Sistemi di elaborazione e trasmissione dell'informazione

Informatica (Università degli Studi di Genova)

RETI

1 – INTERNET

Internet è una rete di calcolatori che interconnette miliardi di dispositivi di calcolo nel mondo. I dispositivi sono detti *host* (ospiti) o sistemi periferici (*end system*). I sistemi sono connessi tra loro tramite una rete di collegamenti (*commutation link*) e commutatori di pacchetti (*packet switch*).
Per eseguire i propri compiti le applicazioni distribuite scambiano messaggi.
Per questi spostamenti nella rete esistono due approcci fondamentali:
Commutazione di circuito: (rete telefonica) scambio di sequenze di bit. Nelle reti a commutazione di circuito le risorse richieste lungo un percorso sono riservate per l'intera durata della sessione di comunicazione. La rete è dinamica, riconfigura il circuito per connettere *host* diversi. Ogni *host* è identificato univocamente da un numero.
Commutazione di pacchetto: (posta) i messaggi possono svolgere una funzione di controllo o contenere dati. La sorgente suddivide i messaggi in parti più piccole chiamate pacchetti. Non è necessaria la comunicazione diretta tra sorgente e destinatario, in quanto la rete è composta da una serie di stazioni che possono ricevere il pacchetto (cioè hanno un'unità di memoria) e trasmetterlo al collegamento successivo. I *router* instradano i pacchetti. Una volta ricevuto l'intero pacchetto, il canale di comunicazione può anche essere chiuso. Saltando da un *router* all'altro (*hop*) si arriva all'*host* di destinazione. Vantaggio: una volta oltrepassato il nodo interno quel canale di comunicazione può essere riutilizzato.

[R1]-----[R2]-----[R3]

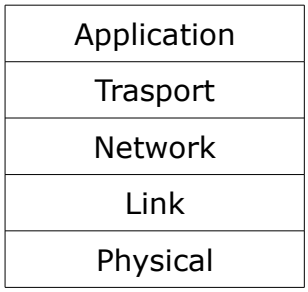
Trasmissione *store and forward* → il commutatore deve ricevere l'intero pacchetto prima di poterne cominciare a trasmettere sul collegamento in uscita il primo bit; la maggior parte dei commutatori di pacchetto utilizza questo meccanismo, che consiste nell'utilizzare un canale di comunicazione alla volta (R1 *forward* – R2 *store*); in seguito si rilasciano R1 ed il canale R1-R2, e si acquisisce il canale R2-R3.
Oltre ad un canale di comunicazione serve un dispositivo che legga ciò che c'è in memoria in R1 e lo sequenzializzi (lo passi un bit alla volta) ad R2. Terminata l'operazione di copia R1 può cancellare la sua copia del pacchetto e riceverne un altro.
Ogni *router* ha una certa quantità di *buffer* (per ricezione e trasmissione). R2 deve avere spazio libero prima di poter cominciare la fase di ricezione del pacchetto; solo dopo aver ricevuto tutti i bit del pacchetto R2 può iniziare la seconda fase, cioè trasmetterlo verso *router* successivo.
Input: canale *input* +2 *buffer*.
Output: canale *output* + 2 *buffer*.
Fase intermedia: solo un *buffer*.

Indirizzamento → IP *address* per identificare un *host* in modo univoco (per dirigere i pacchetti). I pacchetti contengono metadati (es. per implementare la comunicazione bidirezionale servono informazioni sulla provenienza) + *payolad*.
Standard di definizione dei pacchetti + regolazione velocità di trasmissione per evitare la perdita di pacchetti. Modalità predefinita di funzionamento dei *router* per decidere quali pacchetti tenere e quali scartare (ricezione veloce – uscita lenta o viceversa), si può evitare con lo *store and forward*.

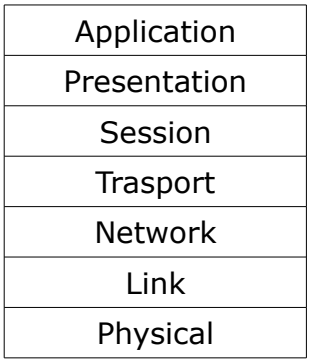
1.1 – Architettura a livelli e stratificazione dei protocolli

Un protocollo definisce il formato e l'ordine dei messaggi scambiati tra due o più entità di comunicazione, così come le azioni intraprese in fase di trasmissione e/o di ricezione di un messaggio o di un altro evento.

Un’architettura a livelli consente di discutere una parte specifica di un sistema articolato e complesso. Stratificazione dei livelli di astrazione.
Lo *stack* dei protocolli di Internet consiste di cinque livelli:



Five-layer Internet protocol stack



Seven-layer ISO OSI reference model

1. **Applicativo** → sede in cui vengono realizzati effettivamente i protocolli di comunicazione. Include molti protocolli, quali HTTP, FTP, e DNS. Un’applicazione in un sistema periferico scambia pacchetti (messaggi) con un’altra applicazione tramite i protocolli.
2. **Trasporto** → trasferisce i messaggi del livello di applicazione tra punti periferici gestiti dalle applicazioni. In Internet esistono due protocolli di trasporto: TCP e UDP. TCP fornisce alle applicazioni un servizio orientato alla connessione, il quale include la consegna garantita dei messaggi a livello di applicazione alla destinazione, e il controllo di flusso. Inoltre fraziona i messaggi lunghi in segmenti e fornisce un controllo della congestione. UDP fornisce un servizio non orientato alla connessione.
3. **Rete** → definisce indirizzi a livello globale, si occupa di traferire i pacchetti, chiamati datagrammi, da un *host* ad un altro. Comprende il protocollo IP, che definisce i campi dei datagrammi e come i sistemi periferici e i *router* agiscono su tali campi. Il livello contiene anche svariati protocolli di instradamento che determinano i percorsi che i datagrammi devono seguire.
4. **Datalink** → (collegamento) per traferire un pacchetto da un nodo (*host* o *router*) a quello successivo sul percorso, il livello di rete passa il datagramma al livello di collegamento. I servizi forniti dipendono dallo specifico protocollo utilizzato.
5. **Fisico** → mentre il compito del livello di collegamento è quello di instradare interi *frame*, il ruolo di questo livello consiste nel trasferire i singoli *bit* del *frame* da un nodo a quello successivo. I protocolli di questo livello dipendono dall’effettivo mezzo trasmissivo.

Per potersi connettere due macchine devono avere gli stessi protocolli standard. Ogni livello in una macchina si interfaccia con lo stesso livello in un’altra.
Incapsulamento → i dati seguono un percorso discendente lungo lo *stack* dei protocolli del sistema del mittente, risalgono e riscendono lungo gli *stack* dei protocolli dei *router* che intervengono a livello di collegamento, e infine risalgono lo *stack* dei protocolli nel sistema del destinatario.

RFC - *Request For Comment* → i protocolli internet vengono standardizzati in modo parzialmente informale, non c’è un’unica organizzazione che li decide. RFC è una bozza di documento, poi avviene la pubblicazione ufficiale tramite la IANA. RFC è il nome generale, poi vengono numerati progressivamente.

Con i primi due livelli di astrazione si possono mettere in comunicazione due macchine, con il livello di rete si può espandere geograficamente.

2 - LIVELLO APPLICATIVO

2.1 – PROTOCOLLO HTTP – Hyper Text Transfer Protocol

v. 1.0 RFC 1945
v 1.1 RFC 2616 – RFC 7230-31-32-33-34-35
Tipo client/server – *Server* in ascolto sulla Porta 80
Basato su trasporto TCP
Il protocollo definisce sia la struttura dei messaggi, che la modalità con cui *client* e *server* si scambiano i messaggi. Una pagina *web* (documento) è costituita da oggetti, cioè da *file* indirizzabili tramite URL. La maggior parte delle *web pages* consiste di un *file* HTML principale e diversi oggetti referenziati da esso.
URL (*Universal Resource Locator*) si compone di due parti: il nome dell'*host* del *server* che ospita l'oggetto e il percorso dell'oggetto. Un *web browser* implementa il lato *client* di HTTP. Un *web server* ospita oggetti *web* indirizzabili tramite URL.
Lo scopo è accedere ad informazioni memorizzate sul *server*, il protocollo distribuisce file dal *file system* del *server* ai *client* che ne fanno richiesta.
Messaggio di richiesta dal *client* → corrispondente messaggio di risposta dal *server*.
I messaggi sono codificati in forma leggibile (stringhe ASCII).
Per prima cosa il *client* inizia una connessione TCP con il *server*, che comporta il *three-way handshake*. Una volta stabilita, i processi *client* e *server* accedono a TCP attraverso i propri *socket*. Le prime parti dell'*handshake* richiedono un RTT, a cui segue il messaggio di richiesta HTTP da parte del *client* combinato con la terza parte dell'*handshake*. Quando il messaggio di richiesta arriva al *server*, quest'ultimo inoltra il *file* sulla connessione TCP. La richiesta-risposta HTTP consuma un altro RTT. Pertanto il tempo di risposta totale è approssimativamente di due RTT, più il tempo di trasmissione del *file* da parte del *server*.

Comandi (*methods*):

- GET: richiedere una risorsa identificata dal campo URL.
- HEAD: richiedere informazioni su una risorsa (metadati).
- POST: invio di ciò che l'utente ha immesso nei campi di un *form*.
- PUT: inviare un oggetto ad un percorso specifico (*directory*) su uno specifico *web server*. Utilizzato anche dalle applicazioni che richiedono di inviare oggetti ai *web server*.
- DELETE: cancellazione di un oggetto su un *server*.

URL → indirizzo nome *host* leggibile + *pathname*.
Il *browser* utilizza il nome dell'*host* per individuare il *server* e stabilire la connessione, utilizza il *pathname* per recuperare la risorsa tramite GET.
Per convenzione i nomi dei *file* vengono specificati nel metodo GET come se fossero *path* assoluti (a partire dalla *root directory* sul *server*), anche se in realtà non è così: il *server* sostituisce l'indicazione della *root* fittizia con quella della *directory* effettiva.

Struttura richieste:
I messaggi di richiesta possono essere costituiti da un numero indefinito di righe (anche una sola). La prima è detta *request line*, quelle successive *header lines*.
Ciascuna riga termina con un carattere di ritorno a capo <CR> (*carriage return*) e uno di carattere di nuova linea <LF> (*line feed*).
Agli *header* segue una riga vuota e il campo *body*, che è vuoto nel caso del metodo GET, ma viene utilizzato nel metodo POST (cioè quando l'utente riempie un *form*).

1. `http://www.dibris.unige.it/index.html`
2. `GET /index.html HTTP/1.1`
3. `Host:  www.dibris.unige.it`
4. `Connection: Close`

2 – contiene tre campi: metodo, URL, versione HTTP.
3 – riga necessaria nonostante sia già in corso una connessione TCP con l'*host*, in quanto questa informazione viene richiesta dalle *cache* dei *proxy*.
4 – includendo questo *header* si sta comunicando al *server* che le connessioni saranno di tipo non persistente: la connessione verrà chiusa dopo aver inviato l'oggetto richiesto.

Struttura risposte:
1 riga di stato + 6 *header* + riga vuota + *body*.
Al metodo segue un'intestazione di lunghezza variabile. Ciascuna riga contiene un *header* tra quelli standardizzati nel protocollo (es. *header host* con il nome + *server* HTTP su una stessa macchina) e si chiude con il terminatore.
Gli *header* sono seguiti da una riga vuota che contiene solo il terminatore <CR><LF>. A questa segue la parte opzionale del *body*.

1. `HTTP/1.1 200 OK`
2. `Connection: Close`
3. `Date: Mon, 23 May 2005 22:38:34 GMT`
4. `Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)`
5. `Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT`
6. `Content-Length: 6821`
7. `Content-Type: text/html; charset=UTF-8`
8. `<CR><LF>`
9. `Inizio byte file (BODY)`

1 – la riga di stato presenta tre campi: nel primo il *server* conferma al *client* la versione in uso. Si sceglie quella più alta che entrambi supportano (ne supportano più di una). Secondo campo codice risposta. Terzo campo spiegazione testuale del codice. A questa seguono gli *header* contenenti le informazioni aggiuntive.
2 – questo *header* viene incluso solo nel caso di connessioni non permanenti. Si comunica al *client* l'intenzione di chiudere la connessione TCP dopo l'invio del messaggio.
3 – indica ora e data di creazione e invio della risposta da parte del *server*. Momento in cui il server recupera la risorsa dal proprio *file system*, lo inserisce nel messaggio ed invia; è sempre in formato GMT.
4 – indica che la riposta è stata generata da un *web server Apache*.
5 – indica l'istante e la data in cui l'oggetto è stato creato o modificato per l'ultima volta. Importante per la gestione dell'oggetto nella *cache*, sia in locale che nei *proxy*.
6 – contiene il numero di *byte* della risorsa inviata.
7 – contenuto è in formato standard Internet MIME (*Multipurpose Internet Mail Extensions*) che estende la definizione del formato dei messaggi di posta elettronica, originariamente definito dal protocollo SMTP, impiegato anche in ambiti diversi, specialmente in contesti di comunicazione o memorizzazione il cui oggetto abbia una codifica non fissata e che debba pertanto essere esplicitata da metadati, come in questo caso.
9 – tipo di dato specificato nel *Content-Type*.

Se il *client* manda un messaggio con il metodo HEAD ottiene la stessa risposta, ma manca il *body*: non viene inviato il file ma soltanto i suoi metadati.

Codici risposta

`HTTP/1.1 404 NOT FOUND`

Risposta non trovata + metadati + contenuti di un *file* HTML con lo stesso contenuto del messaggio di errore.
Diverse classi, tutte a 3 cifre:
1xx: informazioni.
2xx: risposte positive.
3xx: *redirections*.
- 301 MOVED PERMANENTLY
- 304 NOT MODIFIED

4xx: *client error*.
- 400 BAD REQUEST (sintassi protocollo sbagliata)
- 404 NOT FOUND
- 405 METHOD NOT ALLOWED (riconosciuto ma *server* si rifiuta di eseguirlo es.PUT)
- 426 UPGRADE REQUIRED

5xx: server error.
- 500 INTERNAL SERVER ERROR
- 501 NOT IMPLEMENTED
- 507 INSUFFICIENT STORAGE

HTTP 1.0

La prima versione standardizzata nel 1996 prevedeva solo tre metodi (GET, HEAD, POST). Prevede connessioni non permanenti (o persistenti): quando il *client* da via al primo *handshake* la connessione viene aperta, e ottenuta la risposta la connessione viene chiusa. Si ottiene inserendo l'*header* `Connection: Close`
Poiché per ogni oggetto occorre stabilire e mantenere una nuova connessione, ciascun oggetto subisce un ritardo di consegna di due RTT.
Se ad esempio una pagina contiene 10 immagini, sono necessarie 11 richieste, ed ognuna aprirebbe e chiuderebbe la connessione; il vantaggio è che non obbliga il server a mantenere troppe connessioni aperte, ma rende tutto molto lento.

HTTP 1.1

Superati i limiti della versione precedenti tramite l'aggiunta di nuovi metodi e la possibilità di connessioni permanenti (o persistenti): il *server* lascia la connessione TCP aperta dopo l'invio di una risposta, per cui le richieste e le risposte successive tra gli stessi *client* e *server* possono essere trasmesse sulla stessa connessione. C'è un *time out* per evitare di tenere aperte le connessioni indefinitamente.
Inoltre possibilità di implementare meccanismo di risposta in *pipeline*: una volta aperta la connessione il *client* può inviare una serie di richieste consecutive senza aspettare prima di ricevere risposta. Unico vincolo per la correttezza del protocollo: le risposte devono essere inviate nell'ordine in cui sono state ricevute le richieste.
Pipelining + connessioni in parallelo → un *client* può aprire più connessioni con lo stesso *server* e tra queste smistare le richieste.

