Appunti di Sistemi di Elaborazione e Trasmissione dell' Informazione

Roberto Castellotti

November 19, 2020

Contents

1 Funzionamento della Rete

Nello studiare il funzionamento non consideriamo il modello ISO/OSI a 7 livelli in quanto e' uno schema meramente teorico e non corrispondente all' effettivo funzionamento di Internet, studiamo invece il seguente schema a 5 livelli:

[git] ▪ Branch: master @ f5110d9 ▪ Release: 0.70 (2020-11-19)

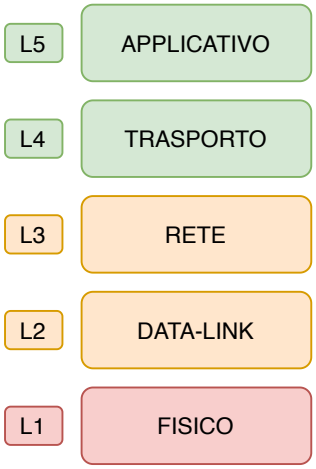

Figure 1: Schema stratificazione rete a 5 livelli

Tecnica Store & Forward La trasmissione di un messaggio tra due host avviene tramite un **muiltihop** (ogni host salva un messaggio e lo forwarda)

1.1 Protocolli di comunicazione

- Al livello 3 il protocollo di riferimento e' il protocollo **IP**, che implementa lo **store and forward multihop**, si divide in:
  - **ipv4** 32 bit (4,294,967,296 possibili indirizzi)
  - **ipv6** 128 bit (2^{128} possibili indirizzi)
- Al livello 2 (data link) il protocollo di riferimento e' **Ehternet 802.3**
- Al livello 5 (applicativo) i protocolli sono vari **(SMTP/HTTP/FTP)**
- Al livello 4 i protocolli di riferimento sono **TCP/UDP**

1.2 User Datagram Protocol

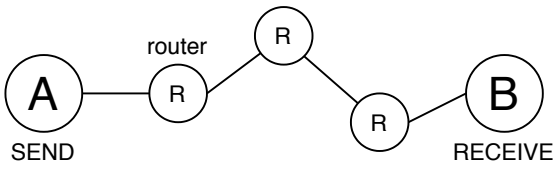

Figure 2: Schema funzionamento User Datagram Protocol

Questa modalita' di trasporto dei dati sostanzialmente aggiunge poco all **IP** (oltre la funzione di multiplexing/demultiplexing (gestione del traffico in uscita e in ingresso, sostanzialmente cio' che si occupa di fare arrivare il traffico al socket giusto sulla porta giusta) unito a un veloce check degli errori), non essendo presente un *handshake* questa modalita' e' anche detta **connectionless**.

Perche' usare UDP:

- non e' necessario stabilire una connessione, quindi e' piu' veloce rispetto a TCP (esepio DNS)
- ha solamente 8 byte di header overhead (TCP ne ha 20)
- non dovendo mantenere lo stato di una connessione (come in TCP) un server puo' servire piu' client

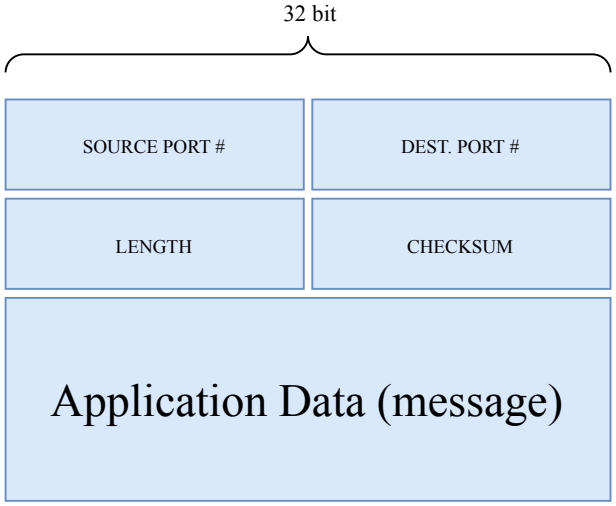

Figure 3: Segmento UDP

Una volta che il messaggio raggiunge la sua destinazione viene segnalata una trap e il sistema operativo processa il messaggio ricevuto.

