ad 1, numero sequenza Z, ack number Y, **porta mittente 25**, porta destinazione effimera. Il secondo segmento conterrà come payload 220 service ready, nessun flag ad 1, numero sequenza Z+1, ack number Y, porta mittente 25, porta destinazione effimera.

8. Dire in quali delle seguenti circostanze il TCP cambia la dimensione della finestra di congestione **cwnd**, e, nel caso, come viene ricalcolata.

Stato	Evento	Cambia la finestra?	Nuova dimensione della finestra
Slow Start	Ricezione di un ACK duplicato		
Slow Start	Scatta un timeout		
Congestion Avoidance	Scatta un timeout		

Soluzione:

Stato	Evento	Cambia la finestra?	Nuova dimensione della finestra
Slow Start	Ricezione di un ACK duplicato	NO	
Slow Start	Scatta un timeout	SI	CongWin = 1 MSS
Congestion Avoidance	Scatta un timeout	SI	CongWin = 1 MSS

9. Si consideri il seguente scenario TCP in cui, per semplicità, non sono indicati i segmenti inviati o re-inviati dal sender TCP al receiver TCP, che si suppone essere quelli necessari per avere i riscontri descritti di seguito.

- Al tempo t_0 il TCP di un host A ha una connessione già stabilita, per la quale ha 4 segmenti full sized in volo (inviati ma non riscontrati) e nessun nuovo dato da spedire, il primo byte dei segmenti in volo è il byte Y, $ssthresh = 6.5 \text{ MSS}$, $cwnd = 5 \text{ MSS}$. Inoltre, non ha ricevuto nessun riscontro duplicato.
- Tra il tempo t_0 e il tempo t_1 riceve 7 riscontri: i primi due non duplicati e con acknumber uguale a $Y + 1 \text{ MSS}$ per il primo, e $Y + 3 \text{ MSS}$ per il secondo. I seguenti 4 riscontri sono tutti duplicati, ed infine, al tempo t_1 , riceve un settimo riscontro con acknumber uguale a $Y + 4 \text{ MSS}$.

Si supponga che non scatti alcun timeout tra t_0 e t_1 . Indicare, per ciascun riscontro ricevuto, lo stato del TCP e i valori di $ssthresh$ e $cwnd$, giustificando la risposta.

Soluzione:

n.1	Riscontro	Cong Window	Threshold	Stato
1	Y+1 MSS	5+1 MSS	6,5 MSS	SS
2	Y+3 MSS	(5+1)+1 MSS*	6,5 MSS	CA
3-4	Y+3	7 MSS	6,5 MSS	CA
5	Y+3	3,5+3 MSS	3,5 MSS	FR
6	Y+3	7,5 MSS	3,5 MSS	FR
7	Y+4	3,5 MSS	3,5 MSS	CA

10. Descrivere in modo dettagliato e mediante uno **pseudocodice** le azioni svolte da un **destinatario TCP** per realizzare il controllo di flusso. Si assuma che ogni segmento ricevuto dal destinatario contenga anche lo pseudoheader (lunghezza segmento TCP). Inoltre, si supponga che la chiusura della connessione venga fatta dal mittente e che i segmenti di chiusura non contengano dati in piggybacking. Non occorre realizzare la parte iniziale delle azioni svolte (quindi le inizializzazioni delle variabili e l'apertura della connessione). Infine, si supponga che si invii un riscontro appropriato per ogni segmento ricevuto, e che il destinatario non debba inviare dati al mittente. Inoltre, per semplicità, si specificino solamente i campi dell'header del riscontro relativi al controllo del flusso e al riscontro. Non occorre realizzare le interazioni con il livello applicativo (processo che legge dal buffer), ma solamente quelle con il mittente TCP. Si hanno a disposizione le seguenti procedure:

```

receive(segm) //per ricevere il segmento segm dal livello di rete
OK(segm) //restituisce true solo se segm è corretto
nuovo(segm.x) //vettore di booleani che vale true se dal campo x di
segm si deduce che i dati contenuti in segm non sono doppioni
(specificare x nella soluzione)
insert(segm.y,finestra) //per inserire segm.y nel buffer di
ricezione nella posizione corretta
calcolacknum(segm) //restituisce il numero di riscontro per il
riscontro associato a segm
send(risp) //per inviare risp al mittente

```

Descrivere il contenuto o la funzionalità delle variabili e delle altre funzioni o procedure eventualmente utilizzate.

Soluzione:

```

finito = false;
while (! finito) {
receive(segmn);
finito = segm.FIN;
#controllare inizio della chiusura della conn.
if (! finito) {
    #se segmento non corrotto
    if (OK(segmn)) {
        #se il segmento contiene nuovi dati
        if (nuovo(segmn.seqnum)) {
            #inserisco i dati nel buffer
            insert(segmn.dat, finestra);
            #nuovo valore di rwnd
            rwnd = rwnd - (segmn.lungtotTCP - segmn.HLEN);
            #flag ack settato a true
            risposta.ACK = true;
            risposta.ACKnum = calcolacknum(segmn);
            risposta.rwnd = rwnd;
        }
        #invia o il nuovo ack o ultimo ack inviato
        send(risposta);
    }
}
}

```

11. Discutere l'affermazione: *poiché FTP e HTTP sono protocolli adatti a trasferire file, possono essere usati indifferentemente.*

Soluzione possibile:

Esempio di risposta (schema per punti):

- Descrivere brevemente obiettivo HTTP e FTP.
 - Entrambi usano TCP (trasferimento affidabile dei dati), elencare le differenze ai fini del trasferimento file.

e.g. **FTP**

- connessione controllo, persistente, inizializzata dal client, porta 21. comandi in formato ASCII a 7 bit
 - connessione dati, inizializzata da server, porta 20, non persistente
 - FTP è Stateful
 - Offre funzionalità aggiuntive rispetto a HTTP per la gestione di file e directory (list, retr, put).

e.g. HTTP

- Un'unica connessione per dati e comandi.
 - Interazione stateless.

3.4 Lo strato di rete



Figura 3.75: Lo strato di rete all'interno dello stack protocollare TCP/IP.

Nello stack protocollare TCP/IP **il livello rete** è collocato tra **il livello trasporto** e **il livello collegamento**, offre servizi al livello trasporto e ne **riceve** dal livello collegamento. Realizza una connessione logica fra host system diversi facenti parte di reti eterogenee. Si occupa di:

- **Suddividere i dati in pacchetti, frammentandoli** nel caso in cui il payload sia troppo grande per entrare in un singolo pacchetto.

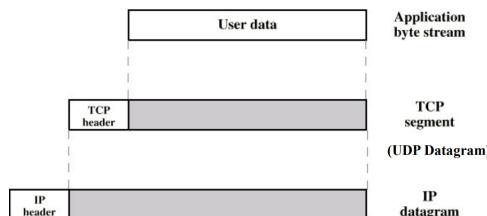


Figura 3.76: Un pacchetto IP.

- **Instradare i pacchetti** così ottenuti dall'host sorgente all'host destinazione. La vastità di Internet, infatti, fa sì che con ogni probabilità sia possibile scegliere **più di un percorso**. Il livello di rete deve trovare **il migliore** tra questi.

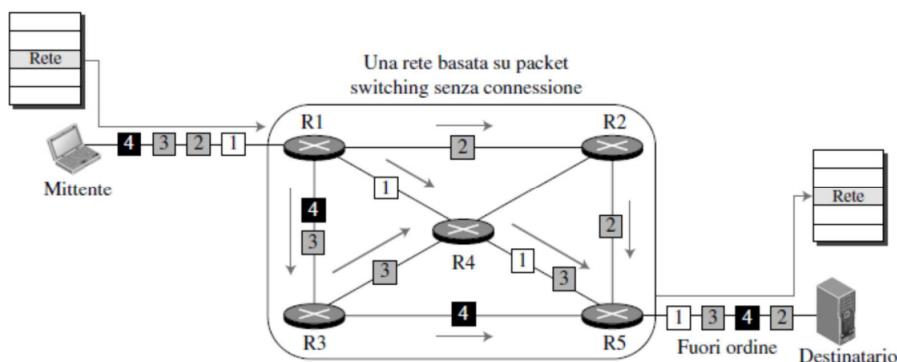


Figura 3.77: Il processo di instradamento in una rete priva di connessione.

- **Inoltrare i pacchetti**, ovvero trasferire il pacchetto arrivato al router sull'**appropriato** collegamento di uscita.

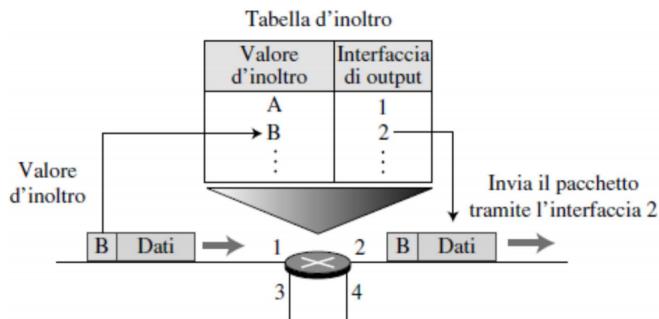


Figura 3.78: Il processo d'inoltro in una rete priva di connessione. La tabella d'inoltro è il risultato del processo di instradamento.

D'altro canto **il livello di rete non si occupa di**:

- **Controllare gli errori nel payload dei pacchetti**; sebbene nel pacchetto sia presente un campo **checksum** la sua funzione è di rilevare errori **solo nell'intestazione**. Inoltre i pacchetti del livello rete possono essere **frammentati** da ciascun router a cui giungono, un eventuale controllo degli errori sul payload sarebbe molto **inefficiente**.
- **Controllare il flusso**; il controllo di flusso è delegato del tutto agli strati superiori, in Internet il livello di rete non fornisce **pressoché** alcun meccanismo di controllo di flusso.
- **Controllare la congestione**; al livello rete non è implementato alcun controllo della congestione.
- **Controllare la qualità del servizio**; tutte le tecniche di controllo della qualità del servizio sono implementate **a livelli superiori**.
- **Offrire sicurezza**; quando il livello di rete è stato progettato la sicurezza non era un problema.

Per la trasmissione dei dati è usata la tecnica di **packet switching**, sia con **approccio a datagramma** che a **circuito virtuale**, l'approccio dipende dalla tecnologia della rete.

- **Approccio a datagramma**: offre un servizio **connection-less**, ogni datagramma è un'unità **indipendente** che viaggia sulla rete e il suo **inoltro** si basa **solamente sull'indirizzo di destinazione** contenuto nella sua **intestazione**. I datagrammi che compongono un messaggio **possono** viaggiare tutti sullo **stesso** percorso o seguire ognuno una strada completamente **diversa**. Il livello di rete di **Internet** è stato progettato per essere in grado di fornire un servizio **connection-less**.

- **Approccio a circuito virtuale:** offre un servizio connection-oriented. Viene stabilita una **connessione logica** prima dell'invio dei datagrammi che compongono il messaggio. I **datagrammi** seguono tutti lo **stesso percorso** all'interno della rete e oltre agli indirizzi sorgente e destinazione, utili nel caso i cui il datagramma debba attraversare una rete che non offre un servizio connection-oriented, contengono anche un'**etichetta di flusso** che identifica il **circuito virtuale** che a sua volta definisce il **percorso virtuale** che il datagramma deve seguire. L'inoltro del datagramma di base quindi sulla sua etichetta di flusso. Per implementare un servizio **connection-oriented** si utilizza un processo composto da **tre fasi**:

1. **Setup:** creazione del circuito virtuale, configurazione dei router intermedi e creazione delle tabelle d'inoltro.
2. **Data transfer:** trasferimento dei dati.
3. **Teardown:** distruzione del circuito virtuale.

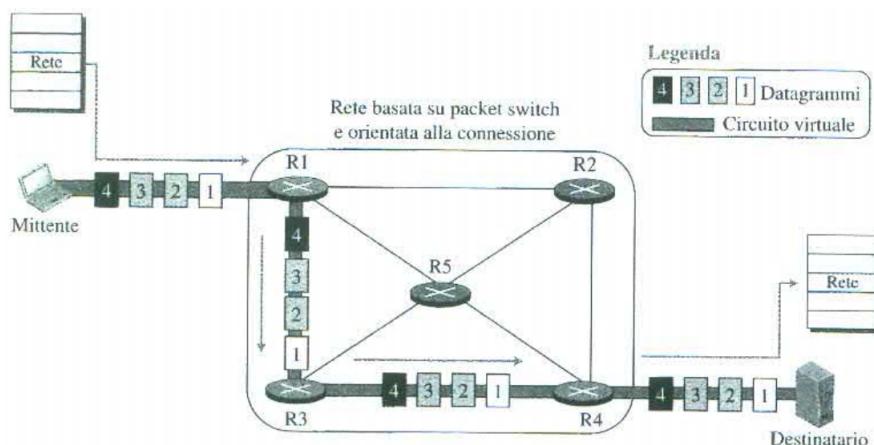


Figura 3.79: Il processo d'istradamento in una rete connection-oriented.

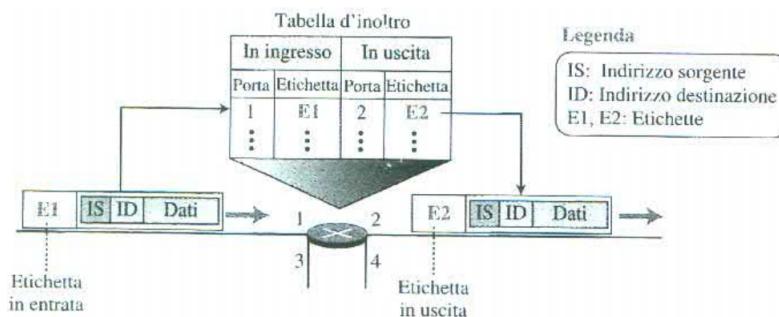


Figura 3.80: Il processo d'inoltro in una rete connection-oriented.

3.4.1 Internet Protocol

Il protocollo IP è un protocollo di tipo **connection-less**, sono perciò assenti sia il circuito virtuale che quello fisico fra i due sistemi terminali. Adotta una **politica di invio** dei datagrammi di tipo **send and pray**, senza controllo degli errori, non garantisce il controllo di flusso e presenta una **politica di gestione della qualità** del servizio di tipo **best effort** ovvero con prestazioni estremamente variabili.

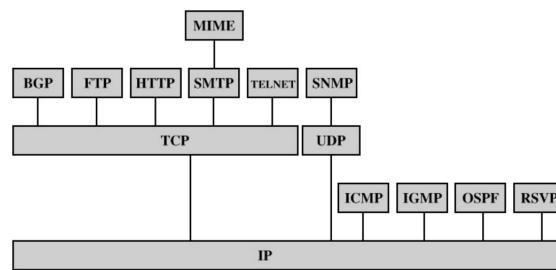


Figura 3.81: I protocolli di livello rete dello stack protocollare TCP/IP. In realtà il protocollo RSVP è un protocollo di livello trasporto.

