

# 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.

# Indice

<b>1</b>	<b>Introduzione</b>	<b>4</b>
1.1	Cos'è internet? . . . . .	4
1.2	Ai confini della rete . . . . .	4
1.2.1	Reti d'accesso e mezzi fisici . . . . .	4
1.2.2	Accesso residenziale: punto-punto . . . . .	4
1.2.3	FTTH - Fiber To The Home . . . . .	5
1.2.4	Accesso aziendale: reti locali (LAN) . . . . .	5
1.2.5	Accesso wireless . . . . .	5
1.3	Al nucleo della rete . . . . .	5
<b>2</b>	<b>Il Livello Applicazione</b>	<b>6</b>
2.1	principi delle applicazioni di rete . . . . .	6
2.1.1	Architetture di rete . . . . .	6
2.1.2	Struttura delle applicazioni di rete . . . . .	7
2.1.3	Indirizzamento . . . . .	7
2.1.4	Protocolli a livello applicazione . . . . .	7
2.1.5	Servizi di trasporto . . . . .	7
2.2	Web e HTTP . . . . .	9
2.2.1	Terminologia . . . . .	9
2.2.2	Introduzione a HTTP . . . . .	9
2.2.3	Cookies . . . . .	10
2.2.4	Cache web . . . . .	11
2.2.5	HTTP/1.0 e HTTP/1.1 . . . . .	11
2.2.6	HTTP/2.0 . . . . .	12
2.2.7	Transport Layer Security (TLS) . . . . .	12
2.2.8	HTTPS . . . . .	13
2.3	FTP - File Transfer Protocol . . . . .	13
2.3.1	Connessione di controllo . . . . .	13
2.3.2	Comandi & Risposte FTP . . . . .	13
2.4	Posta Elettronica . . . . .	13
2.4.1	SMTP . . . . .	14
2.4.2	POP3 . . . . .	14
2.4.3	IMAP . . . . .	15
2.5	DNS . . . . .	15
2.5.1	Servizi DNS . . . . .	15
2.5.2	Struttura del DNS . . . . .	16
2.5.3	Resource Record RR . . . . .	16
2.5.4	Inserire un record . . . . .	16
2.6	Condivisione di file P2P . . . . .	16
2.6.1	Distribuzione di File . . . . .	17
2.6.2	Ricerca delle informazioni . . . . .	18
2.7	Cloud Computing . . . . .	18

---

2.7.1	<i>Content Delivery Network</i> - CDN . . . . .	19
<b>3</b>	<b>Il Livello Applicazione</b> . . . . .	<b>20</b>
3.1	Servizi a livello di trasporto . . . . .	20
3.2	Multiplexing e De-multiplexing . . . . .	20
3.2.1	De-multiplexing . . . . .	21
3.2.2	Porte TCP-UDP . . . . .	22
3.3	Trasporto senza connessione: UDP . . . . .	22
3.3.1	Header . . . . .	22
3.4	trasferimento dati affidabile . . . . .	23
3.4.1	ARQ . . . . .	23
3.4.2	<i>Stop-and-Wait</i> . . . . .	23
3.4.3	Protocolli con <i>pipelining</i> . . . . .	23

# Capitolo 1

## Introduzione

### 1.1 Cos'è internet?

#### Internet

**Host** Una rete di milioni di dispositivi collegati detti "Host"

**Applicazioni di rete** un insieme di applicazioni di rete

**Collegamenti** una rete di collegamenti fisici (rame, fibra ottica, onde elettromagnetiche, ecc...). Frequenza di trasmissione è uguale a ampiezza di banda

**Router** Instrada i pacchetti verso la destinazione finale

**ISP** Internet Service Provider

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

#### Standard

**RFC** Request for Comments

**IETF** Internet 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**

**Velocità** 10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps

**Moderna Configurazione:** sistemi terminali collegati a switch, switch collegato a router

### 1.2.5 Accesso wireless

**Wireless LANs**

**Wi-Fi** 2.4 GHz, 5 GHz, 802.11b/g/n/ac

**Cellular networks** 3G, 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 scegliere 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	Sì	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 sessione 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, specificamente sulla latenza percepita. Obiettivo 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.