Caching

Solo per pagine statiche, che restano per la maggior parte invariate nel tempo.
L'idea è che ottenute le risposte il *client* memorizza il file ricevuto in una sua memoria locale. Ma in seguito questa copia dell'oggetto potrebbe essere scaduta, cioè l'oggetto ospitato nel *web server* potrebbe essere stato modificato. *Cache* non reale. Lo scopo è ridurre i messaggi di risposta. Condizione da rispettare è che il contenuto della pagina rimanga costante, cioè le risorse devono rimanere invariate per un tempo sufficientemente lungo. Questo non è attuabile con le pagine dinamiche, che vengono

create nel momento in cui vengono richieste in base a dati memorizzati in un *database* (in quel momento) es. *web mail*.
Il metodo GET condizionale permette alla *cache* di verificare se i suoi oggetti siano aggiornati. La consistenza della *cache* viene mantenuta attraverso l'uso della data di ultima modifica del file: *header Last-Modified* nelle risposte del *server*. Per verificare se il file in locale è ancora valido, cioè se è aggiornato, si fa una richiesta *conditional* GET:

```
GET /index.html HTTP1.1
If-Modified-Since:  <date>
```

Richiede la risorsa solo se è stata modificata dopo quella data.
Il *server* può rispondere in due modi:

- se la copia del *client* è obsoleta invia quella aggiornata
  ```
  HTTP/1.1 200 OK
  Last-Modified:  <date>
  ```
- se non c'è stata alcuna modifica il *client* può utilizzare quella che ha in *cache*. Il *server* confronta le date per questa operazione.
  ```
  HTTP/1.1 304 NOT MODIFIED
  ```

Altro modo utilizzare *server* ausiliari *proxy* (tra *client* e *server* principale): non contengono copia delle informazioni presenti sul *server* originale, ma fanno da intermediari inoltrando le richieste. La risposta del *server* viene memorizzata nel *proxy* e allo stesso tempo ne viene inviata una copia al *client* → caching sul proxy, la risorsa può essere condivisa da più **client**.

Cookie
HTTP è classificato come protocollo senza memoria di stato (*stateless*) → il *server* non tiene traccia delle azioni precedentemente effettuate: invia i *file* richiesti al *client* senza memorizzare alcuna informazione di stato a proposito del *client*, né delle domande né delle risposte. Questo semplifica l'implementazione del *server*, e consente di sviluppare server in grado di gestire migliaia di connessioni TCP contemporaneamente. Tuttavia è necessario che alcune informazioni siano memorizzate comunque (ad es. il *log-in* degli utenti). Per questo è stata fatta una modifica al protocollo che altrimenti è completamente senza stato, attraverso i *cookie*.
Quattro componenti: un *header* nella riposta HTTP del *server*, un *header* nella richiesta HTTP, un *file* sul sistema del *client* gestito dal *browser*, un *database* sul sito.
I *cookie* sono dei numeri. Il *server* nel messaggio di risposta include l'*header*:

```
Set-Cookie:  1678
```

Il *browser* del *client* aggiunge una riga al *file* dei *cookie* che gestisce, con il numero identificativo associato al nome e all'indirizzo del *server*; il *client* deve memorizzarlo in un *file* locale perché questo deve permanere dopo lo spegnimento della macchina.
Ogni volta che il *client* manda una richiesta deve includere l'*header*:

```
Cookie:  1678
```

Il *server* può gestire il meccanismo di numerazione dei *client* associando un nuovo numero quando riceve una richiesta senza questo *header*. Per le richieste successive il *client* verrà automaticamente riconosciuto.

2.2 – PROTOCOLLO SMTP – Simple Mail Transfer Protocol

RFC 821(1982) – RFC 5321 aggiornato
Basato su trasporto TCP – Porta 25
Trasferisce messaggi dal *mail server* del mittente a quello del destinatario.
Messaggi testuali codificati in ASCII a 7 bit (sottoinsieme).

- Il mittente tramite il proprio *mail user agent* compone ed invia il messaggio, fornendo l'indirizzo del destinatario.
- Lo *user agent* invia il messaggio al proprio *server*, dove viene collocato in una coda.

- Il lato *client* di SMTP vede il messaggio nella coda ed apre una connessione TCP verso un *server* SMTP in esecuzione sul *mail server* del destinatario.
- Dopo un *handshaking* SMTP il messaggio viene inviato sulla connessione TCP.
- Il lato *server* di SMTP riceve il messaggio presso il *mail server* del destinatario.
- Il destinatario potrà accedere al *server*, quando lo ritiene opportuno, tramite il proprio *mail user agent*.

È previsto che gli utenti abbiano un *account* sul *server*: ad ogni utente abilitato è associato un *file* chiamato *mailbox*, attraverso SMTP i messaggi vengono inseriti nel *file*, nel quale l'utente troverà copia del messaggio.

Modalità di comunicazione di tipo asincrono: non è richiesta la presenza di mittente e destinatario contemporaneamente per portare a termine la comunicazione, sono il *mail user agent* e il *server agent* che comunicano tra loro, sempre attivi quando la macchina è in funzione. Di solito SMTP non utilizza *mail server* intermedi per inviare messaggi, anche quando i *mail server* finali sono collocati a grande distanza. Qualora il *mail server* del destinatario sia spento, il messaggio rimane nel *mail server* del mittente, e attende un nuovo tentativo di invio. Prevede comunque la possibilità di utilizzare *server* intermedi (rendendo la comunicazione ancora più asincrona).
Può esserci anche connessione tra più *server* (*peer-to-peer*).

Struttura messaggi
RFC 5322 contiene la definizione delle righe di intestazione e la loro interpretazione.
Header + *payolad* tutto codificato in stringhe di caratteri (ASCII), non numeri.
Le righe dell'*header* contengono stringhe costituite da una parola chiave seguita dai due punti a loro volta seguiti da un valore. Alcune informazioni sono obbligatorie, come l'indirizzo del destinatario, altre sono opzionali.
La lunghezza massima di una riga è di 72 caratteri; le restrizioni sono dovute al periodo di definizione del protocollo.

```
To: keyword a cui segue l'indirizzo <cr><lf>
From:      <cr><lf>
Object:    <cr><lf>
Bcc/cc:    <cr><lf>
<cr><lf><cr><lf>
..
payload
..
<cr><lf>.<cr><lf>
```

Stabilita la connessione è affidabile in quanto *stream* (TCP). L'invio del messaggio è preceduto da uno scambio tra *client* e *server*: il *client* SMTP fa stabilire a TCP una connessione sulla porta 25 verso il *server* SMTP; se quest'ultimo è inattivo il *client* riprova in seguito. Se il *server* è attivo invia il messaggio identificandosi (indica anche gli indirizzi *e-mail* del mittente e del destinatario), il *server* risponde OK e il *client* invia il messaggio vero e proprio. Per confermare la trasmissione il *server* invia ancora OK, poi la connessione viene chiusa. Per ogni messaggio si apre e chiude la connessione. Il *server* verifica se il messaggio sia destinato ad un *client* sulla macchina; se sì appende il messaggio nel *file mailbox*, se no contatta l'altro *server* per effettuare l'*hop* successivo. Se la macchina di destinazione non è attiva non va a buon fine la connessione TCP: il protocollo prevede una serie di tentativi successivi dilazionati, più un *time out* finale dopo il quale si desiste e torna al mittente una segnalazione di errore, sempre sotto forma di posta elettronica (circa 3 giorni).
Se il *server* è configurato come *open-relay* accetta di inoltrare *e-mail* da qualunque parte di Internet senza controlli. Inizialmente questa era la configurazione di *default*, ma poiché non è previsto nessun tipo di autenticazione, venivano pesantemente

bersagliati da *spamming*, e soggetti a *spoofing* (durante la connessione il *client* falsifica l'indirizzo mittente del messaggio).

ESMTP – Extended (Enhanced) Simple Mail Transfer Protocol
Autentica i *client* prima di accettare messaggi. Prevede varie modalità di autenticazione dei *client* più una verifica della consistenza dell'indirizzo IP fornito. Inizialmente lo standard originale riteneva sufficiente che aperta la connessione il *client* si presentasse al *server* tramite HELO + indirizzo *client*; nella versione aggiornatainvece la connessione viene aprta con il comando EHLO + indirizzo *client.* Non blocca lo *spam*, ma si sa chi lo ha inviato.

Protocolli di accesso alla posta
Quattro possibilità per leggere i messaggi ricevuti:

1. *Log-in* e accedere al *file mailbox* sul *server.* Poiché un *server* che gestisce posta esegue lato *client* e *server* di SMTP, se il *server* di posta del destinatario risiedesse sulla sua macchina locale, questa dovrebbe essere sempre accesa e connessa. Per questo l'utente ha in esecuzione lo *user agent* in locale, ma accede ad un *server* condiviso con altri utenti, generalmente gestito dall'ISP.
2. Tramite protocollo POP3 (*Post Office Protocol* v3 - RFC 1939), che entra in azione quando lo *user agent* del *client* apre una connessione TCP verso il *mail server* sulla porta 110. Stabilita la connessione, si procede in tre fasi: autorizzazione, transazione, aggiornamento. Nella prima lo *user agent* autentica l'utente (in chiaro). Nella seconda lo *user agent* recupera i messaggi, e può marcarli per la cancellazione. La terza fase ha luogo dopo che il *client* ha inviato il comando *quit*, che conclude la sessione; in questo momento vengono rimossi i messaggi precedentemente marcati per la cancellazione.
   Permette di avere una diverse macchine *client*, dal *server* si può ottenere una copia scaricata in locale del *file mailbox* (ci si può connettere quando si vuole); si può mantenere la versione originale sul *server* oppure no (il *file* continua a crescere). Altre opzioni erano: vedere tutto l'elenco dei messaggi ricevuti senza doverlo necessariamente scaricare (serviva per connessioni lente).
3. IMAP4 (*Internet Mail Access Protocol* v4 - RFC 3501) protocollo che risolve lo stesso tipo di problemi, ma progettato espressamente per l'accesso di più *client* alla stessa *mailbox*. I messaggi restano sul *server* finché non vengono cancellati, sul *client* vengono scaricati tramite un meccanismo di *caching*, solo quelli ritenuti importanti. Gestisce copie multiple della *mailbox* mantenendo la consistenza dell'informazione.
4. Webmail (HTTP) ci si connette al *web server* e tramite quello alla *mailbox*. I messaggi rimangono sul *server*, non vi si può accedere in locale.

E-mail → per trasmettere dati non ASCII si usa una tecnica di codifica per cui i dati come rappresentazioni di *byte* vengono trasformati in sequenze di caratteri in base 64 (RAD x64 algoritmo originario, poi sistemi più sofisticati definiti dallo standard NINE).

2.3 – PROTOCOLLO DNS – Domain Name System
RFC 1034-35 - Protocollo tipo *client/server*. Servizio di risoluzione dei nomi.
Basato su trasporto UDP - Server in ascolto sulla porta 53.
Gli *host* sono identificati tramite il nome o l'indirizzo IP. Compito principale del servizio è tradurre i nomi nei corrispondenti indirizzi.
Consiste in un *database* distribuito implementato in una gerarchia di DNS *server* + protocollo a livello applicazione che consente agli *host* di interrogare il *database*.
Indirizzo simbolico ← DNS → Indirizzo IPv4-IPv6
Idea base di implementazione: una macchina su cui gira un processo c*lient* interessato alla comunicazione. Altra macchina su cui gira processo *server* DNS con un *socket* in

ascolto sulla porta 53, attraverso un'operazione di BIND (*Berkeley Internet Name Domain* - i DNS *server* sono generalmente macchine UNIX che eseguono questo sofware). Il protocollo è di tipo domanda/risposta.
Il *client* invia la richiesta (*query*) contente l'*hostname* all'interno di un datagramma UDP verso il server, il quale include nel datagramma di risposta l'indirizzo IP corrispondente. Il *client* ha un *time out* per la ricezione della risposta. Sia le richieste che le risposte possono essere più di una all'interno di un singolo datagramma. Lo scambio avviene utilizzando un unico formato per le informazioni composto da quattro campi (sia richieste che risposte).

L'associazione avviene a fronte di un *database* per tutte le macchine della rete, distribuito, gerarchico, e continuamente aggiornato. La distribuzione delle informazioni avviene differenziando i *server*, che svolgono ruoli diversi all'interno della struttura.
La gerarchia del *database* vero e proprio è implementata come struttura ad albero:

- ***Root server***: la radice è concettualmente unica, ma per disponibilità e affidabilità delle informazioni ne esistono 400 duplicati e dislocati in tutto il mondo. Sono gestiti da 13 organizzazioni. Forniscono gli indirizzi IP dei TLD *server*.
- ***Top-Level Domain server***: livello immediatamente inferiore alla radice. Ognuno è specializzato per un sottoinsieme di Internet, gestiscono i domini di primo livello (com, org, net, edu, gov, e quelli nazionali).
- **Server autoritativi**: ogni organizzazione dotata di *host* pubblicamente accessibili tramite Internet deve fornire *record*, altrettanto pubblicamente accessibili, che associno i nomi di tali *host* ad indirizzi IP. Il DNS *server* autoritativo dell'organizzazione ospita questi *record*. (es. unige) Sono ulteriormente suddivisi in altri *server* autoritativi (es. dibris) che si occupano di *file* di configurazione gestiti manualmente dall'amministratore del sistema, tabella con tutte le macchine associate.

Il *server* a cui si connettono i *client* si chiamano locali (***local DNS server***). Il loro scopo è interfacciarsi con una quantità limitata di *client* (a livello LAN). Non appartengono strettamente alla gerarchia dei DNS *server*.
Il punto finale negli indirizzi simbolici (che non viene messo dall'utente, ma direttamente dal *software*) indica la radice.

Record e messaggi DNS
I *server* che implementano il database distribuito memorizzano i record di risorsa (RR – *Resource Record*). Ogni messaggio di risposta ne trasporta uno o molteplici.
Un record è composto da quattro campi:
`(Name, Value, Type, TTL)`

- **TTL** → *time to live* (diverso da quello contenuto nell'*header* TCP), è il tempo residuo di vita di un *record,* e determina quando le informazioni scambiate andranno rimosse dalla *cache*.
- ***Type*** → se *Type=A*, allora *Name* è il nome dell'host, e *Value* il suo indirizzo IP corrispondente. Questo è il *record* più comune.
  Se *Type=NS*, allora *Name* è un dominio, e *Value* è il nome del DNS *server* autoritativo che sa come ottenere gli indirizzi IP degli *host* nel dominio. Questo *record* viene utilizzato per instradare le richieste DNS successive alla prima nella concatenazione delle *query*.
  Se *Type=CNAME* (*canonical*), allora *Value* rappresenta il nome canonico dell'*host* per il sinonimo *Name*. Questo *record* può fornire agli *host* richiedenti il nome canonico relativo ad un *hostname*.
- Se *Type=MX*, allora *Value* è il nome canonico di un *mail server* che ha come sinonimo *Name*. Consente agli *hostname* dei *mail server* di avere sinonimi semplici.

16 *bit*	16 *bit*
ID	Flag
#Richieste	#RR Risposta
#RR autoritativi	#RR addizionali
Sezione Richieste	
Sezione Risposte	
Sezione autoritativa	
Sezione addizionale	

] 12 *byte* sezione di intestazione

] Nome e tipo di una *query*
] RR in risposta alla *query*
] *Record server* autoritativi
] *Record* utili

ID → Identificatore richiesta.
Flag → il primo bit è quello di richiesta/risposta (*query/reply*), che indica se il messaggio è una richiesta (0) o una risposta (1). Un altro *bit* viene impostato nelle risposte quando il DNS *server* è autoritativo per il nome richiesto. Un altro è quello di richiesta di ricorsione (*recursion-desired flag*), che viene impostato quando un *client* desidera che il *server* effettui ricorsione qualora non disponga del *record*.
Sezione richieste → contiene informazioni sulle richieste che stanno per essere effettuate. Include inoltre un campo con il nome che sta per essere richiesto, e un campo che indica il tipo della richiesta su quel nome.
Sezione risposte → contiene i *record* di risorsa relativi al nome richiesto originariamente. Una risposta può contenere più RR, dato che un *hostname* può avere più indirizzi IP (*web server* replicati).
Sezione autoritativa → contiene i *record* di altri *server* autoritativi.

