RETI DI CALCOLATORI

Corso A

Autore
Giuseppe Acocella
2025/26
https://github.com/Peenguino

Ultima Compilazione - October 11, 2025

Contents

1 Introduzione alle Reti

Come facciamo a costruire una rete di computer che sia scalabile e che supporti **svariati tipi di applicazioni** come streaming, messaggistica e videochiamate? Internet ci permette di farlo, e tecnicamente è un insieme di miliardi di host (servers, laptops, smartphones) connessi tra loro.

Internet - Vista dei Componenti Questa connessione è permessa da specifici componenti:

1. **Link di Comunicazione**: Fibra ottica, rame, onde radio, microonde.
2. **Dispositivi di Interconnessione**: I più comuni sono:
   (a) **Switch**: Dispositivo che collega tra loro **più host**.
   (b) **Router**: Dispositivo che collega tra loro **più reti**.
3. **Reti**: Insieme di host, dispositivi di interconnessione e link gestiti da una stessa organizzazione.

Internet - Vista dei Servizi Da un punto di vista di servizi offerti invece Internet:

1. E' un **infrastruttura** che offre servizi alle **applicazioni distribuite**.
2. Offre un **interfaccia di programmazione** (socket) che permette alle applicazioni, distribuite su **host diversi**, di scambiarsi informazioni.

Internet - Vista delle Entità Software

1. **Applicazioni**: Elaborano e si scambiano informazioni.
2. **Protocolli**: Regolano la trasmissione di queste informazioni, definendo quindi il formato e l'ordine dei messaggi scambiati tra due o più entità in comunicazione. Due esempi noti di protocolli internet sono:
   (a) **Trasmission Control Protocol (TCP)**
   (b) **Internet Protocol (IP)**

Standardizzazione di Protocolli I protocolli di Internet vengono standardizzatida diversi enti, come:

1. Internet Engineering Task Force (IETF). Si basano su una procedura detta RFC (Request for Comments) che seguendo passi rigorosi definisce gradualmente nuovi standard per Internet.
2. IEEE 802 LAN Standards.
3. World Wide Web Consortium (W3C): Comunità internazionale che sviluppa standard aperti per favorire lo sviluppo del Web.

1.1 Tipi di Rete per Dimensione

Descrizione dei tipi di rete per estensione fisica:

1. **Personal Area Network** (PAN): Connessione personale, ad esempio una Bluetooth.
2. **Local Area Network** (LAN): Reti circoscritte ad un area limitata, utilizzano cavi o mezzi wireless per connettersi e sono solitamente proprietarie.
3. **Metropolitan Area Network** (MAN): Rete di calcolatori che non supera mai l'estensione fisica indicativa di una "città".
4. **Wide Area Network** (WAN): Reti che cercano di interconnettere host separati da distanze geografiche tramite cavi in fibra ottica o ponti radio.

1.2 Gestione di Internet come Rete di Reti

Ogni rete locale si espone alle altre reti grazie agli **Internet Service Provider** (ISP). Come possiamo immaginare però non connettiamo ogni punto terminale ad ogni altro punto terminale perchè provocherebbe un alta complessità da un punto di vista di collegamenti. Di conseguenza saranno presenti dei **Global ISP** che si occupano della gestione del route della comunicazione. Non esiste un solo **Global ISP** ma **molteplici** e questi si connettono tra loro con dei **Peering Point** su punti d'incontro tra le reti detti **Internet Exchange Point** (IXP).

Content Provider Network Idealmente si segue lo schema descritto sopra, ma a volte grandi aziende (Amazon, Google) si posizionano trasversalmente a questa organizzazione descritta, in modo tale da gestire la propria rete per avvicinare i propri servizi all'utente finale.

1.3 Reti e Mezzi d'Accesso

Descriviamo questi elementi caratterizzanti dell'accesso ad una rete:

1. **Rete d'Accesso**: Rete che connette fisicamente un sistema terminale al **edge router**, ossia il primo router da sorgente a destinazione.
2. **Mezzi d'Accesso**: Come passiamo fisicamente dal calcolatore al edge router:
   (a) **Cablati (Vincolati)**: Doppino, coassiale, fibra.
   (b) **Wireless (Non Vincolati)**: Onde Radio, microonde, infrarossi.