Nel corso del tempo i **protocolli di livello rete** hanno avuto diverse versioni, di particolare interesse sono la **versione 4** e la **versione 6**. L'IP è responsabile della **suddivisione** in pacchetti, dell'**inoltro** e della **consegna** dei datagrammi. Sono inoltre presenti altri 4 **protocolli ausiliari**:

1. L'**Internet Control Message Protocol** o **ICMP** aiuta a gestire alcuni **errori** che possono avvenire nell'inoltro dei datagrammi.
2. L'**Internet Group Management Protocol** o **IGMP** supporta l'IP nella gestione del **multicasting**.
3. L'**Open Shortest Path First** o **OSPF** è un protocollo di **routing**.
4. Il **Resource Reservation Protocol** o **RSVP** è un protocollo di **livello trasporto**, non trasporta dati applicativi bensì implementa un meccanismo, dal lato del ricevente, per riservare risorse lungo la rete.

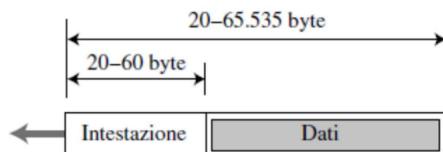


Figura 3.82: La struttura di un datagramma IP.

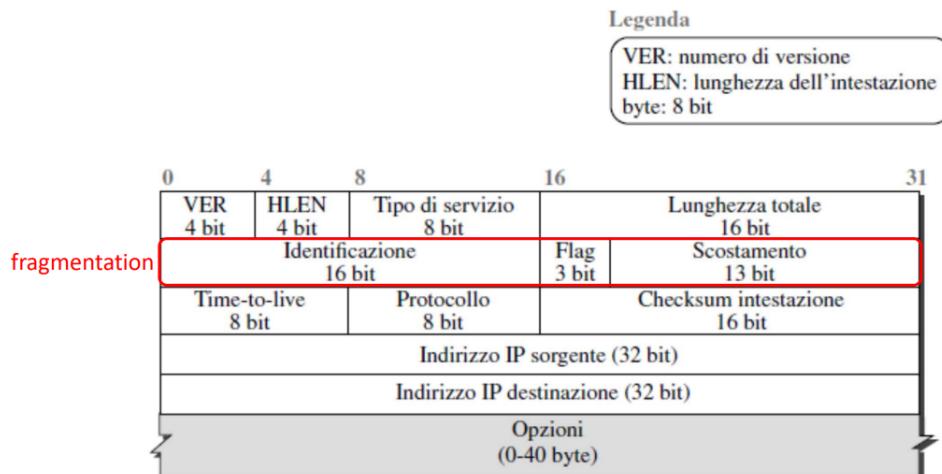


Figura 3.83: Un datagramma IP. I campi evidenziati in rosso riguardano la frammentazione dei datagrammi.

Un **datagramma IP** è un pacchetto di lunghezza variabile composto da due parti: **un'intestazione o header** e **un campo dati o payload**. L'intestazione è lunga da **20** a **60** byte e contiene le informazioni fondamentali per il **routing** e la **consegna** dei datagrammi.

- **Numero di versione**: definisce la versione del protocollo IP, attualmente: **v4** o **v6**.
- **Lunghezza dell'intestazione**: definisce la lunghezza totale dell'intestazione del datagramma misurata in **parole** formate da **4 byte** per via della lunghezza del campo stesso, ovvero 4 bit. L'effettiva lunghezza totale corrisponde al valore di questo campo **moltiplicato per 4**.
- **Tipo di servizio**: campo riservato per i servizi differenziati.
- **Lunghezza totale**: definisce la **lunghezza totale** del **datagramma**.
- **Identificazione, Flag e Scostamento**: questi tre campi riguardano la **frammentazione** del datagramma IP che **avviene** quando la sua **dimensione** è **maggiori** rispetto alla **capacità di trasporto** del **livello di collegamento** sottostante.
- **Time To Live**: campo utilizzato per definire il **numero massimo di salti** (hops) che un datagramma può effettuare da un router all'altro. Il suo **scopo** è quello di **eliminare dalla rete** il traffico superfluo causato da **datagrammi instradati male**. È pari a circa **il doppio del numero massimo di salti tra due host qualsiasi** presenti su Internet. Ogni router decremente il valore del campo di una unità, **quando è pari a 0** il prossimo router **scarta il datagramma**.

Protocollo: è il campo contenente un numero di 8 bit, **individua il protocollo che fa uso di IP**. I numeri sono assegnati **univocamente**. Quando il datagramma arriva a destinazione il valore di questo campo permette di individuare il protocollo a cui consegnare il payload del datagramma. Il campo fornisce **multiplexing alla sorgente e demultiplexing alla destinazione**. Per il livello rete ha quindi una funzione molto simile ai numeri di porta del livello trasporto.

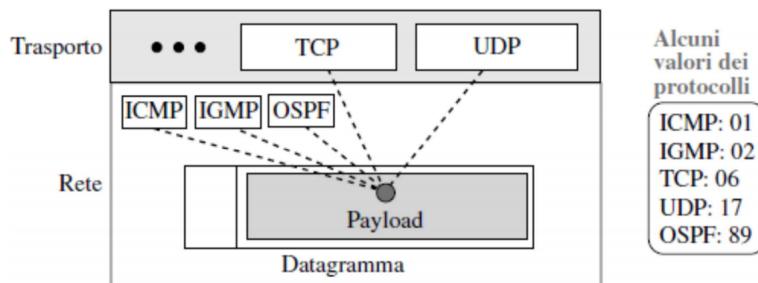


Figura 3.84: Multiplexing e demultiplexing utilizzando il valore del campo protocollo.

- **Checksum dell'intestazione:** il protocollo IP **non** si cura di **individuare** nè tanto meno di **correggere eventuali errori del payload dovuti alla trasmissione**. Tuttavia l'**intestazione** dei datagrammi viene **aggiunta**, durante il processo di encapsulamento, **al livello rete** ed è perciò **responsabilità dell'IP** **verificarne la correttezza**. Inoltre gli **errori** nell'**intestazione** possono comportare la consegna dei datagrammi a host e/o protocolli **sbagliati**. Tali ragioni giustificano la presenza del campo checksum per l'intestazione. **La checksum va ricalcolata in ogni router** a causa della possibile **variazione dei campi dell'intestazione** come il TTL o i campi relativi alla frammentazione.
- **Indirizzi sorgente e destinazione:** contengono rispettivamente l'indirizzo dell'host **sorgente** e quello dell'host **destinatario**. L'host sorgente dovrebbe conoscere a priori il proprio indirizzo IP, mentre quello dell'host destinatario è noto al protocollo di livello superiore che sta usando l'IP. Il valore di questi campi **non dovrebbe cambiare** per tutta la durata della vita del datagramma per ovvie ragioni.
- **Opzioni:** contiene opzioni che possono essere usate per il test o il debug della rete.
- **Payload:** contiene i dati trasportati dal datagramma. Il **payload** è il pacchetto che deriva dagli altri protocolli che fanno uso del servizio offerto da IP, siano essi del livello di trasporto o del livello di rete, come ad esempio ICMP.

Frammentazione

Per giungere a destinazione un **datagramma IP** potrebbe dover attraversare varie reti, ognuna con caratteristiche differenti. Supponiamo ad esempio che il datagramma sia arrivato a un generico router, a quel punto verrà estratto dal frame di livello collegamento, elaborato, **incapsulato in un altro frame** e poi **rispedito**. Il formato e la dimensione del frame in entrata nel router **dipendono** dalla **tecnologia fisica** e da quella di **collegamento** utilizzati per trasportare il frame, analogamente, formato e dimensione in uscita dipenderanno dalle caratteristiche di livello fisico e di livello collegamento della rete su cui il frame verrà inviato.

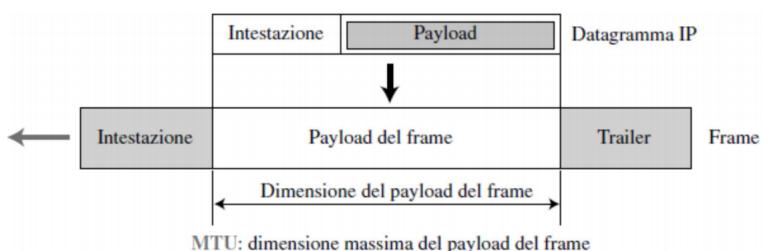


Figura 3.85: Maximum Transfer Unit o MTU.

Ogni protocollo di livello collegamento ha il suo formato di frame e ogni formato di frame ha il suo **MTU** che indica la **dimensione massima del payload che può essere incapsulato nel frame**. In altre parole la dimensione massima dell'intero datagramma IP deve essere minore o uguale all'MTU del livello di collegamento sul quale verrà inoltrato. Quando ciò non dovesse avvenire il datagramma viene **frammentato** in parti più piccole per poter essere trasportato. **N.B.** I frammenti così ottenuti hanno **una propria intestazione**, possono seguire **percorsi diversi** sulla rete e possono essere a loro volta frammentati da qualsiasi entità IP, tuttavia il **riassemblaggio** avviene solo all'host di destinazione, se manca qualche frammento l'intero datagramma viene scartato.

L'host o il router che frammenta il datagramma IP deve, per l'intestazione di ogni frammento, cambiare i seguenti campi: **flag**, **scostamento** e **lunghezza totale** e copiare il resto dei campi del datagramma originale.

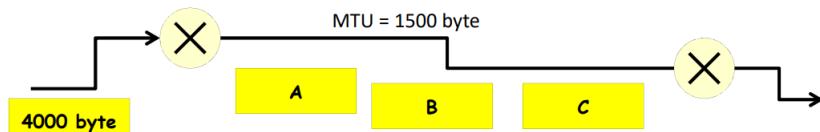
- **Identificazione:** è un numero positivo di 16 bit associato al datagramma dall'host sorgente, **IP utilizza un contatore per etichettare i datagrammi**. La combinazione dell'**identificazione** e degli **indirizzi IP sorgente e destinazione** identifica il datagramma per un tempo ragionevolmente lungo. Il valore del campo **identificazione** viene copiato in tutti i frammenti, quindi **tutti i frammenti** di un messaggio hanno lo **stesso valore** per il campo **identificazione**.

- I tre bit di Flag definiscono tre flag distinti:
 1. Il **bit 0 è riservato**.
 2. Il **bit 1, do not fragment**, vale 0 se il datagramma può essere frammentato e 1 se non deve essere frammentato, in tal caso il dispositivo **scarta** il datagramma se non è in grado di trasmetterlo lungo nessuna rete fisica a cui è collegato e invia un messaggio d'errore ICMP all'host sorgente.
 3. Il **bit 2, more fragments**, vale 0 se il frammento è l'ultimo del datagramma o 1 se non è l'ultimo e ci sono altri frammenti dopo di lui.
- **Scostamento di frammentazione:** mostra la posizione relativa del frammento rispetto al datagramma originario.

Esprime l'offset dei dati del frammento nel datagramma originario misurato in unità di 8 byte. I 13 bit del campo non consentono di rappresentare numeri maggiori di 8191 ed essendo la lunghezza massima del datagramma IP 65535 byte **la divisione per 8 è necessaria**.

Ciò comporta che **l'host sorgente o i router che frammentano il datagramma debbano scegliere la dimensione dei frammenti in modo tale che sia un multiplo di 8**, ad eccezione dell'ultimo frammento.

Nel caso in cui avvengano **multiple frammentazioni** lo **scostamento di frammentazione** è sempre **relativo al datagramma originario**.



Arriva un pacchetto di 4000 byte (header IP incluso).
Considerando 20 byte di intestazione...

Framm.	byte	ID	offset	flag
A	1480	667	0	1
B	1480	667	185 (1480/8)	1
C	1020	667	370 (2960/8)	0<

Figura 3.86: Un esempio di frammentazione, l'offset è espresso in unità di 8 byte.

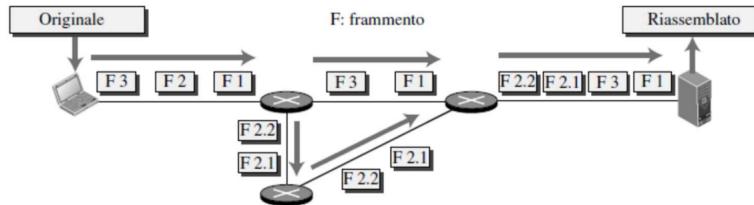
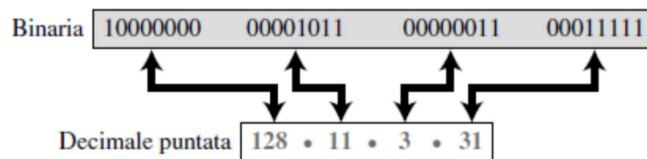


Figura 3.87: I frammenti potrebbero seguire un percorso alternativo.

Indirizzi

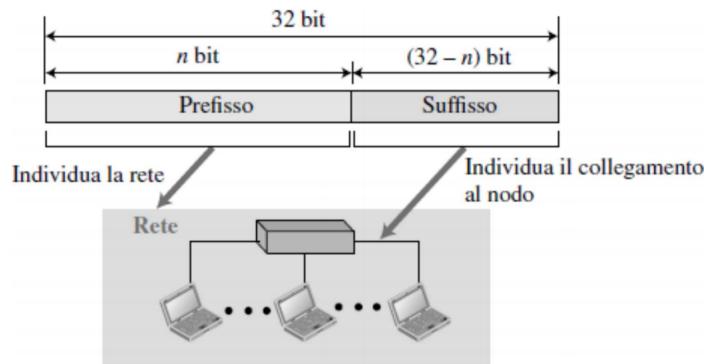
La versione 4 del protocollo IP, comunemente chiamata **IPv4**, usa **indirizzi a 32 bit o 4 byte**, ciò comporta che lo spazio degli indirizzi abbia dimensione pari a 4.294.967.296 indirizzi possibili.

Ogni host è connesso ad Internet attraverso **un'interfaccia di rete**, che marca il confine fra l'host ed il collegamento su cui vengono inviati i datagrammi. Ad ogni interfaccia è **assegnato un indirizzo IP**. I router sono **dispositivi intermedi che devono necessariamente essere connessi ad almeno due collegamenti e per questo motivo hanno almeno due interfacce di rete**.



Come gli indirizzi di ogni altra rete di comunicazione anche **gli indirizzi IPv4 hanno una struttura gerarchica** che si compone di **due parti**:

1. **Prefisso o network ID**: identifica una rete IP su internet.
2. **Suffisso o host ID**: identifica l'host sulla rete.