Sono stati standardizzati due diversi metodi per la ricerca:
Ricorsiva: se il *local DNS server* è in modalità ricorsiva prende carico della richiesta del *client* contattando la *root*; se anche questa è in modalità ricorsiva si scende nell'albero fino ad ottenere la risposta, e si fa il percorso al contrario tornando al *client*.
Iterativa: il *client* manda la richiesta ai livelli dell'albero dalla radice scendendo, trovata la macchina questa risponde al client.
In teoria ogni richiesta DNS può essere iterativa o ricorsiva. In pratica la richiesta dall'*host* iniziale al *server* locale è ricorsiva, mentre le restanti sono iterative.
I *server* eseguono soltanto un *look-up* del loro database interno, è responsabilità del *client* percorrere l'algoritmo fino ad ottenere la risposta finale. Il numero di messaggi per metodo di ricerca è lo stesso. Nella modalità ricorsiva la risposta viene propagata a tutte le macchine coinvolte, in quella iterativa solo il *client* la conosce.

DNS caching
Il DNS sfrutta in modo estensivo il *caching* per migliorare le prestazioni di ritardo e per ridurre il numero di messaggi DNS in rete.
Se si è in modalità ricorsiva tutte le macchine possono fare *caching*, in modalità iterativa può farlo solo il client: la soluzione adottata prevede che il *local DNS server* sia in modalità ricorsiva, mentre tutti gli altri in modalità iterativa; così il *local server* può conservare informazioni e permettere a *client* diversi di accedervi. Possibilità di restituire immediatamente un indirizzo IP senza dover interrogare altri *server*.
La *cache* del *local server* permette di avere risposte non autoritative e velocizzare la comunicazione per i *client*, inoltre evita di sovraccaricare i server con una quantità eccessiva di richieste. Crea però un problema di sicurezza, come gli attacchi di *cache*

poisoning: tramite delle richieste vengono inserite nella *cache* record falsi con un TTL molto alto, cosicché il server DNS lo conserverà per molto tempo; ciò consente di reindirizzare un nome di dominio *web* verso un indirizzo IP diverso da quello originale. Questo può essere la base, tra altre cose, per *phishing* e *spoofing*.
Originariamente il protocollo DNS non prevedeva protezione da questi attacchi, ma da quando si è diffuso l'uso rete per scopi commerciali è stato introdotto un controllo nell'identificatore (primi 2 *byte* dei messaggi), i numeri vengono generati in modo casuale su 16 *bit* e si confrontano per rispondere solo a richieste effettivamente ricevute.
Brute force: attacco che genera in questo caso 2^{16}-1 numeri e manda nel minor tempo possibile tutte le risposte al *server*. Questo basterebbe a superare il controllo dell'identificatore. Per questo viene aggiunta un'altra informazione randomizzata a livello di trasporto: nell'*header* UDP c'è il numero di porta sorgente (usato dal server DNS per contattare gli altri server), invece di usare una porta predeterminata il *server* genera una porta effimera tramite la quale effettuare la richiesta: 16 *bit* ID + 16 *bit source port.* Tutto questo vale se l'attaccante non vede il messaggio di richiesta.

2.4 – PROTOCOLLO FTP – File Transfer Protocol

RFC 775 – Basato su trasporto TCP – *Port* 20-21
In una tipica sessione FTP, l'utente utilizza un *host* (locale) per trasferire file da o verso un *host* remoto. Per accedere ed essere autorizzato a scambiare informazioni con il *file system* remoto, da sempre si prevede che l'utente fornisca un nome identificativo e una password; dopo l'autenticazione sul server può trasferire file tra i due *file system*. Il *server* ha le due porte, il *client* può connettersi sulla 21 e ha a disposizione una serie di comandi per il trasferimento.
L'utente interagisce con il protocollo tramite l'*FTP user agent*, che si occupa di fornire un'interfaccia e, in base alle richieste dell'utente, comunicare al *software* che implementa il *client* FTP quali comandi inviare sul *socket*.
Codifica comandi: forma testuale ASCII a 7 *bit*, 4 caratteri maiuscoli e ciascuno delimitato da un carattere di ritorno a capo e da uno di nuova linea.
`USER username`: il *client* si identifica sul *server*
`PASS password`: invio della *password* al *server*
Riconosciuta l'autenticazione è disponibile tutta una serie di comandi, tra cui:
`LIST`: si chiede al *server* di inviare un elenco di tutti i file contenuti nella *current directory* remota. La risposta viene spedita su una connessione dati nuova e non persistente.
`RETR filename`: *retrieve*, *download* di una copia di un file dalla *current directory* remota; per l'invio del file viene inizializzata una connessione dati.
`STOR filename`: *store*, upload di un file sulla *current directory* del *server*.
Il protocollo utilizza due connessioni TCP *client/server* parallele:
– *Port* 21 → connessione di controllo (comandi, risposte)
– *Port* 20 → trasferimento dati
Risposte del *server*: numero (ID risposta) + stringa di caratteri (tramite Telnet).
Due modalità per il trasferimento dati (connessione normale 8 *bit*):

- Attiva → *default*. Il *client* deve predisporre un altro *server socket,* per la connessione dati, su una porta effimera. Attraverso la connessione di controllo comunica al *server* la porta su cui è in ascolto. Dopo di che si da via al *three-way handshake* TCP e viene stabilita la connessione dati. Questa modalità oggi è considerata non ottimale, soprattutto se associata all'uso di *firewall* (non è ritenuto sicuro accettare *server* FTP in entrata).
- Passiva → attivata esplicitamente dal comando `PASV`. Il *server* risponde sulla connessione di controllo comunicando il numero di porta effimera sulla quale ha deciso di ricevere la connessione dati dal *client*, ed è quest'ultimo a dare via al

three-way handshake. Diversamente dalla modalità attiva bisogna aspettare il *server*, ma viene considerata più sicura in quanto la connessione è sempre in uscita dal *client*, e non in ingresso.

Variante FTP *anonymous* → in questo caso viene superato il limite dell'autenticazione, l'accesso al *server* e aperto a chiunque (per dati di pubblico dominio). Viene fissato *username* = *anonymous*, *password* non considerata; si suggeriva all'utente di inserire nel campo `PASS` il suo indirizzo *e-mail* per tenere traccia in qualche modo del traffico, ma a questo non veniva applicato alcun controllo.
Applicazione tipica di questo protocollo è quella del *download*, è disponibile anche l'*upload* (solitamente per la segnalazione di *bug*), ma come alternativa, non sono coesistenti.
FTP è considerato protocollo da non utilizzare in quanto si passa in chiaro l'autenticazione → SFTP versione cifrata, anche dei dati, a livello applicativo.

2.5 – PROTOCOLLO NTP – Network Time Protocol

Versione attuale v4 RFC 5905 – Basato su trasporto UDP – *Port* 123
Originariamente sviluppato nel 1985. Sincronizzazione degli orologi delle macchine connesse alla rete. Tipo *client/server*.
Serie di *client* che hanno un orologio solitamente al quarzo, si può ritenere affidabile, ma variazioni atmosferiche possono modificarne la frequenza base. L'inizializzazione può essere fatta a mano: errori anche di secondi.
Gli orologi più precisi in assoluto sono quelli atomici (decadimento radioattivo).
Server a cui è associato un orologio atomico: il *client* può richiedere l'ora, ma questa è totalmente inaffidabile a causa dei ritardi nella comunicazione. Si cerca di ovviare misurando il RTT (sulla base dell'orologio locale del *client*, misurando più volte); anche questo non è attendibile per la componente di ritardo della risposta del *server*, inoltre si utilizza come punto di partenza l'ora ancora da sincronizzare.
Modifica allo schema: prese le misurazioni di 4 tempi, invece che 2
Due orologi diversi non possono essere messi a confronto se non sono sincronizzati tra loro, ma si elimina la componente di ritardo del *server*.
t1 = invio richiesta (*client clock*) t2 = ricezione richiesta (*server clock*)
t3 = invio risposta (*client clock*) t4 = ricezione risposta (*server clock*)
RTT = (t4-t1) – (t3-t2) bias
Si ripete più volte la misurazione, e la valutazione avviene soltanto dopo aver accumulato una certa quantità di risultati (si sincronizza solo quando la connessione al *server* è sufficientemente affidabile). Le macchine che collaborano alla realizzazione del protocollo NTP sono suddivise in *strata,* a partire dal livello 0 che è l'orologio atomico. Connessione punto-punto (tipo stampante, tempo connessione predicibile, no traffico condiviso) tramite porta seriale RS232:
Livello 1: *server* connesso localmente all'*atomic clock.*
Livello 2: *server* che si connette al livello 1 (e così via).
Tutti questi *server* utilizzano lo stesso protocollo per la sincronizzazione, che è anche lo stesso del *client* (*peer-to-peer*). Ci possono essere più *server* di uno stesso livello connessi tra loro, in quanto il confronto di informazioni tra *server* dello stesso livello migliora la sincronizzazione, e gli errori vengono compensati meglio.
Essendo un protocollo basato su UDP si può contattare il *server* senza prima aprire una connessione: non c'è *overload* del *server* per il mantenimento delle connessioni. Per lo stesso motivo si possono perdere pacchetti, ma questo si evita tramite l'inserimento di un *time out* lato *client*, legando il protocollo al funzionamento dell'orologio locale; ogni tot secondi il *client* invia una richiesta al *server*.
La configurazione tipica di un *client* prevede un file di configurazione, associato al protocollo NTP, contenente l'elenco degli indirizzi IP dei *server* NTP conosciuti (di vari livelli inferiori a 0). Quando si accende la macchina parte il processo NTPd (*Network*

Time Protocol daemon) che comincia a contattare i *server*: inizialmente la frequenza con cui vengono contattati è elevata, e rallenta dal momento in cui cominciano ad arrivare le risposte. Si può anche evitare di continuare a contattarli tutti, eliminando quelli che ignorano, quelli di livello più basso; oppure servendosi del RTT: si tiene conto delle variazioni, oltre che dei valori, tra l'ultima misurazione e una media dei valori precedenti. La media mobile della variazione darà un'indicazione sull'affidabilità del *server*, si continuano a contattare quelli che mandano risposte a bassa variabilità, non importa se il RTT è alto.
Si prosegue fino a quando i *server* contattati sono solo due o tre, se l'orologio si sta mantenendo sincronizzato si diminuisce la frequenza (raddoppio del tempo).
Altra possibilità è avere un ricevitore GPS sulla macchina (= livello 1).
Modo per risincronizzare l'orologio: diminuire/aumentare la frequenza di *clock* della macchina, così da andare a distribuire l'errore nel tempo.
Problemi di sicurezza dovuti all'imprecisione → no è prevista autenticazione, quindi si presentano problemi di falsificazione degli indirizzi dei protocolli di rete; inoltre, essendo basato su UDP, non prevede il *three-way handshake*; questo comporta che possano arrivare risposte da chiunque con un IP sorgente falsificato, oppure attacchi DdoS tramite *spoofing* degli indirizzi.
Il *clock* sulla scheda madre è separato da quello della CPU, ed è accessibile tramite un ciclo di lettura. Registro che contiene un numero intero incrementato con una frequenza predefinita. È alimentato a batteria, e l'accesso in lettura è costoso in termini di tempo (viene fatto al momento del *bootstrap*), il valore del registro viene copiato in RAM.

3 – LIVELLO DI TRASPORTO

Mentre un protocollo a livello di rete fornisce comunicazione logica tra *host* differenti, un protocollo a livello di trasporto mette a disposizione una comunicazione logica tra processi applicativi eseguiti su *host* differenti. Questi protocolli sono implementati nei sistemi periferici. Lato mittente, il livello di trasporto converte i messaggi provenienti da un processo in pacchetti (o segmenti – *transport-layer segment*), spezzettandoli se necessario, e aggiungendo ad ognuno un'intestazione di trasporto. Il segmento viene passato al livello di rete, tramite il quale viene incapsulato in un datagramma e inviato. I datagrammi sono parole su 32 bit suddivise in porzioni più piccole. I *router* intermedi agiscono soltanto sui campi a livello di rete del datagramma, senza esaminare i campi del segmento incapsulato al suo interno. Lato ricevente, il livello di rete estrae il segmento, mentre quello di trasporto lo elabora perché sia disponibile per il processo destinatario.
UDP (*user datagram protocol*) fornisce alle applicazioni un servizio non affidabile e non orientato alla connessione. TCP (*transmission control protocol*) fornisce alle applicazioni un servizio affidabile e orientato alla connessione.

Multiplexing (MPX) e Demultiplexing (DMPX)

Il compito principale di questi due protocolli è l'estensione del servizio di consegna di IP. Come il servizio di trasporto da *host-to-host* fornito dal livello di rete diventa un servizio di trasporto *process-to-process* per le applicazioni in esecuzione sugli *host*.
Un processo può gestire più *socket* attraverso cui i dati fluiscono dalla rete al processo e viceversa; ciascuno di questi *socket* ha un identificatore univoco il cui formato dipende dal fatto che si tratti di TCP o UDP.
Multiplexing → raccolta di tutti i messaggi in uscita dalla stessa macchina e utilizzo stesso punto di accesso. Azione di radunare frammenti di dati dai diversi *socket* sull'*host* di origine e incapsulare ognuno con intestazioni a livello di trasporto (che

verranno poi utilizzate per il *demultiplexing*) per creare dei segmenti e passarli al livello di rete.
Demultiplexing → raccolta di tutti i messaggi ricevuti e gestione tramite le applicazioni. L'*host* di ricezione indirizza verso il *socket* appropriato il segmento a livello di trasporto in arrivo. Ciascun segmento a livello di trasporto ha vari campi deputati allo scopo: numero di porta di origine e numero porta di destinazione. Il livello di trasporto esamina questi campi per identificare il *socket* di ricezione e quindi vi dirige il segmento. Consiste nel trasportare i dati dei segmenti a livello di trasporto verso il giusto *socket*. I numeri di porta sono su 16 *bit* (0-65535). È il processo che chiede al SO che il particolare *socket* sia messo in comunicazione con una delle porte ancora non assegnate tramite la *syscall bind*. I numeri compresi tra 0 e 1023 sono chiamati numeri di porta noti (*well-known port number*) e sono riservati ad alcuni protocolli applicativi.

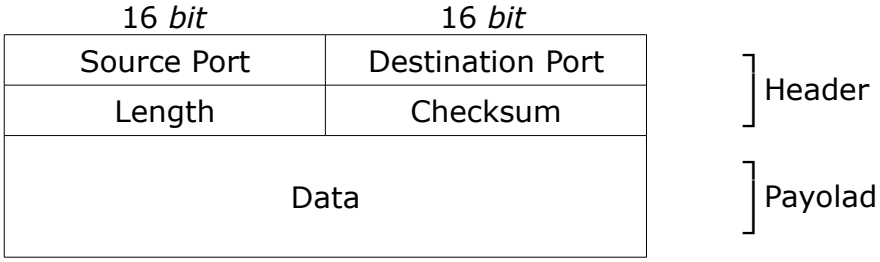
Un *socket* è il punto in cui il codice applicativo di un processo accede al canale di comunicazione. In base alla modalità di connessione si distinguono due tipi principali che corrispondono ai due protocolli di trasporto: *stream socket* (TCP) e *datagram socket* (UDP).

