Appunti di Reti

Luca Facchini
Matricola: 245965

Corso tenuto dal prof. Casari Paolo
Università degli Studi di Trento

A.A. 2024/2025

Sommario

Appunti del corso di Reti, tenuto dal prof. Casari Paolo presso l'Università degli Studi di Trento. Corso seguito nell'anno accademico 2024/2025.
Dove non specificato diversamente, il le immagini e i contenuti sono tratti dalle slide del corso del prof. Casari Paolo (paolo.casari@unitn.it)

Indice

Capitolo 1

Introduzione

1.1 Cos'è internet?

Internet
HostUna rete di milioni di dispositivi collegati detti "Host"
Applicazioni di reteun insieme di applicazioni di rete
Collegamentiuna rete di collegamenti fisici (rame, fibra ottica, onde elettromagnetiche, ecc...). Frequenza di trasmissione è uguale a ampiezza di banda
RouterInstrada i pacchetti verso la destinazione finale
ISPInternet Service Provider

Protocollo Un protocollo definisce il formato e l'ordine dei messaggi scambiati fra due o più entità in comunicazione

Standard
RFCRequest for Comments
IETFInternet Engineering Task Force

1.2 Ai confini della rete

Sistemi Terminali (Host)

1. Fanno girare i programmi applicativi (Web, email, ecc...)
2. Sono situati ai margini della rete

Architettura client-server Host client richiede e riceve i servizi da un programma server in esecuzione su un host server

Peer-to-peer Nessun server fisso, i peer sono client e server allo stesso tempo (es. BitTorrent, Skype)

1.2.1 Reti d'accesso e mezzi fisici

1.2.2 Accesso residenziale: punto-punto

Modem dial-up Fino a 56 kbp/s (mai raggiunti) su linea telefonica, non si può telefonare e navigare contemporaneamente

DSL - Digital Subscriber Line Installazione da parte di un ISP, fino a 1-5 Mbps in upstream e 10-50 Mbps in downstream, linea dedicata

1.2.3 FTTH - Fiber To The Home

Fibra ottica Fino a 2.5 Gbps in upstream e 2.5 Gbps in downstream

1.2.4 Accesso aziendale: reti locali (LAN)

LAN Una Local Area Network, o LAN, collega i sistemi terminali di aziende e università all'edge router

Ethernet
Velocità10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps
Moderna Configurazione: sistemi terminali collegati a switch, switch collegato a router

1.2.5 Acceso wireless

Wireless LANs
Wi-Fi2.4 GHz, 5 GHz, 802.11b/g/n/ac
Cellular networks3G, 4G, 5G

1.3 Al nucleo della rete

La rete solitamente è composta da una maglia di router interconnessi, questi lavorano per trovare la migliore strada per arrivare alla destinazione più velocemente possibile

Commutazione di circuito Esistono delle risorse punto-punto riservate alla comunicazione
Time Division Multiplexing Il tempo è diviso in slot, ogni slot è assegnato ad una connessione diversa
Frequency Division Multiplexing La banda è divisa in frequenze, ogni frequenza è assegnata ad una connessione diversa.

Capitolo 2

Il Livello Applicazione

2.1 principi delle applicazioni di rete

2.1.1 Architetture di rete

Esistono varie architetture per le applicazioni di rete, tra le quali:

- Client-Server
- Peer-to-Peer
- Architetture ibride
- Cloud Computing

Client - Server

In questa architettura esistono due ruoli principali:

Server Il server è un host **sempre attivo** con un **indirizzo permanente** e molto spesso difficile da scalare

Client Il client **comunica col server**, inoltre a differenza del server può **disconnettersi temporaneamente** e inoltre può avere un **indirizzo IP dinamico**. Generalmente i client **non comunicano tra di loro**.

Architettura P2P pura

In questa architettura **non c'è** sempre un server attivo, vengono eseguire **coppie arbitrarie** di host che comunicano tra di loro. Infine i **peer** non devono necessariamente essere sempre attivi e possono avere un **indirizzo IP dinamico**.

Architetture ibride

In queste architetture si ha una combinazione tra client-server e P2P, ad esempio un server con peer che comunicano tra di loro. Un esempio di architettura ibrida è Skype, oppure un applicativo di messaggistica istantanea dove le chat sono P2P ma l'individuazione degli utenti è fatta tramite un server centrale.

Cloud Computing

In questa architettura si ha un insieme di tecnologie che permettono **di memorizzare archiviare e/o elaborare dati** tramite l'utilizzo di risorse distribuite. La creazione di copie di sicurezza dette **backup**

è automatica e l'operabilità si trasferisce online. I dati sono memorizzati in **server farm** generalmente localizzate nei paesi di origine del service provider.

2.1.2 Struttura delle applicazioni di rete

Processi del sistema operativo

Processo: Programma in esecuzione su un host.
All'interno di uno stesso host due processi comunicano utilizzando **schemi interprocesso** (definiti dal S.O.)
Processi su host diversi comunicano tramite **messaggi** scambiati tramite la rete.

Processo client processo che inizia la comunicazione

Processo server processo che attende di essere contattato

Socket

Socket Un processo che invia/riceve messaggi a/da il suo **socket**.
Un socket è analogo ad una porta

2.1.3 Indirizzamento

IP Per identificare un host in modo univoco si usa un **indirizzo IP** che è formato da 32 bit.

Numeri di porta L'indirizzo IP però non è sufficiente ad identificare un processo all'interno dell'host per questo definiamo dei **numeri di porta**.

2.1.4 Protocolli a livello applicazione

Definizioni

I protocolli a livello applicazione definiscono:

- Tipi di messaggi scambiati
- Sintassi dei messaggi
- Semantica dei campi dei messaggi
- Regole per determinare quando e come i processi inviano e ricevono messaggi

Protocolli dominio pubblico Alcuni protocolli sono di pubblico dominio definiti nelle **RFC** (Request for Comments) della **IETF** (Internet Engineering Task Force). Questi consentono interoperabilità tra diversi host, esempi di protocolli a pubblico dominio sono: **HTTP**, **SMTP**...

Protocolli proprietari Altri protocolli sono proprietari, ad esempio Skype.

2.1.5 Servizi di trasporto

Come segliere il protocollo di trasporto

Perdita di dati Applicazioni che richiedono trasmissione affidabile dei dati (es. file transfer) richiedono un protocollo di trasporto affidabile

Temporizzazione Applicazioni che richiedono bassa latenza (es. VoIP) richiedono un protocollo di trasporto con bassa temporizzazione

Throughput Applicazioni che richiedono alto throughput richiedono un protocollo di trasporto con alto throughput

Sicurezza Applicazioni che richiedono sicurezza (es. trasferimento di file) richiedono un protocollo di trasporto sicuro

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

TCP / UDP

TCP **Transmission Control Protocol** è un protocollo di trasporto **affidabile** e **orientato alla connessione**. TCP ha un **controllo di flusso** e **controllo di congestione**, **non offre** temporizzazione e garanzie su un'ampiezza di banda minima, sicurezza.

UDP **User Datagram Protocol** è un protocollo di trasporto **inaffidabile** fra i processi d'invio e di ricezione. UDP **non offre** controllo di flusso, controllo di congestione, temporizzazione, garanzie su un'ampiezza di banda minima, sicurezza.

Applicazione	**Protocollo a livello applicazione**	**Protocollo di trasporto**
Posta elettronica	SMTP [RFC 2821]	TCP
Accesso a terminali remoti	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Trasferimento file	FTP [RFC 959]	TCP
Multimedia in streaming	HTTP [RFC 2616], RTP [RFC 3550]	TCP, UDP
Telefonia Internet	SIP [RFC 3261], RTP [RFC 3550], Proprietario	Tipicamente UDP

2.2 Web e HTTP

2.2.1 Terminologia

Pagina Web Una **pagina web** è costituita da **oggetti**

Oggetto Un **oggetto** può essere una **pagina HTML**, un'**immagine**, un'**applet**, un'**audio**, un'**video**, ...

un file HTML è un **file base** per formare una **pagina web**. Suddetto file è scritto tramite l'**HyperText Markup Language** che include diversi oggetti referenziati

URL Ogni oggetto è referenziato tramite un **URL** (Uniform Resource Locator)

Esempio di URL

`http://www.sito.com/folder/file.html`

http Protocollo di trasferimento

www.sito.com Nome del server

folder Cartella in cui si trova il file

file.html Nome del file

2.2.2 Introduzione a HTTP

Overview L'**HTTP** (HyperText Transfer Protocol) è un protocollo di livello applicazione del web. Sfrutta il modello **client-server** dove il **client** invia una **richiesta** al **server** che risponde con una **risposta** contenente il **contenuto richiesto** e il client visualizza il contenuto.

Usa TCP Il client inizializza una connessione **TCP** con il server sulla porta 80, il server accetta la connessione **TCP** del client e si scambiano messaggi HTTP tra il browser e il web-server. Quando il trasferimento è completato la connessione **TCP** viene chiusa.

Si noti come il protocollo HTTP sia **stateless**, ovvero non mantiene informazioni sullo stato del client.

Connessioni HTTP

Connessioni non persistenti Almeno un oggetto viene trasmesso su una connessione `TCP`.

1. Il client `HTTP` inizializza una connessione `TCP` con un server `HTTP` sulla porta 80
2. Il server `HTTP` sul host in attesa di una connessione `TCP` alla porta 80
3. Il client `HTTP` trasmette un *messaggio di richiesta* con l'*URL* nella socket della connessione `TCP`. Il messaggio indica che oggetto si vuole
4. Il server `HTTP` trasmette un *messaggio di risposta* con l'oggetto richiesto nella socket della connessione `TCP`
5. Il server chiude la connessione `TCP`
6. Il client riceve l'oggetto e visualizza l'oggetto richiesto e all'arrivo del messaggio di risposta chiude la connessione `TCP`

- Il metodo di connessione non persistente richiede 2 round-trip time (`RTT`) per ottenere un oggetto.
- Overhead di connessione `TCP` per ogni oggetto richiesto
- I browser moderni spesso in caso di connessioni non persistenti aprono richieste parallele per ottenere più oggetti contemporaneamente

Connessioni persistenti Più oggetti vengono trasmessi su una connessione TCP

Tipi dei metodi

GET Il client richiede un oggetto al server

POST Il client invia dati al server

HEAD Il client richiede solo l'intestazione dell'oggetto

PUT Il client invia un oggetto al server (da HTTP/1.1)

DELETE Il client cancella un oggetto dal server (da HTTP/1.1)

Messaggio di risposta HTTP

```
HTTP/1.1 200 OK ⇒ Versione del protocollo, codice di stato, frase di stato
Connection close ⇒ Connessione chiusa
Date:  Thu, 06 Aug 1998 12:00:15 GMT ⇒ Data e ora
Server:  Apache/1.3.0 (Unix) ⇒ Server web
Last-Modified:  Mon, 22 Jun 1998 ...  ⇒ Data ultima modifica
Content-Length:  6821 ⇒ Lunghezza del contenuto
Content-Type:  text/html ⇒ Tipo di contenuto
dati dati dati dati dati ...  ⇒ Dati
```

Codici di stato

200 OK ⇒ La richiesta è stata completata con successo l'oggetto richiesto è stato trasmesso

301 Moved Permanently ⇒ Il documento richiesto è stato spostato in un'altra locazione

400 Bad Request ⇒ La richiesta non può essere soddisfatta di errori client

404 Not Found ⇒ Il documento richiesto non è stato trovato sul server

505 HTTP Version Not Supported ⇒ La versione HTTP usata non è supportata dal server

2.2.3 Cookies

I **cookies** sono composti da quattro componenti:

1. Una riga di intestazione nel messaggio di *risposta HTTP*
2. Una riga di intestazione nel messaggio di *richiesta HTTP*
3. Un file mantenuto sul *client*
4. Un database mantenuto sul *server*

Come vengono usati cookies

Cosa contengono

- Autorizzazione
- Carta per acquisti
- Raccomandazioni
- Stato della sessioni dell'utente

Lo Stato

- Mantengono lo stato del mittente e del ricevente per più richieste
- I messaggi HTTP trasportano lo stato

Privacy

- I cookies possono essere usati per tracciare la navigazione dell'utente
- L'utente può fornire al sito nome e l'indirizzo

2.2.4 Cache web

Obbiettivo: soddisfare le richieste degli utenti senza coinvolgere il server d'origine

Cache è una copia di un oggetto mantenuta da un'entità più vicina all'utente

Il Procedimento Il client invia una richiesta al server proxy, il server proxy invia la richiesta al server d'origine se l'oggetto non è in cache, altrimenti il server proxy invia l'oggetto al client.