La funzione SEND termina solamente quiando e' stato scritto il messaggio nei buffer di trasmissione.

Il campo length specifica la lunghezza del messaggio che deve essere inviato (usato poi dal destinatario per salvarlo)

Il protocollo UDP non garantisce il recapito del messaggio, inoltre dal momento che la funzione RECEIVE e' bloccante il destinatario potrebbe rimanere in attesa in eterno nel caso in cui il messaggio andasse perso.

Per avere un feedback sulla ricezione di un messaggio di usa il protocollo TCP.

1.3 Transmission Control Protocol (TCP)

Il protocollo TCP e' caratterizzato da uno stream di connessione bidirezionale, la cui apertura e' detta **three-way handshake**, durante questa fase un host deve fare da SERVER (passivo) e uno da CLIENT (attivo), non ha importanza chi sia server e chi il client, l' unica differenza di comportamento avviene durante la handshake.

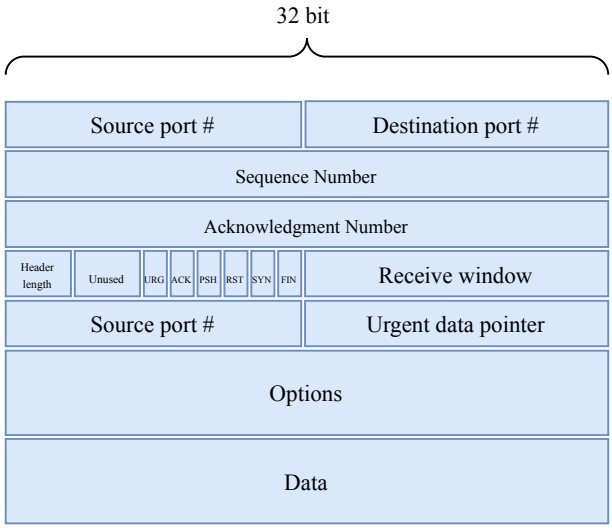

Figure 4: Segmento TCP

sequence number: $\begin{cases} \text{SYN} = 1 & \text{sequence number iniziale} \\ \text{SYN} = 0 & \text{sequence number accumulato dalle trasmissioni} \end{cases}$

acknowledgment number: se ACK=1 il valore del field e' il prossimo sequence number che il sender si aspetta.

FIN: il client vuole chiudere la connessione

RST: il client intende rifiutare la connessione

1.3.1 Three-way handshake

1. il client invia un messaggio SYN al server , viene settato il seq# a un valore random A.
2. il server risponde con un messaggio SYN/ACK, ack# e' settato a $A + 1$, il seq# viene settato a un valore random B
3. il client manda un messaggio ACK al server, seq# viene settato al ack# ricevuto $(A + 1)$ e ack# viene settato a $B + 1$

Inizialmente per il seq# scegliamo un numero casuale per sicurezza (solamente client e server lo conoscono, nessun altro agente puo' mandare messaggi spacciandosi per client/server)

Un host fa passare una determinata quantita' di tempo dopo avere inviato un messaggio, se non riceve un ACK ritrasmette il messaggio, il contenuto viene eliminato dal buffer di trasmissione solo dopo avere ricevuto un ACK, il timeout permette inoltre di correggere eventuali errori di trasmissione.

Il seq# permette di costruire uno stream anche in presenza di errori (mancata trasmissione/sorpasso) in quanto sappiamo esattamente dove storare il payload trasmesso.

Ha senso attendere un ACK solamente per una quantita' di tempo moderata, dipendente dalla velocita' del canale di comunicazione e dalla distanza fisica, il**round-trip time (RTT)** puo' variare, per fare una stima misuriamo un RTT, inizialmente ne viene scelto uno aribitrariamente, successivamente viene scelto un timeout leggermente (in caso di errore vogliamo rimandare il messaggio subito) maggiore al RTT.

Se viene ritrasmesso un valore che era gia' stato ricevuto (viene ricevuto 2 volte lo stesso seq#) significa che non era stato ricevuto un ACK, che quindi viene ritrasmesso.