- **Stream** → può individuare un altro processo, sulla stessa macchina o su un'altra, purché anche lì il *socket* sia di tipo *stream*. Stabilisce un canale bidirezionale tra i due *socket*. Se i due processi connessi si trovano sulla stessa macchina, il SO utilizza lo stesso buffer per invio e ricezione. Se i processi applicativi sono su due *host* diversi è necessario il sistema di comunicazione tramite protocolli Internet. In questa modalità di comunicazione i *byte* fluiscono in sequenza attraverso il *socket*. La *syscall* che permette di inviare può essere la stessa utilizzata per scrivere i file (*write*), oppure *send*. Entrambe implementano il *socket* come se fosse un *file*. Il ricevente può fare un'operazione di lettura da *file* (*read*), oppure una *recv*. I *byte* mantengono il loro ordine: affidabilità del canale, viene garantita la ricezione di tutti i *byte*. Problemi di trasmissione e perdita dati vengono gestiti dai livelli di astrazione inferiori.
- **Dgram** → comunicazione di tipo *datagram* (limite sulla dimensione massima). Una certa quantità di *byte* viene inviata insieme. Non c'è un canale di comunicazione predefinito. Attraverso lo stesso *socket* si possono fare invii a destinatari diversi, basta l'indirizzo. Servizio non affidabile, se arriva il messaggio è intera, altrimenti è perso. Tipo *best-effort delivery service*, cioè se riesce recapita, ma non offre garanzie. Meccanismo di sincronizzazione per cui chi riceve deve sapere la dimensione dei datagrammi (gestito esternamente rispetto ai protocolli di comunicazione).

3.1 – PROTOCOLLO UDP – User Datagram Protocol

RFC 768

Prende i messaggi dal processo applicativo, aggiunge il numero di porta di origine e il numero di porta di destinazione, per le operazioni di MPX e DMPX, più altri due campi, e passa il segmento risultante al livello di rete (che lo incapsula in un datagramma IP). Poiché non prevede formalità intermedie, non introduce alcun ritardo nello stabilire una connessione. Inoltre UDP non conserva lo stato della connessione, e non tiene traccia di parametri come numero di sequenza e di *acknowledgement*. Un *server* dedicato ad una particolare applicazione può generalmente supportare molti più *client* attivi quando questa utilizza UDP. Infine, l'intestazione dei pacchetti UDP ha lunghezza massima di 8 *byte*. Tutti questi possono essere motivi per cui preferire UDP rispetto a TCP.

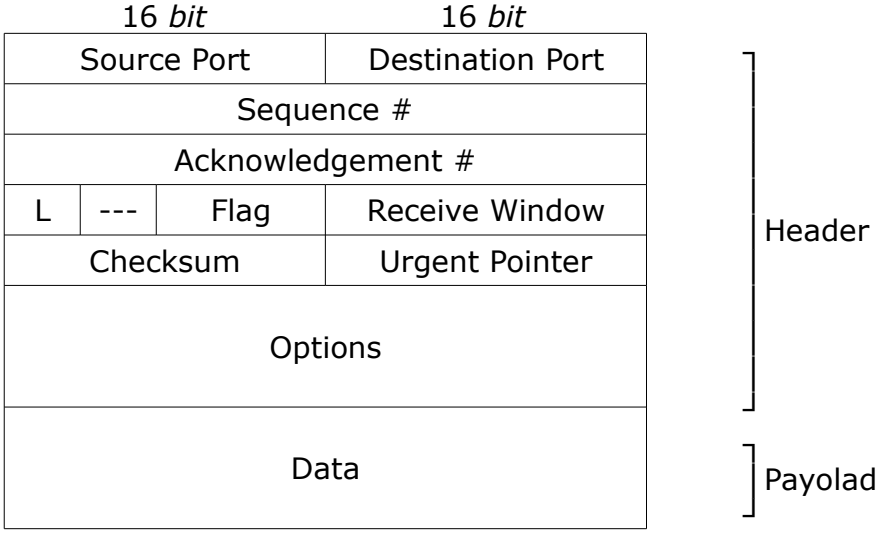


Source port → identifica il numero di porta sull'*host* del mittente del datagramma.
Destination port → identifica il numero di porta sull'*host* del destinatario del datagramma.
Length → contiene la lunghezza totale in *byte* del datagramma UDP (header+dati). Intervallo $0–2^{16}$-1.
Checksum → contiene il codice di controllo del datagramma, utilizzato per determinare se i *bit* del segmento UDP siano stati alterati durante il trasferimento. Il numero è il risultato dell'algoritmo di calcolo definito nell'RFC del protocollo. Viene inserito alla partenza del messaggio, mentre alla ricezione viene ripetuto il calcolo e si confrontano i due risultati per verificare la presenza di errori. Se sì il datagramma viene scartato. Questo controllo esiste in quanto, dato che non sono garantiti né l'affidabilità del singolo collegamento, né il rilevamento di errori in memoria, UDP deve mettere a disposizione un meccanismo di verifica, a livello di trasporto, su base *end-to-end*.
Data → contiene i dati del messaggio.

3.2 – PROTOCOLLO TCP – Transmission Control Protocol

RFC 793 – 1122 – 1323 – 2018 – 2581

TCP viene detto *connection-oriented* in quanto, prima di effettuare lo scambio di dati, i processi devono effettuare il *three-way handshake*, cioè inviarsi reciprocamente alcuni segmenti preliminari per stabilire i parametri del successivo trasferimento.



L'header TCP ha lunghezza variabile (alcuni campi sono opzionali).
Oltre a numero di porta sorgente, numero di porta di destinazione, e *checksum* già visti nella struttura dei segmenti UDP, comprende i campi:

Sequence number → numero del primo *byte* del segmento nel flusso.
Acknowledgement number → numero *byte* ricevuti con successo.
***Header Length* (o *data offset*)** → 4 *bit*. Indica la lunghezza dell'*header* del segmento TCP in multipli di 32 *bit*; tale lunghezza può variare da 20 a 60 *byte*, a seconda della presenza e della lunghezza del campo facoltativo *Options*.
Reserved → 4 *bit* non utilizzati e predisposti per futuri sviluppi del protocollo; dovrebbero essere impostati a zero.
Flag → 8 *bit*.

- CWR (*Congestion Window Reduced*): se impostato a 1 indica che l'*host* sorgente ha ricevuto un segmento TCP con il flag ECE impostato a 1 (aggiunto con RFC 3168).
- ECE (*Explicit Congestion Notification*): se impostato a 1 indica che l'*host* supporta ECN durante il *three-way handshake* (aggiunto con RFC 3168).
- URG: utilizzato per indicare la presenza nel segmento di dati che l'entità mittente, a livello superiore, ha marcato come urgenti. Se impostato a 1 indica che nel flusso sono presenti dati urgenti alla posizione (*offset*) indicata dal campo *Urgent pointer.*
- ACK: utilizzato per indicare che il valore nel campo di *acknowledgement* è valido. Indica se un segmento è stato ricevuto con successo.
- PSH: se impostato a 1 il destinatario dovrebbe inviare immediatamente i dati al livello superiore.
- RST: se impostato a 1 indica che la connessione non è valida; viene utilizzato in caso di grave errore. Utilizzato insieme al flag ACK per la chiusura di una connessione.
- SYN: se impostato a 1 indica che l'*host* mittente del segmento vuole aprire una connessione TCP con l'*host* destinatario e specifica nel campo *Sequence number* il valore dell'*Initial Sequence Number* (*ISN*); ha lo scopo di sincronizzare i numeri di sequenza dei due *host*. L'*host* che ha inviato il SYN deve attendere dall'*host* remoto un pacchetto SYN-ACK.
- FIN: se impostato a 1 indica che l'*host* mittente del segmento vuole *c*hiudere la connessione TCP aperta con l'*host* destinatario. Il mittente attende la conferma dal ricevente (con un FIN-ACK). A questo punto la connessione è ritenuta chiusa per metà: l'*host* che ha inviato FIN non potrà più inviare dati, mentre l'altro *host* ha il canale di comunicazione ancora disponibile. Quando anche l'altro *host* invierà il pacchetto con FIN impostato, dopo il relativo FIN-ACK, la connessione sarà completamente chiusa. (versione lunga 4 messaggi, breve solo 3).

Receive window → 16 *bit*. Campo utilizzato per il controllo di flusso.
Urgent pointer → indica la posizione dell'ultimo *byte* di dati urgenti.
Otptions → facoltativo e di lunghezza variabile.

Handshake a tre vie: 1)il *client* invia un primo segmento al *server* con il *flag* SYN a 1, per indicare una richiesta di connessione. 2) il server risponde SYN-ACK, sincronizzazione e *acknowledgement* per la connessione. 3)il *client* invia ACK e la connessione è stabilita. Questo terzo messaggio può già contenere dati nel *payload*, i due precedenti no. Tutti i messaggi inviati successivamente hanno *bit* ACK attivo + *payload* (vero e proprio trasferimento).

La quantità massima di dati prelevabili e posizionabili in un segmento TCP viene limitata dalla dimensione massima di segmento (MSS); questo valore viene generalmente impostato determinando prima la lunghezza del *frame* più grande che può essere inviato a livello di collegamento dall'*host* mittente, chiamata unità trasmissiva massima (MTU).

Per implementare lo stream servono due contatori, che corrispondono ai campi più importati dell'intestazione. TCP vede i dati come un flusso di *byte* non strutturati, ma ordinati. Nel buffer di ricezione si ricostruisce la sequenza di *byte* indipendentemente dall'ordine di ricezione dei *frame*. Mentre il *Sequence number* serve per indicare il numero del primo *byte* contenuto in quel segmento, rispetto all'intero flusso, il campo *Acknowledgement number* serve al destinatario per dire al mittente quali *byte* ha ricevuto nella ricostruzione dello *stream*. Questo permette di gestire l'eventuale perdita di messaggi.
Nell'implementazione i contatori non partono da 0. Per motivi di sicurezza al momento della connessione viene generato un numero casuale da cui si parte.
1) Il *client* genera n casuale e lo scrive nel campo *Sequence number*, inviandolo nel primo pacchetto con anche il *flag* SYN attivo.
2) Il *server*, ricevuta la richiesta, se accetta la connessione genera il numero casuale e lo aggiunge alla risposta nel campo *Acknowledgement number*.
3)Tutti i messaggi successivi conterranno un numero di sequenza che parte dal numero casuale iniziale, a cui si aggiunge il numero dei *byte* ricevuti.
L'obiettivo primario è quello di distinguere vecchi *frame* rimasti indietro ed evitare che vengano inseriti all'interno del nuovo *stream.* L'aumento della sicurezza risiede nel fatto che dopo il *three-way handshake* bisogna conoscere il numero di sequenza corretta da inserire affinché i messaggi vengano accettati.

Time out e RTT
Nello *stream* i *byte* vengono inviati con l'indicazione del numero di sequenza, e un certo *time out*, scaduto il quale il frammento si considera perso e viene rinviato (buffer trasmissione nel *client*).
Problema dell'impostazione del *time out*; è in base all'effettivo canale di comunicazione sottostante che si calcola (latenza, distanza).
Il protocollo prevede il calcolo tramite RTT: timer ulteriore per valutare il tempo di andata e ritorno del messaggio. Viene fatto partire all'invio di un frammento, e si misura l'istante in cui si riceve ACK di ritorno per quel segmento. Questa è comunque una stima perché i messaggi hanno dimensioni diverse. Sulla misurazione del RTT si basa un *time out* diverso e adatto ad un particolare canale.
$$\mathbf{ERTT = (1 - \alpha)ERTT + \alpha \cdot nuova\ misura\ RTT}$$
$\alpha = 1/8$
Valore iniziale ERTT = 1 sec
Media mobile esponenziale ponderata: per seguire le variazioni del RTT i valori più importanti sono quelli più recenti a partire dall'ultima misurazione. Il peso dei campioni decresce esponenzialmente al procedere degli aggiornamenti.
Viene fatta anche una stima della variabilità:
$$\mathbf{DevRTT = (1 - \beta)DevRTT + \beta \cdot |Nuova\ misura\ RTT - ERTT|}$$
Una volta modificato il RTT si calcola la distanza tra la nuova misurazione e quelle precedenti.
$$\mathbf{Timeout = ERTT + 4 \cdot DevRTT}$$
Se scatta il *time out* e il *frame* viene ritrasmesso non si aggiorna più il RTT, anche se alla seconda volta si riceve ACK, perché non si sa a quale invio sia riferito: si misura solo se va a buon fine la prima trasmissione. La specifica del protocollo prevede che in caso di ritrasmissione il *time out* venga raddoppiato ad ogni invio (senza aggiungere la misura) fino alla ricezione di ACK.
Si riprende a misurare sulla nuova trasmissione se si riceve subito ACK.

Controllo di flusso (flow-control service)
Gli host agli estremi delle connessioni TCP riservano dei *buffer* di ricezione per la connessione; quantità di dati che può essere inviata prima di mandare in *overflow* il

buffer di ricezione del destinatario (non si prendono in considerazione i *buffer* intermedi).
TCP offre un servizio di controllo di flusso alle proprie applicazioni per evitare che il mittente saturi il *buffer* del destinatario: è un servizio di confronto sulla velocità, dato che paragona la frequenza di invio del mittente con quella di lettura del destinatario.
Il destinatario comunica la dimensione del proprio *buffer* di ricezione all'apertura del canale di comunicazione. Il mittente però non vede la condizione attuale di quel *buffer*, deve aspettare la notifica dal destinatario che avviene tramite ACK.
Nello *stream* si cerca di mantenere la quantità di dati inviati per cui non si è ancora ricevuto ACK minore o uguale alla dimensione del *buffer*.
Il servizio viene implementato facendo mantenere al mittente il valore della *Receive window* concordata nel *three-way handshake* (es. riceve ACK per 100 *byte*, solo allora può inviarne altri 100).

Receive window = 10.000 *byte*
Totale da inviare = 20.000 *byte*
Frammenti max 1000 *byte* ciascuno (frame 1[0-999], frame 2[1000-1999] ...)
Un'applicazione ha effettuato una *syscall send* con un *buffer* di dimensione 20.000.
Quando al mittente torna ACK per il primo segmento (0-999) si sposta la finestra di ricezione e si inviano altri 1000 *byte* (si aspetta un RTT). Il destinatario, ricevuto il primo frammento, aspetta 500 millisecondi prima di inviare ACK (serve per ottimizzare la comunicazione). Il mittente può inserire ACK insieme ai dati (*piggy-backing*).
Se arrivano altri dati prima di quel *time out* si implementa ottimizzazione *cumulative ACK*: se il destinatario ha già ricevuto 2 frammenti invia ACK 2000, invece che ACK 1000, e la finestra di ricezione può essere spostata di due unità. Se si riceve ad alta velocità si manda ACK cumulativo per l'ultimo frammento ricevuto (solo se si sono ricevuti tutti i frammenti fino all'ultimo).
Se arrivano frame 1[0-999)] e 3[2000-2999], il destinatario invia ACK normale per questi due, non cumulativo perché manca il secondo frame nel mezzo.
Nel *buffer* di ricezione i *frame* fuori sequenza ma sempre all'interno della *receive window* vengono memorizzati, se vanno oltre vengono scartati.
Può succedere che il destinatario riceva un segmento con numero di sequenza superiore al successivo numero atteso e in ordine; questo rileva un buco nel flusso di dati. Il destinatario non può inviare un ACK negativo al mittente, ma si limita ad inviare nuovamente un *acknowledgement* relativo all'ultimo *byte* di dati che ha ricevuto in ordine: se arrivano frame 1 e 3 si può ripetere ACK 1000 per segnalare che manca il frame intermedio; se in seguito arrivano i frame 4 e 5, il destinatario continua a ripetere solo ACK duplicato per il primo segmento ricevuto. Il mittente considera questo come indice che un *frame* è andato perduto ed esegue ritrasmissione veloce (*fast retransmit*), rispedendo il frammento perduto senza che scatti il *time out.*
Per questa soluzione sono necessari ACK ripetuti per tre volte dopo la prima corretta.
Riempito il buco il destinatario può dare ACK cumulativo fino a dove aveva ricevuto.