1.3.1 Schema dell'Accesso DSL

Uno dei primi schemi d'accesso è stato quello Digital Subscriber Line (DSL), dove la compagnia telefonica rappresentava l'ISP. Risultava quindi necessario splittare la linea telefonica già esistendo, suddividendo il traffico in bande di frequenze diverse:

1. **Canale Telefonico**: $0 - 4kHz$
2. **Upstream**: $4 - 50kHz$
3. **Downstream**: $50kHz - 1MHz$

In questo modo sarà necessario un multiplexer di accesso alla linea digitale (DSLAM) che permetta di separare i due tipi di comunicazione. Esistono **versioni diverse** di **questo schema**, riferiremo infatti allo schema **xDSL**, proprio perchè è possibile migliorare le caratteristiche della connessione ad esempio basandosi su una connessione in **fibra** tra DSLAM ed ISP, ad esempio:

1. **Fibre To The Cabinet (FTTC)**: Standard VDSL e VDSL2.
2. **Fibre To The Derivation Point**.
3. **Fibre To The House (FTTH)**: Caratterizzato dai seguenti elementi:
   (a) Fibra ottica fino all'interno delle abitazioni.
   (b) Fibra uscente dalla centrale locale con terminatore ottico di linea **OLT** viene condivisa da diverse abitazioni.
   (c) Il terminatore ottico di rete **ONT** invece è connesso ad uno splitter tramite fibra dedicata, questo permette la gestione fino al centinaio di abitazioni.
   (d) L'**OLT** si connette al router dell'ISP e tramite questo ad Internet.

1.3.2 Tipi d'Accesso - Cavo, Aziendale, Domestico, Mobile e Satellitare

Descriviamo queste tipologie di accesso alla rete:

1. **Accesso Via Cavo**: Il segnale ottico viene convertito in segnale elettromagnetico e inviato sulle linee di cavi coassiali per la distribuzione del segnale alle varie abitazioni. La velocità potrebbe degradare a causa della distanza o della congestione del canale condiviso tra più abitazioni.
2. **Accesso Aziendale**: Nelle aziende i sistemi sono collegati al router di frontiera con una LAN.
3. **Accesso Domestico**: Utilizzate tecnologie Ethernet o WiFi per creare LAN domestiche.
4. **Accesso Mobile**: Infrastruttura radio mobile definita dagli ISP, può subire degrado di prestazioni a causa di distanza oppure ostacoli.
5. **Accesso Satellitare**: Accesso permesso da trasmettitori terrestri che si connettono a satelliti geostazionari (GEO) oppure satelliti a bassa quota (LEO). Questo sistema risulta essere il più duttile e non richiede complesse installazioni, ma potrebbe risultare inferiore da alcuni punti di vista come la latenza più alta.

1.4 Nucleo della Rete - Commutazione Circuito vs Pacchetto

Come determiniamo il percorso effettuato da un messaggio? Come trasferiamo dalla porta di uscita a quella di ingresso di due calcolatori? A queste domande risponde il tipo di commutazione.

Commutazione di Circuito Si instaura un cammino tra gli host in comunicazione e tutte le risorse del canale sono dedicate a loro. Questo causa un utilizzo esclusivo del canale e conseguente spreco di risorse. Per poter permettere molteplici comunicazioni utilizzando questo tipo di commutazione esistono due politiche:

1. **Frequency Division Multiplexing (FDM)**: Divido il canale in bande di frequenze e le dedico alle diverse comunicazioni.
2. **Time Division Multiplexing (TDM)**: Dedico tutto il canale alla coppia di host ma per una finestra di tempo limitata.

Commutazione di Pacchetto Il flusso dati di una comunicazione viene suddiviso in **pacchetti**, e gli viene assegnato un **header** che mantenga dei metadati riguardanti il flusso dati originale. Questo tipo di instradamento segue queste fasi:

1. **Suddivisione** in **pacchetti** ed **assegnamento header**.
2. **Instradamento** del singolo **pacchetto indipendentemente** dagli altri. Pacchetti di flussi dati diversi possono condividere i canali di comunicazione.
3. Fase di **Store and Forward**, ossia prima che il commutatore (router) consegni i pacchetti dovrà prima aspettare che siano del tutto arrivati. Il tipo di Multiplexing quindi è statistico e non prefissato. Seguendo questa politica potremmo incorrere anche in problemi di **perdita di pacchetti** in base al tipo di gestione del buffer di ciascun commutatore. Non abbiamo quindi alcuna garanzia sulle prestazioni utilizzando questa tecnica.