Originariamente l'indirizzo IPv4 era stato progettato con un prefisso di **lunghezza fissa**, ma la necessità di identificare **reti di varie dimensioni** ha portato all'introduzione di un prefisso a **lunghezza variabile**. Sono state previste **tre lunghezze di prefissi** e l'intero **spazio degli indirizzi** è stato **diviso in cinque classi**.

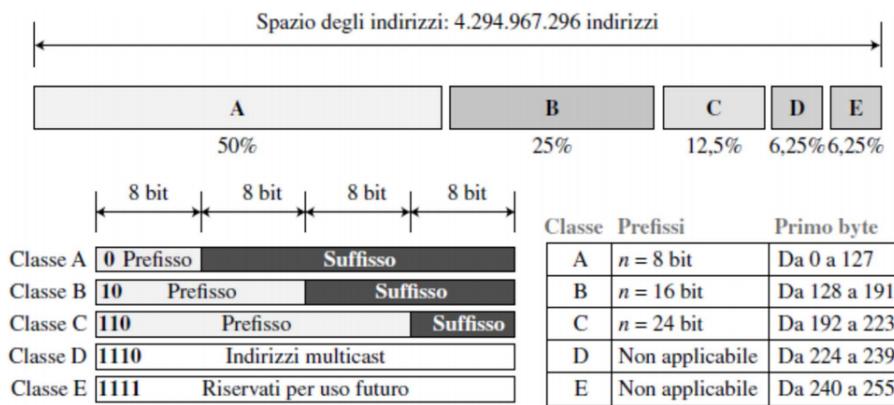


Figura 3.88: Class addressing.

Nella **classe A** la lunghezza della parte di rete è **8 bit** ma poichè il primo bit, che è 0, identifica la classe si hanno solo **7 bit per identificare 128 reti e 24 bit per circa 16 milioni di host**. Gli indirizzi di classe A vanno dal 0.0.0.0 al 127.255.255.255.

Nella **classe B** la lunghezza della parte di rete è di 16 bit ma poichè i primi due sono uguali a $(10)_2$ e identificano la classe si hanno **14 bit per identificare le reti**, circa 16000, e **16 bit per circa 64.000 host**. Gli indirizzi di classe B vanno dal 128.0.0.0 al 191.255.255.255.

Nella **classe C** la lunghezza della parte di rete è di 24 bit, i primi tre bit identificano la classe e sono uguali a $(110)_2$. Si hanno **21 bit per le reti IP e 8 bit per 256 host**. Gli indirizzi di classe C vanno dal 192.0.0.0 al 223.255.255.255.

La **classe D** è riservata per il **multicasting**. I suoi indirizzi vanno dal 224.0.0.0 al 239.255.255.255. La **classe E** è riservata per **uso futuro**. I suoi indirizzi vanno dal 240.0.0.0 al 255.255.255.255.

Analizziamo alcuni indirizzi per fare un esempio: **15.10.10.90**: si tratta di un indirizzo di classe A, poichè il primo numero è compreso fra 0 e 127. Dunque i seguenti campi indicano l'host: 10.10.90 nella rete IP: 15.

130.250.42.53: si tratta di un indirizzo di classe B, poichè il primo numero è compreso fra 128 e 191. Dunque i primi due campi indicano la rete IP: 130.250, gli altri indicano l'host: 42.53.

196.234.12.14: si tratta di un indirizzo di classe C, poichè il primo numero è compreso fra 192 e 223. Dunque i primi tre campi indicano la rete IP 196.234.12, e l'ultimo indica l'host: 14.

L'**indirizzamento con classi**, sebbene consenta di risalire al prefisso senza l'aggiunta di ulteriori informazioni, è **diventato obsoleto** a causa dello **scorretto assegnamento degli indirizzi** e del **conseguente esaurimento** degli stessi. È chiaro che la **soluzione a lungo termine** sia di **adottare degli indirizzi più lunghi** (IPv6). Ciò comporterebbe però di modificare anche la **struttura dei datagrammi IP**.

Per tale motivo e anche a causa dell'affermarsi degli **Internet Service Providers** durante gli anni '90, si è ideata, oltre alla soluzione a lungo termine, anche una **soluzione a breve termine** facente sempre uso dell'**indirizzo IPv4** ma non dell'**indirizzamento con classi**.

Gli ISP sono organizzazioni che forniscono accesso a Internet a persone o altre organizzazioni il cui interesse è solo quello di utilizzare i servizi offerti ma non di far parte di Internet o di offrire a loro volta servizi. Un ISP ottiene un insieme molto ampio di indirizzi e lo suddivide in gruppi.

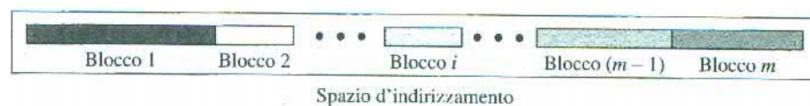


Figura 3.89: Indirizzamento senza classi, con spazio degli indirizzi diviso in blocchi di lunghezza variabile.

Tale soluzione, chiamata **indirizzamento senza classi**, prevede che lo spazio degli indirizzi venga suddiviso in blocchi di lunghezza variabile. Il **prefisso di un indirizzo individua il blocco (la rete)** e il **suffisso individua il dispositivo**.

Un **prefisso grande** individua una **rete più piccola** e viceversa un **prefisso piccolo** individua una **rete più grande**.

Per individuare il **prefisso**, data l'assenza delle classi, è stata aggiunta all'indirizzo una barra slash "/" che separa l'indirizzo dalla lunghezza del prefisso. Tale notazione è definita **CIDR** o **Class-less Inter Domain Routing**.

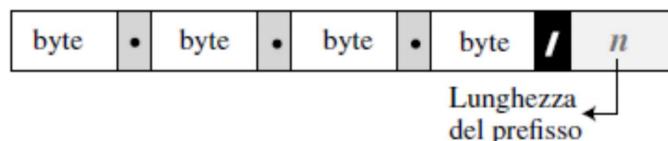


Figura 3.90: CIDR.

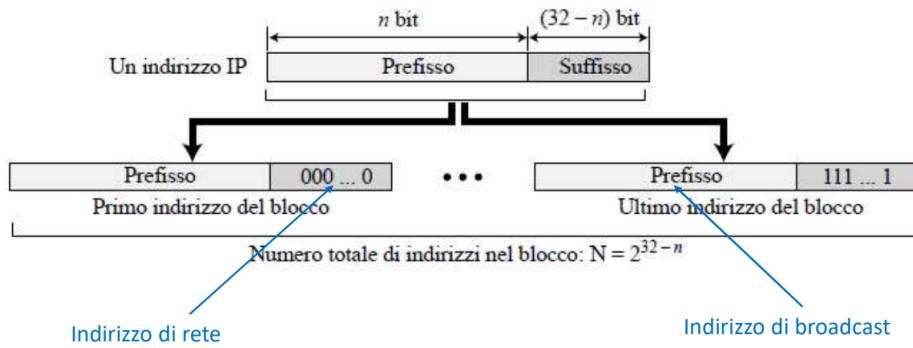


Figura 3.91: Estrazione delle informazioni nell’indirizzamento senza classi.

1. Il **numero degli indirizzi nel blocco** è $N = 2^{32-n}$.
2. Il **numero di host** è dato da $N - 2$, devo sottrarre l’indirizzo di rete e l’indirizzo di broadcast. [*spiegazione avanti*]
3. Il **primo indirizzo** è l’**indirizzo di rete** ed è ricavato tenendo invariati i primi n bit e azzerando tutti gli altri.
4. L’**ultimo indirizzo** è l’**indirizzo di broadcast** ed è ricavato tenendo invariato i primi n bit e impostando ad 1 tutti gli altri.

Un altro modo per ricavare il primo e l’ultimo indirizzo del blocco è usare la **maschera dell’indirizzo**, ovvero un numero di 32 bit in cui i **primi n bit** sono impostati a 1 e gli altri $32 - n$ a 0, con **n lunghezza del prefisso**.

La **maschera dell’indirizzo** o **subnet mask** può essere utilizzata molto **efficientemente** da un **programma** per ricavare le informazioni di un blocco usando tre operatori sui bit: **NOT**, **AND** e **OR**.

1. Il **numero degli indirizzi nel blocco** è:

$$N = \text{NOT}(\text{maschera}) + 1.$$

2. Il **primo indirizzo del blocco** è:

(qualsiasi indirizzo del blocco) **AND** (maschera)

3. L’**ultimo indirizzo del blocco** è:

(qualsiasi indirizzo del blocco) **OR** (**NOT**(maschera))

L’**indirizzo di rete** è usato per instradare i datagrammi.

Esempio:

All'indirizzo IP: **150.217.8.42** è definita la netmask **255.255.255.0**.

La rete o subnet effettiva ha indirizzo **150.217.8.0/24**.

```
Indirizzo IP 150.217.8.42 10010110 11011001 00001000 00101010
Subnet Mask 255.255.255.0 11111111 11111111 11111111 00000000
AND           150.217.8.0 10010110 11011001 00001000 00000000
```

Gli indirizzi **150.193.4.1/22**, **150.193.8.18/22**, **150.193.11.3/22** appartengono alla stessa sottorete o subnet?

```
Indirizzo IP 150.193.4.1 10010110 11000001 00000100 00000001
Subnet Mask 255.255.252.0 11111111 11111111 11111100 00000000
AND           150.217.8.0 10010110 11000001 00000100 00000000
```

```
Indirizzo IP 150.193.8.18 10010110 11000001 00001000 00010010
Subnet Mask 255.255.252.0 11111111 11111111 11111100 00000000
AND           150.217.8.0 10010110 11000001 00001000 00000000
```

```
Indirizzo IP 150.193.11.3 10010110 11000001 00001011 00000011
Subnet Mask 255.255.252.0 11111111 11111111 11111100 00000000
AND           150.217.8.0 10010110 11000001 00001000 00000000
```

Il primo indirizzo **non** vi appartiene.

Esiste un modo più adatto a noi esseri umani? Sì, basta che **confrontiamo i primi 22 bit degli indirizzi** per arrivare alla stessa conclusione.

```
Indirizzo IP 150.193.4.1 10010110 11000001 000001 | 00 00000001
Indirizzo IP 150.193.8.18 10010110 11000001 000010 | 00 00010010
Indirizzo IP 150.193.11.3 10010110 11000001 000010 | 11 00000011
```

Altro esempio:

Indirizzo IP: **167.199.170.82/27**, maschera: **255.255.255.224**. Si ha che:

```
Primo indirizzo del blocco o i. di rete = (address) AND (mask)
10100111 11000111 10101010 01010010 AND
11111111 11111111 11111111 11100000
10100111 11000111 10101010 01000000
```

```
Ultimo indirizzo del blocco o i. di broadcast = (address) OR (NOT(mask))
10100111 11000111 10101010 01010010 OR
00000000 00000000 00000000 00011111
10100111 11000111 10101010 01011111
```

Curiosità

Per quale motivo UDP e TCP si preoccupano di controllare l'integrità di informazioni di livello network se anche **IP usa un suo checksum**?

Stando a **David Patrick Reed**, noto per essere l'ideatore del protocollo UDP nonché una delle principali menti dietro allo stack protocollare TCP/IP:

As I was there in 1976, when we split TCP into IP, TCP, and other protocols, such as UDP, for the decision to separate the checksums and to create a pseudo-header, here is the rationale, which is highly relevant.

TCP and UDP are end-to-end protocols. In particular, the TCP checksum is "end-to-end". It is a "private matter" between end points implementing the TCP layer, guaranteeing end-to-end reliability, not hop-by-hop reliability.

IP is a wrapper for TCP, which instructs the transport layer, the gateways and routers, where the packet is to be transported, how big it is, and how it may be fragmented in the process of delivery.

The Source Address, Destination address, length, etc. are part of the meaning of the TCP frame - in that the end point machines use that information in the TCP application.

Thus the function of source address, destination address, etc. are "shared" because they are meaningful to both layers IP and TCP. Rather than include the same information twice in the packet format, the concept of a "virtual header" was invented to encapsulate the idea that IP is not allowed to change the SA and DA because they are meaningful.

Further, in the case of end-to-end encryption, in 1976 we had a complete design by Steven T. Kent, my office mate, which was blocked by NSA from being deployed, it is essential that all end-to-end meaning be protected. The plan was to leave the source address and destination address in the clear, but encrypt the rest of the TCP payload, including the checksum. This would protect against a man-in-the-middle attack that delivered valid packets with an incorrect source address.

This was a careful design decision, wrecked irrevocably by the terrorists who invented NAT, which doesn't allow end-to-end encryption, because NAT is inherently a "man-in-the-middle" attack!

The rise of the middleboxen have now so thoroughly corrupted the Internet protocol design that it's not surprising that the original designs are difficult to decode. If we actually had end-to-end encrypted TCP, now impossible because of the NATs, we would have a much more secure and safe Internet, while preserving its open character. Instead we have a maze of twisty, disconnected passages, vulnerable to a zillion hackers.

Assegnazione dei blocchi di indirizzi

Uno dei **principali problemi** dell'**indirizzamento senza classi** è rappresentato dall'**assegnazione dei blocchi**. La responsabilità di questo compito ricade su un ente globale chiamato **ICANN** o **Internet Corporation for Names and Numbers**. L'**ICANN** non assegna gli indirizzi ai singoli utenti bensì **assegna grandi blocchi di indirizzi a ISP** o organizzazioni di dimensione comparabile.

Per il corretto funzionamento del CIDR bisogna operare due restrizioni sui blocchi assegnati:

1. Il **numero degli indirizzi assegnati N deve essere una potenza di 2**. N sarà uguale a 2^{32-n} e $n = 32 - \log_2 N$. Se N non fosse una potenza di 2 n non sarebbe un numero intero.
2. Il blocco assegnato deve essere uno spazio libero **contiguo** e **il primo indirizzo deve essere divisibile per il numero degli indirizzi nel blocco**. La ragione è che il primo indirizzo deve essere composto dal prefisso seguito da $32 - n$ zeri e sarà uguale a: $(prefisso)_{10} \times N$

Esempio:

Supponiamo che un ISP richieda un blocco 190 indirizzi.

Ne otterrà 256, poiché 2^8 è la potenza di 2 immediatamente maggiore di 190. Supponiamo che ottenga il seguente blocco: **14.24.74.0/24**

L'ISP vuole ora partizionare il blocco ottenuto in **tre sottoblocchi** da 120, 60 e 10 indirizzi, rispettivamente. Come può disegnare tali sottoblocchi?

Partiamo dal sottoblocco più grande, quello da 120 indirizzi. La potenza di 2 maggiore di 120 è $2^7 = 128$. Quindi $32 - 7 = 25$ bit per la rete. Il blocco è 14.24.74.0/25. Il primo indirizzo 14.24.74.0 e l'ultimo sarà 14.24.74.127.

00001110.00011000.01001010.00000000

Secondo sottoblocco $60 < 2^6 = 64$. Quindi $32 - 6 = 26$ bit per la rete. Il blocco è 14.24.74.128/26. Il primo indirizzo sarà 14.24.74.128 e l'ultimo sarà 14.24.74.191.