Vantaggi Riduzione del tempo di risposta, riduzione del traffico di rete, riduzione del carico sui server d'origine

Perchè viene usata Viene usata per ridurre il tempo di risposta e il traffico di rete, in certe situazioni delle istituzioni si possono dotare di un cache interna per ridurre il traffico di rete verso l'esterno e per ridurre il tempo di risposta.

`GET` **condizionale** Il client può chiedere al server proxy di inviare l'oggetto solo se è stato modificato, in caso contrario il server proxy invia un messaggio di risposta con codice 304 (Not Modified) e l'oggetto non viene inviato. Il controllo viene eseguito tramite un **header** `If-Modified-Since` che contiene la data dell'ultima modifica dell'oggetto.

2.2.5 HTTP/1.0 e HTTP/1.1

HTTP/1.0

- Connessioni non persistenti
- Ogni oggetto richiede una connessione TCP separata
- Non supporta proxy
- Non supporta cache

HTTP/1.1

- Connessioni persistenti
- Pipelining
- Host Virtuale
- Cache
- Cookies
- Connessioni persistenti

- Pipelining
- Host Virtuale
- Cache
- Cookies

2.2.6 HTTP/2.0

HTTP/2 rappresenta una evoluzione di **HTTP/1.1**, il protocollo è focalizzato sulle prestazioni, specificatamente sulla latenza percepita. Obbiettivo di `HTTP/2` è di avere una unica connessione per browser.

Framing binario

Nuovo livello di framing binario per incapsulare i messaggi `HTTP`, in questo modo la semantica `HTTP` rimane invariata ma la codifica in transito è differente. Tutte le comunicazioni `HTTP/2` sono suddivise in messaggi più piccoli, ognuno dei quali codificano un formato binario, inoltre sia il client che il server possono inviare messaggi in qualsiasi momento.ù

Stream, messaggi e frame

Tutte le comunicazioni vengono eseguite all'interno di una connessione TCP bidirezionale, ogni **stream** ha un identificativo univoco con priorità. Ogni messaggio è un messaggio `HTTP` logico (richiesta/risposta). Il frame è la più piccola unità di comunicazione di un certo tipo specifico di dati.

Multiplexing di richieste e risposte In **HTTP/1.x** se il client esegue più richieste in parallelo per migliorare le prestazioni deve usate `TCP` multiple (`HTTP/1.1 o HTTP/1.2`) oppure aprire una nuova connessione (`HTTP/1.0`). Grazie al **framing binario** di `HTTP/2` è possibile rimuovere queste limitazioni consentendo il **multiplexing** di richieste e risposte.

Priorità degli streamL'ordine nel quale i frame vengono inviati dal client o dal server influenza le prestazioni, per questo motivo `HTTP/2` supporta di associare a ciascun `stream` una priorità e delle dipendenze. Infatti ogni stream può avere un peso tra 1 ovvero il peso minimo e 256 ovvero il peso massimo, inoltre uno stream può avere un elenco di dipendenza su altri stream. Grazie a questa funzionalità il client costruisce un "**albero di priorità**" in modo da ottimizzare il caricamento della pagina.

Server Push Il server può inviare più risposte per una singola richiesta (se ad esempio è necessaria una dipendenza per il caricamento della pagina) in modo da ridurre il tempo di caricamento della pagina senza dover attendere la richiesta del client.

2.2.7 Transport Layer Security (TLS)

Il `TLS` ovvero **Transport Layer Security** è un protocollo crittografico che permette una comunicazione sicura da sorgente a destinatario fornendo: **Autenticazione**, **Integrità dei dati** e **Confidenzialità**. [1]
Il funzionamento del `TLS` può essere riassunto in tre fasi:

1. Negoziazione fra client e server per stabilire l'algoritmo di crittografia da usare
2. Scambio delle chiavi per la crittografia e autenticazione della comunicazione
3. Cifratura simmetrica dei dati e autenticazione dei dati

[1] Più sulla sicurezza di computer e reti in: "Appunti di Introduction to Computer and Network Security" di Luca Facchini

2.2.8 HTTPS

`HTTPS` è un protocollo di comunicazione sicura che estende `HTTP` aggiungendo una crittografia tramite `TLS`. Il protocollo `HTTPS` usa la porta 443 e permette tutti i vantaggi di `TLS` come l'autenticazione, l'integrità dei dati e la confidenzialità. Questo però non significa che tutto il traffico dei livelli inferiori sia crittografato, infatti solo il traffico (header e dati) del livello applicazione è crittografato.

2.3 FTP - File Transfer Protocol

FTP Il **File Transfer Protocol** è un protocollo di trasferimento di file che permette di trasferire file tra un host e un server. FTP è un protocollo **stateful** che mantiene lo stato del client e del server durante la sessione. Lo standard FTP è definito nella **RFC 959** e usa una porta standard di **21**.

2.3.1 Connessione di controllo

Connessione di controllo La connessione di controllo è usata per inviare comandi tra il client e il server. I comandi sono inviati in **ASCII** e i comandi sono **case-insensitive**. La connessione di controllo è **stateful** e mantiene lo stato del client e del server durante la sessione. La connessione di controllo usa la porta **21**, mentre la connessione dati usa la porta **20**, questo è un esempio di protocollo con **controllo fuori banda**.

2.3.2 Comandi & Risposte FTP

Comandi FTP

USER *username* Autenticazione con l'username

PASS *password* Autenticazione con la password

LIST Mostra i file nella directory corrente

RETR *filename* Recupera un file dalla directory corrente

STOR *filename* Memorizza un file nella directory corrente

Risposte FTP

331 Username OK, password richiesta

125 Connessione dati aperta, inizio trasferimento

425 Connessione dati non aperta

452 Errore di memorizzazione

2.4 Posta Elettronica

Introduzione Per la gestione della posta elettronica esistono 3 componenti principali:

- Agente utente
- Server di posta
- Simple Mail Transfer Protocol (SMTP)

Agente utenteè detto anche "mail reader" e permette di comporre, modificare e leggere i messaggi di posta elettronica. I messaggi in uscita o in arrivo vengono memorizzati sul server di posta che è sempre attivo.

Server di posta Contiene la **Casella di posta** che contiene i messaggi in arrivo, ha una **coda di messaggi** in uscita ed usa il **protocollo SMTP** tra server di posta per inviare messaggi di posta elettronica, in quanto il protocollo **SMTP** richiede che il server ricevente sia sempre in ascolto.

2.4.1 SMTP

Il protocollo **SMTP** (Simple Mail Transfer Protocol) è un protocollo di livello applicazione che permette di inviare messaggi di posta elettronica tra server di posta. Il protocollo **SMTP** usa la porta **25** ed è un protocollo **stateless**.

Fasi del trasferimento Il trasferimento di un messaggio di posta elettronica avviene in tre fasi:

Handshaking Il client apre una connessione **TCP** con il server di posta, il server risponde con un messaggio di benvenuto

trasferimento Il client invia il messaggio, il server accetta il messaggio e lo deposita nella casella di posta del destinatario

Chiusura Il client chiude la connessione

Iterazione comando/risposa I comando usano **ASCII** a 7 bit e sono **case-insensitive**, le risposte sono codificate con un codice a tre cifre.

Note finali

- Il protocollo usa connessioni **persistenti**
- Il protocollo richiede che il messaggio (intestazione e corpo) sia nel formato **ASCII** a 7 bit
- Il protocollo prevede che `<CR><LF>.<CR><LF>` sia usato per terminare il messaggio

Formato dei messaggi di posta elettronica

Intestazione contiene i mittenti, i destinatari, il soggetto, la data e l'ora

riga vuota separa l'intestazione dal corpo

Corpo contiene il testo del messaggio

2.4.2 POP3

Il protocollo **POP3** (Post Office Protocol 3) è un protocollo di livello applicazione che permette di scaricare i messaggi di posta elettronica dal server di posta. Il protocollo **POP3** usa la porta **110** ed è un protocollo **stateful**.

Fasi del trasferimento Il trasferimento di un messaggio di posta elettronica avviene in tre fasi:

autorizzazione Il client apre una connessione **TCP** con il server di posta, il client si autentica con il server

trasferimento Il client scarica i messaggi di posta elettronica

Chiusura Il client chiude la connessione

Comandi POP3

USER Autenticazione

PASS Password

LIST Lista dei messaggi

RETR Recupera un messaggio

DELE Cancella un messaggio

QUIT Chiude la connessione

2.4.3 IMAP

Il protocollo **IMAP** (Internet Message Access Protocol) è un protocollo di livello applicazione che permette di scaricare i messaggi di posta elettronica dal server di posta. Il protocollo **IMAP** usa la porta **143** ed è un protocollo **stateful**.

Fasi del trasferimento Il trasferimento di un messaggio di posta elettronica avviene in tre fasi:

autorizzazione Il client apre una connessione **TCP** con il server di posta, il client si autentica con il server

trasferimento Il client scarica i messaggi di posta elettronica

Chiusura Il client chiude la connessione

Comandi IMAP

LOGIN Autenticazione

SELECT Seleziona una casella di posta

FETCH Recupera un messaggio

STORE Modifica lo stato di un messaggio

LOGOUT Chiude la connessione

2.5 DNS

Introduzione

Domain Name Sysyem Il **DNS** consiste in un *database distribuito* implementando una gerarchia di *server DNS*. Il *DNS* è un protocollo a livello applicazione che consente agli host e ai router di comunicare per *risolvere* i nomi degli host in indirizzi IP.