Confronto Caratteristiche Commutazione Circuito/Pacchetto Elenchiamo pro e contro di ciascuna tecnica:

1. **Commutazione Circuito - Pro**:
   (a) Prestazioni garantite.
   (b) Tecnologie di switching efficienti.
   (c) Tariffazioni per ISP semplici da definire.
2. **Commutazione Circuito - Contro**:
   (a) Segnalazione e configurazione delle tabelle di switching.
   (b) Poca ottimizzazione del uso di risorse.

3. **Commutazione Pacchetto - Pro**:
   (a) Utilizzo ottimale delle risorse.
   (b) Non richiede fase di setup e tabelle di switching predefinite.
4. **Commutazione Pacchetto - Contro**:
   (a) Tecnologie di inoltro non efficienti.
   (b) Ritardi variabili.
   (c) Rischio di congestione.

1.5 Metriche - Prestazioni della Rete (Latenza, Throughput, Prod. Rate/Ritardo)

Descriviamo due metriche fondamentali delle prestazioni della rete:

1. **Latenza**: Tempo richiesto dal primo bit partito dalla sorgente fino all'arrivo a destinazione. Questa definizione è ideale, infatti dipende anche dal tipo di commutazione, quella di pacchetto con Store & Bound introduce altre difficoltà. Nello specifico le **cause specifiche** di **ritardo** nella **commutazione di pacchetto** sono:
   (a) **Ritardo di Elaborazione del Nodo**
      i. Controllo errori sui bit.
      ii. Determinazione porta di uscita.
   (b) **Ritardo di Accodamento**
      Componente di ritardo più complessa da stabilire, si basa su una **condizione di stabilità** ossia un rapporto $\rho = \frac{L\,a}{R}$ dove a è velocità media di arrivo, R velocità di trasmissione ed L lunghezza media dei pacchetti in bit. La condizione $\rho < 1$ è detta **condizione di stabilità**.
      i. Attesa di trasmissione.
      ii. Dipende da intensità e tipo di traffico.
   (c) **Ritardo di Trasmissione L/R**
      Tempo impiegato dal router per trasmettere un pacchetto sul link, con $R(bit/sec)$ indichiamo il rate, ossia la velocita di trasmissione del collegamento mentre con L (bit) la lunghezza del pacchetto. Il ritardo sarà quindi $d_{trasm} = \frac{L}{R}$. Quindi rappresenta il ritardo di passaggio nel commutatore.
   (d) **Ritardo di Propagazione**
      Quanto un bit ci mette per spostarsi fisicamente, rappresenta il ritardo di passaggio nel link.

   Il ritardo complessivo è calcolato come somma di tutti i ritardi calcolati sopra.

2. **Throughput** Rappresenta la capacità effettiva di un link da host A verso host B, difficile da definire formalmente, ma diamo questi due riferimenti:

   (a) **Throughput Istantaneo**: Velocità in bit/sec a cui host B riceve in ogni istante.

   (b) **Throughput Medio**: Rapporto tra quantità L di dati trasferiti e il tempo T impiegato per il loro trasferimento, ovvero $\frac{L}{T}$.

   Un potenziale bottleneck influenzerebbe pesantemente questa metrica, in caso quindi di più di un collegamento q_i il calcolo sarà $min(q_1, ... , q_n)$. Il Throughput quindi è differente dalla velocità di trasmissione del collegamento che si pone come upper bound alla metrica del Throughput.

3. **Prodotto Rate per Ritardo** Il prodotto rate per ritardo rappresenta il "volume" del collegamento, ossia il massimo numero di bit che possono riempire il collegamento ad un certo istante.

1.6 Modello Stratificato

Lo schema base di gestione delle reti è di tipo **stratificato**, ossia ciascun livello ha le proprie caratteristiche e responsabilità seguendo i due criteri di:

1. **Separation of Concern**: Divisione di interessi e responsabilità tra i livelli.
2. **Information Hiding**: Si espone solo l'interfaccia di ciascun livello ma non la sua implementazione.

Quindi ogni livello fornisce **servizi** al livello superiore e scambia informazioni con i livelli adiacenti tramite **interfacce**. In questo caso anche in caso di rifattorizzazione di un livello gli altri livelli sarebbero indipendenti dalle modifiche apportate a meno di modifica dell'interfaccia. Bisogna quindi definire dei protocolli che guidino la gestione di questi livelli.