Controllo di congestione (congestion control)
La ritrasmissione di pacchetti tratta un sintomo della congestione di rete, cioè la perdita di uno specifico segmento a livello di trasporto, ma non le cause stesse della congestione, cioè i tentativi da parte di troppe sorgenti di inviare dati a ritmi troppo elevati. In questo caso si necessita di meccanismi per adeguare l'attività dei mittenti in relazione al traffico. Vengono messi in pratica due principali orientamenti al controllo di congestione:
- Controllo di Congestione *end-to-end* → poiché il livello IP non offre supporto esplicito al livello di trasporto in merito alla congestione, questa dev'essere dedotta dai sistemi periferici sulla base dell'osservazione del comportamento della rete. TCP deve necessariamente utilizzare questo approccio. La perdita di segmenti TCP (indicata da

time out o ACK triplicati) viene considerata congestione; TCP allora diminuisce l'ampiezza l'ampiezza della propria finestra. Non c'è analisi dei nodi intermedi perché si lavora a livello di trasporto e quelli sono a livello di rete, cioè inferiore. UDP non ha alcun tipo di controllo.
- Controllo di congestione assistito dalla rete → agisce sui nodi intermedi per evitare che vengano persi frammenti (su di essi possono esserci altri flussi di dati che occupano i *buffer*). I *router* forniscono un *feedback* esplicito sullo stato di congestione della rete. L'informazione di concatenazione viene fornita dalla rete al mittente in due modi: un avviso diretto da *router* a mittente tramite un *chockepacket*, oppure un *router* che imposta un campo specifico in un pacchetto che fluisce dal mittente al destinatario.

TCP deve utilizzare il controllo di congestione *end-to-end*, anziché quello assistito dalla rete, dato che il livello IP non offre ai sistemi periferici *feedback* esplicito sulla congestione. L'approccio scelto da TCP consiste nell'imporre a ciascun mittente un limite alla velocità di invio sulla propria connessione in funzione della congestione di rete percepita. Il meccanismo di controllo di congestione fa tenere traccia agli estremi della connessione della variabile *congestion window*: impone un vincolo alla velocità di immissione di traffico sulla rete da parte del mittente; la quantità di dati inviata dal mittente per cui non si è ancora ricevuto l'*acknowledgement* non può eccedere il minimo tra i valori di *congestion window* e *receive window*
MIN(CongW, RecvW)
in presenza di una congestione eccessiva, uno o più *buffer* dei *router* lungo il percorso vanno in *overflow*, causando l'eliminazione di un datagramma. Il datagramma eliminato costituisce, a sua volta, un evento di perdita presso il mittente (*time out* o ACK duplicati), il quale lo considera come un'indicazione di congestione sul percorso tra sé e il destinatario.
Considerando invece una rete priva di congestione, TCP si serve degli *acknowledgement* ricevuti per controllare la propria finestra di congestione: se gli ACK arrivano con una frequenza relativamente bassa, allora la finestra verrà ampliata piuttosto lentamente; se arrivano con una frequenza alta, allora la finestra verrà ampliata più rapidamente. Per questo si dice che TCP è auto-temporizzato (*self-clocking*).

Principi guida del controllo di congestione in TCP

- Un segmento perso implica congestione, quindi i tassi di trasmissione del mittente TCP dovrebbero essere decrementati quando un segmento viene perso.
- Un ACK indica che la rete sta consegnando i segmenti del mittente al destinatario, e quindi il tasso di trasmissione del mittente può essere aumentato quando arriva un ACK non duplicato.
- Rilevamento della larghezza di banda. Avendo ACK che indicano che il percorso è privo di congestione, la strategia TCP per regolare la velocità di trasmissione è incrementarla in risposta all'arrivo di ACK finché non si verifica un evento di perdita. Il mittente TCP aumenta quindi la sua velocità di trasmissione per rilavare a che tasso trasmissivo comincia a verificarsi congestione, rallenta e quindi inizia nuovamente la fase di rilevamento per trovare se il punto di inizio della congestione sia cambiato.

L'algoritmo di controllo di congestione TCP (RFC 5861) si compone di tre fasi principali. *Slow start* e *congestion avoidance* sono componenti obbligatorie di TCP, e differiscono nel modo in cui aumentano la grandezza della CongW in risposta agli ACK ricevuti (la prima più velocemente della seconda). *Fast recovery* è suggerita, ma non obbligatoria per i mittenti TCP.

– ***Slow start*** → quando si stabilisce una connessione TCP, il valore di CongW viene inizializzato ad 1 MSS (*Maximum Segment Size*), il che comporta una velocità di invio iniziale di circa MSS/RTT. Durante questa fase iniziale il valore di CongW si incrementa di 1 MSS dal valore iniziale ogni volta che un segmento trasmesso riceve un ACK. Se il segmento riceve un ACK prima che si verifichi un vento di perdita, il mittente incrementa la CongW di 1 MSS ed invia 2 segmenti di dimensione massima; questi ricevono a loro volta ACK e il mittente incrementa la CongW di 1 MSS per ciascuno di essi, portandola a 4 MSS e così via. Se c'è un vento di perdita indicato dallo scadere del *time out*, il mittente pone il valore di CongW = 1, ed inizia nuovamente la fase di *slow start*. Inoltre pone il valore della variabile ssthresh (*slow start threshold*) = CongW/2 (metà del valore che aveva CongW quando la congestione è stata rilevata). Il secondo modo in cui la fase di *slow start* può terminare è legato direttamente al valore di ssthresh: quando il valore di CongW è uguale a quello di ssthresh, la fase di *slow start* termina e TCP entra nella fase di *congestion avoidance*. Ultimo modo in cui *slow start* può terminare è quando vengono rilevati 3 ACK duplicati, nel qual caso TCP effettua una ritrasmissione veloce ed entra nello stato di *fast recovery*.
– ***Congestion avoidance*** → quando TCP entra in questa fase, il valore di CongW è circa la metà di quello che aveva l'ultima volta in cui era stata rilevata la congestione. In questo caso TCP incrementa il valore di CongW di 1 MSS ogni RTT. Ciò si può ottenere in diversi modi: un approccio comune è l'incremento, da parte del mittente, della propria CongW di MSS · (MSS/CongW) *byte* ogni volta che riceve un nuovo ACK. Quando si verifica un *time out*, l'algoritmo di *congestion avoidance* si comporta nello stesso modo di *slow start*: il valore è CongW posto ad 1 MSS e il valore di ssthresh viene impostato alla metà del valore di CongW al momento del *time out*. In caso di ACK duplicati, TCP dimezza il valore di CongW e imposta il valore di ssthresh pari a metà del valore di CongW al momento del ricevimento dei 3 ACK duplicati. Infine TCP entra nella fase di *fast recovery*.
– ***Fast recovery*** → durante questa fase il valore di CongW è incrementato di 1 MSS per ogni ACK duplicato ricevuto relativamente al segmento perso che ha causato l'entrata di TCP in questa modalità. Quando arriva un ACK per il segmento perso, TCP entra nello stato di *congestion avoidance* dopo aver ridotto il valore di CongW. Se si verifica un *time out* vi è invece la transizione da *fast recovery* a *slow start*: il valore di CongW è posto a 1 MSS e quello di ssthresh a metà del valore di CongW nel momento in cui si è riscontrato l'evento di perdita. *Fast recovery* è una componente raccomandata, ma non obbligatoria di TCP. Mentre la prima versione di TCP (*Tahoe*) portava in modo incondizionato la CongW a 1 MSS ed entrava in *slow start* dopo qualsiasi evento di perdita, la versione più recente (*Reno*) adotta invece *fast recovery*.

4 – LIVELLO DI RETE

Il ruolo principale del livello di rete è quello di trasferire pacchetti da un *host* ad un altro. Per svolgerlo si identificano due importanti funzioni:

- Inoltro (*forwarding*): quando un *router* riceve un pacchetto, lo deve trasferire sull'appropriato collegamento di uscita. Nel caso più generale un pacchetto può anche essere bloccato, o duplicato e inviato su più collegamenti di uscita. Si riferisce all'azione locale con cui il *router* trasferisce i pacchetti da un'interfaccia di ingresso a quella di uscita. Solitamente implementato in hardware.
- Instradamento (*routing*): il livello di rete deve determinare il percorso che i pacchetti devono seguire tramite algoritmi di instradamento. Indica il processo globale di rete che determina i percorsi dei pacchetti dalla sorgente alla destinazione. Poiché avviene su scale temporale più grandi è solitamente implementato in software.

Per inoltrare i pacchetti, i *router* estraggono da uno o più campi dell'intestazione dei valori che utilizzano come indice nella tabella di inoltro (*forwarding table*).

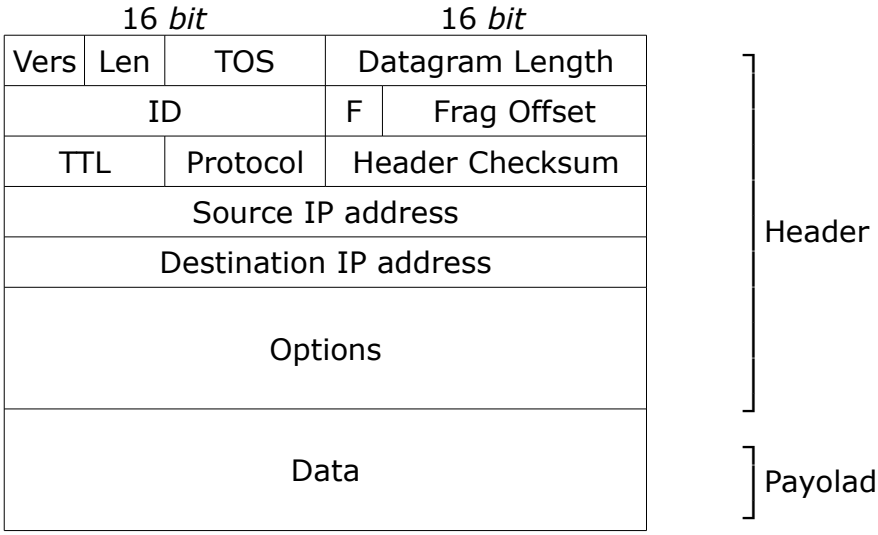
Nella generica architettura di un *router* si possono identificare quattro componenti:

- Porte di ingresso: svolgono le funzioni a livello fisico di terminazione di un collegamento in ingresso al *router*; svolgono funzioni di collegamento per operare con le analoghe funzioni all'altro capo del collegamento di ingresso; svolgono funzione di ricerca in modo che il pacchetto inoltrato nella struttura di commutazione esca sulla porta di uscita corretta.
- Struttura di commutazione: connette fisicamente le porte di ingresso a quelle di uscita.
- Porte di uscita: memorizzano i pacchetti provenienti dalla struttura di commutazione e li trasmettono sul collegamento in uscita.
- Processore di instradamento: esegue i protocolli di instradamento, gestisce le tabelle di inoltro e le funzioni sui collegamenti attivi, ed elabora la tabella di inoltro per il *router*.

4.1 – PROTOCOLLO INTERNET – IPv4

IPv4 RFC 791 – IPv6 RFC 2460 – 4291

Struttura datagrammi IPv4



Numero di versione → 4 *bit*. Consentono al *router* la corretta interpretazione del datagramma, in quanto versioni diverse corrispondono a formati diversi.
Header length → 4 *bit*. Poiché un datagramma IP può contenere un numero variabile di opzioni, questo campo indica dove effettivamente iniziano i dati in esso contenuti. Senza opzioni la lunghezza dell'intestazione è di 20 *byte*.
Tipe of service → 8 *bit*. Servono per distinguere diversi tipi di datagrammi.
Datagram lenght → lunghezza totale del datagramma IP, misurata in *byte*. Dimensione massima 65.535 *byte*, anche se raramente si superano i 1500 per non eccedere la lunghezza massima del campo dati dei *frame* Ethernet.
Identificatore → serve per differenziare i pacchetti, in particolare per quelli frammentati. Si usa in combinazione a *flag* e *fragmentation offset*.
Flag → 3 *bit*. Per rilevare la presenza di pacchetti frammentati.
Fragmentation offset → 13 *bit*. Serve per riposizionare i pacchetti frammentati. Specifica l'ordine che i frammenti avevano nell'originario datagramma IP.

Time-To-Live → 8 *bit*. Incluso per assicurare che i datagrammi non circolino per sempre nella rete. È un contatore, inizializzato con un certo valore deciso dall'host mittente, che viene decrementato di un'unità ogni volta che il datagramma viene elaborato da un *router* (*hop*). Individua quindi il numero massimo di *hop* che un datagramma può fare prima di arrivare a destinazione. Tipicamente il mittente non ha idea di quanti *hop* dovrà fare il datagramma, e questo rischia di essere scartato: quando il contatore arriva a 0 il *router* scarta il datagramma e invia un messaggio di errore all'indirizzo sorgente (protocollo ICMP). Allo stesso modo se il TTL ha un valore troppo alto può consumare risorse di rete. Il TTL viene utilizzato per l'applicazione *Traceroute*: invia un pacchetto al destinatario di cui si vuole ricavare il percorso di traceroute con il campo TTL impostato ad 1. Il primo router che lo riceverà, constatando che il campo TTL ha raggiunto lo 0, invierà un errore al mittente (ICMP Time Exceeded). L'applicazione memorizzerà l'indirizzo IP del primo *router*, quindi invierà un nuovo pacchetto con TTL impostato a 2. L'operazione verrà ripetuta finché il pacchetto non sarà arrivato al destinatario.
Protocollo → 8 *bit*. Indica il *next level protocol*, cioè lo specifico protocollo a livello di tarsporto al quale vanno passati i dati del datagramma. Il valore 6 indica TCP, il valore 17 UDP.
Header Checksum → consente ai router di rilevare gli errori sui *bit* nei datagrammi ricevuti.
Indirizzi IP sorgente e destinazione → alla creazione del datagramma l'*host* inserisce il proprio IP nel campo *source* e quello della destinazione nel campo *destination*.
Options → questi campi consentono di estendere l'intestazione IP. Dato che possono avere lunghezza variabile, non è possibile determinare a priori dove comincerà il campo dati. Inoltre, dato che i datagrammi possono richiedere o non richiedere l'elaborazione delle opzioni, il tempo necessario per questa operazione su un *router* può variare in modo significativo. Per questi ed altri motivi le opzioni IP sono state eliminate dall'intestazione IPv6.
Dati → nella maggior parte dei casi questo campo contiene il segmento a livello di trasporto (TCP o UDP). Può trasportare anche altri tipi di dati, quali i messaggi ICMP. Idea di incapsulamento: gli *header* si susseguono, ad uno segue l'altro come *payolad*.

I datagrammi IPv4 hanno 20 byte di *header*, escludendo le opzioni. I datagrammi non frammentati che trasportano segmenti TCP ne hanno 40 *byte* complessivi (20 IP + 20 TCP, a cui si aggiunge il messaggio di livello applicativo).