2.5.1 Servizi DNS

- Traduzione degli hostname in indirizzi IP
- Host aliasing - Un host può avere più nomi
- Mail server aliasing - Un host può avere più server di posta
- Payload distribution - Distribuzione del carico tra i server

Perchè non centralizzare DNS

- Singolo punto di fallimento
- Traffico di rete
- Database centralizzato distante
- Manutenzione

2.5.2 Struttura del DNS

In generale i server DNS sono organizzati in una struttura gerarchica a **albero** dove il nodo radice è il server DNS radice (13 al mondo) esistono dei server di DNS di nomi di primo livello (com) (TLD) e infine i server di DNS autoritativi usati per un dominio di secondo livello (google.com)

Server DNS locali Ogni ISP ha un server DNS locale che si occupa di tradurre i nomi degli host in indirizzi IP

2.5.3 Resource Record RR

Resource Record Un RR è una tupla che contiene i seguenti campi:

- `Name` - Il nome del dominio
- `Value` - Il valore del campo
- `Type` - Il tipo di record
- `TTL` - Il tempo di vita del record

Tipi di RR

- `A` - Indirizzo IP - **name**: `hostname` **value**: `IP`
- `NS` - Server di nomi - **name**: `dominio` **value**: `hostname`
- `CNAME` - Nome canonico - **name**: `alias` **value**: `hostname`
- `MX` - Mail server - **name**: `dominio` **value**: `hostname`

2.5.4 Inserire un record

Esempio Abbiamo avviato la nuova società

- Registriamo il nome "foo.com" presso un `registrar`
- Otteniamo un indirizzo IP per il nostro server web (host)
- Diamo al nostro registrar l'indirizzo IP del nostro server web e il nome del nostro server web. Esempio records: `(foo.com, dns1.foo.com, NS)`, `(dns1.foo.com, 211.211.211.211, A)`

2.6 Condivisione di file P2P

La condivisione di file in modalità P2P non prevede un *server* sempre attivo, ma un numero arbitrario di coppie di *host* o *peer* che comunicano direttamente tra di loro. I *peer* non devono essere sempre attivi e inoltre possono cambiare indirizzo IP.

2.6.1 Distribuzione di File

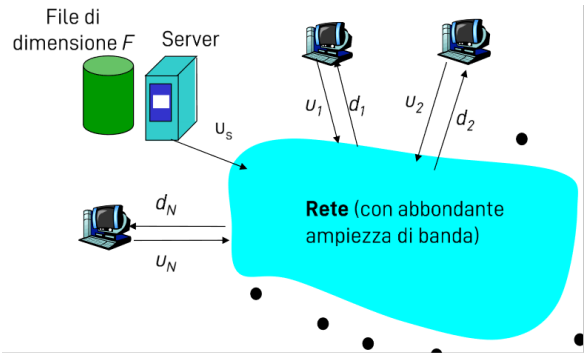

Figura 2.1: Distribuzione di file in modalità P2P

Domanda: Tenendo a mente la soprastante figura, ci si può chiedere: Quanto tempo ci vuole per distribuire un file da un server a N *peer*?
Consideriamo: U_s il *bitrate* in uscita del collegamento di accesso del server, U_i bit rate di upload del collegamento di accesso dell i-esimo *peer* e d_i il *bitrate* di download del collegamento di accesso dell i-esimo *peer*.

Soluzione *server-client* In una architettura *server-client* il tempo di distribuzione è dato dal server che deve inviare N copie moltiplicato per la dimensione del file F il tutto diviso per il *bitrate* in uscita del server U_s, quindi in formula: $T = \frac{N \cdot F}{U_s}$. Ora il client i-esimo impiega il tempo F/d_i per scaricare il file. Per calcolare il tempo per distribuire il file F su N client è uguale al massimo tra il tempo impiegato dal server per la trasmissione (U_s) e dal tempo massimo impiegato da un client per scaricare il file (d_i), quindi $d_{cs} = \max\left\{\frac{F}{U_s}, \frac{F}{\min(d_i)}\right\}$. Notare come il tempo di distribuzione aumenta linearmente rispetto al numero di *peer*.

Soluzione per *P2P* In una architettura *P2P* per calcolare il tempo di distribuzione dobbiamo prima sapere quanto tempo impiega *server* per inviare la prima copia del file $T_s = \frac{F}{U_s}$ e poi necessitiamo di sapere quanto tempo impiega l'i-esimo client per scaricare il file: $T_i = \frac{F}{d_i}$. Dato che poi il *peer* che ha scaricato il file poi può trasmetterlo a sua volta allora il può veloce tasso di *upload* è: $U_s + \sum_i U_i$. Il tempo totale è dato dunque dal massimo tra: tempo di invio prima copia, tempo massimo di un peer e il tempo di invio di tutte le copie ($\frac{NF}{U_s \sum_i U_i}$), quindi in modo formulare: $d_{P2P} = \max\left\{\frac{F}{U_s}, \frac{F}{\min(d_i)}, \frac{NF}{U_S + \sum_i u_i}\right\}$. Notare come il tempo di distribuzione a meno di un tempo costate per l'invio della prima copia si riduce all'aumentare del numero di *peer*.

BitTorrnet

Introduzione Nel protocollo di *BitTorrent* si usa la distribuzione di file in modo P2P ma sono presenti dei *server* detti *tracker* che mantengono una lista di *peer* partecipanti alla rete. Un nuovo *peer* che vuole scaricare un file si connette al *tracker* e ottiene la lista che stanno scaricando il file. Il *peer* scarica il file da più *peer* contemporaneamente, andando quindi a costituire una rete *torrent*.

Caratteristiche I file vengono divisi in *chunk* da $256kB$, quando un peer entra a far parte del torrent non possiede nessun *chunk*, dunque si registra al server *tracker* che gli assegna una lista di *neighbors* ovvero vicini dai quali scaricherà i *chunk* del file, quando un *chunk* è scaricato il *peer* lo condivide con gli altri alimentando l'effetto *tit-for-tat*. Una volta che il file è scaricato nella sua completezza il *peer* può lasciare la rete egoisticamente (*leech*) o contribuire a questa (*seeder*)

2.6.2 Ricerca delle informazioni

I sistemi di tipo P2P devono in qualche modo fornire un indice della posizione dei *peer* e di in quali di questi si può trovare un particolare file, solitamente questo viene fatto attraverso una *Distributed hash table*.

File sharing *e-mule*

In un sistema come *e-mule* l'indice tiene traccia dinamicamente della posizione dei file che i *peer* condividono, questi condividono i file disponibili. Un nuovo *peer* cerca nell'indice quello che vuole trovare e poi stabilisce una connessione diretta al *peer* contenete il file cercato.

Messaggistica istantanea

Nel caso della messaggistica istantanea l'indice crea una corrispondenza tra utenti e posizione (ip), quando un utente si registra all'applicazione informa il server della sua posizione, per inviare un messaggio ad un utente il *peer* chiede al server la posizione dell'utente e poi stabilisce una connessione diretta.

Directory centralizzata

Nel caso di *napster* invece quando un *peer* si connette alla rete informa il server centrale del suo indirizzo ip e del contenuto condiviso, se un altro *peer* cerca il contenuto dal server centrale allora questo restituisce l'indirizzo ip del *peer* che condivide il file e il *peer* può scaricare il file direttamente.

Problemi In primo luogo questa *directory centralizzata* costituisce un *Single point of failure*, inoltre essendo un solo punto questo è un importante *bottleneck* infine se vengono condivisi file protetti da *copyright* il server centrale ne è responsabile.

Query flooding

Un sistema che adotta il *query flooding* è completamente distribuito senza un server centrale, ogni *peer* mantiene un indice locale dei file condivisi e quando un *peer* cerca un file invia una *query* a tutti i *peer* vicini che se non hanno il file cercato inoltrano la *query* ai loro vicini e così via. Questo sistema è molto efficiente ma può generare un grande traffico di rete. Una volta trovato il file viene istituita una connessione diretta TCP tra i due *peer*. Questo sistema è molto efficiente ma può generare un grande traffico di rete ed è soggetto ad attacchi di tipo *DoS*, se viene richiesto un file inesistente la *query* viene inoltrata a tutti i *peer* della rete.

2.7 Cloud Computing

Introduzione Il *cloud computing* prevede che uno o più *server* reali, organizzati in una architettura ad alta affidabilità e fisicamente collocati in un *data center* del fornitore del servizio. Il fornitore espone delle interfacce per elencare e gestire i propri servizi, un utente amministratore usa queste selezionando i servizi richiesti per accedervi o amministrarlo, infine un utente finale può accedere ai servizi tramite un'interfaccia web o un'applicazione.

Criticità

- **Privacy & Sicurezza** - I dati sono memorizzati in un server remoto
- **Continuità del servizio** - I servizi sono soggetti a interruzioni e necessitano di connessione internet
- **Problemi internazionali di natura economico-politica** - I dati possono essere soggetti a leggi di paesi diversi
- **Difficoltà di migrazione** - I dati possono essere difficili da migrare una volta che sono stati caricati

2.7.1 *Content Delivery Network* - `CDN`

Introduzione Una *CDN* è un sistema di server distribuiti che lavorano insieme per fornire contenuti web ad utenti finali con prestazioni elevate e alta affidabilità. Una *CDN* è composta da *server* detti *edge server* che sono distribuiti in tutto il mondo, un *server* centrale detto *origin server* che contiene i contenuti originali e un *server* detto *DNS server*

Esempio

- Un utente richiede un oggetto
- Il *DNS server* determina il *edge server CDN* più vicino all'utente
- Il *edge server* richiede l'oggetto all'*origin server* e lo memorizza
- Il *edge server* invia l'oggetto all'utente
- Il *edge server* memorizza l'oggetto per un certo periodo di tempo
- Se un altro utente richiede lo stesso oggetto il *edge server* lo invia direttamente
- Se l'oggetto non è più richiesto il *edge server* lo elimina
- Se l'oggetto cambia il *edge server* lo richiede all'*origin server*

Conclusione "There is no cloud, it's just someone else's computer"

Capitolo 3

Il Livello di Trasporto

Obbiettivi

- Capire i principi che sono alla base dei servizi di livello di trasporto:
  - Multiplexing/de-multiplexing
  - Trasferimento dati affidabile
  - Controllo di flusso
  - Controllo di congestione
- Descrivere i protocolli del livello di trasporto di Internet:
  - TCP: Trasposto orientato alla connessione
  - Controllo di congestione TCP
  - UDP: Trasporto non orientato alla connessione

3.1 Servizi a livello di trasporto

Introduzione I **protocolli di trasporto** forniscono la comunicazione logica tra processi applicativi di host diversi. I protocolli di trasporto vengono eseguiti negli host "terminali" ovvero quelli che generano o consumano i dati. Dal lato di inviante il protocollo di trasporto divide in diversi segmenti i dati ricevuti dal livello di applicazione e li invia al livello di rete. Dal lato di ricevente il protocollo di trasporto riassembla i segmenti ricevuti e li invia al livello di applicazione.

TCP Il **TCP** (Transmission Control Protocol) è un protocollo di trasporto orientato alla connessione. Il TCP fornisce un trasferimento affidabile dei dati, controllo di flusso e controllo di congestione.

UDP L'**UDP** (User Datagram Protocol) è un protocollo di trasporto non orientato alla connessione. L'UDP non fornisce trasferimento affidabile dei dati, controllo di flusso e controllo di congestione.

Servizi non disponibili Al momento in internet non è disponibile un servizio di garanzia su ritardi (latenza), e non è disponibile un servizio di garanzia sulla banda (velocità di trasferimento).

3.2 Multiplexing e De-multiplexing

Introduzione Il **multiplexing** è il processo di invio di dati da più socket a un'unica connessione, per identificare il socket di destinazione si utilizza un **port number**. Il **de-multiplexing** è il processo di invio dei dati ricevuti al socket corretto in base al port number.

3.2.1 De-multiplexing

Come funziona In primo luogo quando l'*host* riceve un segmento IP contenente: IP del mittente, IP del destinatario, protocollo di trasporto, porta di destinazione e porta di sorgente. L'*host* utilizza l'indirizzo IP del destinatario e la porta di destinazione per inviare il segmento al processo corretto.

De-multiplexing senza connessione

Per eseguire il de-multiplexing senza connessione si crea un socket per ricevere i dati. Il socket è ora associato a una porta ed a un indirizzo IP. Quando l'*host* riceve il segmento `UDP`, viene controllato che il numero di porta di destinazione sia uguale alla porta del socket. Se il numero di porta non corrisponde il segmento viene scartato. Se invece il numero di porta corrisponde il segmento viene inviato al processo associato al socket.