Cosa è un Protocollo In protocollo si specificano **sintassi** di un messaggio, la **semantica** di un messaggio e **azioni** da intraprendere in caso di ricezione di messaggio.

1.6.1 Stack Protocollare TCP/IP

Si compone di 5 livelli (ciascun livello sarà guidato a sua volta da protocolli):

1. **Livello Applicazione**: Rappresenta delle applicazioni di rete, oppure collegamenti logici e scambi di messaggi end-to-end tra due processi su host diversi.

   (a) **Protocolli**: **ftp**, **smtp**, **http**.

   (b) **Entità Informazione del Livello**: L'informazione a questo livello è detta **messaggio**.

2. **Livello Trasporto**: Trasferimento dati end-to-end tra processi su host remoti.
   (a) **Protocolli**: **tcp**, **udp**.
   (b) **Entità Informazione del Livello**: L'informazione a questo livello è detta:
      i. **Segmento** se in **Protocollo TCP**.
      ii. **Datagramma Utente** se in **Protocollo UDP**.
3. **Livello Rete**: Trasferimento e instradamento di **datagrammi** dalla sorgente alla destinazione.
   (a) **Protocolli**: **IP** o **ICMP**.
   (b) **Entità Informazione del Livello**: L'informazione a questo livello è detta **datagramma**.
4. **Livello Link**: Trasferimento dati in **frame** attraverso il collegamento tra elementi di rete vicini.
   (a) **Protocolli**: **PPP** o **Ethernet**.
   (b) **Entità Informazione del Livello**: L'informazione a questo livello è detta **frame**.
5. **Livello Fisico**: Trasferimento dei bit di un frame sul mezzo trasmissivo.

1.6.2 Stack Protocollare ISO/OSI - Cenno Storico

A differenza dello Stack TCP/IP, che era stato definito dall'IETF, l'ISO/OSI voleva essere uno standard effettivo e formale, definito invece dall'ISO. Si differenziava dall'TCP/IP perchè aggiungeva due livelli ossia Presentazione e Sessione (compresi nel livello Applicazione nel TCP/IP). Non dimostrò nessun pratico vantaggio al TCP/IP quindi viene solo usato come riferimento formale.

1.7 Breve Introduzione alla Sicurezza di Reti

Un software malevolo può penetrare in un host e può rendere questo host parte di una botnet che distribuisce richieste spam. Esistono varie tecniche:

1. **DDoS** (Distributed Denial of Service):
   (a) **Bandwidth Flooding**: Inondazione di pacchetti
   (b) **Connection Flooding**: Inondazione di richieste TCP
2. **Sniffing**: Man in the middle host C collegato ad un mezzo condiviso può ottenere copia di ogni pacchetto trasmesso tra due host A e B.
3. **Spoofing**: Man in the middle host C che immette pacchetti spacciandosi per un altro utente tra due host A e B.

Parametri e Criteri di Sicurezza Esistono dei metodi per difendersi da potenziali attacchi:

1. **Autenticazione**: Dimostrazione della propria identità.
2. **Crittografia**: Garantisce confidenzialità delle informazioni.
3. **Integrità** ed **Autenticità**: Assicurare che i dati non siano stati alterati e provengano da fonti affidabili.
4. **Controllo Accessi**: Limitare l'uso di risorse in base ai ruoli.
5. **Utilizzo di Firewall**: Filtrare il traffico tramite dispositivi hardware o software dedicato.

2 Livello Applicazione

2.1 Introduzione a Paradigmi, Protocolli e Tipi d'Utilizzo

Livello che definisce programmi applicativi che comunicano tra i vari host. Si seguono in genere due paradigmi di comunicazione tra host in questo livello:

1. **Client/Server**: Host server sempre attivo che attende richieste dei vari host client.
2. **Peer to Peer**: Tutti i client alla pari, spesso utilizzato nella condivisione di file.

Protocolli La comunicazione si basa su protocolli aperti (HTTP, SMTP) oppure proprietari. Una web app ad esempio si basa sul protocollo HTTP, invece un client di posta elettronica può utilizzare il protocollo SMTP.

Socket e API Ogni processo, essendo che comunicherà con **processi di altre macchine**, dovra essere **identificato** dall'**IP della macchina** (da 32 bit) e dalla **porta dedicata al processo** (da 16 bit). Questo avviene grazie all'astrazione dei socket, che utilizzando IP e porta permettono ai processi di scrivere/leggere da ciascun lato.