Tecnica di piggybacking: ACK non viene mandato subito, un host aspetta qualche istante per vedere che non si voglia mandare piu' di un mesaggio (puo' essere disabilitato per ottimizzare trasmissione)

Svantaggi del protocollo TCP

- prima di ogni trasmissione sia necessario un handshake (2 RTT di tempo),
- necessaria differenziazione tra client e server per handshake

2 I Socket

La virtualizzazione della rete avviene attraverso i **socket**, una specie di porta che permette a informazioni di entrare e uscire, questo deve essere messo in comunicazione con la NIC che attraverso un collegamento raggiunge la NIC dell' host destinatario, e da li' al socket destinatario.

Possiamo immaginare un socket come un file (la comunicazione di tipo stream si adatta molto bene all' idea di intendere la comunicazione come una operazione di lettura e scrittura su file).

2.1 Socket di tipo Stream

Una volta stabilita la connessione questa avviene attraverso due buffer (TX e RX) avvengono una serie di syscall:

Lato Client

- **SOCKET:** permette di aprire una connessione (returna un **file descriptor**)
- **CONNECT:** viene dato avvio al three-way handshake lato client

Lato Server

- **SOCKET:** permette di aprire una connessione (returna un **file descriptor**)
- **BIND:** viene stabilito il numero di porta da usare perla ricezione
- **LISTEN:** informa l' OS che si aspetta una richiesta di connessione sulla porta specificata da **BIND**
- **ACCEPT:** ricezione SYN e invio SYN/ACK, restituisce un socket che verra' usato per la comunicazione, il primo viene usato solamente per ricevere SYN e inviare SYN/ACK

3 Domain Name Server (DNS)

Il protocollo DNS converte gli indirizzi a cui siamo abituati negli indirizzi ipv4 delle macchine in rete affinche' sia piu' semplice navigare, la connessione avviene usando il protocollo UDP seguendo questo schema:

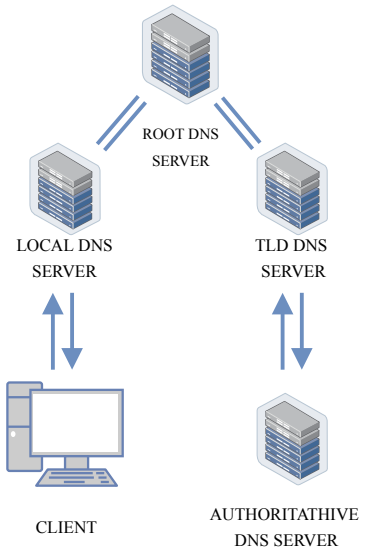

Figure 5: Schema funzionamento generale DNS

Il client invia un datagramma di richiesta (www.google.com) e il server risponde con un indirizzo ip (216.58.198.46), esistono due algortimi di ricerca per i nomi:

- **algoritmo ricorsivo:** la richiesta arriva al **Local DNS Server** che inoltra al **Root DNS Server** che a sua volta contatta il **TLD DNS Server**

che si connette al **Server Autoritativo**, ora la risposta deve fare il percorso all' indietro, quindi ogni server rimane occupato per attendere la risposta.

- **algoritmo iterativo:** la richiesta arriva al **Local DNS Server** che risponde con l' ipv4 del **Root DNS Server**, e cosi' via fino ad arrivare al server autoritativo, questo fa in modo che la connessione sia **stateless**, e' molto piu' efficiente per i server ma il client e' costretto a gestire tutto l' algoritmo

La soluzione che si adootta e' una combinazione delle due: fino al **Local DNS Server** l' algoritmo e' ricorsivo, da li' in poi si usa la versione iterativa, in questo modo i server rimangono liberi, e il carico dell' algoritmo iterativo e' affidato al **Local DNS Server**.