00001110.00011000.01001010.10000000

Terzo sottoblocco $10 < 2^4 = 16$. Quindi $32 - 4 = 28$ bit per la rete. Il blocco è 14.24.74.192/28. Il primo indirizzo sarà 14.24.74.192 e l'ultimo sarà 14.24.74.207

00001110.00011000.01001010.11000000

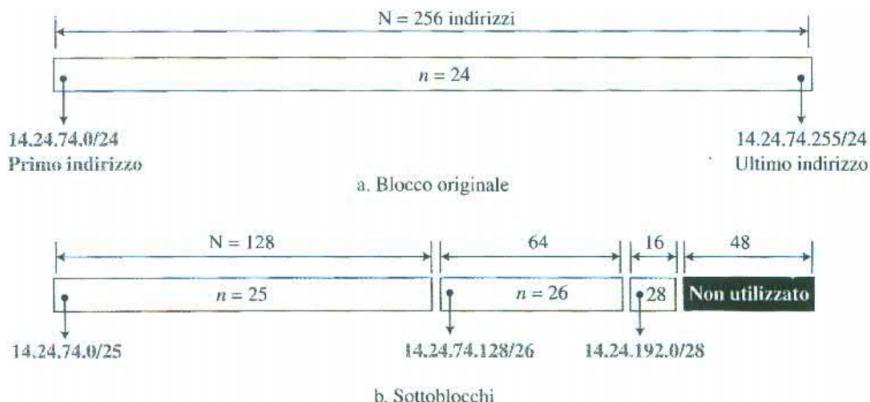


Figura 3.92: Il blocco dell'esempio di prima partizionato.
L'indirizzo dell'ultimo blocco in figura è sbagliato.

Aggregazione degli indirizzi

Per contenere il numero degli indirizzi di rete è la possibile **aggregare gli indirizzi** in vari blocchi per ottenere un blocco più grande. L'instradamento può poi essere effettuato usando il prefisso del blocco aggregato in tempo molto minore.

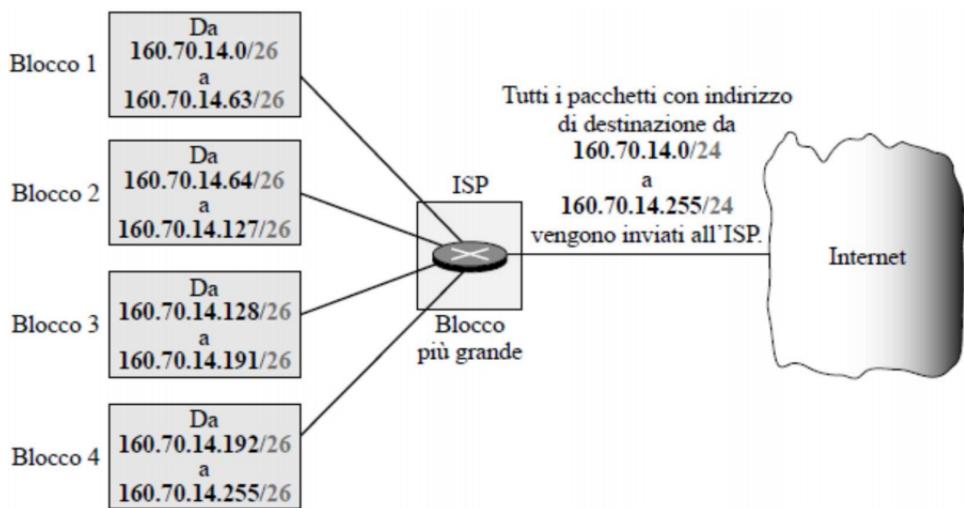


Figura 3.93: Un ISP ha assegnato a 4 organizzazioni piccoli blocchi di indirizzi. L'ISP aggrega i 4 blocchi in 1 singolo blocco e annuncia al resto della rete solo il blocco aggregato. Sarà poi sua responsabilità inoltrare i datagrammi all'organizzazione corretta.

Indirizzi speciali

- **0.0.0.0 this-host**, è usato quando un host ha necessità di inviare un datagramma ma non conosce il proprio indirizzo IP.
- **127.0.0.1 loopback**, il datagramma con questo indirizzo di destinazione non lascia l'host locale. Per test e debug.
- **255.255.255.255 limited broadcast**, usato quando un router o un host devono inviare un datagramma a tutti i dispositivi che si trovano all'interno della loro rete. I router bloccano la propagazione alla sola rete locale.
- **10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 169.254.0.0/16**, quattro blocchi di indirizzi privati per reti locali.
- **Blocco 224.0.0.0/4**, indirizzi multicast.

DHCP

Una volta che un blocco è stato assegnato a un'organizzazione l'amministratore di rete può assegnare **manualmente** gli indirizzi ai singoli host o router. L'assegnazione degli indirizzi può però anche essere **automatizzata** mediante il protocollo **DHCP** o **Dynamic Host Configuration Protocol**. Il DHCP è un programma **client-server** di livello **applicazione**, viene usato per **assegnare indirizzi IP** permanentemente agli host e ai router di una rete, o anche temporaneamente, a richiesta degli host.

Quest'ultima funzione è particolarmente **utile nel caso in cui** ad esempio si voglia fornire un servizio a 4000 persone avendo però solo 1000 indirizzi, assumendo che non più di un quarto delle persone si connetta contemporaneamente.

Oltre all'indirizzo IP **DHCP** fornisce all'host anche il **prefisso della propria rete**, **l'indirizzo del router di default** e **l'indirizzo IP del server dei nomi di dominio**. Illustriamone il funzionamento:

1. Il **client DHCP** dell'host che vuole entrare in rete crea un messaggio **DCHPDISCOVER** nel quale imposta, non avendo altre informazioni, solo il campo **transaction id** ad identificare la richiesta. **Tutti i messaggi DCHP vengono incapsulati in un datagramma UDP** con p.s 68 (client) e p.d. 67 (server) **ed esso, successivamente, in un datagramma IP**. In questo caso l' i.s vale 0.0.0.0 e i.d.d. 255.255.255.255.
2. Il **server DHCP** risponde un messaggio **DCHPOFFER** contenente l'indirizzo IP offerto al client, l'indirizzo IP del server e un **lease time**, il tempo a disposizione dell'host per usare l'indirizzo IP assegnatogli.

L'i.d.d. è ancora l'indirizzo di broadcast in modo tale che **anche gli altri server DHCP ricevano l'offerta e nel caso ne inviano una migliore** qual'ora possano.

3. Il **client**, che potrebbe aver ricevuto più offerte, a questo punto **sceglie l'alternativa migliore** e invia un messaggio **DHCPREQUEST** al server che l'ha proposta.

L'i.d.d. è sempre quello di broadcast **per comunicare agli altri server che la loro offerta non è stata accettata**.

4. Il server a questo punto, se l'indirizzo IP offerto è ancora valido, risponde con un messaggio **DHCPACK** altrimenti con un messaggio **DHCPNACK**. L'i.d.d. è sempre il broadcast **per comunicare agli altri server che la richiesta è stata o meno soddisfatta**.

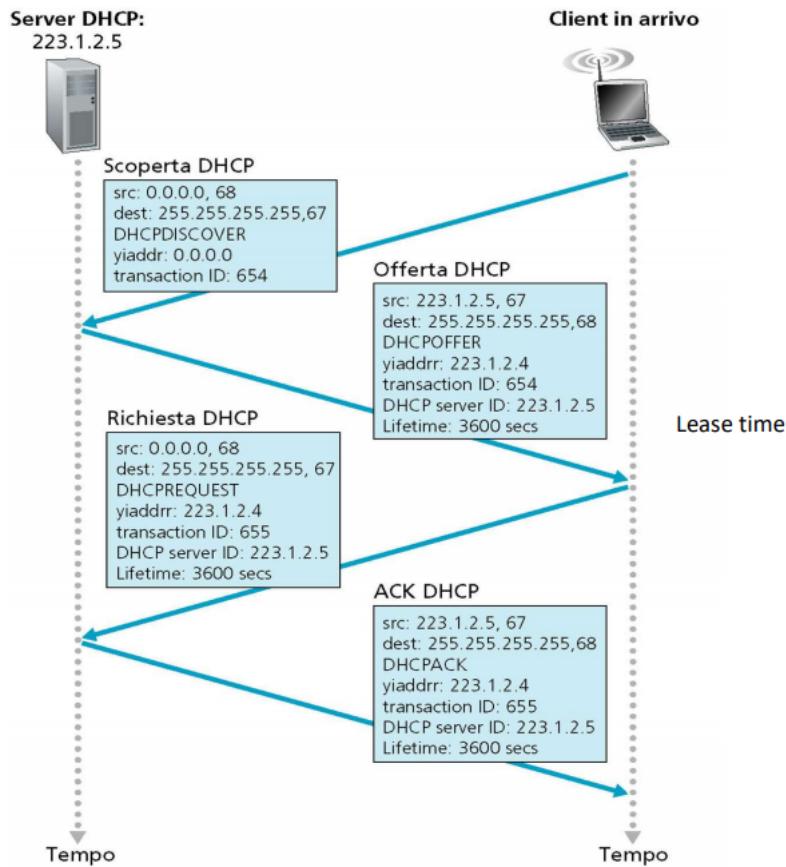


Figura 3.94: Funzionamento del DHCP.

Inoltro dei datagrammi IP

Inoltrare un datagramma significa **collocarlo nel giusto percorso** che lo porterà a destinazione.

Per **forwarding diretto** s'intende l'invio di un datagramma che ha come destinazione un host nella stessa rete del mittente. Il datagramma **non esce dalla rete** e l'invio è **diretto** sul destinatario, non viene interpellata nessun'altra entità. L'indirizzo di destinazione al livello collegamento è quello del destinatario stesso.

Per **forwarding indiretto** s'intende l'invio di un datagramma che ha come destinazione un **host di un'altra rete** rispetto a quella del mittente. Nell'inoltro si interpellano varie altre entità chiamate **routers**. L'indirizzo di destinazione al livello link è quello del router.

È bene far presente che in entrambi i casi le condizioni necessarie affinché tutto funzioni sono che **esista un collegamento diretto a livello link tra tutti gli host che appartengono a una stessa sottorete e ogni host coinvolto abbia un indirizzo IP corretto, cioè con uguale network ID e con host ID univoco nella sottorete**.

Le **due condizioni insieme** diventano condizione **necessaria e sufficiente** perché la comunicazione **funzioni**.

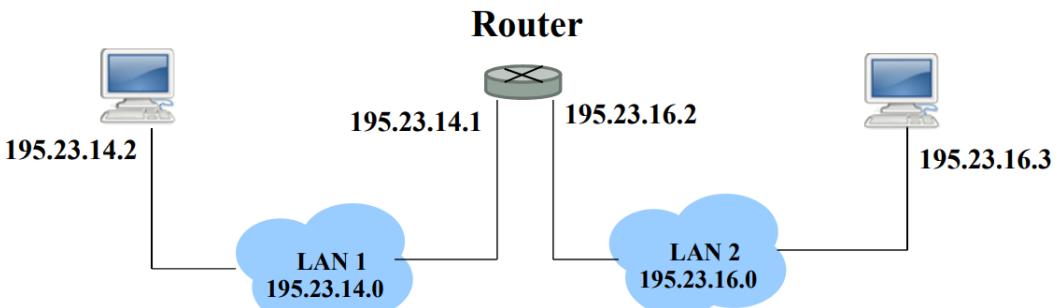


Figura 3.95: Esempio di rete IP.

Come detto in precedenza, ogni host connesso a una rete IP è dotato di un interfaccia di rete e ad ogni interfaccia di rete è associato un indirizzo IP **univoco**.

Il **router è un dispositivo intermedio** che svolge funzioni di indirizzamento al livello IP ed è dotato di almeno due interfacce di rete.

Quando un **host** deve inviare un datagramma, o un **router** deve inoltrarne uno ricevuto, fanno entrambi riferimento alla loro **tabella d'inoltro** per ricavare il nodo della rete successivo a cui inviare il datagramma.

Nell'indirizzamento senza classi una **tabella d'inoltro deve includere quattro informazioni:** la maschera, l'indirizzo di rete, il numero dell'interfaccia e l'indirizzo IP del router successivo. Le prime due informazioni sono spesso combinate.

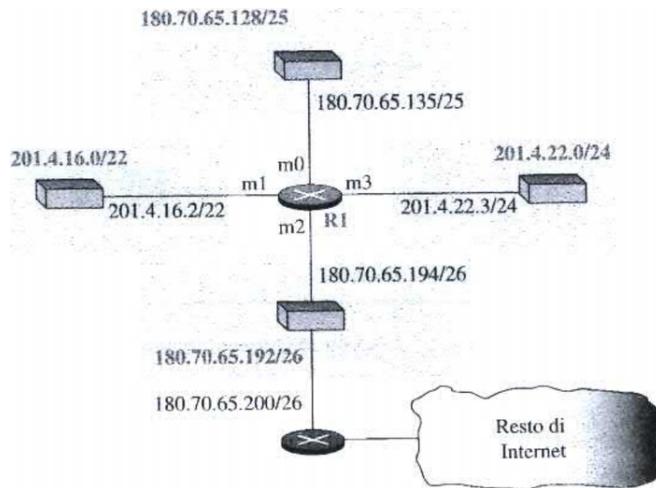


Figura 3.96: Una configurazione di hosts.

Indirizzo di rete/maschera	Salto successivo	Interfaccia
180.70.65.192/26	-	m2
180.70.65.128/25	-	m0
201.4.22.0/24	-	m3
201.4.16.0/22	-	m1
Default	180.70.65.200	m2

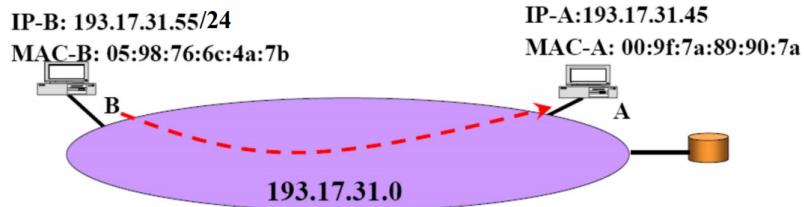
Figura 3.97: La tabella d'inoltro del router R1.

Il modulo d'inoltro si occupa di effettuare le ricerche nella tabella. La ricerca avviene nel modo seguente: per ogni riga gli **n** bit a sinistra dell'indirizzo di destinazione, che costituiscono il **prefisso**, sono lasciati **invariati** e i **restanti settati a zero**. Se l'indirizzo di rete risultante **combacia** con l'indirizzo presente nella riga allora anche le informazioni delle altre due colonne, relative alla riga, vengono estratte. **Altrimenti la ricerca continua**.