Caratterizzazione di un Applicazione Ciascuna applicazione dovrà scegliere un protocollo da utilizzare in base alla sua caratterizzazione ed in base a specifici criteri:

1. **Throughput**: Tasso effettivo di trasferimento dati, ad esempio un client di mail non avrà bisogno di un alto Throughput.
2. **Integrità dei Dati**: Richiesta d'integrita del flusso dei dati, ad esempio una stream potrebbe non richiedere un integrità assoluta.
3. **Sensibilità ai Ritardi**: Richiesta di una bassa latenza, ad esempio un videogioco multiplayer potrebbe richiedere una bassa latenza.

L'analisi di questi criteri in relazione all'applicazione permette la scelta del protocollo di trasporto (livello inferiore), quindi nel nostro caso se TCP (orientato allo stream) o UDP (orientato al messaggio).

2.2 TCP vs UDP nel Contesto del Livello Applicazione

Si sceglie un protocollo del livello trasporto in base alle caratteristiche dell'applicazione in questione. Descriviamone le caratteristiche:

1. **TCP**
   (a) Orientato allo stream.
   (b) Necessario un primo setup tra client e server, questo ha un **costo** in termini di **latenza**.
   (c) Utilizzo di un buffer limitato d'appoggio per lo stream.
   (d) Nessuna garanzia su proprietà di **Timing** e **Throughput**.
   (e) Attualmente utilizzato in diversi contesti.
2. **UDP**
   (a) Orientato al messaggio.
   (b) Non necessario alcun setup.
   (c) Nessuna garanzia su proprietà di **Timing**, **Throughput**, **Trasferimento Dati**, **Controllo Flusso**, **Ampiezza Minima di Banda**.
   (d) Si accettano potenziali perdite di pacchetti.

2.2.1 API e Socket

API - Application Programming Interface Un API è un insieme di regole che un programmatore deve rispettare per utilizzare delle risorse. Se un processo a livello applicazione vuole comunicare un messaggio ad un altro host allo stesso livello allora dovrà interagire con il sistema operativo che implementa i sottostanti livelli dello stack TCP/IP attraverso un interfaccia.

Interfaccia Socket API che viene utilizzata da interfaccia tra gli strati di applicazione e trasporto, è l'API di Internet per eccellenza. Dato che un host abbiamo detto essere individuato grazie ad un IP da 32 bit, identificheremo un processo e la relativa socket con la porta da 16 bit.

Per fare un esempio di API basato su uno dei protocolli del livello trasporto citati prima, mostreremo una bozza di setup tra client e server in TCP e qualche pseudocomando:

1. **Client**:

```
1    //Apertura Connessione
2    connection TCPopen(IPaddress, int)
3
```

2. **Server**:

```
    //Binding
    int tcpbind(int port)
    //Attesa Richieste
    connection TCPaccept(int)
    //Spedire Dati
    void TCPsend(connection, data)
    //Ricevere Dati
    data TCPreceive(connection)
    //Chiusura Connessione:
    void TCPclose(connection)

```

Meccanismi di Sicurezza in TCP/UDP Nativamente questi livelli non implementano alcun meccanismo di sicurezza, dunque è stato aggiunto un layer con un socket TLS (Transport Layer Security) che garantisce proprietà come cifratura o integrità dei dati.

2.3 Protocollo HTTP - HyperText Transfer Protocol

Protocollo del livello Applicazione che risulta essere **generico**, **stateless** e **basato sulla tipizzazione** che non deve per forza essere HTML, si negozia infatti il tipo di contenuto.

Inizialmente nato per le pagine HTML, attualmente utilizzato anche per la comunicazione tra applicazione tramite API REST che permettono esposizione di servizi.

Dati un client ed un server, il client tramite un **browser** ottiene una **pagina HTML** e tante **risorse dereferenziabili** con **URL**.

URI - Uniform Resource Identifier Permette l'identificazione di una risorsa:

1. **URN - Uniform Resource Name**: Sottoinsieme dell'URI che deve identificare univocamente una risorsa anche quando questa cessa di esistere.
2. **URL - Uniform Resource Locator**: Sottoinsieme dell'URI che identifica una risorsa attraverso il suo meccanismo d'accesso.