Per ragioni di efficienza esistono 13 copie del **Root DNS Server** distribuite su scala geografica, inoltre gli internet provider realizzano su loro macchine copie del **Root DNS Server** in modo da ridurre il RTT e accellerare la risoluzione dei nomi, inoltre ogni server fa caching delle richieste, quindi spesso si interrompe il flusso di comunicazione prima di arrivare al server autoritativo desiderato

Record DNS e Messaggi I **server DNS** che implementano il database con i dati necessari alla traduzione storano dei **resource record(RR)**, ogni datagramma di risposta di un server contiene una o piu' RR. Ogni RR e' rappresentato da una tupla di 4 elementi contente i seguenti fields:

`(Name,Value,Type,TTL)`

Il TTL (time to live) determina dopo quanto tempo il record debba essere cancellato dalla cache, gli altri valori dipendono dal `Type`:

- Se `Type=A` `Name` e' un hostname e `Value` e' l' indirizzo IP dell' hostname, ad esempio: `(rcastellotti.dev, 164.90.227.145, A)`
- Se `Type=NS` `Name` e' un dominio e `Value` e' l' hostname del server DNS autoritativo che sa come ottenere l' indirizzo IP degli host del dominio specificato, ad esempio: `(foo.com, dns.foo.com, NS)`
- Se `Type=CNAME` `Name` e' un alias `Value` e' il canonical hostname, ad esempio: `(foo.com, relay1.bar.foo.com, CNAME)`
- Se `Type=MX` `Name` e' un alias `Value` e' il canonical name di un mail server, ad esempio: `(foo.com, mail.bar.foo.com, MX)`, questi record permettono ai mail server di avere alias semplici, inoltre una azienda puo' avere lo stesso alias per un web server e un mail server, per ottenere il primo e' necessaria una query per un `CNAME` record, per il secondo una query per un `MX` record.

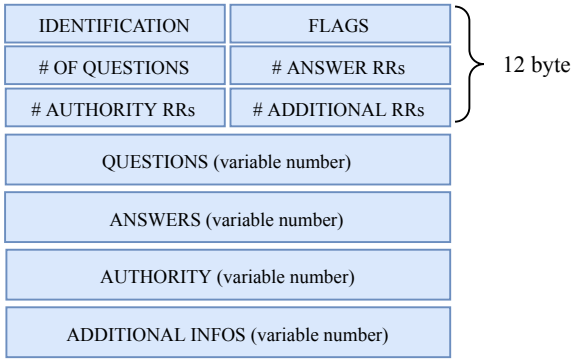

Figure 6: Struttura di un messaggio DNS

- IDENTIFICATION contiene un int a 16 bit che identifica la query (copiato poi nella risposta per permettere al client di matchare richieste e risposte)
- FLAGS:
  - **query/reply** (0/1)
  - **authoritative flag:** il DNS sever e' autoritativo per l' hostname desiderato
  - **recursion desired:** il client desidera che il DNS server usi ricorsione
  - **recursion available:** il DNS server in questione supporta la ricorsione
- QUESTIONS contiene informazioni sulla query che viene fatta (Name, Type)
- **ANSWERS:** contiene uno o piu' RR per il name della query
- **AUTHORITY:** contiene record dei server autoritativi
- **ADDITIONAL INFOS:** informazioni aggiuntive eventuali

Per approfondire eventuali problemi di sicurezza consultare https://www.cloudflare.com/learning/dns/dns-security (in particolare DNS cache poisoning).

4 NTP Protocol (Network Time Protocol)

Il protocollo applicativo NTP si occupa della sincronizzazione degli orologi in rete, prima di approfondirlo vediamo come come le macchine misurano il tempo.

Ogni macchina ha un **Hardware Clock** che si occupa di mantenere il conteggio del tempo quando il dispositivo e' spento (esiste una piccola bateria per alimentarloo), durante la fase di bootstrap un porcesso fa una lettura e da quel momento in poi viene mantenuta una versione software del CLOCK per questioni di efficienza, la stuttura dati per il counter e':

- 1 int che misura i secondi a partire da un istante predefinito
- 1 int che misura i nanosecondi

I valori dell' istante di partenza e della dimensione dei counter dipendono dalla implementazione considerata, le macchine UNIX considerano epoch = 01/01/1970 e usano come counter integer a 32 bit con segno per i secondi e unsigned per i nanosecondi, UTC considera come istante iniziale il 01/01/1900 e usa un counter con int unsigned, in ogni caso questi counter prima o poi overflowano. Per ovviare a questo problema e' gia' stato definito un nuovo standard a 64+64 bit che migliorera' anche la precisione di rappresentazione del tempo.