Supponiamo che R1 debba inoltrare un datagramma con il seguente indirizzo di destinazione: **180.70.65.140**. Il router esegue i seguenti passaggi:

1. La maschera del primo indirizzo nella tabella, /26, è applicata all'indirizzo di destinazione. Il risultato è 180.70.65.128, che non combacia con 180.70.65.192.
2. Successivamente la maschera del secondo indirizzo nella tabella, /25, è applicata all'indirizzo di destinazione. Il risultato è 180.70.65.128, che combacia con 180.70.65.128, quindi l'indirizzo del salto e il numero di interfaccia vengono estratti e usati per inoltrare il datagramma.

Rete locale coincidente con rete / sottorete IP



1. L'entità IP di B deve spedire un pacchetto all'indirizzo IP-A

2. B conosce l'indirizzo IP-B della propria interfaccia e dal confronto con IP-A capisce che A si trova nella stessa rete

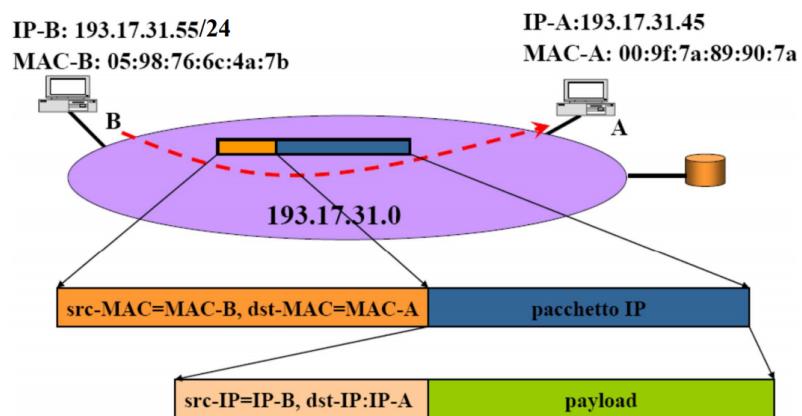
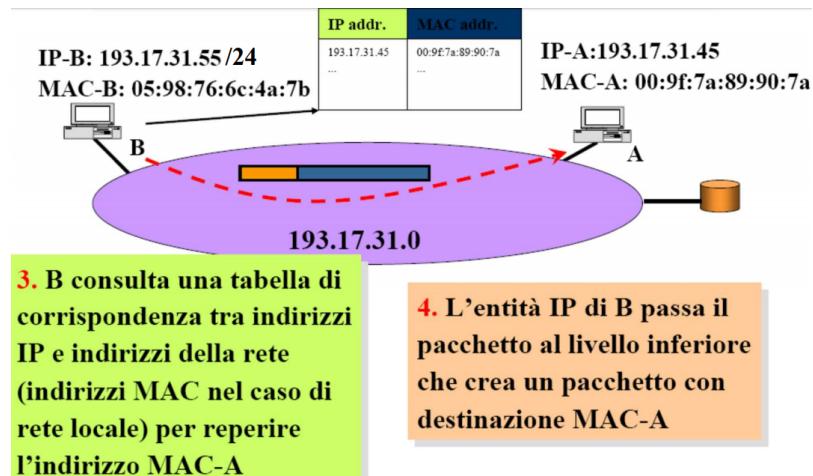
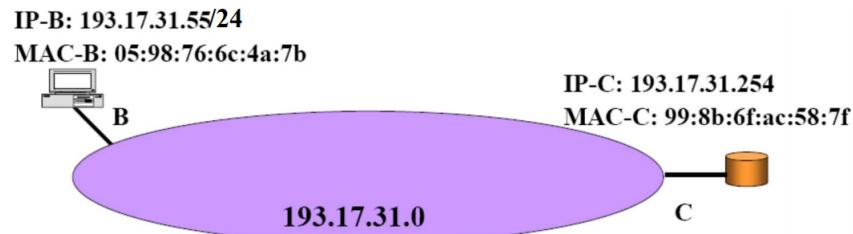
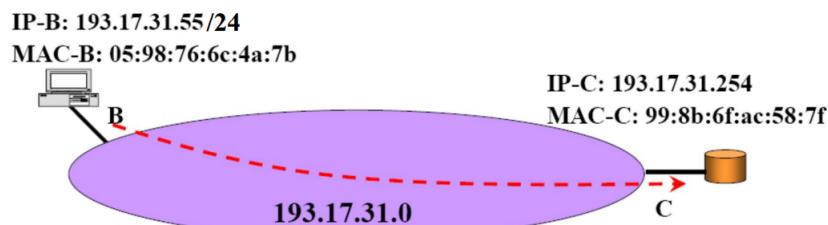


Figura 3.98: Inoltro diretto negli host.



1. L'entità IP di B deve spedire un pacchetto all'indirizzo IP-D=131.17.23.4

2. B conosce l'indirizzo IP-B della propria interfaccia e dal confronto con IP-D capisce che D NON si trova nella stessa rete



3. B deve dunque inoltrare il pacchetto ad un router (di solito è configurato un solo default router)

4. B recupera l'indirizzo MAC del router nella tabella di corrispondenza e passa il pacchetto al livello inferiore

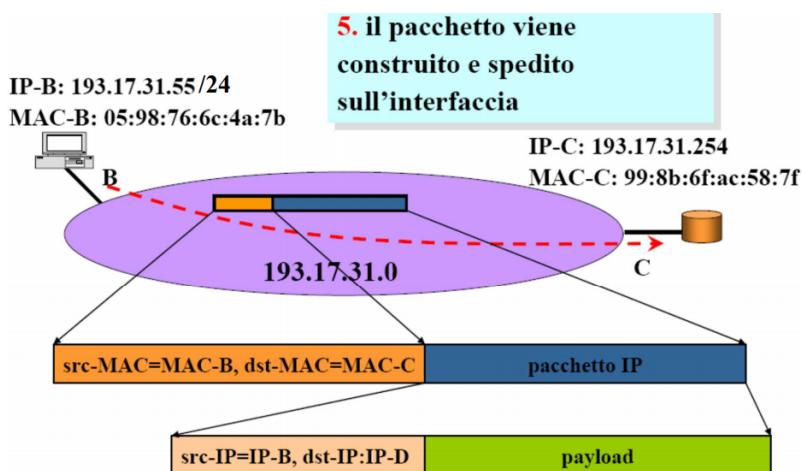
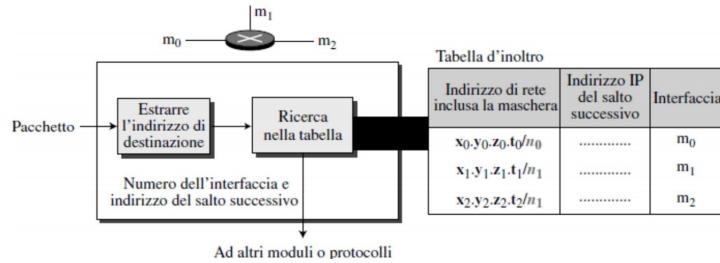


Figura 3.99: Inoltro indiretto negli host.



L'**indirizzamento con classi** permette al router di avere una tabella di indirizzamento contenente **una riga per ogni rete esterna** alla rete a cui appartiene il router.

Utilizzando un **indirizzamento senza classi** si verifica inevitabilmente un **aumento delle righe**, poiché lo spazio degli indirizzi è diviso in blocchi più piccoli delle classi, con un conseguente **aumento del tempo richiesto per effettuare l'operazione di ricerca**.

Per attenuare questo problema, come detto in precedenza, è stato ideato il meccanismo di **aggregazione degli indirizzi**.

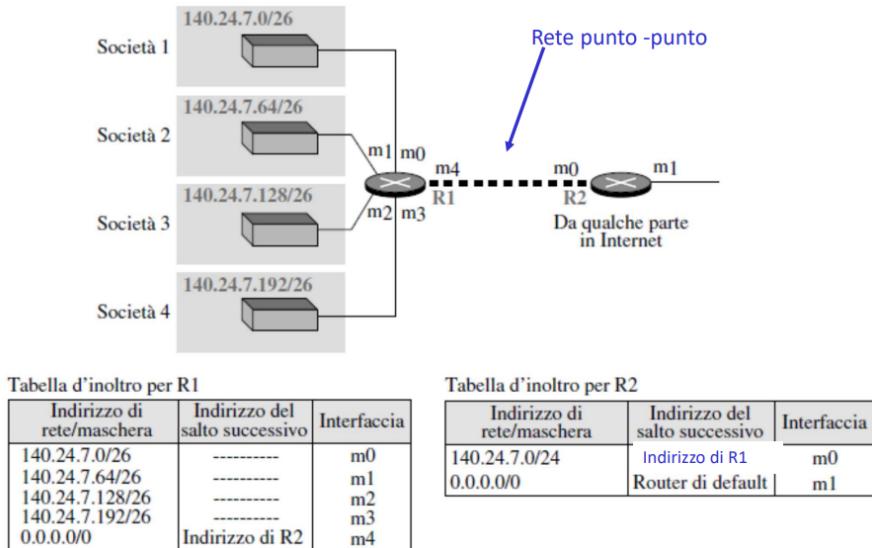


Figura 3.100: Inoltro e aggregazione degli indirizzi.

Il vantaggio è osservabile in figura: il router R2 ha una tabella di indirizzamento molto più corta del router R1, qualunque datagramma con indirizzo di destinazione compreso tra 140.24.7.0 e 140.24.7.255 verrà inviato a R1 che si occuperà poi di consegnarlo alla rete appropriata. In questo caso gli indirizzi dal 140.24.7.0 al 140.24.7.255 sono stati **aggregati** in un unico blocco.

N.B. L'aggregazione è possibile solo con blocchi **contigui**.

Un'altra soluzione possibile per risolvere il **problema della dimensione eccessiva delle tabelle di inoltro** è quella di implementare una gerarchia delle tabelle.

Attualmente Internet ha una struttura gerarchica essendo divisa in **dorsali**, **ISP nazionali**, **ISP regionali** e **ISP locali**.

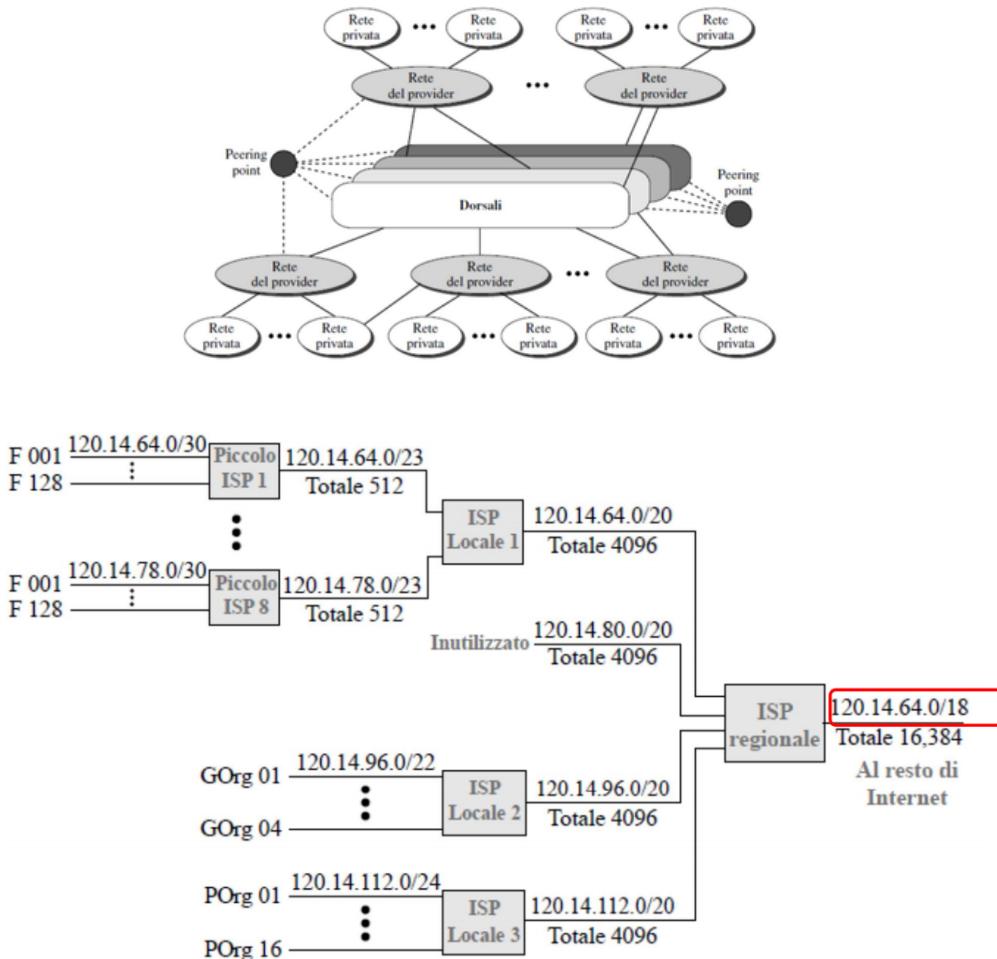


Figura 3.101: Routing gerarchico con ISP.

In figura un ISP regionale, a cui è stato assegnato un blocco di 16.384 indirizzi a partire dal 120.14.64.0, ha deciso di dividerlo in 4 sotto blocchi, ciascuno con 4096 indirizzi.

Ognuno di questi blocchi è stato poi assegnato a un ISP locale che ha a sua volta effettuato una divisione in sotto blocchi di 512 indirizzi.

Il **vantaggio**, similmente prima, è che il resto di **Internet non è consapevole di tale divisione**. Tutti i singoli clienti degli ISP locali sono **noti** al resto della rete mediante l'indirizzo 120.14.64.0: **in ogni router del mondo vi è una sola riga per tutti questi clienti**.

La **tabella d'inoltro** viene ordinata dalla maschera più lunga alla maschera più corta, secondo il principio della **Longest Matching Mask**.