#### 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 usare 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 stream** L'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**.<sup>1</sup>

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

---

<sup>1</sup>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 utente** è 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/risposta** I comandi 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

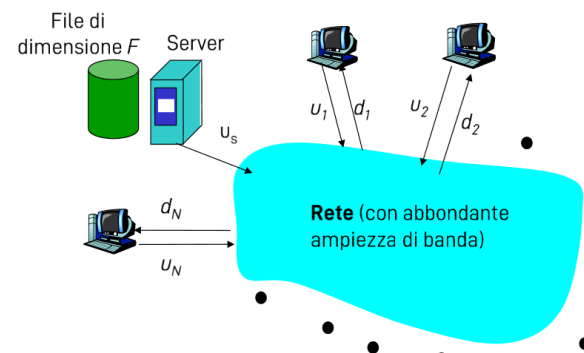


Figura 2.1: Distribuzione di file in modalità P2P

**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 più 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 costante per l'invio della prima copia si riduce all'aumentare del numero di *peer*.

#### BitTorrent

**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* contenente 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 Applicazione

### 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: Trasporto 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.

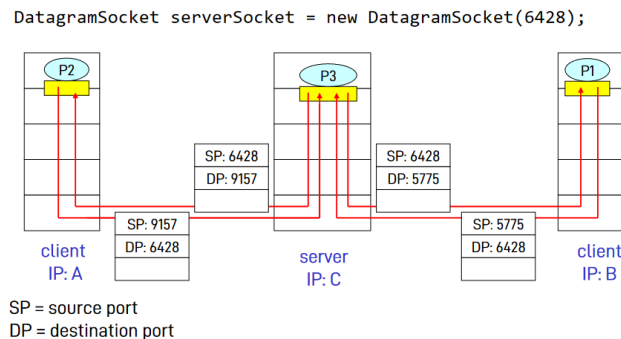


Figura 3.1: De-multiplexing senza connessione

#### 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).<sup>1</sup>

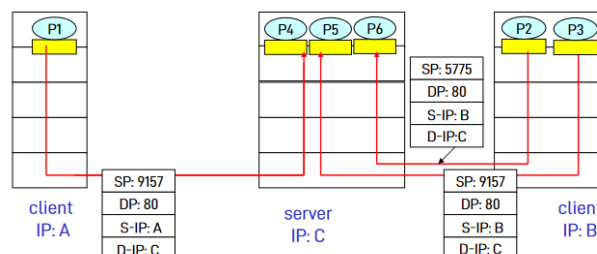


Figura 3.2: De-multiplexing orientato alla connessione

<sup>1</sup>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.

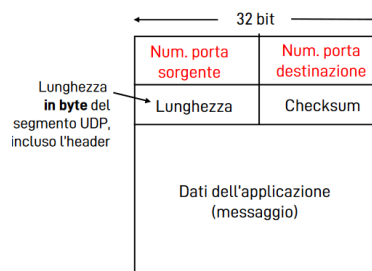
Inoltre è importante dire che le porte di sorgente e di destinazione non sono le stesse in quanto la porta di sorgente è una porta dinamica assegnata dal sistema operativo.

## 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



Struttura del segmento UDP

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  $R = 1G\text{-bit/s}$ ,  $15ms$  di ritardo di propagazione, lunghezza del messaggio  $L = 8000\text{bit}$ , allora il tempo di trasmissione sarà:  $T_{trans} = \frac{L}{R} = \frac{8000}{10^9} = 8\mu s$ . Mentre il *throughput* percepito a livello applicativo sarà:  $\frac{L}{T_{trans} + RTT} = \frac{8000}{8\mu s + 30ms} = 33Kbps$ .<sup>2</sup> 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*.

<sup>2</sup>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  $W_T$ .

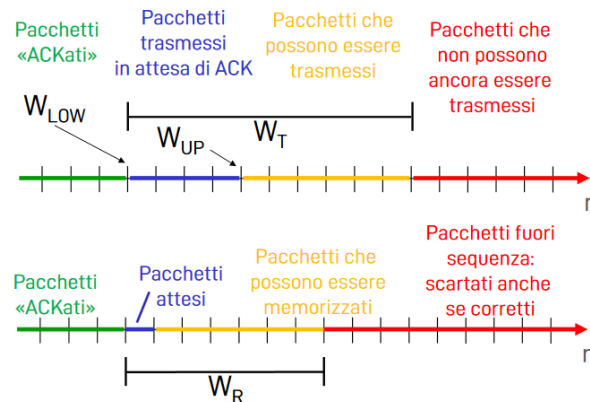


Figura 3.4: Finestre di trasmissione e di ricezione