4.1 Protocollo NTP

Il protocollo NTP prevede un server dotato di un orologio molto preciso da interrogare per sincronizzare gli altri orologi, tuttavia e' facilmente intuibile che ci sia un problema di lag tra quando viene fatta la richiesta e quando viene ricevuta una risposta. Per ovviare a questo problema vengono ripetute 4 misurazioni del tempo:

- t_1: il client invia la richiesta
- t_2: il server riceve la richiesta
- t_3: il server invia la risposta
- t_4: il client riceve la risposta

Ogni volta che bisogna "aggiustare" l' ora su un client non viene modificato il counter (potrebbe causare svariati problemi), ma viene cambiata la frequenza del clock, in questo modo poco alla volta viene sistemato l' orario.

Gli orologi piu' precisi al mondo sono gli orologi atomici (in Italia ne abbiamo uno a Torino), la precisione viene classificati attraverso alcuni strati:

- s0 $\rightarrow$ orologio atomico (estremamente stabile)
- s1 $\rightarrow$ server che legge dall' orologio atomico (connessione con porta seriale, molto piu' stabile e prevedibile)
- s2 $\rightarrow$ server NTP che si collega a un server s1

- s3 → server NTP che si collega a un server s2
- s15 → stato di mancanza di sincronizzazione (prima di collegarsi a un server)

All' aumentare dello strato si perde precisione e si aggiunge dell' errore dovuto alla trasmissione.

Durante una fase di configurazione del protocollo NTP un client inizia una serie di round di connessione con i server, in base alle risposte ricevute e ai timing di risposta viene effettuata una scrematura delle macchine che si possono ritenere affidabili, e' importante notare che se facciamo un grande numero di richieste possiamo fare una media degli RTT, e' quindi piu' importante un RTT stabile rispetto a un RTT piu' basso ma instabile.

Tra i vari problemi di sicurezza da considerare ricordiamo che qualcuno modificando l' orario potrebbe fare accettare dei certificati digitali scaduti.

5 Protocollo FTP

Il protocollo FTP e' un protocollo ideato per scambiare dati tra due macchine connesse in rete attraverso una connessione TCP, dopo che e' stata stabilita una connessione e l' utente si e' autenticato puo' trasferire file dal file system locale a quello del server e viceversa.

Una caratteristica del protocollo FTP e' l' uso di due connessioni TCP parallele: una per **control connection** (porta 21) e una per **data connection** (porta 20), si dice che il protocollo invia informazioni di controllo **out-of-band**. La control connection rimane sempre aperta, mentre la data connection viene aperta e chiusa ogni volta che si desidera inviare un file.

Durante la connessione il server FTP deve mantenere uno **stato** dell' utente, in particolare deve associare la control connection con l' utente specifico, e deve mantenere traccia della `pwd` dell' utente. Questo limita molto il numero di sessioni che FTP riesce a mantenere simultaneamente.

5.1 Comandi e risposte FTP

I comandi FTP (client to server) e le risposte (server to client) sono inviate attraverso la connessione di controllo in formato ASCII a 7 bit, ogni comando e' formato da 4 caratteri ASCII maiuscoli con opzionali argomenti, ecco alcuni comandi:

- `USER rc`
- `PASS ilovebears`
- `LIST:` richiesta di un `ls` nella directory remota
- `RETR filename:` ricerca e invio di un file
- `STORE filename:` put di un file nella directory remota

Le risposte invece sono codificate da numeri e un messaggio opzionale:

- `331: Username OK, password required`
- `125: Data connection already open; transfer starting`
- `425: Can't open data connection`
- `452: Error writing file`

Per ulteriori informazioni: https://tools.ietf.org/html/rfc959

6 Email

La posta elettronica e' un medium di comunicazione asincrono, un messaggio parte dallo **user agent** del mittente, arriva nel **mail server** del mittente, da qui viene inviato al **mail server** del destinatario, e da qui finalmente arriva nella **mailbox** del destinatario. Il server mail del mittente deve essere in grado di mantenere una **message queue** per i messaggi che non riescono ad essere trasferiti, dopo un arco di tempo prestabilito questa queue viene svuotata e il mittente riceve una notifica di failure.

Il principale protocolo application-layer e' SMTP, un protocollo dotato di due side: client e server.