Frammentazione dei datagrammi IPv4
Non tutti i protocolli a livello di collegamento possono trasportare pacchetti della stessa dimensione a livello di rete. La massima quantità di dati che un *frame* a livello di collegamento può trasportare è detta unità massima di trasmissione (MTU *Maximum Transmission Unit*). Dato che, per il trasporto da un *router* all'altro, i datagrammi IP sono incapsulati in *frame* a livello di collegamento, la MTU di questo protocollo pone un limite rigido alla lunghezza dei datagrammi IP. Il problema risiede nel fatto che le tratte del percorso tra mittente e destinatario possono utilizzare differenti protocolli a livello di collegamento e possono avere differenti MTU.
I frammenti in cui vengono suddivisi i datagrammi IP devono essere riassemblati prima di raggiungere il livello di trasporto alla destinazione; per evitare di sovraccaricare i *router* interni questo compito è affidato ai sistemi periferici.
I campi ID, offset di frammentazione e flag nell'header IP servono all'host di destinazione per individuare i frammenti di un datagramma proveniente dalla stessa origine e stabilire come debbano essere assemblati.

Indirizzamento IPv4
Generalmente un *host* ha un solo collegamento con la rete, sul quale l'implementazione di IP dell'*host* invia il datagramma. Il confine tra *host* e collegamento fisico viene detto interfaccia. Un *router* invece ha molti collegamenti, e quindi molte interfacce. Dato che *host* e *router* possono inviare e ricevere datagrammi, IP richiede che tutte le interfacce abbiano un proprio indirizzo IP. Quindi tecnicamente l'indirizzo è associato all'interfaccia, e non all'*host* o *router* che la contiene. Ogni interfaccia ha un IP globalmente univoco, tranne quelle gestite da NAT, che connette una rete privata a quella pubblica: tutti gli *host* privati sono visti all'esterno con lo stesso indirizzo pubblico, che solitamente è quello del *router*.
Gli indirizzi IPv4 sono lunghi 32 *bit* (ne esistono quindi 2^{32}) scritti in notazione decimale puntata; sono composti da quattro otteti che codificano l'equivalente decimale di ogni *byte* (ognuno da 0 a 255).
1.2.3.4/30 → indirizzo/(*sub)net mask* (*bit* della parte *network*). *Network* e *host* sono separati perché la parte di *routing* è fatta solo sull'indirizzo di rete: idea di Internet come una collezione di LAN di varie dimensioni, all'interno delle quali l'instradamento è realizzato dal livello *datalink*, mentre il *routing* serve per passare da una rete all'altra.
La strategia di assegnazione degli indirizzi Internet è detta *classless interdomain routing* (CIDR), che generalizza la nozione di indirizzamento di sottorete. L'IP viene diviso in due parti: a.b.c.d/x dove gli x *bit* più a sinistra costituiscono la porzione di rete dell'indirizzo, e sono detti prefisso di rete. Generalmente, ad un organizzazione viene assegnato un blocco di indirizzi contigui con un prefisso comune per tutti gli IP dei dispositivi che si trovano al suo interno. I rimanenti 32-x *bit* di un indirizzo possono essere quindi utilizzati per distinguere i dispositivi interni dell'organizzazione, e i *router* della rete interna se ne serviranno per indirizzare al dispositivo destinatario.
Prima dell'adozione di CIDR si utilizzava uno schema di indirizzamento noto come *classful addressing*, che suddivideva gli indirizzi in:
Reti classe A → 8 *bit* rete + 24 *bit host*
Reti classe B → 16 *bit* rete + 16 *bit host*
Reti classe C → 24 *bit* rete +8 *bit host*

Net mask: dato l'indirizzo serve per sapere quanti sono i *bit* di rete e quanti quelli per l'*host*. È una sequenza di 32 *bit* in cui quelli dell'*host* sono a 0, quelli dell'indirizzo di rete sono a 1. Non c'è più la divisione in classi, invece si prende una decisione arbitraria su come dividere l'indirizzo tramite la *netmask,* che viene fornita insieme a quest'ultimo, e inserita all'interno del driver della scheda di interconnessione di rete. Il *router* fa l'AND tra indirizzo di rete e *netmask*.

Alcuni indirizzi IP vengono trattati in modo particolare:
127.0.0.1 – *local host* → è associato a tutti gli *host* della rete. Non c'è instradamento, ma è utilizzato solo dai processi che girano su una stessa macchina. Il SO riconosce l'indirizzo e non manda i messaggi al *router* della rete, ma li reindirizza all'interno della stessa macchina. Sempre associato alla macchina locale.
Indirizzi nell'intervallo 10.0.0.0 – 10.255.255.255 (indirizzi privati) → mai usati per instradamento all'interno della rete IP, possono essere usati solo all'interno di una LAN. Privati perché ogni LAN può decidere di usarli in modo arbitrario, e non c'è problema di duplicazione perché non vengono instradati. Ci sono altri intervalli di indirizzi privati, come 192.168.0.0 – 192.168.255.255 (rete classe B).

Broadcast 255.255.255.255: caso particolare di *multicast.* Quando un *host* emette un datagramm con questo indirizzo destinazione, il messaggio viene consegnato a tutti gli *host* sulla stessa sottorete. Nella rete Internet è implementata l'idea di *broadcast* a livello locale, ma non di *routing*. A livello di trasporto supporta *datagram*.

L'indirizzo *broadcast* contiene tutti 1 nella parte *host* (indirizzo di una macchina messo in OR con la negazione della netmask). La funzionalità viene solitamente implementata solo a livello di rete locale. Si accede a tutte le macchine contemporaneamente. Implementata a livello datalink 1.2.3.4/30 diventa 1.2.3.7/30
Es. di *unicast*: *socket* di tipo *stream*.
Multicast: stesso messaggio replicato a più di un destinatario contemporaneamente.

4.2 – PROTOCOLLO DHCP – Dynamic Host Configuration Protocol

Basato su livello di trasporto UDP – *server Port* 67 – *client Port* 68
Consente ad un *host* di ottenere un indirizzo IP in modo automatico, così come apprendere informazioni aggiuntive (maschera di sottorete, *gateway*, *local* DNS *server*). Viene detto protocollo *plug-and-play* o *zero-configuration* per la sua capacità di automatizzare la connessione degli *host* alla rete. Importante per gli amministratori di rete, che altrimenti dovrebbero svolgere questi compiti manualmente.
Protocollo di tipo domanda/risposta → *client* chiede al *server* informazioni di configurazione per poter comunicare in rete (possibile perché a livello locale l'instradamento è realizzato a livello *datalink*) così si può predisporre il livello *network* uscendo dalla rete locale.
Il protocollo si articola in quattro passaggi, nei quali vengono scambiati 4 messaggi:

- ***D****iscovery*: l'identificazione da parte del *client* del *server* DHCP col quale interagire avviene tramite un messaggio *DHCP discover*. Il server DHCP cambia a seconda della rete locale: si usa una comunicazione di tipo *broadcast*, con indirizzo sorgente 0.0.0.0 (per convenzione, l'*host* non lo ha ancora) e indirizzata a 255.255.255.255, cioè a tutte le macchine connesse alla LAN; se tra esse c'è il *server* DHCP in ascolto sulla porta 67 risponde, mentre gli altri ignorano il messaggio. Il messaggio è un datagramma IP, quindi basato su UDP; non sarebbe possibile servirsi di TCP, che prevede il *three-way handshake*, scelta obbligata in quanto non si conosce né l'IP della rete alla quale è collegato, né quello del DHCP *server*.
  All'interno del primo messaggio c'è un *frame* che contiene gli indirizzi MAC.
  Dest → indirizzo broadcast FF.FF.FF.FF.FF.FF
  Source → quello della scheda di rete dell'host.
- ***O****ffer*: i *server* che riceveno il messaggio di identificazione rispondono al *client* con un messaggio *DHCP offer* utilizzando il proprio indirizzo MAC.
  IP *source*: quello del server → MAC source: quello del server
  IP *dest*: *broadcast* (liv 3) → MAC dest: quello del client (liv 2)
  L'offerta arriva solo alla macchina che aveva fatto richiesta, non a tutte quelle della sottorete. Nel *payolad* c'è l'indirizzo IP che il *server* ha deciso di assegnare al *client*, la *net mask*, uno o più *server* DNS locali, il *gateway*.
  All'interno di una LAN possono esserci più *server* DHCP locali. Il protocollo prevede che entro un certo *time out* (chiamato *lease time*) il client scelga una delle offer che ha ricevuto (per evitare di consumare risorse inutilmente).
- ***R****equest*: scelta una tra le *offer* dei *server*, il *client* risponde mandando un messaggio *DHCP request* che viene nuovamente indirizzata in broadcast a tutti
  IP *source*: 0.0.0.0 → MAC *source*: indirizzo client DHCP
  IP *dest*: 255.255.255.255 → MAC *dest*: FF.FF.FF.FF.FF.FF
  Nel *payload* il *client* chiede al *server* di assegnargli l'indirizzo IP; il *server* che si riconosce nel *payolad* assegna definitivamente l'indirizzo, mentre gli altri registrano che l'offerta è stata rifiutata e possono offrire l'indirizzo IP ad altri.
- ***A****cknowledge*: il *server* risponde alla *request* mandando un messaggio DHCP ACK in unicast, rivolto solo all'*host client*, il quale può utilizzare l'IP per la durata della concessione. Il *client* memorizza anche l'indirizzo del *gateway* e quello del *local* DNS *server*, così può accedere anche alla rete esterna.

Poiché quando un nodo si connette ad una nuova sottorete, DHCP gli rilascia un nuovo IP, non si può mantenere una connessione TCP ad un'applicazione remota (il nodo mobile si sposta da una sottorete all'altra).

4.3 – NAT – Network Address Translation

RFC 2663 – RFC 3022

Molte delle centinaia di migliaia di reti private esistenti utilizzando un identico spazio di indirizzamento per scambiare pacchetti fra i loro dispositivi. Ovviamente, quelli inviati su Internet globale non possono utilizzare questi indirizzi come sorgente e destinazione, in quanto gli indirizzi privati hanno significato soltanto all'interno di una data rete. I *router* NAT si occupano di effettuare questo passaggio.

LAN con indirizzi 192.168.-.-
[*host*] 192.168.0.1
[*host*] 192.168.0.2 → 192.168.0.0 → *router* NAT → 1.2.3.4 → [*host*] 2.3.4.5

Nell'*header* IP l'indirizzo sorgente è 192.168.0.2, quello destinazione è 2.3.4.5 (a questo segue l'*header* UDP con i numeri delle porte). Il messaggio viene instradato a livello *datalink,* che modifica sia l'IP mittente che la porta mittente, producendo un *frame* diverso. Indirizzo sorgente 192.168.0.2 → 1.2.3.4
Indirizzo destinatario invariato, come la porta di destinazione (es. 1020). La porta sorgente invece cambia (es.10000 da 35210). Il messaggio viene instradato dal *router* Internet, e il destinatario non consce l'indirizzo della macchina perché si sfrutta l'indirizzo pubblico del *router*. Se il destinatario vuole rispondere avrà come indirizzo destinazione 1.2.3.4 e porta destinazione definita dal *router* NAT (10000), il quale ricorda che quella porta era stata usata per fare uscire i messaggi dalla macchina tramite una tabella di traduzione (che contiene l'associazione da 10000 a 35210).
I *router* NAT hanno due interfacce di rete diverse, una che si rivolge all'interno e una che si rivolge all'esterno della rete locale.
- Porte da 0 a 1023 sono predefinite e riservate.
- Porte effimere (non ci sono costantemente) vengono utilizzate quando si lancia un'applicazione, e questa chiede di usare dei *socket* con modalità *client* per connettersi verso altre macchine (numeri alti). Tipicamente un *router* NAT utilizza porte effimere, che corrispondono ad indirizzi che non sono stati standardizzati. Limitazione a questo meccanismo è il numero limitato di porte disponibili.
Port forwarding → traduzione degli indirizzi di destinazione, invece che di quelli sorgente, permette di far entrare le richieste dall'esterno e indirizzarle in una macchina all'interno.
Col protocollo TCP il NAT funziona solo se l'*host* interno è in modalità *client* (attiva). Se si vuole mettere un *server* all'interno della LAN (modalità passiva) si deve specificare che il messaggio deve essere ributtato all'interno della LAN e non è destinato ad un'applicazione.

4.4 – PROTOCOLLO INTERNET – IPv6

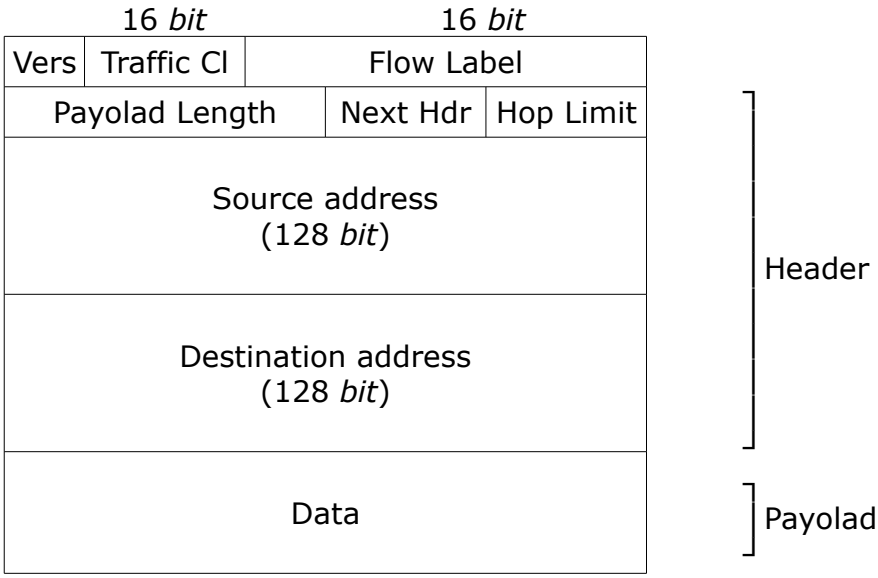
IPv4 RFC 791 – IPv6 RFC 2460 – 4291

Iniziato a sviluppare all'inizio degli anni '90 in quanto lo spazio di indirizzamento IP a 32 *bit* stava cominciando ad esaurirsi.

Struttura datagrammi IPv6

Indirizzamento esteso → IPv6 aumenta la dimensione dell'indirizzo a 128 bit in modo tale che diventino praticamente inesauribili. Oltre agli indirizzi *unicast* e *multicast*, IPv6 supporta anche indirizzi *anycast*, che consentono di consegnare un datagramma a un qualsiasi *host* all'interno di un gruppo. Questa caratteristica potrebbe essere

usata per inviare un GET HTTP al più vicino di una serie di siti contenenti un dato documento.
Intestazione ottimizzata di 40 *byte* → una serie di campi IPv4 è stata eliminata o resa opzionale. La risultante intestazione ha lunghezza fissa e consenta una più rapida elaborazione dei datagrammi IP.
Etichettatura dei flussi → la definizione elusiva di flusso di IPv6 consente l'etichettatura di pacchetti che appartengono a flussi particolari per i quali il mittente richiede una gestione speciale, come una qualità di servizio diversa da quella di *default* o un servizio in tempo reale.



Numero di versione → 4 *bit*. Per IPv6 il numero è ovviamente 6.
Traffic class → 8 *bit*. Simile a TOS in IPv4. Può essere utilizzato per attribuire priorità a determinati datagrammi all'interno di un flusso o provenienti da specifiche applicazioni.
Flow label → 20 *bit*. Utilizzato per identificare un flusso di datagrammi.
Payload length → 16 *bit*. Indica il numero di *byte* nel datagramma che seguono l'intestazione a lunghezza fissa.
Next header → 8 *bit*. Utilizza gli stessi valori del campo *next level protocol* in IPv4. Identifica il protocollo a cui verranno consegnati i dati del datagramma, ma non solo, in quanto permette anche di concatenare più *header* IP successivi, e supplisce alla mancanza delle opzioni.
Hop limit → 8 *bit*. Corrisponde al TTL di IPv4. Il contenuto è un numero che viene decrementato di un'unità da ciascun *router* che inoltra il datagramma. Quando si azzera il datagramma viene eliminato.
Indirizzi sorgente e destinazione → formato a 128 *bit* descritto nell'RFC 4291.
Dati → *payload* che viene passato al protocollo specificato nel campo di intestazione successiva quando il datagramma IPv6 raggiunge la sua destinazione.