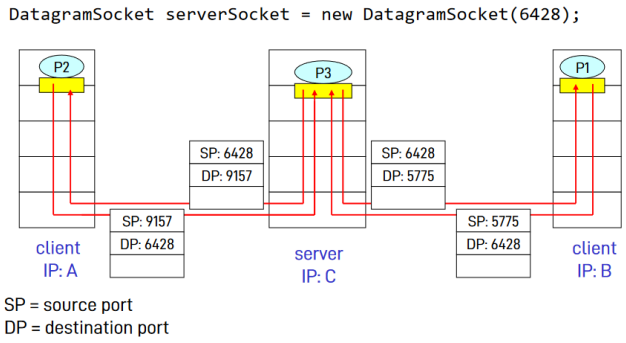

Figura 3.1: De-multiplexing senza connessione

De-multiplexing orientato alla connessione

Quando si utilizza un protocollo orientato alla connessione (`TCP`), il processo di de-multiplexing è leggermente diverso. Il socket infatti è costituito da quattro parametri: indirizzo IP del mittente, indirizzo IP del destinatario, numero di porta di sorgente e numero di porta di destinazione. Quando l'*host* riceve un segmento `TCP` controlla che i quattro parametri del socket corrispondano ai quattro parametri del segmento. Se i parametri non corrispondono il segmento viene scartato, altrimenti viene inviato al processo associato al socket. Un *host* può supportare più socket contemporaneamente purché cambi almeno uno dei quattro parametri, inoltre i *web server* sono un chiaro esempio di applicazione che utilizza più socket contemporaneamente (su `HTTP/1.0` un socket per ogni richiesta).[1]

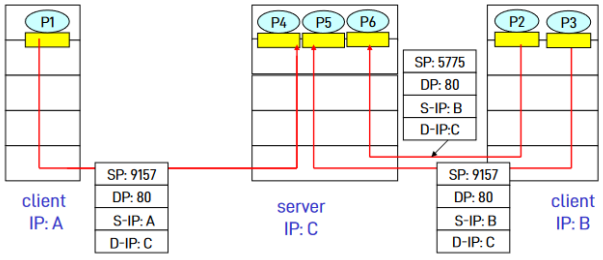

Figura 3.2: De-multiplexing orientato alla connessione

[1] Se viene allocata una porta ad una connessione, la porta non può essere utilizzata da altre connessioni, quindi nel caso di un *web server* è vero che questo ascolta sulla porta 80, ma quando un client si connette al server, il server apre un socket con una porta dinamica.

3.2.2 Porte TCP-UDP

La destinazione finale di un segmento non è un host ma un processo. L'interfaccia tra l'applicazione e il livello di trasporto è chiamata **socket** o **porta** (nel caso di UDP e TCP). Un **socket** è un indirizzo IP e un numero di porta. Un **port number** è un numero a 16 bit che identifica un processo all'interno di un host. Esiste una mappatura biunivoca tra un **port number** e un processo. I servizi standard utilizzano porte ben note con valori tra 0 e 1023. I processi non-standard e le connessioni in ingresso a un client usano numeri fino a 25535 (16 bit).

Numeri di porte I numeri di porta si classificano come segue:

Statici Per i servizi standard, es. HTTP (80), FTP (21), SSH (22), Telnet (23), SMTP (25), POP3 (110), IMAP (143), HTTPS (443), ecc.

Dinamici (o "ephemeral") per le connessioni in uscita o per porte allocate dinamicamente, es. client web, client FTP, client SSH, client Telnet, client SMTP, client POP3, client IMAP, client HTTPS, ecc.

Inoltre è importante dire che le porte di sorgente e di destinazione non sono le stesse in quanto la porta di sorgente è una porta dinamica assegnata dal sistema operativo.

3.3 Trasporto senza connessione: UDP

Caratteristiche L'**UDP** (User Datagram Protocol) è un protocollo di trasporto senza connessione, offre un servizio *best effort* e non fornisce garanzie di consegna, ordine o duplicazione dei dati. Questo in quanto non ha *handshake* iniziale e non mantiene alcuno stato di connessione.

Perché esiste UDP Non richiede di stabilire una connessione, è semplice e veloce, Header di segmento corti, senza controllo di congestione.

3.3.1 Header

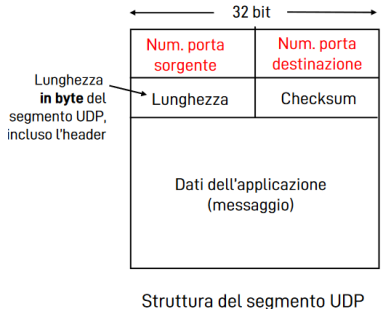

Figura 3.3: Header di un segmento UDP

Porta di sorgente Numero di porta del processo mittente.

Porta di destinazione Numero di porta del processo destinatario.

Lunghezza Lunghezza del segmento in byte.

Checksum Utilizzato per rilevare errori nel segmento.

Checksum

Il checksum è un campo a 16 bit che viene utilizzato per rilevare errori nel segmento. Questo viene calcolato da entrambe le parti: viene trattato il contenuto come una sequenza di 16 bit e si sommano tutti i bit (se presente riporto questo viene sommato a sua volta) e viene eseguito il complemento a 1. Il mittente invia il checksum calcolato nel segmento e il ricevente calcola il checksum del segmento ricevuto e lo confronta con il checksum ricevuto. Se i due checksum non corrispondono il segmento viene scartato.

3.4 Principi del trasferimento dati affidabile

3.4.1 ARQ

ARQ o *Automatic Repeat reQuest* è una classe di protocolli che "cercano" di recuperare i segmenti persi o danneggiati. Questa classe usa dei pacchetti speciali per notificare al mittente che un segmento è stato perso o danneggiato. Questi pacchetti speciali sono chiamati ACK (Acknowledgement) e NACK (Negative Acknowledgement).

Esempi di protocolli basati su ARQ

- *Stop-and-Wait*
- *Go-Back-N*
- *Selective Repeat*
- TCP
- il protocollo MAC (al livello 2) dei sistemi WiFi

3.4.2 *Stop-and-Wait*

Nel protocollo *Stop-and-Wait* il mittente invia una PDU e ne mantiene una copia in memoria, imposta dunque un *timeout* su quel PDU. Attende poi un ACK dal ricevente, se non riceve l'ACK entro il *timeout* invia nuovamente la PDU. Se invece riceve l'ACK controlla che questo non contenga errori, che sia il numero di sequenza corretto e che sia per la PDU inviata. Se tutto è corretto invia la prossima PDU. Il ricevente quando riceve una PDU controlla che il numero di sequenza sia corretto e che la PDU non abbia errori, se tutto è corretto invia un ACK al mittente, de-capsula la PDU ai livelli superiori. Se sono presenti errori nella PDU il ricevente esegue il *drop* della PDU.

Efficienza dello *Stop-and-Wait*

Assumendo una banda $R = 1G - bit/s, 15ms$ di ritardo di propagazione, lunghezza del messaggio $L = 8000bit$, allora il tempo di trasmissione sarà: $T_{trans} = \frac{L}{R} = \frac{80000}{10^9} = 8\mu s$. Mentre il *throughput* percepito a livello applicativo sarà: $\frac{L}{T_{trans}+RTT} = \frac{8000}{8\mu s+30ms} = 33Kbps$.[2] Dunque anche se la nostra banda è di $1Gbps$ il *throughput* percepito a livello applicativo è di $33Kbps$, l'efficienza dunque è: $\frac{T_{trans}}{T_{trans}+RTT} = \frac{0.008}{0.008+30} = 0.00027$ ovvero 0.027%.

3.4.3 Protocolli con *pipelining*

I protocolli con *pipelining* permettono di inviare più segmenti successivi senza attendere l'ACK del segmento precedente, si allarga dunque il range dei pacchetti di sequenza accettabili. Questo permette di aumentare l'efficienza del trasferimento dati. Esempio di protocolli con *pipelining* sono *Go-Back-N* e *Selective Repeat*.

[2] Aggiungiamo della formula il RTT ovvero il *Round Trip Time* che è il tempo che impiega un pacchetto per andare dal mittente al ricevente e ritornare indietro, nel nostro caso lo aggiungiamo per il pacchetto di ACK ed è di $30ms$.

Throughput in presenza di *pipelining*

Assumiamo la stessa situazione precedente ed un *pipelining* di $N = 3$ allora il *throughput* sarà: $\frac{3L}{T_{trans}+RTT} = \frac{24000}{8\mu s+30ms} = 100Kbps$, questo è un miglioramento del 300% rispetto allo *Stop-and-Wait*. In generale il *throughput* rispetto al *pipelining* con N segmenti in parallelo è: $\frac{N \cdot L}{RTT+T_{trans}}$. Il parametro N è detto *window size* o "dimensione della finestra".

Definizioni

Finestra di trasmissione W_T Insieme di `PDU` che il mittente può inviare senza attendere un `ACK` del ricevente.

Grande al massimo come la memoria allocata dal sistema operativo.

$|W_T|$ indica la dimensione della finestra.

Finestra di ricezione W_R Insieme di `PDU` che il ricevente può ricevere può accettare e memorizzare.

Grande al massimo come la memoria allocata dal sistema operativo.

Puntatore *low* W_{LOW} Indica il primo segmento della finestra di trasmissione W_T.

Puntatore *high* W_{HIGH} Indica l'ultimo segmento già trasmesso della finestra di trasmissione W_T.

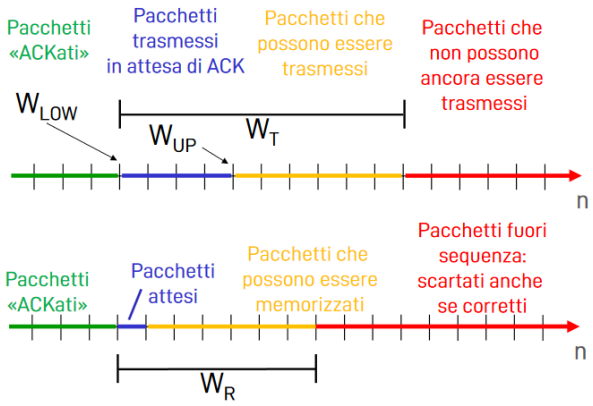

Figura 3.4: Finestre di trasmissione e di ricezione

Acknowledgements - `ACK`

Esistono vari tipi di `ACK` a seconda del protocollo utilizzato, abbiamo dunque:

- `ACK` individuale il cui compito è quello di indicare la corretta ricezione di uno specifico pacchetto - `ACK(`n`)` vuol dire ho ricevuto il pacchetto n.
- `ACK` cumulativo il cui compito è quello di indicare la corretta ricezione di tutti i pacchetti fino a quello specificato - `ACK(`n`)` vuol dire ho ricevuto tutti i pacchetti fino a n (escluso), mi aspetto il pacchetto n.
- `ACK` negativo o `NACK` il cui compito è quello di indicare la mancata ricezione di un pacchetto - `NACK(`n`)` vuol dire non ho ricevuto il pacchetto n, invialo nuovamente.
- Esiste poi la tecnica del "**Piggybacking**", ovvero l'inserimento dell'`ACK` (di un pacchetto precedente) all'interno di un pacchetto dati successivo.