6.1 Protocollo SMTP (RFC 5321)

Il protocollo SMPT (Simple Mail Transmission Protocol) e' usato in larghissima scala, tuttavia si tratta di un protocollo legacy con caratteristiche arcaiche, per esempio il body di tutti i messaggi (non solo gli header) devono essere in 7-bit ASCII, questo al giorno d' oggi comporta un encode e un decode continuo. Vediamo una schematizzazione di un common scenario Alice invia una mail a Bob

- Alice scrive il messaggio e fornisce al suo user agent l' indirizzo mail di bob `bob@smth.com`
- L' user agent invia il messaggio al mail server dove viene inserito in una queue
- Il lato client di SMTP che gira sul mail serve di Alice vede il messaggio nella queue, apre quindi una connessione TCP a un server SMTP che gira sul mail server di Bob (porta 25).
- dopo un handshaking iniziale il client SMTP invia il messaggio attraverso la connessione TCP
- il lato server di SMTP che gira sul server di Bob poisiziona il messaggio nella mailbox di Bob

```
S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.com
S: 250 smtp.example.com, I am glad to meet you
C: MAIL FROM:<bob@example.com>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Bob Example" <bob@example.com>
C: To: Alice Example <alice@example.com>
C: Cc: theboss@example.com
C: Date: Tue, 15 Jan 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice.
C: Test message:5 header fields and 4 lines in message body.
C: Your friend,
C: Bob
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye
{The server closes the connection}
```

Listing 1: Esempio SMTP, da https://en.wikipedia.org/wiki/SMTP

- Bob quando vuole con il suo user agent accede alla mailbox sul server.

Ecco un esempio di comunicazione tra un client e un server STMP Osserviamo che in gergo ASCII ogni messaggio termina con `CRLF.CRLF` (carriage return e line feed) SMTP implementa delle **connessioni user-persistent**, vale a dire che se un utente vuole mandare piu' messaggi puo' farlo usando la stessa connessione TCP.

```
From: bear@rcastellotti.dev
To: kitto@rcastellotti.dev
Subject: Cats are qter than bears and other hilarious jokes.
```

Listing 2: Esempio di header di un messaggio

Fino a questo momento abbiamo fatto una assunzione molto forte: ovvero che Bob si colleghi manualmente al suo mail server e esegua un programma in grado di leggere la mailbox, questo era cio' che si faceva negli anni 80-90, recentemeente sono stati definiti dei protocolli di accesso come **POP3** (Post Office Protcol v3) e **IMAP** (Internet Mail Access Protocol) che si occupano di trasferire le e-mail dal mail server allo user agent del destinatario.

6.2 POP3

Il protocollo POP3 e' definito nell' RFC 1939, prevede una connessione TCP da parte dello user agent con il mail server sulla porta 110 e successivamente 3 fasi:

1. **Authorization:** lo user agent invia username e password (in chiaro)
2. **Transaction:** lo user agent legge messaggi, puo' inoltre cancellarli e fare altre operazioni, incluso ottenere statistiche sui messaggi.
3. **Update:** dopo che lo user agent invia il comando `quit` il mail server elimina i messaggi indicati durante la fase precedente.

7 Livello di trasporto lezione del 26 10

Il TCP offre altre due funzionalita' molto importanti:

- **controllo di flusso**
- **controllo di congestione**

Il controllo di flusso serve per ridurre le perdite dei messaggi. Quando inviamo un messaggio attraverso un controllo di integrita' possiamo determinare se il messaggio ricevuto sia da considerare valido o da scartare. Nelle reti moderne le perdite dei messaggi sono estremamente rare e sono principalmente dovuto

al fatto che nei buffer di ricezione di un host non ci sia spazio per archiviare il datagramma. La cosa piu' intelligente da fare e' rallentare il flusso dei dati per evitare un overflow nei buffer di ricezione.