Sintassi di un URL Un URL segue questo schema per essere definito:

$$\boxed{< \text{scheme} > // < \text{user} > : < \text{password} > @ < \text{host} > : < \text{port} > / < \text{path} >}$$

1. $< \text{user} > : < \text{password} >$ è attualmente deprecata.
2. $< \text{port} >$ è spesso omessa, dato che se non definita viene utilizzata la porta standard 80.
3. $< \text{path} >$ segue lo stesso schema di un file system, dato che storicamente i server web erano statici e l'accesso ad una risorsa avveniva tramite il suo path.

Istanziando al protocollo HTTP come $< \text{scheme} >$ otteniamo:

$$\boxed{\text{http} // < \text{host} > : < \text{port} > / < \text{path} > ? \ < \text{query} >}$$

Client/Server HTTP Protocollo basato su richiesta/risposta tra client e server. La **risposta** è **stateless**, ossia non dipende dalla richiesta effettuata. Il protocollo su cui si basa, del livello trasporto, è il TCP, dato che risulta necessario prima **setuppare** una connessione **tra server e client**.

2.3.1 Versioni HTTP, Persistenza e Pipelining

HTTP/1.0 - Connessione Non Persistente Nella versione HTTP 1.0 la connessione non era persistente, e per ciascuna risorsa richiesta veniva aperta una nuova connessione TCP tra client e server. Questo comportava un overhead significativo, poiché ogni richiesta (ad esempio per immagini, fogli di stile, script) necessitava di una nuova connessione, aumentando la latenza e il carico sul server.

HTTP/1.1 - Connessione Persistente e Pipelining Con HTTP/1.1 è stata introdotta la **connessione persistente**: una singola connessione TCP può essere riutilizzata per più richieste e risposte tra client e server, riducendo l'overhead di apertura e chiusura delle connessioni. Inoltre, HTTP/1.1 supporta il **pipelining**, cioè la possibilità di inviare più richieste bufferizzando tramite una pipe di richieste. Tuttavia, il pipelining non è stato ampiamente adottato a causa di problemi di gestione dell'ordine delle risposte e di compatibilità.

2.3.2 Messaggi HTTP

In questo protocollo il messaggio può essere o una **richiesta** o una **risposta**. E' facilmente riconoscibile il tipo di messaggio già dalla sua **startline**, ossia la prima linea del messaggio.

1. **HTTP Request**: Composta da due parti:

   (a) **Request Line**:

   METODO URI VERSIONE_HTTP

   ad esempio:

   GET /dump/page.html HTTP/1.1

   (b) **Header**: Insieme di coppie (nomi-valori), che mantengono informazioni riguardanti formati preferiti, charset o encodings, ad esempio:

   Accept: text/plain; text/html, image/png, application/json

   Accept-Charset iso-8859-5, unicode-1-1

   Durante questa fase è possibile che si instauri una negoziazione tra client e server sulla versione di HTTP da utilizzare. Dato che la comunicazione partirà sempre da una richiesta lato client, in base al confronto tra le versioni dell'HTTP in utilizzo di client e server si stabilisce quale usare.

2. **HTTP Response**:

   (a) **Status Line**:

   VERSIONE_HTTP STATUS_CODE FRASE

   ad esempio:

   HTTP/1.1 200 OK

   dove lo status code segue questo criterio:

1xx	Informational
2xx	Success
3xx	Redirection
4xx	Client Error
5xx	Server Error

   (b) **Response Headers**: Informazioni relative alla risposta che non possono essere inserite nella Status Line, come ad esempio la **location della risorsa** oppure il **software** utilizzato dal **server** per handlare la richiesta.

(c) **Entity Headers**: Mantiene informazioni come:

Content-Base	URI Assoluta per dereferenziare il body
Content-Encoding	Codifica del body
Content-Language	Lingua del body
Content-Type	Tipo del body
Expires	Validità Temporale, usato nel Caching
Last Modified	Ultima modifica della risorsa, usato nel Caching

2.3.3 Content Negotiation

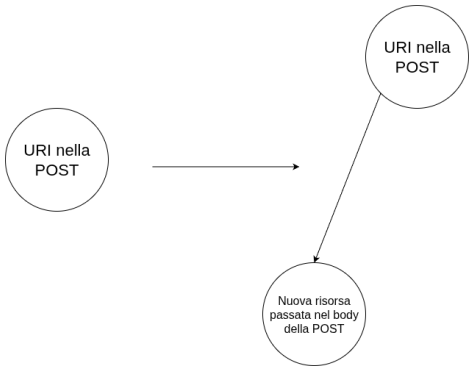
Il client conosce l'URL più generico della risorsa a cui vuole riferirsi. In base ai parametri forniti nell'header della richiesta sarà il server a dereferenziare l'URL più specifico, magari in base alle informazioni sulla lingua, il tipo di compressione o il formato.