Go-Back-N

Quando si sceglie di usare il protocollo del tipo *Go-Back-N* questo consiste in: il mittente invia fino ad un numero n di pacchetti senza aver ricevuto prima ACK, quando un pacchetto è stato ricevuto correttamente viene inviato un ACK cumulativo, se un pacchetto non è stato ricevuto allora i pacchetti successivi vengono scartati in attesa del pacchetto mancante. Dopo un periodo di *timeout* il mittente invia nuovamente tutti i pacchetti a partire dal pacchetto mancante, basandosi sull'ultimo ACK ricevuto. La finestra di trasmissione W_T è dunque composta da n pacchetti e non viene spostata finché non si riceve un ACK cumulativo, mentre la finestra di ricezione W_R è composta da un solo pacchetto.

Selective Repeat

Nel paradigma del *selective repeat* vengono usati ACK singoli, inoltre è presente una finestra di ricezione W_R composta da m pacchetti, ciò significa che anche se un pacchetto ricevuto fuori sequenza viene ricevuto allora questo viene comunque "salvato" all'interno di un buffer in attesa del pacchetto nell'ordine corretto. Anche il mittente in caso di ACK fuori sequenza conserva in memoria questo dato e non lo scarta. Quello che succede se un pacchetto non viene ricevuto ma qualche pacchetto (fino a $m-1$) successivo viene ricevuto correttamente è che il mittente invia nuovamente solo il pacchetto mancante, mentre i pacchetti successivi, se sono già stati ACK'ati, non vengono inviati nuovamente e la trasmissione riprende dal primo pacchetto non ACK' ato.

Spazio dei numeri di sequenza Solitamente se si hanno k bit a disposizione allora si usa un periodo pari a 2^k, ovvero il periodo massimo con quello spazio di bit. Le finestre di trasmissione per non avere conflitti devono avere somma inferiore al periodo, quindi $|W_T| + |W_R| < 2^k$.

3.5 Trasporto orientato alla connessione TCP

TCP - vari standard RFC Il TCP o *Transmission Control Protocol* è un protocollo di trasporto orientato alla connessione, è stato standardizzato nel RFC 793 e successivamente aggiornato con il RFC 1122, RFC 1323, 2018, 2581 ed è in continuo aggiornamento. Questo prevede una connessione punto-punto tra mittente e destinatario, è presente un flusso di byte affidabile e consegnato in ordine senza limiti, è presente un meccanismo di *pipelining* e di controllo di congestione per non sovraccaricare la rete. La connessione inoltre (anche se a livelli inferiori non lo è) è *full-duplex* ovvero entrambi i lati possono inviare e ricevere dati contemporaneamente. Inoltre il TCP è un protocollo *stateful* ovvero mantiene uno stato della connessione, infatti si dice che è orientato alla connessione. Infine ha presente un flusso controllato, il trasmettitore non può inviare dati se il ricevente non è pronto a riceverli.

3.5.1 Struttura di un pacchetto TCP

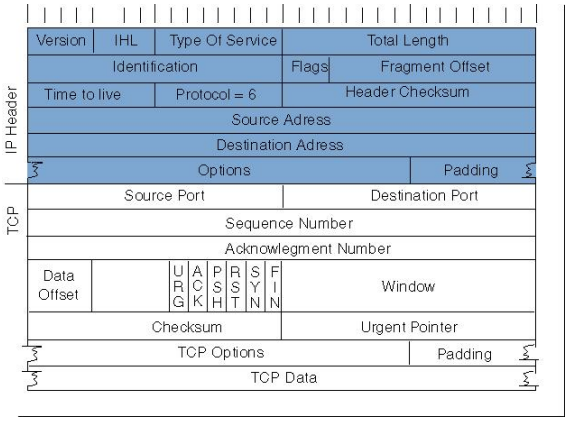

Figura 3.5: Struttura di un pacchetto TCP
[3]

Nel pacchetto TCP sono presenti i seguenti campi principali:

Source Port Porta di sorgente.

Destination Port Porta di destinazione.

Sequence Number Numero di sequenza del primo byte del segmento.

Acknowledgement Number Numero di sequenza del prossimo byte atteso.

Data Offset Lunghezza dell'header in parole di 32 bit.

URG Flag che indica la presenza di dati urgenti.

ACK Flag che indica la presenza di un campo ACK.

PSH Flag che indica che i dati devono essere passati al livello superiore.

PST Flag che indica l'inizio di una connessione.

SYN Flag che indica la sincronizzazione dei numeri di sequenza.

FIN Flag che indica la chiusura della connessione.

Window Dimensione della finestra di ricezione. (RWND)

Checksum Utilizzato per rilevare errori nel segmento (contiene oltre ai parametri TCP anche i parametri IP come l'indirizzo IP del mittente e del destinatario, la lunghezza del segmento, il protocollo di trasporto, ecc.).

Urgent Pointer Puntatore ai dati urgenti.

TCP Options Opzioni aggiuntive. (opzionali)

Padding Padding per allineare il segmento a 32 bit.

Data Dati del segmento.

Finestra di ricezione (RWND) La finestra di ricezione è un campo a 16 bit che indica la dimensione della finestra di ricezione del ricevente. Questo campo è utilizzato per il controllo di flusso, infatti il mittente non può inviare dati se la finestra di ricezione del ricevente è piena. La finestra di ricezione è un campo a 16 bit, quindi la dimensione massima della finestra di ricezione è di $2^{16} - 1 = 65535$ byte. In base alla velocità della banda questo campo può essere modificato per evitare che il mittente non sfrutti tutta la banda disponibile.

Numeri di sequenza ACK di TCP I numeri di sequenza di TCP sono a 32 bit, questo significa che il numero di sequenza può variare tra 0 e $2^{32} - 1 = 4294967295$. Il numero di sequenza nella direzione mittente-ricevente può essere diverso da quello nella direzione ricevente-mittente, questo perché i numeri di sequenza sono indipendenti nelle due direzioni, inoltre non è detto che i numeri di sequenza inizino da 0, infatti possono iniziano solitamente da un numero casuale. Durante la trasmissione di un pacchetto mittente-ricevente può essere allegato anche un campo ACK per la conferma della ricezione del pacchetto precedente tra ricevente-mittente (stessa cosa per la direzione opposta).

Lunghezza massima segmento MSS e MTU

In quanto il TCP lavora per byte cerca sempre di non inviare un singolo byte solo in quanto sarebbe uno spreco di risorse e di banda. Allo stesso tempo non si può inviare un segmento troppo grande in quanto potrebbe essere frammentato a livello di rete. Viene dunque introdotta una "lunghezza massima" detta MSS (*Maximum Segment Size*) che indica la lunghezza massima di un segmento TCP. La MSS è calcolata come la MTU (*Maximum Transmission Unit*) che è la lunghezza massima di un pacchetto che può essere inviato su una rete a livello di collegamento. A sua volta la MTU viene calcolata da passati al livello *data-link* e può variare da rete a rete. La MSS si riferisce non alla lunghezza di tutto il segmento TCP ma solo al *payload*, ovvero il campo dati del segmento TCP.

Come si sceglie MSS? Non esistono meccanismi per comunicarlo, viene dunque adottato un modello del tipo *trial& error*, ovvero il mittente invia un segmento con una MSS di dimensione X se si nota che i livelli inferiori sopportano la dimensione X allora si aumenta la dimensione della MSS, altrimenti se si nota che qualche messaggio inizia ad essere perso si riduce la dimensione della MSS.

Valodi di default:

- MTU di ethernet: 1500 byte (payload inseribile al livello 2)
- Header IP: 20 byte
- Header TCP: 20 byte
- MSS di default: 1460 byte

"*Least maximum*" La più piccola MTU impostabile per IP è di 576 byte, questo dunque la MSS di minima impostabile è di 536 byte.

3.5.2 Setup della connessione TCP - *handshake*

La procedura di apertura di una connessione TCP è detta *three-way handshake*, questa procedura è composta dai seguenti passaggi:

1. **Host A** invia un segmento TCP con il flag SYN impostato a 1 e la porta di sorgente A e la porta di destinazione B.
2. **Host B** riceve il segmento TCP e invia un segmento TCP con il flag SYN impostato a 1 e il flag ACK impostato a 1 (avvenuta la ricezione del segmento di SYN) e la porta di sorgente B e la porta di destinazione A.
3. **Host A** riceve il segmento TCP e invia un segmento TCP con il flag ACK impostato a 1 (avvenuta la ricezione del segmento di SYN) e la porta di sorgente A e la porta di destinazione B.

In tutti questi passaggi il numero di ACK non si riferisce al numero di sequenza del segmento ricevuto ma al numero di sequenza del prossimo segmento atteso. Questo *handshake* è necessario per sincronizzare i numeri di sequenza tra mittente e ricevente.

3.5.3 Chiusura della connessione TCP

La procedura di chiusura di una connessione TCP è detta *tearDown*, questa richiede che la connessione sia chiusa in tutte e due le direzioni. Esiste una maniera "gentile" per chiudere la connessione che prevede l'invio di un segmento TCP con il flag FIN impostato a 1. A questo punto la controparte riceve il segmento TCP e invia un ACK & FIN, ora la connessione è *half-closed* ovvero la connessione è chiusa in una direzione ma aperta nell'altra. Quando il primo host riceve ACK & FIN, ora la connessione è chiusa in entrambe le direzioni. Questo meccanismo è necessario per evitare che i segmenti TCP vengano persi durante la trasmissione.

Chiusura con RST Se un host invia un segmento TCP con il flag RST (reset) impostato a 1 allora la connessione viene chiusa immediatamente senza attendere risposta dall'altra parte. Questo meccanismo è utilizzato per chiudere una connessione in modo "brusco" in caso di problemi.

3.5.4 Tempi RTT e RTO

Il TCP deve "impostare" un *timeout* per l'invio e per la ricezione dei segmenti, questo *timeout* è detto RTO (*Retransmission TimeOut*), deve essere dunque un valore superiore al RTT (*Round Trip Time*) ovvero il tempo che impiega un pacchetto per andare dal mittente al ricevente e ritornare indietro. Il RTT può variare nel tempo, quindi il RTO deve essere impostato in modo dinamico, se è troppo basso si rischia di inviare prematuramente un segmento, se è troppo alto si rischia di "aspettare" per troppo tempo. Quindi in sostanza và prima stimato il RTT e poi impostato il RTO, stimiamo il RTT tramite un *sampleRTT* ovvero il tempo tra l'invio del pacchetto e la ricezione dell'ACK. Per mantenere il tempo aggiornato ma non troppo sensibile ai "picchi" che si possono verificare sulla rete viene utilizzata la seguente formula:

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

Dove α è un parametro che indica la "sensibilità" del tempo, se α è basso allora il tempo sarà poco sensibile ai picchi, se α è alto allora il tempo sarà molto sensibile ai picchi, questa è una media mobile esponenziale ponderata. Solitamente $\alpha = 0.125$.
Per impostare RTO non ci avvaliamo solamente di questo dato appena ricavato ma anche della deviazione standard del RTT ovvero

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

Dove β è un parametro che indica la "sensibilità" della deviazione standard, se β è basso allora la deviazione standard sarà poco sensibile ai picchi, se β è alto allora la deviazione standard sarà molto sensibile ai picchi, questa è una media mobile esponenziale ponderata. Solitamente $\beta = 0.25$.
Infine il RTO viene calcolato come:

$$\text{RTO} = \text{EstimatedRTT} + 4 \cdot \text{DevRTT}^4$$

3.5.5 Controllo di flusso RWND

Il TCP implementa un meccanismo di controllo di flusso per evitare che il mittente invii troppi dati al ricevente, questo meccanismo è basato sulla finestra di ricezione W_R, il mittente non può inviare dati se la finestra di ricezione del ricevente è piena. La finestra di ricezione è un campo a 16 bit, quindi la dimensione massima della finestra di ricezione è di $2^{16} - 1 = 65535$ byte. Il mittente invia dati fino a $\min(W_T, W_R)$, dove W_T è la finestra di trasmissione del mittente e W_R è la finestra di ricezione del ricevente. Il ricevente invia un segmento TCP con il campo Window impostato alla dimensione della finestra di ricezione, il mittente legge questo campo e regola la dimensione della finestra di trasmissione in base a questo valore.