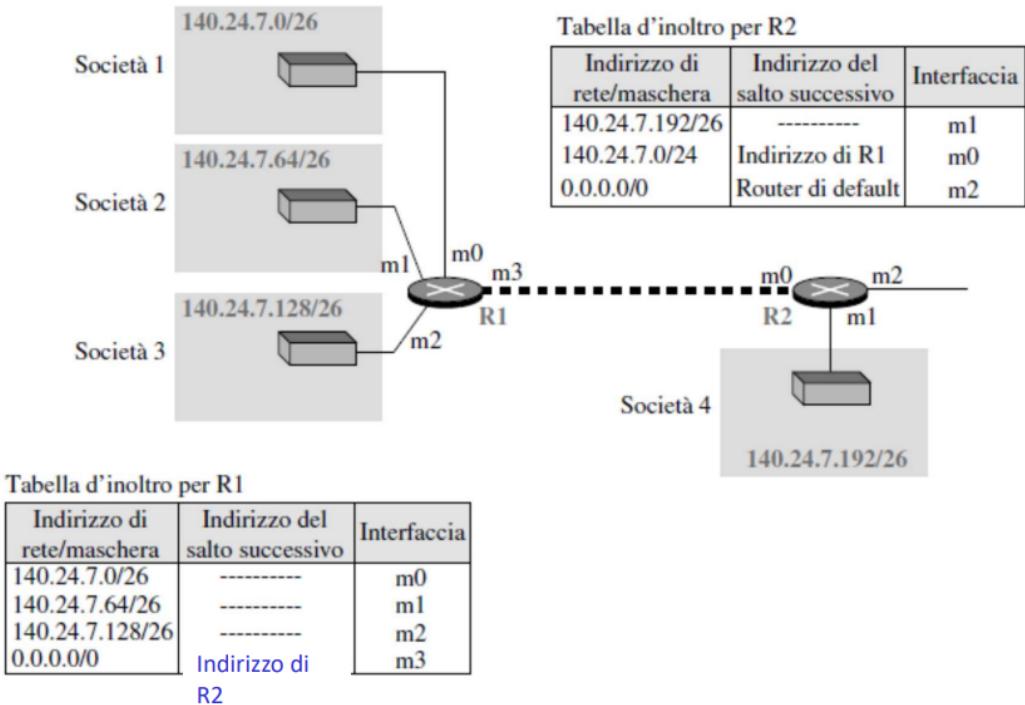


Figura 3.102: Longest Mask Matching.

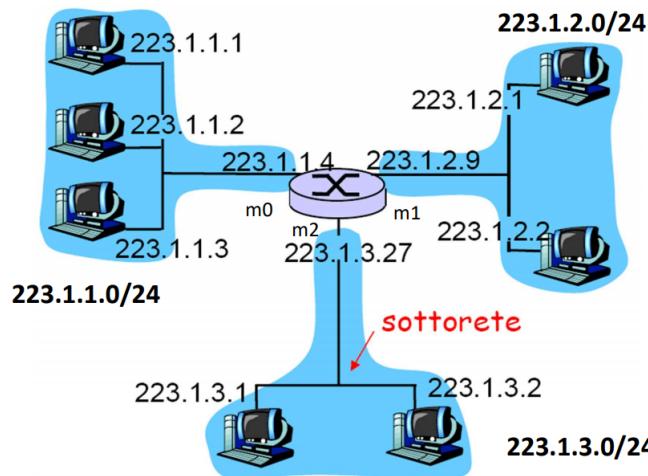


Figura 3.103: L'indirizzo IP delle interfacce di un router appartiene al blocco che individua le sottoreti con cui si interfaccia.

NAT

Il **NAT** o **Network Address Translation** è una tecnologia che **consente** a una **rete privata** di usare una serie di **indirizzi privati** per la comunicazione **interna** e almeno un **indirizzo globale** per la **comunicazione con il resto del mondo**.

L'accesso a Internet è realizzato tramite un router abilitato alla **NAT**, il traffico in ingresso e in uscita hanno come **indirizzo di destinazione** e **indirizzo sorgente** l'**indirizzo IP del router in questione**.

Ciò si effettua nel seguente modo: tutti i **datagrammi in uscita** dalla rete privata passano attraverso il router NAT che **sostituisce l'indirizzo IP sorgente** del datagramma con **l'indirizzo NAT globale del router**. Analogamente tutti i **datagrammi in entrata** passano a loro volta attraverso il router NAT che **sostituisce l'indirizzo IP destinazione** del datagramma, ovvero l'indirizzo NAT globale del router, con **l'indirizzo privato appropriato**.

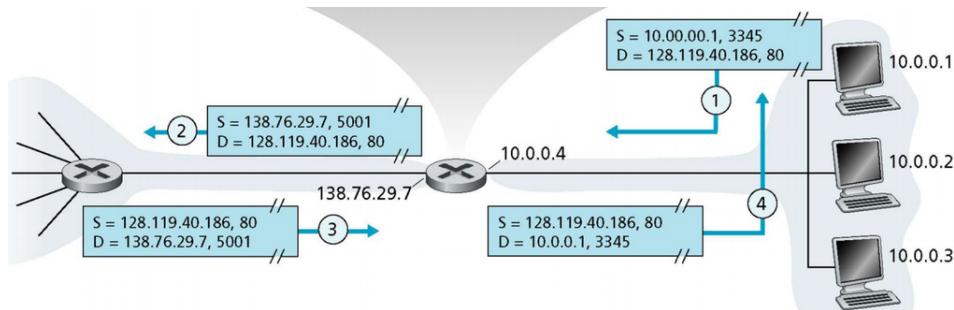


Figura 3.104: Datagrammi in entrata ed uscita da un router NAT.

A prima vista il processo di traduzione degli indirizzi potrebbe sembrare complesso ma non lo è affatto. Partiamo dalla struttura della tabella d'inoltro del router NAT.

Quando il router sostituisce l'indirizzo sorgente nel datagramma in uscita **prende nota del suo indirizzo di destinazione** e inserisce nella sua tabella d'inoltro la riga con il **seguente valore**:

[indirizzo sorgente privato — indirizzo destinazione]

Quando dalla destinazione giungerà la risposta, il router sarà in grado di **indirizzare correttamente il datagramma** verso l'host della rete privata **utilizzando tale associazione**.

N.B. Il meccanismo NAT prevede che sia **sempre** la rete privata ad iniziare la comunicazione.

Se il router NAT ha un **solo indirizzo IP globale solo un host per volta potrà comunicare con un determinato host esterno**. Come fare se si desidera far comunicare **più hosts contemporaneamente**?

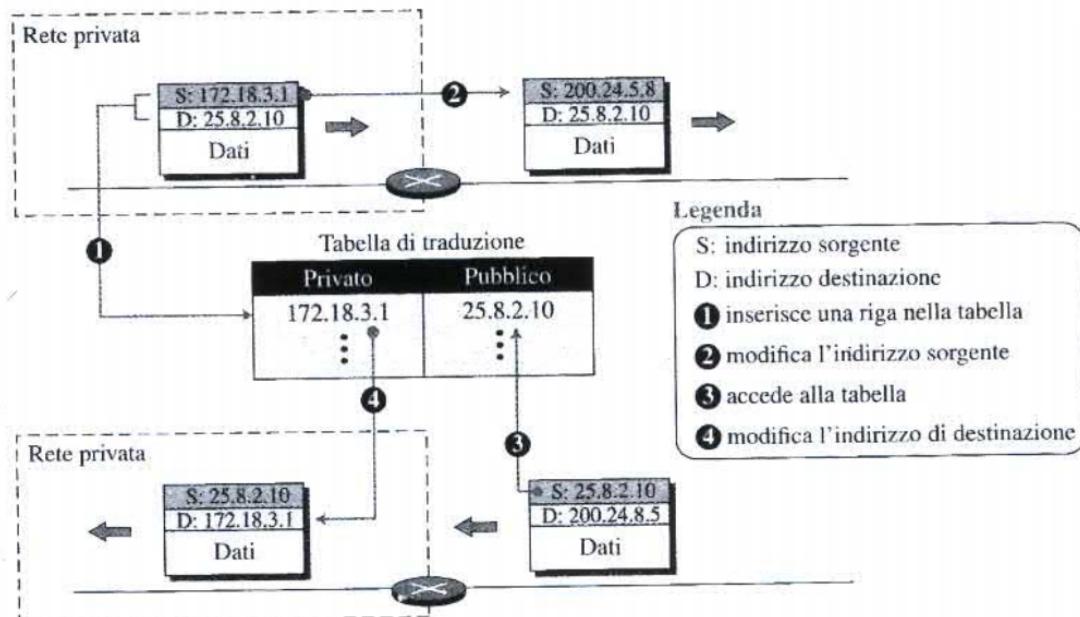


Figura 3.105: Traduzione NAT.

Supponiamo che **due host sulla stessa rete privata** debbano accedere **contemporaneamente** al server HTTP in esecuzione sull'host esterno di indirizzo 25.8.3.2.

Con una tabella d'inoltro composta da **sole due colonne non sarebbe possibile**, non ci sono abbastanza informazioni per indirizzare correttamente i datagrammi, si deve per forza tenere traccia del numero di porta. Se si aggiungono altre **tre colonne** alla tabella d'indirizzamento del router NAT, contenenti rispettivamente i **numeri di porta sorgente** (privata) e **destinazione** (esterna), e **il protocollo di trasporto** si elimina ogni possibile **ambiguità**.

Indirizzo privato	Porta privata	Indirizzo esterno	Porta esterna	Protocollo di trasporto
172.18.3.1	1400	25.8.3.2	80	TCP
172.18.3.2	1401	25.8.3.2	80	TCP
...

Figura 3.106: Tabella di traduzione a cinque colonne. I due host possono ora connettersi contemporaneamente al server HTTP.

N.B. Il meccanismo appena descritto funziona solo se i numeri di **porta privati** dei due host che accedono al server sono **scelti in modo univoco dal router NAT**.

Infatti è la **combinazione** (porta privata - indirizzo di destinazione), in tal caso, ad **individuare gli host** sulla rete privata.

ICMP

L'**ICMP** o **Internet Control Message Protocol** è un protocollo di livello rete pensato per sopperire alle mancanze del **protocollo IP** circa:

- Segnalazione e correzione degli errori.
- Invio di richieste sullo stato di un sistema remoto.

Sebbene sia un protocollo del livello di rete, i **messaggi ICMP** non vengono trasportati dal livello collegamento, bensì **vengono prima incapsulati all'interno di datagrammi IP** per poi essere passati al livello di collegamento. I pacchetti ICMP vengono instradati dai router **prima** dei pacchetti IP ordinari.

N.B. ICMP non corregge gli errori, si limita a **segnalarli**. La vera e propria correzione è lasciata ai protocolli di livello più alto.

I **messaggi ICMP** si dividono in **due categorie**: **messaggi di segnalazione errore** e **messaggio di interrogazione**. I **primi** sono usati dai routers o dagli hosts **per segnalare possibili errori** incontrati durante l'elaborazione dei datagrammi IP, i **secondi** permettono ad un host o ad un amministratore di rete di **chiedere informazioni ad un router o ad un host**.

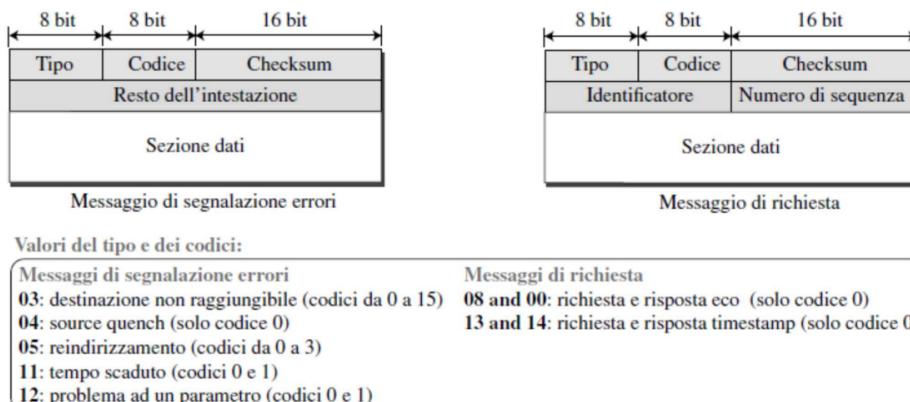


Figura 3.107: La struttura dei messaggi ICMP.

I **messaggi di errore sono sempre inviati alla sorgente originale** poichè le uniche informazioni presenti nel **datagramma IP** sono l'indirizzo sorgente e l'indirizzo di destinazione.

I **messaggi di errore contengono l'intestazione originale del datagramma IP che li ha generati e i primi 8 byte del loro payload**. I primi 8 byte del payload, infatti, contengono **informazioni sui numeri di porta**, per i protocolli TCP e UDP, e **i numeri di sequenza**, per il protocollo TCP. Sono informazioni necessarie per informare la sorgente dell'errore.

Per snellire il processo di segnalazione degli errori **ICMP segue le seguenti regole:**

- Per i **datagrammi IP frammentati**, i messaggi ICMP sono relativi al **solo** frammento numero 0.
- I messaggi ICMP **non** sono mai inviati in risposta a pacchetti IP con un indirizzo mittente che **non** rappresenti un host unico e.g. 0.0.0.0, 127.0.0.1 e indirizzi IP speciali.
- I messaggi ICMP non sono **mai** inviati in risposta a messaggi di errore ICMP, ma possono essere inviati in risposta a **messaggi ICMP di interrogazione**.

I messaggi di richiesta ICMP invece possono essere utilizzati **senza una particolare relazione con un datagramma IP** e son sempre incapsulati in datagrammi IP.

Sono usati per **verificare il funzionamento di un host o un router** sulla rete, per **trovare il tempo necessario a un datagramma per raggiungere la destinazione** e molto altro ancora...

Presentiamo di seguito **varie applicazioni** il cui funzionamento si basa sui messaggi di interrogazione ICMP.

- Il programma ping è usato da un host per **verificare il funzionamento di un altro host**.

Si basa in particolare sui **messaggi di richiesta e risposta eco dell'ICMP**. Un host invia una richiesta eco, tipo 8 e codice 0, a un altro host che, se attivo, può rispondere con una risposta eco, tipo 0 e codice 0.

Fornisce anche una misura dell'RTT e in maniera molto grossolana, **può anche misurare l'affidabilità e la congestione del router tra due host** inviando una sequenza di messaggi richiesta-risposta.

```
+ ~ ping -c 5 www.di.unipi.it
PING www.di.unipi.it (131.114.3.24) 56(84) bytes of data.
64 bytes from www.di.unipi.it (131.114.3.24): icmp_seq=1 ttl=60 time=1.75 ms
64 bytes from www.di.unipi.it (131.114.3.24): icmp_seq=2 ttl=60 time=2.40 ms
64 bytes from www.di.unipi.it (131.114.3.24): icmp_seq=3 ttl=60 time=2.71 ms
64 bytes from www.di.unipi.it (131.114.3.24): icmp_seq=4 ttl=60 time=3.15 ms
64 bytes from www.di.unipi.it (131.114.3.24): icmp_seq=5 ttl=60 time=3.16 ms

--- www.di.unipi.it ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 11ms
rtt min/avg/max/mdev = 1.748/2.634/3.162/0.527 ms
+ ~ |
```

Figura 3.108: Il comando ping.

- Il comando **traceroute** è utilizzato per **tracciare il percorso di un datagramma dalla sorgente alla destinazione** tramite l'identificazione dell'indirizzo IP di tutti i router che vengono visitati lungo il percorso.

Solitamente il programma viene impostato per un **massimo di 30 salti**, che sono quasi sempre sufficienti per raggiungere la destinazione.