2.3.4 Metodi dell'HTTP

Descriviamo le caratteristiche di ciascun metodo messo a disposizione da questo protocollo:

1. **OPTION**: Permette di listare quali siano i metodi chiamabili su una risorsa.
   (a) **Sintassi**: $\boxed{\text{OPTION } [RISORSA]}$
2. **GET**: Permette la richiesta di una risorsa, anche in maniera **parziale** o **condizionale**.
   (a) **Sintassi**: $\boxed{\text{GET } [RISORSA]}$

   Un comune utilizzo di una GET condizionale è quella basata sull'*If-Modified-Since*, che permette di ottenere la risorsa solo se questa è stata modificata.
3. **HEAD**: Permette la richiesta solo della *response line* e dell'*header* di una risorsa, senza il suo *body*.
   (a) **Sintassi**: $\boxed{\text{HEAD } [RISORSA]}$
4. **POST**: Permette che il server accetti l'*entità* passata nel *body* della richiesta come **nuova subordinata** della risorsa identificata dall'*URI*.
   (a) **Sintassi**: $\boxed{\text{POST } [RISORSA]}$



Sarà quindi responsabilità del server trovare un nuovo URI per la nuova risorsa.

5. **PUT/DELETE**: Permette di inserire (PUT) o cancellare (DELETE) una risorsa all'URI passato al metodo.

   (a) **Sintassi**: PUT/DELETE $[RISORSA]$

   In questo caso invece il client dovrà conoscere a priori l'URI della nuova risorsa che sta inserendo.

Metodi Safe ed Idempotenti Definiamo queste due proprietà e listiamo i metodi che le rispettano:

1. **Metodi Safe**: Non hanno effetti collaterali, dunque non modificano la risorsa, quindi i metodi: GET, HEAD, OPTION e TRACE.
2. **Metodi Idempotenti**: Hanno lo stesso effetto se eseguiti una o n volte sulla stessa risorsa, quindi i metodi: GET, HEAD, PUT, DELETE, OPTION, TRACE.

2.3.5 Web Caching

Immaginando di avere una rete locale, assumendo di avere $\{host_1, ..., host_k\}$ nella rete, cosa succede se $host_1$ richiede una risorsa su un host remoto $host_x$ e lo stesso viene fatto da $host_2$ subito dopo? Questo procedimento attualmente non viene per nulla ottimizzato, di conseguenza sono state formulate politiche di caching che permettono, tramite l'utilizzo di host intermedi, di mantenere le informazioni riguardanti le richieste per non ripeterle.

1. **Server Proxy**: Un approccio di **Web Caching** infatti è quello del **Server Proxy**, che si pone tra gli host della rete locale ipotizzata ed il server remoto. Se il server proxy cachasse infatti le richieste allora il secondo host potrebbe acquisire quelle infomazioni senza effettuare richieste al server remoto ma solo richiedendole al server proxy stesso.
2. **User Agent Cache**: Un metodo alternativo è quello che si basa sul mantenimento in locale da parte del browser delle risorse visitate da un utente.

2.3.6 Cookies

Il protocollo HTTP è per definizione stateless, di conseguenza risulta necessario un meccanismo che permetta di mantenere uno stato utente riguardante la sua navigazione su una applicazione web. Vengono quindi utilizzati per autenticazione, ricordare il profilo utente e creare sessioni su un protocollo stateless.

Si vuole quindi avere la possibilità di numerare e far riconoscere gli utenti sul sito presentando ogni volta un cookie. Il meccanismo nello specifico si basa su questi passaggi:

1. **Riga di intestazione** nel **messaggio di risposta** HTTP.
2. **Riga di intestazione** nel **messaggio di richiesta** HTTP.
3. **Cookie file** mantenuto dal **browser** dell'user.
4. **Database** di gestione **cookie** del **server web**.

Cookie First/Third Party Esistono due tipologie diverse di cookie:

1. **First Party Cookie**: Creati direttamente dal sito web che l'utente sta visitando per mantenere informazioni riguardanti lo stato.
2. **Third Party Cookie**: Memorizzano informazioni sensibili riguardanti gli utenti, sono creati e memorizzati da un dominio diverso rispetto a quello del sito web visitato.