[4] 4 è una costante alla quale tutto il mondo si è accordato ed è usato come misura di sicurezza.

3.6 Principi di controllo di congestione

Informalmente la congestione si può tradurre come "troppi trasmettitori stanno mandando troppi dati e la **rete** non riesce a gestirli". Quindi il problema è nella rete e non nel ricevitore. Questo problema si può verificare come: pacchetti persi (*buffer overflow*) o ritardi (*queueing delay*). Il controllo di congestione è un insieme di tecniche che permettono di evitare che la rete vada in congestione. Il controllo di congestione è un problema molto complesso e non esiste una soluzione al problema, esistono però delle tecniche che permettono di mitigare il problema.

3.6.1 Cause/costi della congestione

Scenario 1

Assumiamo di avere due trasmettitori e due ricevitori, un *router* con una coda di dimensione infinita, la capacità del link in uscita è R e non possono esserci ritrasmissioni, allora il *throughput* massimo per ogni trasmettitore è $R/2$, ma se entrambi i trasmettitori inviano dati contemporaneamente allora il ritardo salirà asintoticamente con l'avvicinarsi a $R/2$.

Scenario 2

Assumiamo di avere due trasmettitori e due ricevitori, un *router* con una coda di dimensione finita, la capacità del link in uscita è R e il mittente ritrasmette i pacchetti in timeout, allora considerando il tasso di arrivo dall'applicazione del mittente λ_{in} e il tasso percepito dal destinatario λ_{out} e il fatto che il mittente invii dati solo quando il router ha spazio nel buffer allora ci troveremmo nel caso ideale ed abbiamo a disposizione un *throughput* di $R/2$ per ogni trasmettitore. Se invece il mittente invia dati senza preoccuparsi dello stato del router invia e re-invia i pacchetti in caso di timeout allora per un input di λ_{in} pari a $R/2$ il *throughput* in uscita sarà di $R/4$ (per via delle ritrasmissioni).

3.7 Controllo di congestione `TCP`

Alcune cose da dire Innanzitutto bisogna dire che non esiste un solo algoritmo per il controllo di congestione, esistono infatti molte varianti, ognuna di queste è stata introdotta per rimuovere delle limitazioni della versione precedente. Inoltre l'implementazione di un algoritmo o di un altro dipende spesso dal sistema operativo. Tutte le implementazioni di `TCP` ragionano in *byte*.

Caratteristiche Il controllo di congestione **adatta il tasso di trasmissione** sulla base delle condizioni della rete, inoltre lo scopo è quello di evitare di **saturare** e **congestionare** la rete.

Approcci possibili Esistono due approcci possibili per il controllo di congestione:

- **Controllo di congestione *end-to-end***

  Non coinvolge la rete

  Si capisce se c'è congestione osservando perdite di pacchetti o ritardi

  Metodo usato da `TCP`

- **Controllo di congestione assistito dalla rete**

  I router forniscono feedback ai trasmettitori

  Un singolo bit per indicare la congestione

3.7.1 TCP *congestion control: additive increase multiplicative decrease* (AIMD)

Approccio

Il mittente aumenta il tasso di trasmissione cercando di occupare la banda disponibile e diminuisce il tasso di trasmissione quando rileva una perdita. Questo algoritmo segue due passi fondamentali:

- **Additive Increase** Aumenta il tasso di trasmissione di 1 MSS ogni RTT finché non si rileva una perdita.
- **Multiplicative Decrease** riduce la finestra (tipicamente di un fattore $\frac{1}{2}$) quando si rileva una perdita.

Perché usare AIMD

Per ottenere un *fairness* tra i trasmettitori, infatti se k sessioni TCP si dividono uno stesso *link* ed è presente un *bottleneck* allora la banda percepita da ogni trasmettitore sarà di $\frac{R}{k}$.
Due sessioni in competizione sullo stesso *link* di banda R allora poniamo un grafico sul quale l'asse delle ascisse è la banda percepita della sessione 1 e l'asse delle ordinate è la banda percepita della sessione 2.

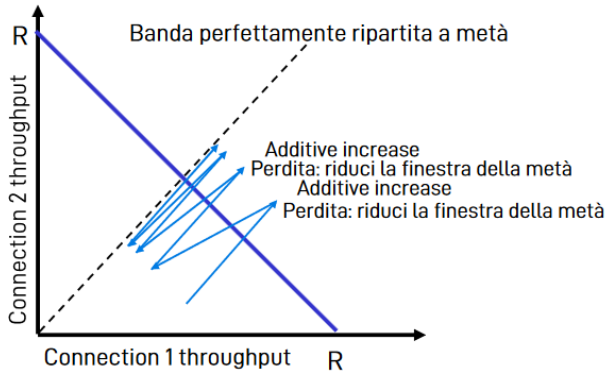

Figura 3.6: Grafico di congestione

Dal grafico si vede come nel tempo le connessioni oscillano verso il punto di intersezione, questo è dovuto al fatto che entrambe le connessioni aumentano la loro banda fino a che non si satura il link, a quel punto entrambe rilevano una perdita e riducono la loro banda dello stesso fattore, questo porta ad un *fairness* tra le due connessioni.

3.7.2 Meccanismi per il controllo di congestione

Il controllo di congestione gestisce l'adattamento della cosiddetta finestra di congestione ($CWND$ = numero di byte che il mittente può inviare).
In TCP ci sono diversi algoritmi per il controllo di congestione, tra i più famosi ci sono:

- **In assenza di perdite**

  Slow Start

  Congestion Avoidance

- Per migliorare l'efficienza di TCP in caso si verifichino perdite

  Fast Retransmit

  Fast Recovery

In ogni caso: $|W_T| = \min(\texttt{CWND}, \texttt{RWND}) = \min(\texttt{CWND}, |W_R|)$.

Slow Start

Il *Slow Start* è un algoritmo che prevede che per ogni `ACK` ricevuto aumento di 1 `MSS` la finestra di congestione, in questo modo la finestra aumenta esponenzialmente. Questo algoritmo è utilizzato quando la connessione è appena stata aperta e non si conosce la banda disponibile. Il *Slow Start* termina quando la finestra di congestione raggiunge una soglia detta `SSThresh` (*Slow Start Threshold*) e si passa al *Congestion Avoidance*.

Algoritmo della fase *slow start*

1. **Inizializzazione**
   - `CWND` = 1 `MSS`
   - `SSThresh` = `RWND` (o `RWND`/2)
2. `ACK` **valido ricevuto**:
   - `CWND` = `CWND` + 1 `MSS`
   - Sposto W_{LOW} al primo segmento non `ACK'ato`
   - Se `CWND` $\geq$ `SSThresh` allora passo alla fase di *Congestion Avoidance*
   - Trasmetto nuovi segmenti (compresi tra W_{LOW} e W_{HIGH})
3. **Se scatta un *timeout***:
   - Abbasso `SSThresh` a $\max(\texttt{CWND}/2, 2)$
   - Aumento `RTO` = 2 `RTO`
   - Reimposto `CWND` = 1 `MSS`
   - Ritrasmetto il segmento in timeout

Congestion Avoidance

Il *Congestion Avoidance* è un algoritmo che prevede che per ogni `ACK` ricevuto aumento di $\texttt{MSS} \cdot \frac{\texttt{MSS}}{\texttt{CWND}}$ la finestra di congestione, in questo modo la finestra aumenta linearmente. Quindi per ogni `RTT` in cui ricevo tutti gli `ACK` attesi allora aumento di un segmento. Questo algoritmo segue un comportamento lineare e non esponenziale.

Algoritmo della fase *congestion avoidance*

- **Se ricevo un `ACK` valido**:
  - `CWND` = `CWND` + $\frac{\texttt{MSS}}{\texttt{CWND}}$ (in byte!)
  - Sposto W_{LOW} al primo segmento non `ACK'ato`
  - Trasmetto nuovi segmenti (compresi tra W_{LOW} e W_{HIGH})
- **Se scatta un *timeout***:
  - Passo alla fase di *Slow Start*
  - Abbasso `SSThresh` = $\max(\texttt{CWND}/2, 2)$
  - Aumento `RTO` = 2 `RTO`
  - Reimposto `CWND` = 1 `MSS`
  - Ritrasmetto il segmento in timeout

Parametri i quali possono essere modificati

- **CWND** (*Congestion Window*) Dimensione della finestra di congestione.
- **SSThresh** (*Slow Start Threshold*) Soglia per il *Slow Start*.
- **RTOT** (*Retransmission TimeOut*) Tempo di ritrasmissione.
- W_{LOW} & W_{HIGH} Puntatori alla finestra di trasmissione.

Fast Retransmit

Il *Fast Retransmit* è un algoritmo che prevede che se il mittente riceve tre `ACK` duplicati allora ritrasmette il segmento richiesto dal `ACK` duplicato, inoltre il fatto che il mittente riceva tre `ACK` duplicati indica che c'è una congestione sulla rete si passa dunque alla fase di *Fast Recovery*. Inoltre prendo in considerazione il valore di `RECOVER`=W_{UP} per determinare quanti segmenti sono stati trasmessi nella rete, in questo modo posso capire quando il processo di *Fast Retransmit* è terminato con successo.

Fast Recovery

Quando ricevo il 3° `ACK` duplicato entro in *Fast Recovery*, in questa fase avvengono diversi passaggi per evitare di saturare la rete:

- **Al 3° `ACK` duplicato**:
  - `SSThresh` = `CWND`/2
  - Supponendo di aver perso solo il segmento in questione: `CWND` = `SSthresh` + 3 `MSS`
  - Non sposto W_{LOW}
- **Se arrivano altri `ACK` duplicati allora**:
  - `CWND` = `CWND` +1 `MSS`
  - Non sposto W_{LOW}
- **Quando arriva un `ACK` valido** (che comprende `RECOVER`):
  - `CWND` = `SSthresh`
  - Passo alla fase di *Congestion Avoidance*
  - Sposto W_{LOW} al primo segmento non `ACK'ato`
- **Se arriva un `ACK` che <u>non</u> comprende `RECOVER`**:
  - Ritrasmetto il primo segmento non `ACK'ato`
  - `CWND` = `CWND`-(numero di segmenti senza `ACK`)+1
  - Sposto W_{LOW} al primo segmento non `ACK'ato`

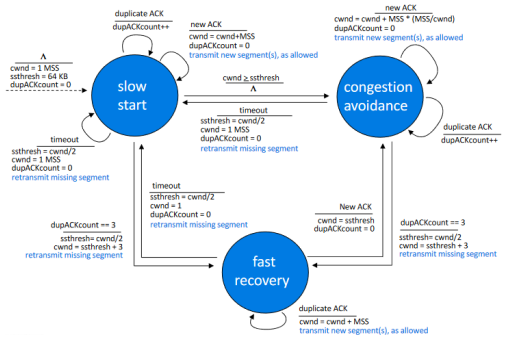

Figura 3.7: Macchina a stati di `TCP`

Problemi di equità (*fairness*) in TCP

In quanto le applicazione multimediali usano di rado TCP per la trasmissione (ma usano UDP), queste non sono soggette ai controlli di congestione, quindi le connessioni TCP sono penalizzate. Questo avviene in quanto le connessioni UDP trasmettono a velocità costante indipendentemente da fattori esterni.
Se invece abbiamo due *host* che usano TCP e uno apre (ad es.) 9 connessioni mentre l'altro ne apre 1 allora il primo avrà un *throughput* di $\frac{R}{2}$ mentre il secondo avrà un *throughput* di $\frac{R}{10}$.