Il programma **traceroute** ha un funzionamento **molto diverso da ping**. **ping** è basato su **due messaggi interrogazione**. **traceroute** è invece **implementato per mezzo di due messaggi di segnalazione degli errori: tempo scaduto e destinazione non raggiungibile**.

Sebbene sia un programma di livello applicazione ne esiste **solo il client**, non esiste un programma **traceroute server**. Non è infatti necessario raggiungere il livello applicazione dell'host destinatario.

traceroute opera nel seguente modo: genera un **datagramma UDP**, usando un numero di porta che con alta probabilità non è in uso nell'host destinatario, che verrà poi incapsulato in un datagramma IP con un valore **TTL pari a 1**.

Il **primo router sul percorso verso la destinazione scarta questo datagramma IP** e invia un messaggio **errore ICMP di tempo scaduto**.

Tramite l'indirizzo IP sorgente contenuto nel datagramma IP che trasportava il messaggio di errore ICMP, adesso **traceroute** conosce l'indirizzo IP del primo router. Invia quindi un altro datagramma UDP all'interno di un datagramma IP con **TTL stavolta impostato a 2** e così via.

In totale verranno inviati $n+1$ datagrammi IP, n saranno scartati dai router intermedi e l'ultimo verrà scartato dall'host destinatario.

```
→ ~ traceroute www.di.unipi.it
traceroute to www.di.unipi.it (131.114.3.24), 30 hops max, 60 byte packets
 1 _gateway (10.100.0.1)  1.904 ms  2.205 ms  2.313 ms
 2 jaut-cser.unipi.it (131.114.191.77)  2.629 ms  3.185 ms  3.911 ms
 3 jser-jaut.unipi.it (131.114.191.17)  4.314 ms  5.147 ms  5.419 ms
 4 jser-jfib.unipi.it (131.114.191.50)  5.390 ms  5.942 ms  6.264 ms
 5 www.di.unipi.it (131.114.3.24)  7.922 ms  7.883 ms  7.800 ms
→ ~ █
```

Figura 3.109: Traceroute.

La maggior parte dei programmi **traceroute** invia **tre messaggi a ogni dispositivo**, con lo stesso valore di TTL, per poter **effettuare una stima del round-trip-time**.

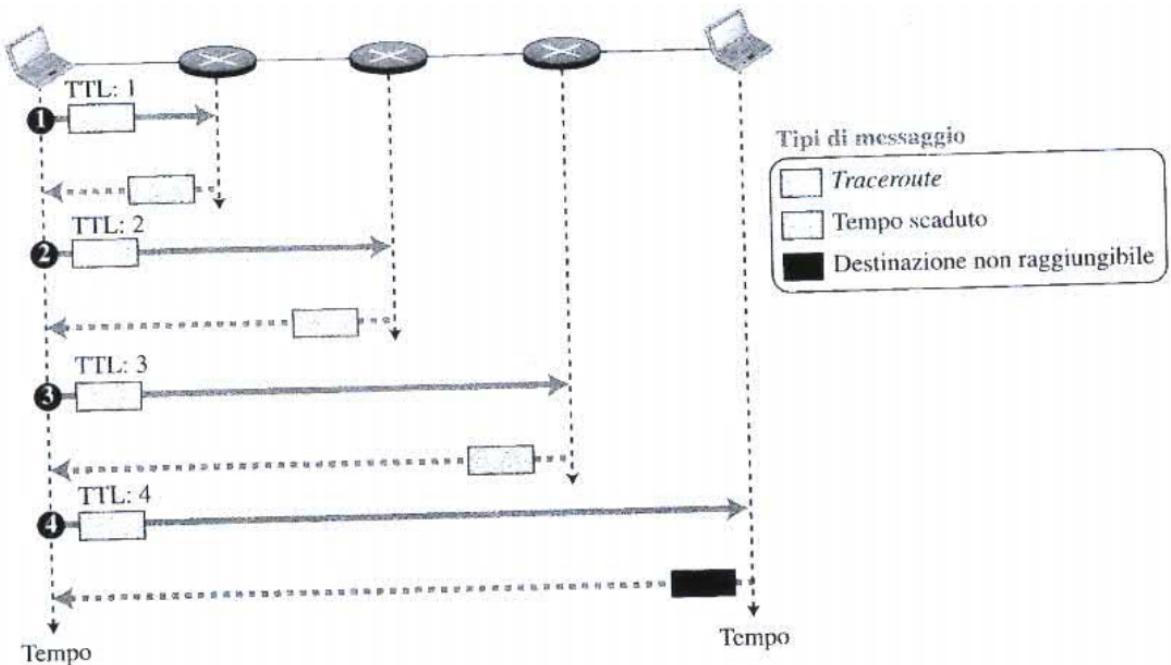


Figura 3.110: Il procedimento descritto in precedenza.

L’ultimo datagramma IP viene scartato dall’host inviando un messaggio ICMP diverso da tempo scaduto, ovvero, di **destinazione non raggiungibile**, non essendoci nessuna applicazione in ascolto sulla **porta specificata**.

Ricevuto il messaggio di destinazione non raggiungibile traceroute riconosce che la destinazione finale è stata **raggiunta**.

Esercizi

- Il sistema autonomo di una grande azienda è logicamente diviso in due sottosistemi: un sottosistema che usa indirizzi IP pubblici nel blocco 113.141.0.0/21, e uno che usa indirizzi IP privati nel blocco 192.168.0.0/16. Quali sono le maschere di rete, in notazione decimale puntata, usate nel sistema autonomo?

Soluzione: Ponendo uguali a uno i bit della maschera e azzerando gli altri, otteniamo:

```
/21 -> 11111111 11111111 11111000 00000000
                  255.255. 248.0
/16 -> 11111111 11111111 00000000 00000000
                  255.255.0.0
```

2. Un router IP connette tre sotto reti: 100.100.100.0/24, 111.99.99.0/24 e 200.200.200.0/24,, con MTU (Maximum Transfer Unit) pari a 900, 1000, e 1400, rispettivamente.

I datagrammi gestiti dal router rispettano il formato IPV4, in particolare contengono i campi IP-S (indirizzo sorgente), IP-D (indirizzo destinazione), L (lunghezza del datagramma, inclusa l'intestazione), Id (identificatore del datagramma), F (flag di frammentazione) e O (offset).

Si considerino i seguenti datagrammi ricevuti dal router:

	IP-S	IP-D	L	Id	F	O
D1	100.100.100.130	200.200.200.127	890	66	0	0
D2	111.99.99.12	100.100.100.150	1000	56	0	0
D3	200.200.200.28	100.100.100.65	1390	123	0	0
D4	200.200.200.28	111.99.99.12	1398	98	0	0

Per ognuno di questi datagrammi dire da quale sotto rete è stato ricevuto, e quali datagrammi il router invia in uscita specificandone anche la sotto rete di destinazione. *Nella frammentazione fare molta attenzione ai 20 byte d'intestazione.*

Soluzione: Il datagramma D1 è ricevuto dalla sotto rete: 100.100.100.0/24 ed è inviato sulla sotto rete: 200.200.200.0/24 e **non** è frammentato in uscita.

	IP-S	IP-D	L	Id	F	O
D1	100.100.100.130	200.200.200.127	890	66	0	0

Il datagramma D2:

	IP-S	IP-D	L	Id	F	O
D2,1	111.99.99.12	100.100.100.150	900	56	1	0
D2,2	111.99.99.12	100.100.100.150	120	56	0	110

Per il datagramma D3 si ha:

	IP-S	IP-D	L	Id	F	O
D3,1	200.200.200.28	100.100.100.65	900	123	1	0
D3,2	200.200.200.28	100.100.100.65	510	123	0	110

Infine per D4:

	IP-S	IP-D	L	Id	F	O
D3,1	200.200.200.28	100.100.100.65	900	123	1	0
D3,2	200.200.200.28	100.100.100.65	510	123	0	110

3.4.2 Struttura di un router

Com'è fatto un router?

Un router è **uno switch di terzo livello** composto da **quattro** componenti principali:

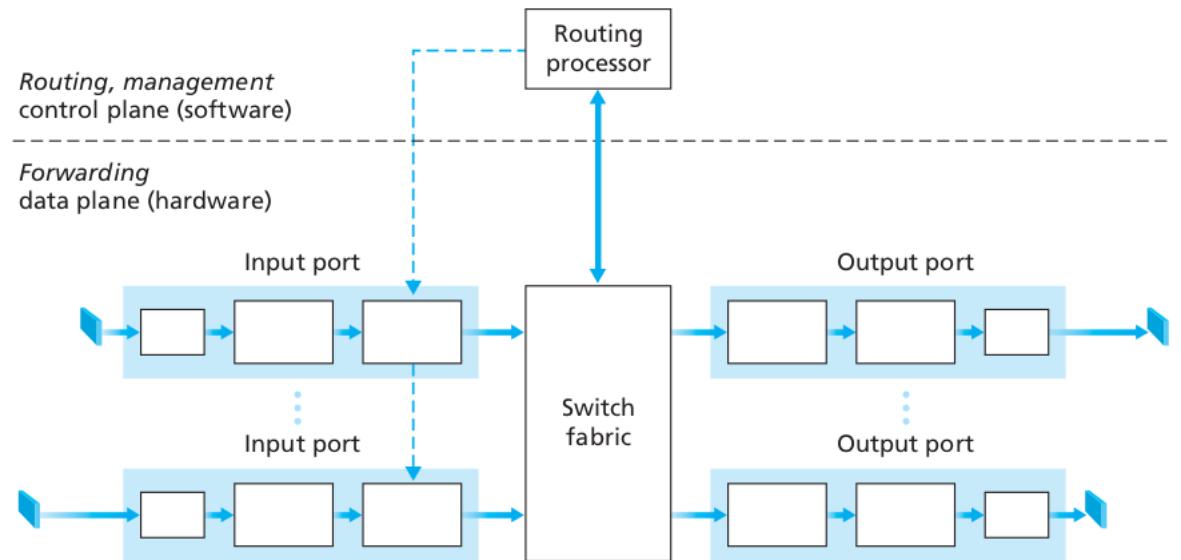
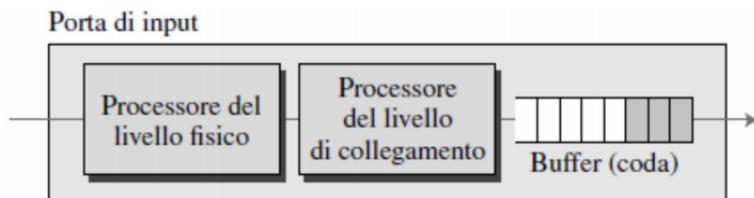


Figura 3.111: La struttura di un router.

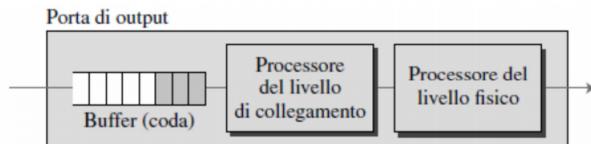
- **Porte di input:**



Le porte di input svolgono diverse funzioni chiavi: sono il punto di sbocco nel router del collegamento fisico, decapsulano i frame in entrata e ne verificano la correttezza, consegnano i frame contenenti le informazioni dei protocolli di routing al processore di routing e, specialmente, nei router moderni, **effettuano il lookup nella forwarding table**.

Per far ciò sono dotate di un **processore di livello fisico** che analizza il segnale e lo traduce in bit, un **processore di livello link** che opera al livello di frame e un **buffer** per memorizzare i pacchetti in attesa di essere inoltrati alla switch fabric.

- **Porte di output:**



Le **porte di output** memorizzano i pacchetti ricevuti dalla switch fabric e li trasmettono poi sul collegamento in uscita. In sostanza svolgono lo stesso lavoro delle porte di input ma al contrario.

Se il collegamento dovesse essere bidirezionale le porte di output saranno accoppiate con delle porte di input.

- **Switch fabric:**

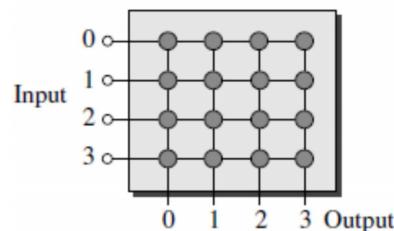


Figura 3.112: Cross-bar switch, gli switch sono disposti a forma di matrice.

La **switch fabric** connette le porte di input e le porte di output. La velocità con cui vengono trasmessi i datagrammi dalle porte di input a quelle di output influenza sulla dimensione delle code di input/output e sul ritardo complessivo nella consegna del datagramma.

- **Processore di routing:**

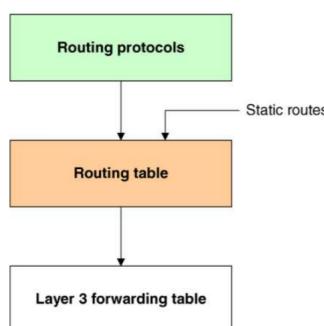


Figura 3.113: Le tabelle d'instradamento sono il risultato dell'esecuzione degli algoritmi d'instradamento.

Il **processore di routing** esegue i protocolli di routing e costruisce ed aggiorna la tabella d'inoltro.

Le tabelle d'instradamento sono simili alle tabelle d'inoltro ma sono entità separate, una tabella d'instradamento infatti può contenere molteplici percorsi verso uno stesso router, una d'inoltro invece **solo uno**.

La tabella d'instradamento può essere creata **dinamicamente** mediante i protocolli di routing ma anche **staticamente** tramite l'aggiunta da parte dell'amministratore del router di **routes statiche**.

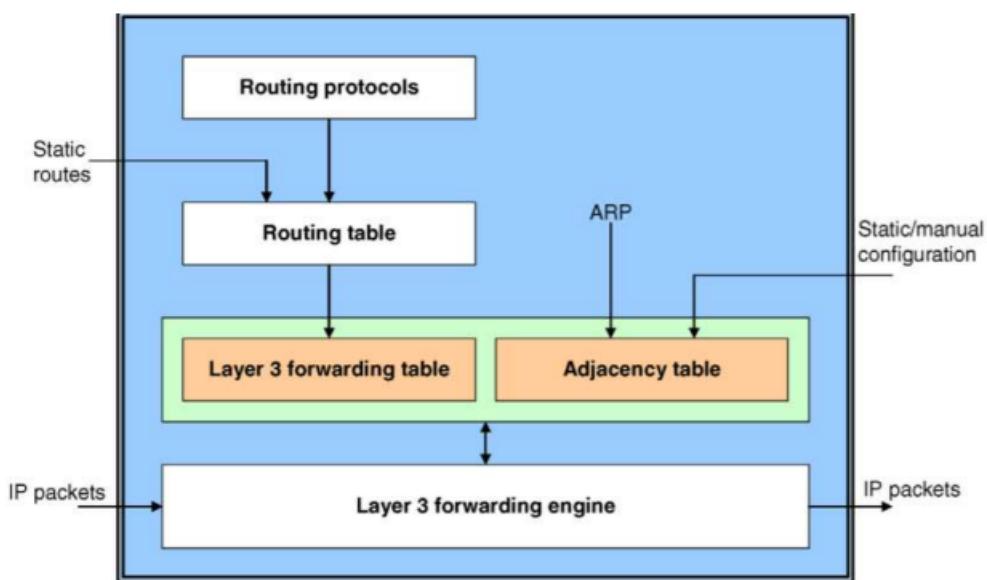


Figura 3.114: La forwarding table e il forwarding engine compongono la parte decisionale del router o data plane. La parte di controllo o control plane invece comprende i protocolli di routing e il routing engine che vanno ad alimentare la routing table.