Confronto con il formato precedente:
Frammentazione/riassemblaggio → IPv6 non consente nessuna di queste due operazioni sui *router* intermedi, che possono essere effettuate soltanto da sorgente o destinazione (al di fuori di questo protocollo). Se un *router* riceve un datagramma IPv6 troppo grande per essere inoltrato sul collegamento di uscita, lo elimina ed invia al mittente un messaggio di errore ICMP; il mittente potrà rinviare i dati con una dimensione di datagramma IP inferiore. Frammentazione e riassemblaggio sono

operazioni che consumano tempo; trasferire l'onere di questa funzionalità dai *router* ai sistemi periferici rende molto più rapido l'instradamento IP all'interno della rete.
Checksum dell'intestazione → Dal momento che i protocolli a livello di trasporto (TCP e UDP) e di collegamento (Ethernet) calcolano un loro *checksum*, i progettisti di IP lo hanno probabilmente ritenuto ridondante. Inoltre avendo l'*hop limit* associato al *cheksum*, questo dovrebbe essere ricalcolato ad ogni *router* e il carico di lavoro sarebbe eccessivo.
Opzioni → questo campo non fa più parte dell'intestazione, ma è una delle possibili intestazioni successive cui punta l'*header* IPv6 (es. usare la crittografia).

Attualmente si è nella fase di transizione da v4 a v6. Il problema è che mentre i nuovi sistemi IPv6 sono retrocompatibili, i sistemi IPv4 non sono in grado di gestire datagrammi IPv6.
Con IPv6 si è aggiunto un livello di rete esteso nello *stack* dei livelli di un host (non sui *router* intermedi) implementato tramite il campo *next header*. Si velocizza il traffico sui nodi intermedi in quanto questi considerano soltanto l'*header* principale IPv6; le estensioni sono elaborate soltanto dall'*host* destinatario.
Tunneling → approccio alla transizione da v4 a v6 più diffusamente adottato. Si ignora la struttura a livelli e si mischiano le intestazioni indipendentemente da essi. Se il mittente è un nodo Ipv4 e il destinatario IPv4, l'ultimo *hop* deve essere fatto da una macchina che conosca entrambe le versioni (doppio *header*); l'ultimo *router* elimina l'*header* IPv6 e consegna il resto all'*host*.
Se due nodi IPv6 sono connessi da un insieme di *router* intermedi IPv4 (tunnel) il nodo mittente inserisce l'intero datagramma IPv6 nel payload di un datagramma IPv4, e verrà estratto e instradato dal ricevente.

4.5 – ROUTING IP – ALGORITMI DI INSTRADAMENTO
Il loro scopo è determinare i percorsi tra le sorgenti e i destinatari attraverso la rete di *router*. Per formulare i problemi di instradamento si utilizza un grafo, nel quale i nodi rappresentano i router, e gli archi i collegamenti fisici tra essi; ad ogni arco è associato un peso che ne indica il costo. Gli algoritmi decidono la direzione che prenderanno i datagrammi sulla base dell'indirizzo di destinazione: quando il datagramma arriva prende subito la strada determinata in precedenza, se l'algoritmo lavorasse dopo l'arrivo del datagramma il suo tempo di esecuzione sarebbe troppo alto.
Ogni *router* contiene un processore che controlla strutture dati di tipo coda in entrata e in uscita e decide dove inviare cosa smistando le code in ingresso in quelle in uscita. Inoltre usa una *forwarding table*, calcolata in precedenza, che tiene traccia delle corrispondenze indirizzo di destinazione/canale di uscita, ma lavorando soltanto sui prefissi, poiché se lavorassero sugli interi indirizzi la loro dimensione sarebbe troppo grande(es. su IPv4 i primi 5 *bit*). Gli algoritmi di *routing* si occupano di costruire ed aggiornare continuamente queste tabelle.

Due tecniche principali basate sullo studio dei grafi:
– Algoritmo di instradamento centralizzato → calcola il percorso a costo minimo tra una sorgente e una destinazione avendo una conoscenza globale e completa della rete. l'algoritmo riceve in ingresso tutti i collegamenti tra i nodi e i loro costi. Questi algoritmi sono detti ***link-state*** poiché devono essere consci di ciascun collegamento della rete; questo è possibile in quanto periodicamente tutti i *router* inviano pacchetti contenenti le loro informazioni (identità + costi collegamenti connessi al nodo di invio) in *broadcast* e le tabelle vengono aggiornate.
Per il calcolo del percorso a costo minimo si utilizza l'algoritmo di Dijkstra: è iterativo, e dopo la k-esima iterazione, i percorsi a costo minimo sono noti a k nodi di destinazione e, tra i percorsi a costo minimo verso tutti i nodi di destinazione, questi k percorsi hanno k costi più bassi. Quando termina, si ha per ciascun nodo il suo

predecessore lungo il percorso a costo minimo dal nodo di origine. La tabella di inoltro in un nodo, può pertanto essere costruita memorizzando, per ciascuna destinazione, il nodo del successivo *hop* sul percorso a costo minimo fino alla destinazione.
Più semplice da implementare se la rete non è troppo estesa.
– Algoritmo di instradamento decentralizzato → il percorso a costo minimo viene calcolato in modo iterativo e distribuito. Nessun nodo possiede informazioni sullo stato globale della rete, ma calcola il percorso tramite un processo iterativo e lo scambio di informazioni soltanto con i nodi ad esso adiacenti. Nell'instradamento ***distance-vector***, ogni nodo effettua una parte del calcolo, elaborando un vettore di stima dei costi (distanze) verso tutti gli altri nodi della rete. L'algoritmo funziona in modo dichiarativo diffondendo ai vicini immediati le proprie connessioni e i costi relativi ai canali. Dopo il primo scambio ogni nodo ha anche la mappa dei propri vicini immediati (cammini minimi fino ad un massimo di 2 *hop*) e si prosegue diffondendo i cammini a distanza 3, per costruire quelli a distanza 4 e così via. Il vantaggio risiede nella quantità di comunicazione rispetto al *broadcast*, e anche nella quantità di memoria impiegata.

Un altro criterio per classificare gli algoritmi di *routing* è suddividerli in algoritmi statici, nei quali i percorsi cambiano raramente e di solito come risultato di un intervento umano (modifica manuale di una tabella di inoltro), e algoritmi dinamici, che determinano gli instradamenti al variare del volume di traffico o della topologia della rete.
Un terzo criterio classifica gli algoritmi in base alla loro sensibilità (*load-sensitive* o *load-insensitive*) al carico della rete. In un algoritmo sensibile al carico i costi dei collegamenti variano dinamicamente per riflettere il livello corrente di congestione. Gli attuali algoritmi di instradamento Internet sono insensibili al carico (OSPF, BGP), dato che il costo di un collegamento non riflette esplicitamente il suo attuale, o recente, livello di congestione.

Instradamento all'interno dei sistemi autonomi.
Bisogna considerare anche la scalabilità, in quanto al crescere del numero di *router*, memorizzare e comunicare le informazioni di instradamento diventa proibitivo.
E l'autonomia amministrativa, in quanto Internet è una rete di ISP che desiderano gestire liberamente i propri *router* (es. scelgono l'algoritmo di instradamento) o nascondere all'esterno aspetti dell'organizzazione interna della rete.
Questi problemi possono essere risolti organizzando i *router* in AS (*autonomous systems*), gruppi generalmente posti sotto lo stesso controllo amministrativo.
I router di un AS seguono lo stesso algoritmo detto protocollo di instradamento interno al sistema autonomo.
– OSPF (*Open Shortest Path Fist*) → v2 RFC 2328. protocollo tipo *link-state* che utilizza il *flooding*, inondazione di informazioni riguardo lo stato dei collegamenti, e l'algoritmo di Dijkstra per la determinazione del percorso a costo minimo. Ogni volta che si verifica un cambiamento nello stato di un collegamento, il *router* invia le informazioni via *broadcast* a tutti gli altri *router* del sistema autonomo; inoltre procede ad un invio periodico dello stato dei collegamenti, anche se questo non è cambiato. I messaggi OSPF vengono trasportati direttamente da IP come protocollo superiore. Il protocollo controlla inoltre che i collegamenti siano operativi tramite messaggio HELLO inviato ad un vicino connesso, e consente ai *router* OSPF di accedere ai *database* sullo stato dei collegamenti della rete contenuti nei *router* confinanti.
Dal punto di vista della sicurezza, OSPF prevede l'autenticazione (semplice, tutti i router con la stessa password, in chiaro, che deve essere inclusa nei pacchetti ricevuti. MD5, basata su chiavi segrete che il mittente aggiunge all'*hash* del contenuto; ricevente calcola la funzione e confronta la chiave preconfigurata con

quella del pacchetto) , per cui durante lo scambio di informazioni tra i *router* si prendono in considerazione solo quelle provenienti da fonti autenticate.
– BGP (Border Gateway Protocol) → determina i percorsi tra sorgente e destinazione che interessano più sistemi autonomi, e ISP diversi. È tipo *distance-vector* decentralizzato ed asincrono. Mette a disposizione di ciascun *router* un modo per ottenere informazioni sulla raggiungibilità dei prefissi di sottorete da parte dei sistemi confinanti, consentendo a ciascuna sottorete di comunicare la propria esistenza al resto di Internet; ma anche un modo per determinare i percorsi ottimali verso le sottoreti, in quanto un router che viene a conoscenza di più cammini verso uno specifico prefisso esegue localmente BGP sulla base delle informazioni di raggiungibilità e delle politiche del sistema.

4.6 – ICMP – Internet Control Message Protocol
RFC 792 – LIVELLO DI TRASPORTO – complemento al protocollo IP.
Viene utilizzato da *host* e *router* per scambiarsi informazioni a livello di rete. Il suo utilizzo principale è quello della gestione delle situazioni di errore.
Dal punto di vista dell'architettura si trova subito sopra ad IP, dato che i messaggi ICMP vengono trasportati come *payolad* nei datagrammi IP. Se un *host* riceve un datagramma IP che specifica ICMP come protocollo di livello superiore, allora effettua il *demultiplexing* dei contenuti, come farebbe per contenuti TCP o UDP.
I messaggi ICMP hanno due ID numerici: uno per il tipo di messaggio, e uno per differenziare diversi codici all'interno di uno stesso tipo; contengono inoltre l'intestazione e i primi 8 *byte* del datagramma IP che ha provocato la generazione del messaggio, in modo che il mittente possa determinare il datagramma che ha causato l'errore.
Tipo 3 – errori all'interno della rete IP. Codice:
0 *Destination network unreachable* (tutti gli indirizzi caratterizzati da quella *netmask*)
1 *Destination host unreachable*
2 *Destination protocol unreachable* (non predisposto a far girare quel protocollo)
3 *Destination port unreachable* (porta a livello di trasporto)
6 *Destination network unknown*
7 *Destination host unknown* (sottorete definita ma non contiene quel numero di *host*)
Tipo 11 codice **0** – *TTL exceeded*

Segnalazioni addizionali:
Tipo 8 codice **0** – *Echo request* (all'invio si attiva un timer e simula l'arrivo di un msg)
Tipo 0 codice **0** – *Echo reply* (per implementare *ping* risposta)
Tipo 9 codice **0** – *Router announce* (nuovo *router* aggiunto sulla rete)
Tipo 4 codice **0** – con questo protocollo di può gestire anche l'interazione fra *router*; con questo si ha una segnalazione da parte di un *router* verso i suoi vicini per la congestione (non implementato).

É stat definita una nuova versione di ICMP per IPv6 (RFC 4443) in cui sono stati introdotti nuovi tipi e codici e ridefiniti quelli vecchi.

4.7 – SNMP – Simple Network Management Protocol
SNMPv2 RFC 3416 – LIVELLO APPLICATIVO **–** tipo richiesta/risposta
La rete è costituita da molteplici e complessi componenti *hardware* e *software*.
Componenti chiave della gestione della rete:
Server di gestione → applicazione controllata dal responsabile ed eseguita da un calcolatore del NOC (*Network Operations Server*). Il suo compito è quello di sovrintendere alla raccolta, all'elaborazione e all'analisi e/o alla visualizzazione delle informazioni di gestione.

Dispositivo di rete gestito → componente *hardware* della rete (*host, router, bridge, hub, modem*, stampante) in cui si trovano molti oggetti da gestire, che sono gli effettivi componenti *hardware* di questo (es. schede di rete). Oppure un insieme di parametri di configurazione per *hardware* e *software* (protocollo OSPF).
Base di dati gestionali → MIB (*Management Information Base*) informazioni archiviate associate ad ogni oggetto da gestire. Il linguaggio di specifica dei dati SMI (*Structure of Management Information*) viene utilizzato per assicurare che sintassi e semantica dei dati di gestione della rete siano ben definite e non ambigue.
Agente di gestione → processo che comunica con il *server* di gestione ed esegue sul dispositivo le azioni decise dal *server*.
Protocollo di gestione → componente tramite il quale il *server* di gestione può richiedere lo stato dei dispositivi ed agire su di essi attraverso gli agenti. Questi ultimi utilizzano il protocollo per informare il *server* di eventuali anomalie.

SNMP trasporta informazioni tra *server* di gestione ed agenti. La sua più comune modalità di utilizzo è quella richiesta/risposta: un'entità di gestione SNMPv2 invia un richiesta ad un agente SNMPv2 che, a seguito di questa, compie azioni e invia una risposta. Altro utilizzo tipico si verifica quando un agente, senza che gli sia pervenuto alcuna richiesta, invia un messaggio *trap* al *server* di gestione in cui segnala il verificarsi di una situazione eccezionale che ha prodotto una variazione dei valori degli oggetti MIB.
SNMPv2 definisce sette tipi di messaggi, detti PDU (Protocol Data Unit):

- *GetRequest, GetNextRequest, GetBulkRequest* sono inviate da un'entità di gestione all'agente del dispositivo da gestire per richiedere uno o più valori di oggetti MIB. Queste tre PDU differiscono in quanto la prima può richiedere qualunque valore MIB, la seconda ripetuta in sequenza può richiedere elenchi o tabelle di oggetti, la terza restituisce grandi gruppi di dati.
- *SetRequest* usata da un *server* di gestione per impostare il valore di uno o più oggetti MIB in un dispositivo. Il *server* ricevente risponde con la PDU *Response* contenente "noError" in caso di effettiva modifica del parametro.
- *InformRequest* è utilizzata da un *server* di gestione per trasmettere informazioni MIB ad un'altra entità di gestione.
- *Response* viene inviata dal dispositivo al *server* di gestione per restituire l'informazione richiesta.
- Il messaggio *trap* è un evento asincrono, cioè non è la risposta ad una richiesta, ma viene emesso quando si verifica uno degli eventi di cui l'entità di gestione aveva richiesto la notifica. Ne esistono varie tra cui quelle relative all'avvio a freddo (senza procedura di *reboot*, in conseguenza ad un malfunzionamento o un calo di tensione), o a caldo (con procedura di *reboot*, gestito dall'amministratore) di un dispositivo; alla disponibilità o meno di un collegamento, alla perdita di raggiungibilità di un vicino o alla mancata autenticazione. Il *server* di gestione non è tenuto a fornire un riscontro ai messaggi *trap* ricevuti.