3.7.3 Altri algoritmi più recenti per il controllo di congestione

Negli algoritmi visti fino ad ora il processo di "rallentamento" della trasmissione avveniva solo in caso di perdita di pacchetti, questo però permette di regolare la banda solo quando è troppo tardi, in quanto deve avvenire una perdita prima che il mittente si renda conto che c'è una congestione. Per ovviare a questo problema sono stati introdotti nuovi algoritmi che permettono di regolare la banda. Questi algoritmi sono:

- CUBIC
- BBR
- QUIC

CUBIC

L'algoritmo CUBIC fa variare la lunghezza della finestra di congestione secondo una funzione cubica nel tempo, questo ne migliora la scalabilità e la stabilità. Questo algoritmo è stato introdotto nel kernel di Linux a partire dalla versione 2.6.19, mentre in Windows è stato introdotto a partire dal 2017.

Principi del funzionamento Per un migliore utilizzo e stabilità della rete CUBIC usa sia sia la parte concava che quella convessa della funzione cubica per regolare la finestra di congestione.

$$\texttt{CWND}_{cubic}(t) = C(t-K)^3 + \texttt{CWND}_{max}$$

Dove C è una costante, $K = \sqrt[3]{\frac{\texttt{CWND}_{max}(1-\beta)}{C}}$ e $\texttt{CWND}_{max}$ è la dimensione massima della finestra di congestione. Per lo standard RFC 8312 $C = 0.4$ e $\beta = 0.7$, ma dopo che è stata rilevata una congestione allora $\beta = 0.5$. Inoltre questo algoritmo è "TCP-*friendly*" ovvero non penalizza i flussi TCP legacy che condividono la stessa rete.

BBR - *Bottleneck Bandwidth and Round-trip propagation time*

L'algoritmo BBR è un algoritmo di controllo di congestione che cerca di massimizzare il *throughput* e minimizzare il ritardo. Questo algoritmo è stato introdotto da Google nel 2016 e si basa non sul rilevamento di perdite ma su due parametri:

- **Bottleneck Bandwidth** La banda disponibile sul *bottleneck*.
- **Round-trip propagation time** Il tempo di propagazione del pacchetto.

Il funzionamento a grandi linee prevede la trasmissione di pacchetti ad una velocità che non *dovrebbe* saturare la rete. Questo infatti è progettato per ridurre la finestra di congestione prima che si verifichi una perdita, in questo modo si dovrebbe riuscire a limitare ritrasmissioni inutili. Un vantaggio di BBR è quello che solo il *server* lo deve implementare e non anche il *client*. Il concetto usato è quello di *pacing* ovvero inserisco nuovi pacchetti nella CWND solo quando il nodo più lento della rete è pronto a riceverli.

Migliore produttività Secondo *Google* BBR con un *link* a 10 Gbps che invia dati lungo un percorso con RTT di 100ms con tasso di perdita dell'1% riesce a raggiungere un *throughput* di $3,3$Mbit/s con CUBIC e di 9100Mbit/s con BBR. Questo è ideale nel caso di connessioni HTTP/2 che sfruttano una singola connessione per trasmettere dati.

Latenza inferiore BBR riesce a mantenere una latenza inferiore rispetto a CUBIC in quanto riesce a mantenere la banda costante e non satura la rete. Vari studi (sempre di *Google*) hanno dimostrato che su un collegamento di 10 Mbps con RTT di 40 ms ed un *bottleneck* di 1000 pacchetti la latenza di BBR è di soli 43 ms contro i 1090 ms di CUBIC.

QUIC - *Quick UDP Internet Connections*

QUIC è un protocollo di trasporto sviluppato da Google nel 2012 e si prefissa il raggiungimento di due obbiettivi:

- Evitare fenomeni di *head-of-line blocking*
- Ridurre la latenza di TCP

QUIC può essere implementato a livello applicazione, oltre che a livello di *kernel*. Lo *use case* di questo dovrebbe essere quello delle connessioni HTTP/3. Il principio di funzionamento di questo è che i pacchetti vengono trasmessi tramite una connessione UDP e non TCP, questo permette di evitare i problemi di *head-of-line blocking* in quanto se un pacchetto viene perso allora non si bloccano tutti i pacchetti successivi. Inoltre QUIC permette di ridurre l'*overhead* di connessione in quando incorpora in se stesso lo scambio delle chiavi (o *handshake*) di TLS.

3.7.4 Conclusioni

Meglio dunque TCP o UDP?

La scelta tra TCP e UDP dipende da cosa si vuole fare, se si vuole trasmettere dati in modo affidabile e si vuole evitare di saturare la rete allora si deve usare TCP, se invece si vuole trasmettere dati in modo veloce e non si vuole preoccuparsi di perdite di pacchetti allora si deve usare UDP. Questa scelta però non è così libera come sembra, in quanto se si vuole usare il protocollo QUIC necessitiamo di connessione UDP ma molta della nostra infrastruttura blocca le connessioni di questo tipo in quanto non avviene un controllo di congestione e quindi si rischia di saturare la rete. Google ha provato a mostrare come QUIC sia migliore di TCP cercando di "sbloccare" la rete per questo tipo di connessioni, detto ciò i prodotti della serie *chromium* aprono in contemporanea una connessione TCP e una connessione UDP e scelgono quella che ha il *throughput* migliore.

Cambio di rete

Con le connessioni TCP le *socket* vengono identificate dalla quadrupla: (IP M.,IP D., Porta M., Porta D.), se si cambia rete allora si cambia anche l'indirizzo IP e quindi la connessione TCP viene persa. Con QUIC invece la connessione viene mantenuta in quanto la *socket* è identificata da un ID e non dall'indirizzo IP.

Capitolo 4

Il livello di rete

4.1 Visione d'insieme

Obbiettivo del livello di rete L'obbiettivo principale del livello di rete è quello di permettere la comunicazione tramite reti diverse attraverso apparecchi detti *router* i quali hanno il compito di inoltrare le informazioni verso la destinazione.

Funzioni principali Esistono due funzioni principali del livello di rete:

Inoltro *forwarding* Questa è una operazione a livello locale che consiste nel prendere un pacchetto in ingresso e inoltrarlo verso l'uscita corretta.

Instradamento *routing* Questa è una operazione a livello globale che consiste nel determinare il percorso migliore per inoltrare un pacchetto verso la destinazione. Per questa operazione si utilizzano degli algoritmi di *routing*.

Queste due funzioni sono legate tra loro, ma possono essere isolate. Infatti convenzionalmente distinguiamo con *control plane* la parte del livello di rete che si occupa dell'istradamento e con *data plane* la parte che si occupa dell'inoltro. Questa distinzione è utile per capire come funzionano i router.

Data Pane Il *data plane* ha funzione a livello locale ad ogni *router*, questo è il livello che determina come inoltrare un *datagram* fornendo la funzione di *forwarding*.

Control Plane Il *control plane* ha funzione a livello globale, questo è il livello che determina dove inoltrare un *datagram* fornendo la funzione di *routing*.

4.2 Come è fatto un router

Visto a livello "alto" un router è composto da tre livelli principali:

Terminazione di linea Questo è il livello più basso del router, è composto da un'interfaccia di rete che si occupa di ricevere i pacchetti e di inviarli al livello successivo.

Protocollo di livello *data link* Questo livello si occupa di ricevere i pacchetti dal livello precedente e di inviarli al livello successivo. Inoltre si occupa di fare il controllo degli errori e di gestire il flusso.

Inoltro e *buffer* Questo è il livello che si occupa di inoltrare i pacchetti verso la destinazione. Inoltre si occupa di fare il *buffering* dei pacchetti in caso di congestione.

4.2.1 Sistemi di commutazione

I sistemi di commutazione trasferiscono i pacchetti dalle porte di ingresso all'uscita appropriata. Definiamo come **tasso di comunicazione** la frequenza alla quale i pacchetti vengono portati dall'ingresso all'uscita (spesso è un multiplo della velocità di comunicazione)

Commutazione a memoria

Questo è il metodo più semplice, i pacchetti vengono memorizzati in un buffer comune e poi inoltrati verso l'uscita appropriata. Questo metodo è molto semplice ma ha il problema che la velocità di inoltro è limitata dalla velocità di accesso alla memoria.

Commutazione a bus

Questo metodo consiste nel collegare le porte di ingresso e di uscita tramite un bus, sempre comune a tutte le porte. Questo metodo risulta lento in quanto non possono essere trasferiti più pacchetti contemporaneamente anche se le porte di ingresso e di uscita sono diverse. Il `Cisco 5600` è un esempio di router che utilizza questo metodo e riesce a trasferire fino a 32 Gbit/s.

Commutazione a matrice

Questo metodo consiste nel collegare le porte di ingresso e di uscita tramite una matrice di commutazione. Questo metodo è molto veloce in quanto permette di trasferire più pacchetti contemporaneamente in quanto se le porte sono differenti allora basta attivare i vari collegamenti della matrice. Questo metodo è molto veloce ed ispirato ai primi commutatori telefonici. Il `Cisco 12000` è un esempio di router che utilizza questo metodo e riesce a trasferire fino a 60 Gbit/s.

4.2.2 Accodamenti

Gli accodamenti sono utilizzati per evitare la perdita di pacchetti in caso di congestione dell'apparecchio di rete. Questo è un problema molto comune in quanto i router sono dispositivi molto veloci e le porte di uscita sono molto più lente. Per evitare la perdita di pacchetti si utilizzano delle code che permettono di memorizzare i pacchetti in attesa di essere inoltrati. Le code possono essere formate in ingresso, quando una stessa porta di uscita è condivisa da più porte di ingresso e quindi una porta di uscita può essere congestionata. Le code possono essere formate in uscita, quando una porta di uscita ha un *link* più lento rispetto alla velocità di inoltro dei pacchetti tramite le porte di ingresso o il commutatore di pacchetto.

Quanta memoria serve per i *buffer* Secondo `RFC 3439` la quantità di memoria necessaria per i buffer è data dalla formula:

$$M = \frac{RTT \cdot C}{\sqrt{N}}$$

Dove:

M è la memoria necessaria per il buffer

RTT è il tempo di round trip

C è la capacità del collegamento

N è il numero di connessioni

Meccanismi di *scheduling* I meccanismi di *scheduling* sono utilizzati per decidere quale pacchetto inoltrare quando si ha la possibilità di inoltrare più pacchetti. Il meccanismo più semplice è il *First In First Out* (`FIFO`) che inoltra i pacchetti in ordine di arrivo. Inoltre viene applicata una politica di scarto dei pacchetti in caso di buffer pieno. Questa politica può essere:

Drop Tail Questa politica scarta i pacchetti in arrivo quando il buffer è pieno.

Random Early Detection Questa politica scarta i pacchetti in arrivo in modo casuale quando il buffer è pieno.

Priority Drop Questa politica scarta i pacchetti in arrivo in base alla priorità.

La politica di scarto dei pacchetti dipende dall'implementazione del router.