Alcune precisazioni:

Una volta costruita, la **tabella d'inoltro** viene **copiata** e **inoltrata** a tutte le porte in modo tale che possano automaticamente effettuare la decisione di forwarding.

Definizione 3.4.1. Il **control plane** fornisce le informazioni necessarie al **data plane** o **forwarding plane** per creare e aggiornare le tabelle d'inoltro in base ai cambiamenti di stato nella topologia della rete.

Definizione 3.4.2. Il **data plane** è la parte del router che si occupa di inoltrare il traffico da un'interfaccia all'altra.

3.4.3 Routing

Il processo attraverso il quale si calcola il percorso da seguire per consegnare un datagramma è detto **routing**.

Il **routing** si distingue in due tipi: quando il datagramma deve essere consegnato a **un solo destinatario** si parlerà di **routing unicast**, quando ci sono più destinatari invece di **routing multicast**.

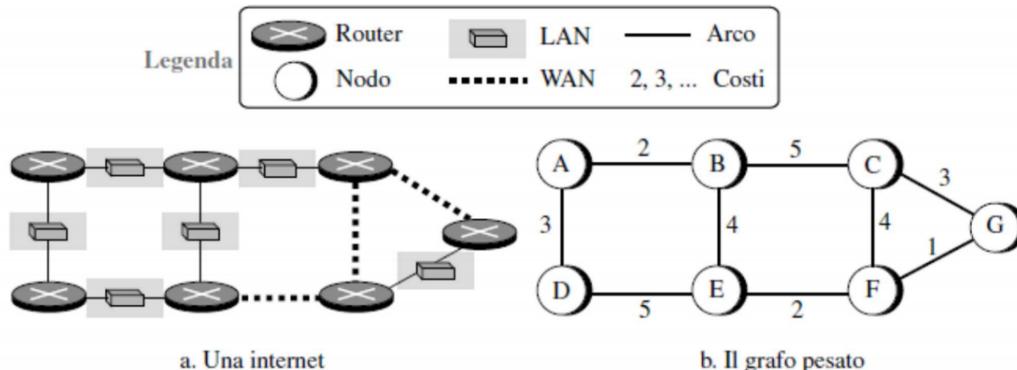


Figura 3.115: Una internet può essere rappresentata come un grafo pesato.

L'astrazione di rappresentare Internet come un grafo pesato consente di riassumere l'obiettivo del processo di routing con una **domanda chiave**:

Qual è il percorso di costo minimo tra due nodi?

Per rispondervi ogni router dovrà trovare il **percorso di costo minimo** tra **sè stesso e gli altri routers**.

In una rete con N routers ci sono $N \times (N - 1)$ percorsi minimi totali, poiché per ogni router ce ne sono $N - 1$ verso tutti gli altri escluso sè stesso.

Definizione 3.4.3. Un **algoritmo di routing** è un algoritmo che, dato un insieme di routers interconnessi, trova il **percorso di costo minimo**, nel caso di un grafo pesato, o, alternativamente, il percorso più breve da un router sorgente a un router destinazione.

Gli **algoritmi di routing** possono essere divisi secondo i seguenti criteri:

- **Statici:** l'operatore aggiorna **manualmente** le righe della tabella di routing. Sono usati per reti di piccole dimensioni.
- **Dinamici:** sono algoritmi che aggiornano **dinamicamente** la tabella di routing in base ai cambiamenti nella topologia della rete e al traffico.
- **Decentralizzati:** gli algoritmi di routing locali **non** possiedono l'intera **conoscenza** della **topologia della rete**. Possiedono solo le informazioni relative ai costi dei cammini verso i nodi adiacenti al nodo sorgente.

Il cammino di costo minimo viene calcolato **iterativamente** e in modo **distribuito**.

- **Globali:** ricevono in input la **conoscenza completa** circa la topologia della rete. Il cammino di costo minimo può essere calcolato in modo **centralizzato** (e.g. da un server) o **distribuito**.

Equazione di Bellman-Ford

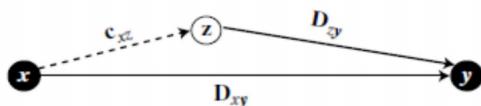
Definizione 3.4.4. Un **albero di costo minimo** è un albero con il router sorgente che fa da radice e che visita tutti gli altri nodi dell'albero seguendo sempre il percorso meno costoso tra quelli possibili.

Nel **distance vector routing** ogni nodo crea il proprio **albero di costo minimo** con le sole informazioni che possiede circa i nodi a lui vicini. Gli **alberi incompleti** così ottenuti vengono inoltrati da ogni nodo ai suoi vicini. Dopo un certo numero di iterazioni l'algoritmo **converge** restituendo l'**albero di costo minimo**.

Il fulcro del **distance vector routing** è l'**equazione di Bellman-Ford**:

$$D_{xy} = \min(D_{xy}, (C_{xz} + D_{zy}))$$

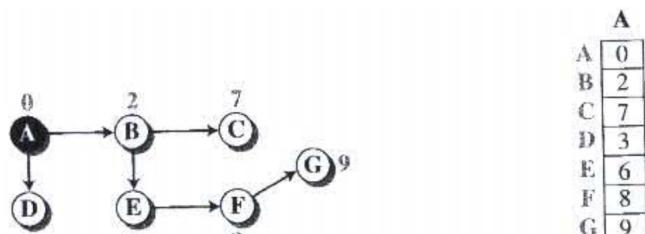
Quest'equazione viene usata per trovare il cammino di costo minimo tra un nodo sorgente X e un nodo destinazione Y attraverso i nodi intermedi.



b. Aggiornamento di un percorso
usando un nodo intermedio

Figura 3.116: Rappresentazione grafica.

Definizione 3.4.5. Un **vettore distanza** è un array monodimensionale che rappresenta un albero di costo minimo.



a. Albero per il nodo A

b. Vettore distanza per il nodo A

Descriviamo ora il **distance vector algorithm**:

1. Ogni nodo invia una copia del proprio vettore distanza a ciascuno dei suoi vicini.
2. Quando un nodo x riceve un nuovo vettore distanza DV, da qualcuno dei suoi vicini, lo salva e usa la formula di Bellman-Ford per aggiornare il proprio vettore distanza come segue:

$$D_{xy} = \min\{c(x, v) + D_{vy}\} \quad \forall v \in N_{neighbor}(x)$$

- (a) Dove D_{xy} è pari alla stima del costo minimo dal nodo x al nodo y , x mantiene il vettore distanza $D_x = [D_{xy} \quad \forall y \in N]$.
 - (b) Il nodo x conosce il costo verso ciascun vicino v : $c(x, v)$.
 - (c) Ogni nodo mantiene i vettori distanza dei suoi vicini. Per ogni vicino v , il generico nodo x mantiene $D_v = [D_{vy} \quad \forall y \in N]$.
3. Tutti i nodi continuano a cambiare i propri DV in maniera asincrona, finché ciascuna stima dei costi D_{xy} converge al costo minimo $d_x(y)$.

L'algoritmo è **decentralizzato** e **dinamico**: è **iterativo** e **asincrono**, viene eseguito da ogni nodo in modo indipendente, e ogni iterazione locale è causata da:

- Il cambio del costo di uno dei collegamenti locali.
- La ricezione da qualche vicino di un vettore distanza aggiornato.

Ogni nodo notifica i suoi vicini solo quando il proprio DV cambia; i vicini avvisano i vicini **solo se necessario**.

Ciascun nodo:

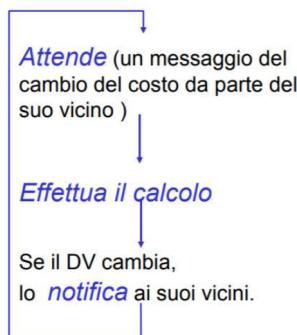


Figura 3.117: L'algoritmo schematizzato.

Esempio:

Sia questa la **configurazione iniziale** dei vettori distanza dei nodi di una rete.

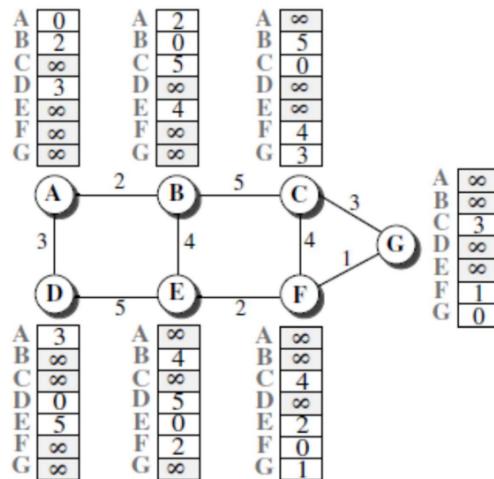


Figura 3.118: Configurazione iniziale.

Supponiamo si susseguano i due eventi:

Nuovo B	Vecchio B	A
A 2	A 2	A 0
B 0	B 0	B 2
C 5	C 5	C ∞
D 5	D ∞	D 3
E 4	E 4	E ∞
F ∞	F ∞	F ∞
G ∞	G ∞	G ∞

a. Primo evento: B riceve una copia del vettore di A.

Nuovo B	Vecchio B	E
A 2	A 2	A ∞
B 0	B 0	B 4
C 5	C 5	C ∞
D 5	D 5	D 5
E 4	E 4	E 0
F 6	F ∞	F 2
G ∞	G ∞	G ∞

b. Secondo evento: B riceve una copia del vettore di E.

Si osservi come cambiano i distance vectors.

Pseudocodice del **distance vector routing**:

```

Distance_Vector_routing() {
    // Inizializzazione:
    // creazione dei vettori
    // istanza iniziali del nodo

    D[ me_stesso ] = 0;

    for (y = 1 to N) {
        if ( $y \in \text{nodi\_vicini}$ )
            D[y] = c[ me_stesso ][y];
        else  $D[y] = \infty$ ;
    }

    speidisci il vettore {D[1], ..., D[n]} a tutti i vicini;

    repeat(sempre) {
        // Aggiornamento:
        // usare il vettore ricevuto
        // dal nodo vicino per aggiornare
        // quello locale

        wait(un vettore D_w da un vicino w o un qualsiasi
             cambiamento negli archi);

        for (y = 1 to N)
            D[y] = min_v [ c[ me_stesso ][v]+Dv[y] ];

        if (cambiamento nel vettore)
            speidisci il vettore {D[1], ..., D[n]} a tutti i vicini;
    }
}

```

C'è però un **problema**, con il routing basato su distance vectors i decrementi di costo si diffondono rapidamente tra i nodi mentre i **peggioramenti** del costo si diffondono molto più **lentamente**. Ad esempio in caso di guasto:



Figura 3.119: Count to infinity.

Supponiamo che due i due routers in figura, A e B, siano collegati alla rete X. Se improvvisamente dovesse saltare il collegamento tra X e A, e A aggiornasse il suo DV con il peggioramento del costo, ma non inviasse **immediatamente** a B il suo DV aggiornato e anzi B lo inviasse ad A, A crederebbe che B abbia trovato un altro modo per raggiungere X. Di conseguenza A aggiornerebbe il suo DV e lo invierebbe a B, che lo aggiornerebbe a sua volta e così via, fino ad arrivare a un costo pari a **infinito**.

Questo scenario è chiamato **count to infinity**.

Una possibile soluzione a questo problema è chiamata **split horizon**: consiste nel non inoltrare le informazioni circa il cammino di costo minore attraverso l'interfaccia dalla quale le si hanno ricevute. Ad esempio: se B ritiene che il costo minore per raggiungere X passi da A, non invierà ad A tale informazione, ovvero $D_B[X]$.

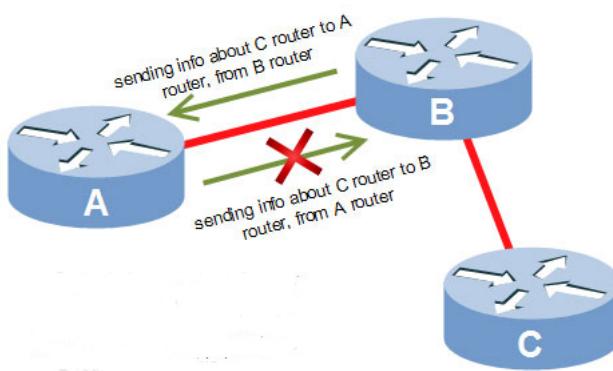


Figura 3.120: Split horizon.

Un'altra soluzione è chiamata **poisoned reverse**: consiste, nell'inviare, attraverso l'interfaccia da cui si hanno ricevuto informazioni circa un percorso, al posto del reale costo, il valore infinito. Nel nostro esempio quindi B invierà ad A $D_B[X] = \infty$, avendo ottenuto l'informazione circa il cammino di costo minore verso X dall'interfaccia che dà verso A.

N.B. Poisoned reverse non risolve completamente il problema del count to infinity, se infatti avessimo un guasto che coinvolge **tre o più nodi** tale strategia si rivelerebbe inefficace.

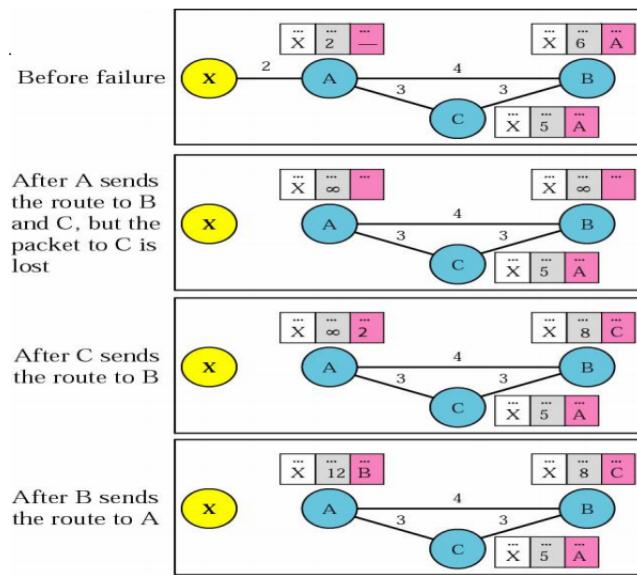


Figura 3.121: Instabilità a tre nodi.