Ogni datagramma TCP contiene nel suo header una **receive window** <PIPPIPPERO>, gia' inserito nel primo messaggio SYN ed indica la dimensione del buffer di ricezione, quando il server risponde inserisce la sua receive window. Man mano che si ricevono messaggi la receive window viene aggiornata. Un host non invia mai una quantita' di buffer che farebbe overfloware il destinatario, viene salvato il messaggio in un buffer locale in attesa di ricevere un messaggio che comunica la disponibilita' di spazio. Per evitare una situazione di deadlock (il destinatario ha svuotato il suo RX buffer ma il mittente non lo sa) viene inserito un timeout nella comunicazione, una volta esaurito il timeout viene mandato un messaggio da 1 byte anche se la receive window era zero, questo causa una risposta e quindi un aggiornamento dello status di disponibilita' di spazio nel buffer di ricezione.

Controllo di congestione Altro modo per ridurre la perdita dei datagrammi Non e' implementato a livello di IP (quando viene perso un datagramma viene semplicemente inviato un messaggio di errore di tipo ICMP). La soluzione adottata e' di tipo end-to-end (ignora la comunicazione multi-hop a livello IP), si definisce una **congestion window**, ovvero la dimensione del piu' piccolo dei buffer dei router della rete, questo valore deve essere stimato in quanto non e' stato implementato un modo di comunicarlo, l' algoritmo che se ne occupa e' cosi' descritto:

- **slow start**: si parte con un valore piccolo di congestion window (solitamente minore della dimensione del messaggio), se riceve la risposta di acknowledgment, ovvero se non e' successo nulla la volta successiva aumenta in maniera esponenziale la dimensione dei datagrammi. Se in un certo momento non si riceve acknowledgment significa che la rete e' stata saturata e si passa alla modalita' successiva.

- **congestion avoidance:** si torna al valore precedente di congestion window, e si ricomincia di nuovo dall' inizio fino ad arrivare a quello che precedentemente era stato individuato come valore massimo, poi cresce in maniera lineare

Se un host vuole inviare una quantita' notevole di dati e l' host che riceve ha specificato una receive window abbastanza grande comunque dobbiamo considerare delle limitazioni della rete: un datagramma IP ha come dimensione massima 2^16 byte, ulteriormente ristretto a 1500 byte nel caso del protocollo ethernet per esempio(MTU). Il protocollo TCP invia datagrammi di dimensione di MTU Se un messaggio viene diviso in vari datagrammi ovviamente l' acknowledgment e' inviato solo alla ricezione di tutti i messaggi.

8 Protocollo IP

Dispositivi hardware: I router hanno il compito di fare arrivare a destinazione i datagrammi che sono inviati, sono delle macchine di calcolo connesse a molte altre in modo da fare raggiungere la destinazione piu' velocemente possibile.

Per configurare l' algoritmo che invia da un router all' altro e' mantenuta una tabella che associa indirizzi ip e porte del router successivo corrispondenti, ovviamente non possiamo prevedere una tabella di 2^32 righe, per ridurre la dimensione non ci si basa sull' intero indirizzo ip (32bit) ma solo una piccola porzione (ad esempio 8 bit) considerando che possiamo avere piu' bit per gli indirizzi che sono piu' vicini e che quindi conosco meglio e meno bit per host che sono piu' lontani (altri host conosceranno meglio gli indirizzi).

Gli indirizzi IP sono divisi in due parti: NETWORK e HOST, ogni amministratore di una rete puo' decidere quanti dedicarne a cosa.

Per capire quale sia il metodo migliore per "mappare una rete" possiamo immaginare due approcci:

- **approccio globale (link state):** ogni router comunica con tutti gli altri router della rete informando gli altri router delle sue connessioni, in questo modo ci si riconduce presto a un grafo ed esistono algoritmi (Dijkstra) per stabilire il cammino minimo tra due nodi.
- **approccio locale (distance vector):** ogni router comunica con i suoi vicini le sue connessioni, in questo modo ogni router e' in grado di inviare un messaggio al router piu' vicino all' indirizzo di destinazione (tutti gli indirizzi "lontani" saranno mappati con un minore numero di bit, chiaramente questa indicazione non e' molto precisa)

Su reti di piccole dimensioni la soluzione migliore e' quella **Link State**, per reti di grandi dimensioni questa situazione e' difficile da implementare, quello che succede nella realta' e' che Internet e' una rete di sottoreti organizzate tramite **Link State**.

ICMP serve ai router per potere dialogare tra di loro e costruire delle tabelle di routing.