Nonostante le PDU possano essere convogliate da qualunque protocollo di trasporto, di solito costituiscono il *payolad* di un datagramma UDP.
Il campo identificato della richiesta (*Request ID*) è utilizzato dall'entità di gestione per numerare le sue richieste e dall'agente per la risposta, quindi lo si può utilizzare per rilevare la perdita di richieste o risposte. È compito dell'entità di gestione decidere se ritrasmettere una richiesta non riscontrata entro un certo tempo. SNMP non prevede particolari procedure di ritrasmissione, ma richiede solamente che l'entità di gestione sia responsabile di frequenza e durata delle ritrasmissioni.
SNMPv3 aggiunge caratteristiche di sicurezza e amministrazione.

5 - LIVELLO DI COLLEGAMENTO

Livello 2 dello *stack* dei protocolli. I canali di comunicazione che collegano nodi adiacenti lungo un cammino sono collegamenti (*link*), quindi i datagrammi che devono essere trasferiti da un *host* sorgente ad un *host* destinazione devono essere trasportati lungo ciascun collegamento nel percorso, da un estremo all'altro. Su ogni collegamento, un nodo trasmittente incapsula il datagramma in un frame del livello di collegamento (*link-layer frame*) e lo trasmette lungo il collegamento stesso.

Indirizzi MAC
Host e *router* hanno indirizzi a livello di collegamento. In realtà non sono propriamente i nodi, ma i loro adattatori (schede di rete) a possedere questi indirizzi. Un host o un router con più interfacce di rete avrà più indirizzi a livello di collegamento associati ad esse, come accadeva per gli indirizzi IP.
Tuttavia, gli *switch* a livello di collegamento non hanno indirizzi di collegamento associati alle loro interfacce che connettono ad *host* e *router*, questo perché il loro compito è trasportare i datagrammi tra *host* e *router* in modo trasparente.
Gli indirizzi a livello di collegamento si indicano con indirizzo fisico, indirizzo LAN, indirizzo MAC (più utilizzato).
Per molte LAN (come Ethernet e 802.11) l'indirizzo MAC è lungo 6 byte, il che consente di avere 2^{48} indirizzi possibili; sono generalmente espressi in esadecimale. Una proprietà degli indirizzi MAC è che non esistono due schede di rete con lo stesso indirizzo (sono gestiti da IEEE); l'indirizzo MAC di una scheda di rete ha una struttura non gerarchica e non cambia mai.
Quando una scheda di rete vuole spedire un *frame*, vi inserisce l'indirizzo MAC di destinazione e lo immette nella LAN. Uno *switch* può occasionalmente fare *broadcast* di un frame in ingresso su tutte le sue interfacce in uscita, per cui una scheda di rete può ricevere un *frame* non indirizzato a lei. Ogni scheda di rete controllerà se l'indirizzo MAC di destinazione corrisponde al proprio. In caso affermativo, la scheda estrae il datagramma e lo passa verso l'alto nello *stack* dei protocolli del nodo a cui appartiene. In caso contrario scarta il frame.
Gli indirizzi MAC devono soddisfare il requisito di essere diversi; questo problema è stato risolto adottando un approccio gerarchico a due livelli: 3 *byte* identificano il produttore del dispositivo + 3 *byte* a disposizione del contruttore per produrre 16 miliardi di indirizzi diversi.

5.1 – PROTOCOLLO ARP – Address Resolution Protocol
RFC 826 – LIVELLO APPLICATIVO – Basato su UDP – Tipo domanda/risposta
Risoluzione degli indirizzi da IP a MAC. Importante differenza con DNS è che quello esegue l'operazione di traduzione per *host* localizzai in qualunque punto di Internet, mentre questo risolve soltanto gli indirizzi IP per i nodi della stessa sottorete.
Nella RAM dei nodi vi è una tabella ARP che contiene la corrispondente tra indirizzi IP e MAC, oltre che un valore relativo al TTL che indica quando bisognerà eliminare una data voce dalla tabella.
Se il nodo trasmittente non ha una voce nella tabella per il nodo di destinazione, contruisce un pacchetto ARP di richiesta, il cui scopo è interrogare tutti gli altri nodi della sottorete (*peer-to-peer*) riguardo l'indirizzo MAC corrispondente all'indirizzo IP da risolvere. I pacchetti ARP di richiesta e di risposta hanno lo stesso formato.
La richiesta viene inviata in *broadcast* a livello MAC, e *unicast* a livello IP. Tutte le schede di rete della sottorete trasferiscono il pacchetto al proprio nodo, che controlla se il proprio indirizzo IP corrisponde a quello di destinazione indicato nel pacchetto ARP. Per questo, a livello IP, l'unica macchina che prende in considerazione la richiesta è quella con l'indirizzo corretto. Il nodo richiedente può quindi aggiornare la sua

tabella ed inviare il datagramma IP incapsulato in un *frame* il cui MAC di destinazione è quello del nodo che ha risposto alla richiesta ARP precedente.
Poiché al protocollo è associata una *cache* (per ridurre le richieste), che inizialmente è vuota e si riempie inviando richieste e ricevendo risposte, ARP è soggetto ad attacchi *cache poisoning; spoofing* (manipolazione dei dati trasmessi in una rete telematica) di risposte ARP contraffatte, per realizzare attacchi ti tipo *man-in-the-middle*:
l'attaccante intercetta la comunicazione e può modificarla, leggerla, anche in presenza di uno *switch* perché evita la comunicazione *broadcast* sulla rete locale, e indirizzarlo direttamente alla macchina agendo sul livello *datalink*.
Poiché tutti ricevono la richiesta non si può superare questa situazione come per DNS, ma si **disabilita il protocollo ARP** sulla rete. La rete avrà una configurazione statica, e la *cache* viene sostituita da un file di configurazione nel quale, per ogni macchina della rete, si va a scrivere la corrispondenza degli indirizzi. Il file è inserito in ogni */etc* delle macchine della rete, e il suo aggiornamento è manuale.

5.2 – Ethernet
Attualmente la tecnologia per LAN cablate più diffusa. È stata ideata a metà degli anni '70 (caso coassiale come bus per connettere i nodi), ed è stata la prima LAN ad alta velocità con vasta diffusione.
Alla fine degli anni '90 la maggior parte delle aziende e delle università avevano installazioni Ethernet con topologia a stella, basata su un *hub*. Un *hub* è un dispositivo a livello fisico che agisce sui singoli *bit*, piuttosto che sui frame; quando un *bit* arriva ad un interfaccia, l'*hub* semplicemente lo rigenera, ne amplifica la potenza, e lo trasmette su tutte le altre interfacce. È una LAN *broadcast* poiché ogni volta che l'*hub* riceve un bit da una delle sua interfacce, ne manda una copia a tutte le altre. Se l'*hub* riceve due *frame* da due diverse interfacce contemporaneamente si verifica una collisione e i nodi che hanno creato i *frame* devono ritrasmetterli.
All'inizio del 2000 Ethernet ha sperimentato un altro cambiamento rivoluzionario: le installazioni continuavano ad usare una topologia a stella, ma l'*hub* è stato sostituito da uno *switch*, che è privo di collisioni ed è un vero e proprio commutatore di pacchetti *store-and-forward* (si limita ad operare a livello 2).

Struttura dei frame Ethernet
Invio di un datagramma IP da un *host* all'altro, sulla stessa LAN Ethernet (il *payolad* può trasportare anche altri pacchetti a livello di rete).
Scheda di rete A con indirizzo MAC AA.AA.AA.AA.AA.AA
Destinazione B con indirizzo MAC BB.BB.BB.BB.BB.BB
La scheda di rete incapsula il datagramma IP in un *frame* Ethernet e lo passa al livello fisico, B lo ottiene allo stesso livello, estrae il datagramma e lo passa al livello di rete.

Preambolo	MAC Dest	MAC Source	T	… Dati …	CRC

Preambolo → 8 *byte*. Sequenza predefinita, nella quale sette *byte* hanno i *bit* 10101010, mentre l'ultimo 10101011. i primi 7 *byte* del preambolo servono per le schede riceventi e per sincronizzare il loro *clock* con quello del trasmittente; questo perché la scheda mittente cercherà di trasmettere il pacchetto a 10Mbps, 100Mbps, o 1Gbps, ma si verificherà sempre una variazione rispetto al tasso previsto che non è conoscibile a priori dalle altre schede di rete sulla LAN. Gli ultimi due *bit* dell'ottavo *byte* avvisano la scheda di rete di destinazione che il resto sta arrivando.
Indirizzo di destinazione → 6 *byte*. Contiene l'indirizzo MAC della scheda di rete di destinazione. I pacchetti ricevuti con un indirizzo MAC diverso da quello indicato in questo campo (o da quello *broadcast*) vengono scartati.
Indirizzo sorgente → 6 *byte*. Indirizzo MAC della scheda di rete che trasmette.

Tipo → 2 *byte*. Indica il protocollo di livello superiore (rete), consente ad Ethernet di supportarne diversi. Poiché gli *host* possono supportare vari protocolli di rete ed utilizzarne di differenti per differenti applicazioni, la scheda di rete di destinazione deve sapere a quale protocollo passare il contenuto del campo dati del *frame* ricevuto.
Dati → da 46 a 1500 *byte*. Dato che la MTU per Ethernet è 1500 *byte*, se il datagramma contenuto supera questo valore, l'*host* deve frammentarlo. Se invece la sua dimensione è inferiore a quella minima, il campo dati dovrà essere *stuffed* fino a raggiungere quel valore. Nell'es. i *byte* di riempimento verranno rimossi servendosi del campo *header length* dell'intestazione del datagramma IP.
Cyclic Redundancy Check → 4 *byte*. Si utilizza l'algoritmo CRC 32. I codici del controllo a ridondanza ciclica sono detti polinomiali, in quanto è possibile vedere la stringa di *bit* da trasmettere come un polinomio i cui coefficienti sono i *bit* della stringa, e le operazioni su di essa come aritmetica polinomiale. È un controllo di integrità in cui i dati d'uscita sono ottenuti elaborando i dati di ingresso, i quali vengono fatti scorrere ciclicamente in una rete logica. Consente alla scheda di rete destinazione di rilevare la presenza di un errore nei *bit* del *frame*. Copre una maggiore quantità di cognizione di errore rispetto al *checksum* a 16 *bit*. Il CRC si trova a livello hardware sulla scheda di rete.

Switch a livello di collegamento
Riceve i *frame* in ingresso e li inoltra sui collegamenti in uscita. Spesso è trasparente ai nodi, i quali indirizzano i *frame* ad altri nodi, e li inviano sulla LAN senza sapere che uno *switch* lo riceverà e inoltrerà agli altri nodi.
Il filtraggio (*filtering*) è la funzionalità dello switch che determina se un frame debba essere inoltrato a qualche interfaccia o scartato. L'inoltro (*forwarding*) consiste nell'individuazione dell'interfaccia verso cui il *frame* deve essere diretto, e quindi nell'invio verso quell'interfaccia.
Le operazioni di inoltro e filtraggio di uno *switch* sono eseguite mediante una tabella di commutazione (*switch table*), composta da voci che contengono: l'indirizzo MAC del nodo, l'interfaccia dello *switch* che conduce al nodo, il momento in cui la voce per quel nodo è stata inserita nella tabella. Quando riceve un *frame*, ne confronta l'indirizzo di destinazione con la propria tabella, se non trova corrispondenza lo manda in *broadcast* su tutte le sue interfacce (tranne quella su cui lo ha ricevuto). Se trova corrispondenza tra l'indirizzo di destinazione e l'interfaccia su cui lo ha ricevuto non occorre inoltrarlo e lo scarta. Se trova corrispondenza tra l'indirizzo destinazione ed un'altra interfaccia (diversa da quella di ricezione) il *frame* deve essere inoltrato al segmento di LAN collegato a quell'interfaccia, quindi lo pone in quel *buffer*.
Si dice che gli *switch* abbiano proprietà di autoapprendimento, in quanto la tabella è inizialmente vuota, e le informazioni vengono archiviate in essa ogni volta che un *frame* viene ricevuto, fino ad avere una tabella completa nel momento in cui tutti i nodi della LAN avranno inviato un *frame*. Inoltre, dopo un dato periodo detto *aging time*, se uno *switch* non riceve *frame* da un determinato indirizzo sorgente, lo cancella dalla tabella.

Vantaggi relativi all'utilizzo degli *switch* (piuttosto che collegamenti *broadcast*):
Eliminazione delle collisioni → in una LAN priva di *hub* non vi è spreco di banda a causa della collisioni. Gli *switch* mettono i *frame* nei *buffer* e non ne trasmettono più di uno su ogni segmento di LAN in un certo istante.
Collegamenti eterogenei → poiché uno *switch* isola un collegamento da un altro, i diversi collegamenti nella LAN possono funzionare a velocità diverse, e possono utilizzare mezzi trasmissivi diversi.
Gestione → facilita la gestione di rete. Può facilmente individuare se una scheda di rete ha un malfunzionamento e disconnetterla internamente. Raccolgono anche statistiche sull'uso della banda, tasso di collisioni e tipi di traffico.

5.3 – Rete Wireless

Le componenti di una rete *wireless* possono essere combinate in vari modi, e formare diverse tipologie di rete.

- *Host wireless* → gli *host* sono dispositivi periferici che eseguono applicazioni.
- Collegamenti *wireless* → l'*host* si connette alla stazione base o ad un altro *host* attraverso un canale di comunicazione *wireless*. Differenti tecnologie comportano diversi tassi trasmissivi e distanze di trasmissione. Le due caratteristiche principali di questi collegamenti sono area di copertura e frequenza del collegamento.
- Stazione base → è responsabile dell'invio e della ricezione dei pacchetti tra gli *host wireless* ad essa associati. Con il termine associato di indica che un *host* si trova nell'area di copertura della *base station*, e la utilizza per trasmettere dati verso il resto della rete (ripetitori di cella nelle reti cellulari, *access point* nelle LAN 802.11). Gli *host* associati ad una stazione base sono considerati come operanti in modalità infrastruttura, in quanto tutti i tradizionali servizi di rete (assegnazione indirizzi, instradamento) sono forniti attraverso la stazione. Nelle reti ad hoc gli *host* non hanno alcuna infrastruttura a cui connettersi e devono provvedere da soli a questi servizi.
  Processo di *handoff* → quando un *host* si sposta dall'area di copertura di una stazione a quella di un'altra, cambiando il sui punto di collegamento con la rete globale.
- Infrastruttura di rete → la rete più ampia alla quale l'host wireless potrebbe volersi connettere.

A livello più alto si possono classificare le reti *wireless* secondo due criteri: il primo considera se un pacchetto nella rete attraversa uno o più collegamenti *wireless*, il secondo considera se vi è o meno un'infrastruttura.

Hop singolo, con infrastruttura → queste reti hanno una stazione base che si collega ad una rete cablata più grande; inoltre, tutte le comunicazioni hanno luogo tra la stazione base e gli *host* su un singolo *hop wireless*. (802.11 in una biblioteca, reti cellulari 4G LTE)

Hop singolo, senza infrastruttura → non vi è alcuna stazione base collegata alla rete wireless; uno dei nodi di questa rete può però coordinare la trasmissione degli altri nodi. (Bluetooth, 802.11 in modalità ad hoc)

Hop multipli, con infrastruttura → è presente una stazione base, collegata tramite cavo alla rete più grande. Tuttavia, alcuni nodi potrebbero dover fare affidamento per le loro comunicazioni su altri nodi *wireless* per comunicare con la stazione base. (reti di sensori, reti *mesh wireless*)

Hop multipli, senza infrastruttura → non c'è una stazione base, e i nodi possono dover ritrasmettere i messaggi a molti altri nodi per raggiungere la destinazione. I nodi possono anche essere mobili, e la connettività tra loro cambiare, come in una classe di reti chiamata *mobile ad hoc network* (MANET). Se i nodi mobili sono veicoli la rete è una *vehicular ad hoc network* (VANET).