4.3 Il protocollo IP

4.3.1 Il formato del *datagram* IP (IPv4)

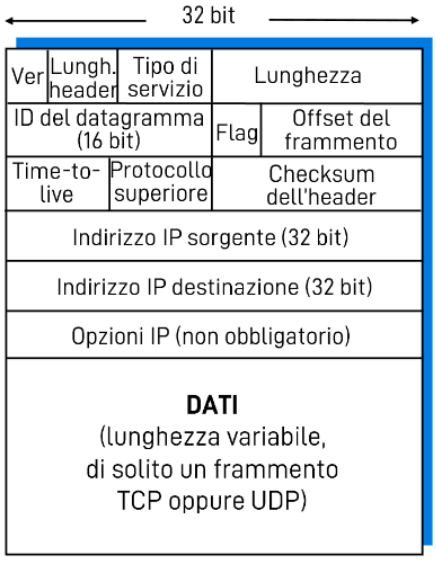

Figura 4.1: Il formato del *datagram* IP (IPv4)

Il formato del *datagram* IP è composto da 20 byte di intestazione e da un campo dati. Il campo dati può contenere fino a 65.535 byte. Di seguito si riportano i vari campi dell'intestazione:

VER (4 bit) Questo campo contiene la versione del protocollo IP utilizzato.

Lunghezza *header* (4 bit) Questo campo contiene la lunghezza dell'intestazione in parole da 32 bit. (=5 se non ci sono opzioni)

Tipo di servizio - ToS (8 bit) Questo campo contiene informazioni sul tipo di servizio richiesto.

Lunghezza totale (16 bit) Questo campo contiene la lunghezza totale del *datagram* in byte.

Identificativo (16 bit) Questo campo contiene un numero univoco per il *datagram*.

Flag (3 bit) Questo campo contiene i flag per il frammento. Il primo bit è il bit di *Don't Fragment*, il secondo bit è il bit di *More Fragment* e il terzo bit è il bit di *Fragment Offset*.

Offset (13 bit) Questo campo contiene l'offset del frammento. (Espresso in multipli di 8 byte)

***Time To Live* - TTL** (8 bit) Questo campo contiene il numero di *hop* massimo che il *datagram* può fare.

Protocollo (8 bit) Questo campo contiene il protocollo di trasporto che si trova nel campo dati.

Checksum (16 bit) Questo campo contiene il checksum dell'intestazione.

Indirizzo IP sorgente (32 bit) Questo campo contiene l'indirizzo IP sorgente.

Indirizzo IP destinazione (32 bit) Questo campo contiene l'indirizzo IP destinazione.

Opzioni (variabile) Questo campo contiene le opzioni del *datagram*.

Padding (variabile) Questo campo contiene il padding per allineare l'intestazione a multipli di 32 bit.

4.3.2 MTU e Frammentazione

La MTU (*Maximum Transmission Unit*) è la dimensione massima di un pacchetto che può essere trasmesso su un collegamento, ogni *hardware* specifica il proprio MTU. Se un *datagram* è più grande della MTU allora il *datagram* viene frammentato in pacchetti più piccoli. Questo processo è chiamato *frammentazione*. I pacchetti frammentati vengono poi ricomposti alla destinazione.

Valori Standard MTU Per alcuni tipi di collegamenti sono stati definiti dei valori standard di MTU. Ad esempio per le reti Ethernet la MTU è di 1500 byte, per le reti WLAN 802.11 la MTU è di 2304 byte,.... In un collegamento tra due *host* possono essere presenti due valori di MTU diversi

EsempioSupponendo che per raggiungere un host B si debba passare prima da una rete con 1500 byte di MTU e poi da una rete con 1000, tutto ciò tramite un router R. Allora quando il *router* R riceve il datagram di 1500 byte lo frammenta in due pacchetti di 1000 byte e 500 byte. I pacchetti vengono quindi inoltrati alla destinazione. Quando l'host B riceve i pacchetti li ricomponi e li passa al livello di trasporto.

4.3.3 Indirizzamento e NAT

Indirizzi IP

Un indirizzo IP è composto da 32 bit ed è associato ad un'interfaccia di rete. Una interfaccia è una connessione tramite mezzo fisico o logico, solitamente ogni *host* ha una o più interfacce di rete.

Caratteristiche indirizzi IP Gli host e i router devono usare le stesse convenzioni per gli indirizzi IP, inoltre ogni indirizzo IP deve essere unico e raggiungibile da un qualsiasi punto di internet. Quando si invia un pacchetto IP si invia l'indirizzo IP sorgente e l'indirizzo IP destinazione. I *router* sono apparati di rete che quando ricevono un pacchetto IP decidono dove inoltrarlo in base all'indirizzo IP di destinazione.

Norazione indirizzo IP Gli indirizzi IP sono composti da 4 gruppi di 8 bit e sono scritti in notazione decimale a punti. Ogni gruppo di 8 bit è espresso in decimale e separato da un punto. Gli indirizzi disponibili vanno dà: 0.0.0.0 a 255.255.255.255

Gerarchie indirizzi IP Gli indirizzi IP sono organizzati in una struttura gerarchica. Inoltre solitamente sono divisi in due parti:

Parte di rete Questa parte identifica la rete a cui appartiene l'indirizzo IP.

Parte di *host* Questa parte identifica l'*host* all'interno della rete.

Classi di indirizzi IP Gli indirizzi IP sono divisi in classi sulla base dei primi bit dell'indirizzo e dalla lunghezza del prefisso:

Classe A Gli indirizzi di classe A hanno il primo bit a 0 e sono composti da 8 bit di rete e 24 bit di *host*. Gli indirizzi vanno da 0.0.0.0 a 127.255.255.255 e sono riservati per le reti molto grandi.

Classe B Gli indirizzi di classe B hanno i primi due bit a 10 e sono composti da 16 bit di rete e 16 bit di *host*. Gli indirizzi vanno da 128.0.0.0 a 191.255.255.255 e sono riservati per le reti di medie dimensioni.

Classe C Gli indirizzi di classe C hanno i primi tre bit a 110 e sono composti da 24 bit di rete e 8 bit di *host*. Gli indirizzi vanno da 192.0.0.0 a 223.255.255.255 e sono riservati per le reti di piccole dimensioni.

Classe D Gli indirizzi di classe D hanno i primi quattro bit a 1110 e sono riservati per i *multicast*.

Classe E Gli indirizzi di classe E hanno i primi quattro bit a 1111 e sono riservati per usi futuri.

Assegnazione indirizzi IP

Gli indirizzi IP sono assegnati dalla ICANN (*Internet Corporation for Assigned Names and Numbers*) che riserva una intera classe ai ISP (*Internet Service Provider*) e poi questi assegnano gli indirizzi ai propri clienti. Gli indirizzi IP sono assegnati in modo gerarchico e quindi un ISP può assegnare un intero blocco di indirizzi ad un altro ISP e questo può assegnare un blocco di indirizzi ad un altro ISP e così via.

4.3.4 Indirizzamento *classless*

In quanto ci si è accorti che con la suddivisione degli indirizzi in classi si stava sprecando molti indirizzi, si è deciso di passare ad un indirizzamento *classless*. Questo tipo di indirizzamento permette di avere una suddivisione più flessibile degli indirizzi, in quanto possiamo richiedere al nostro ISP solo un *range* di indirizzi e non una classe intera, ad esempio l'ISP può usare un prefisso di 26 bit per identificare la rete nel mondo e successivamente usare i restanti 6 bit per identificare tutti gli *host* della rete. In questa situazione se un ISP ha acquistato una rete di classe C e ha bisogno di dividere la rete in quattro clienti allora può dividere la rete in 4 sotto-reti e assegnare un prefisso di 26 bit (24 per la classe C e 2 per le sotto-reti) e i restanti 6 bit per gli *host* di ogni sotto-rete.

Maschere di rete

In quanto ora non si ha più una suddivisione fissa degli indirizzi, si è deciso di introdurre le *maschere di rete*. Queste maschere sono composte da 32 bit e sono composte da una parte di 1 quelli che identificano il prefisso di rete e da una parte di 0 che identificano gli *host* della rete. Ad esempio la maschera di rete per una rete di classe C è 11111111.11111111.11111111.00000000.

Peché si usa? La maschera di rete viene usata per identificare la rete di appartenenza di un indirizzo IP. Per fare ciò si fa un'operazione di AND tra l'indirizzo IP di destinazione e la maschera di rete. Se il risultato è uguale all'indirizzo di rete allora l'indirizzo IP appartiene alla rete. Quindi se il prefisso di rete è 128.10.0.0 ovvero:

$$10000000.00001010.00000000.00000000$$

e la maschera di rete è 255.255.0.0 ovvero:

$$11111111.11111111.00000000.00000000$$

e l'indirizzo IP di destinazione di un determinato pacchetto è: 128.10.2.3 ovvero:

$$10000000.00001010.00000010.00000011$$

allora facendo l'operazione di AND tra l'indirizzo IP e la maschera di rete si ottiene:

$$\begin{array}{l} 10000000.00001010.00000010.00000011 \\ \underline{11111111.11111111.00000000.00000000} \\ 10000000.00001010.00000000.00000000 = 128.10.0.0 \end{array}$$

Quindi l'indirizzo IP appartiene alla rete e il pacchetto viene inoltrato a quella determinata porta di uscita.

Notazioni CIDR

CIDR o *Classless Inter-Domain Routing* è un metodo per rappresentare le maschere di rete. Questo metodo consiste nel rappresentare la maschera di rete con un prefisso di bit. Ad esempio la maschera di rete /24 è uguale alla maschera di rete 11111111.11111111.11111111.00000000, ovvero una maschera di rete di classe C, quindi i primi 3 gruppi di bit sono appartenenti all'*network-id* e l'ultimo gruppo di bit è appartenente all'*host-id*. La notazione prevede inoltre che gli indirizzi IP siano rappresentati nel seguente modo: ddd.ddd.ddd.ddd/m dove ogni singolo d rappresenta un gruppo di bit e m rappresenta il numero di bit del prefisso di rete.

Esempio L'indirizzo 193.168.32.199/26 ha un prefisso di rete di 26 bit, quindi il *network-id* è:

11000001.1010100.000000010.11

e l'*host-id* è

00111

Esistono dunque $2^6 = 64$ indirizzi IP nella rete.

Inoltro con CIDR L'inoltro con CIDR è molto semplice e non differisce dall'inoltro con le classi. Infatti si fa l'operazione di AND tra l'indirizzo IP di destinazione e la maschera di rete e si confronta il risultato con l'indirizzo di rete. Se il risultato è uguale all'indirizzo di rete allora l'indirizzo IP appartiene alla rete e il pacchetto viene inoltrato alla porta di uscita corretta. Nella situazione in cui ci siano più reti che corrispondono al risultato dell'operazione di AND allora si sceglie la rete con il prefisso più lungo in quanto si presume che questa sia la rete più specifica e quindi la più breve.

Aggregazione dei percorsiL'aggregazione dei percorsi è una tecnica che permette di ridurre il numero di percorsi che un router deve memorizzare. Questa tecnica consiste nel raggruppare più reti in un'unica rete più grande. Questa tecnica è molto utile in quanto permette di ridurre il numero di percorsi che un router deve memorizzare e quindi di velocizzare l'inoltro dei pacchetti. Esempio se un ISP controlla tutte le reti 200.23.16.0/23, 200.23.18.0/23. . . 200.23.30.0/23 allora può raggruppare tutte queste reti in un'unica rete 200.23.16.0/20