2.4 Protocollo Telnet

Protocollo di terminale il cui scopo è quello di permettere l'uso interattivo di macchine remote. Chiaramente i due calcolatori devono poter comunicare a prescindere dai sistemi operativi che utilizzano, di conseguenza si utilizza un **interfaccia minima** a **caratteri**.

Protocollo TELNET [RFC 854] Questo modello standardizzato si basa su un programma client ed uno server, nello specifico si seguono questi passi:

1. Il client stabilisce una **connessione TCP** con il server.
2. Il **client** acquisisce l'**input dei tasti** e lo invia al server, successivamente **accetta i caratteri** che il **server manda indietro** e li visualizza sul terminale utente.
3. Il server accetta la connessione TCP e trasmette i dati al sistema operativo locale.

Il client si connette alla porta 23 del server e la connessione TCP persiste per la durata della sessione di login.

NVT - Network Virtual Terminal L'obiettivo di TELNET è quello di poter operare con quanti più calcolatori diversi e di conseguenza sistemi operativi eterogenei. Di conseguenza si definisce un **terminale virtuale** che permette la conversione dei comandi dei vari terminali locali e viceversa, creando una vera e propria interfaccia comune di riferimento. Si definiscono quindi dei set di caratteri universali per trasformare quelli delle macchine locali in comunicazione tra loro.

SSH - Secure Shell Lo standard attuale per comunicazioni tra macchine remote via terminale è SSH, ossia un architettura di protocolli nata per sostituire TELNET ed i suoi problemi di sicurezza. Uno dei servizi principali di SSH è quello di tunneling, infatti grazie al port forwarding possiamo mettere in comunicazione due processi su due macchine diverse mappando le loro porte a quella di SSH, lasciando quindi viaggiare la comunicazione tra i due processi tramite il tunnel SSH.

2.5 Protocollo SMTP - Simple Mail Transport Protocol

Uno dei protocolli applicativi più antichi, si basa su due "protagonisti":

1. **User Agent**: Client dell'user che permette l'editing e l'invio di messaggi di posta.
2. **Mail Server**: Server dei provider che devono archiviare i messaggi ricevuti, dato che il tipo di comunicazione tra i vari User Agent è asincrona, di conseguenza dovranno immagazzinare in una coda i messaggi in uscita.

Indirizzo Mail - Sintassi Solitamente un indirizzo è indicato con:

local-part @ domain-name

dove con *local-part* si indica la specifica "cassetta" dell'user nel mail server, mentre *domain-name* indica l'effettivo mail server.

Spooling dei Mail Server I Mail Server seguono una tecnica di spooling, ossia tentano di inviare i messaggi di posta settando una connessione TCP con la macchina destinazione. Se la connessione viene aperta e il trasferimento va a buon fine allora il Mail Server cancella la copia della mail in locale. Altrimenti, se qualcosa dovesse andare storto ritenterebbe la connessione TCP in maniera periodica fino ad un certo intervallo di tempo, se viene superato allora si scarta il messaggio, notificando il mittente della condizione.

Gestione Alias dei Mail Server Gestione a puntatori di indirizzi di casella postale, ne esistono di due tipi:

1. **Uno a Molti**: Un singolo indirizzo alias riferisce a più indirizzi mail reali.
2. **Molti a Uno**: Un singolo indirizzo mail reale ha molti indirizzi alias che lo riferiscono.

Protocollo SMTP [RFC5321] Si stabilisce una connessione TCP sulla porta 25, si segue una politica PUSH da parte di ogni client fino a quello precedente al destinatario. Quindi dal primo user fino all'ultimo Mail Server tutti agiranno da client nei confronti del server destinatario successivo. Tre fasi:

1. Handshaking a livello applicativo.
2. Trasferimento messaggi in coda.
3. Chiusura.

Comandi SMTP Questo standard definisce specifici comandi:

1. HELO < client identifier >
2. MAIL FROM: < reverse-path >< CRLF >
3. RCPT TO: < forward-path >< CRLF >
4. DATA
5. QUIT

Formato Messaggi SMTP I comandi sono formato ASCII, mentre le risposte sono ASCII a 7 bit. Nello specifico, il messaggio è così composto:

1. **Header**: Contiene le linee d'intestazione, come ad esempio *To*, *From*, *Subject*, che **non** corrispondono ai parametri passati ai comandi SMTP citati prima.
2. **Body**: Contiene il messaggio effettivo, composto da caratteri ASCII a 7 bit.

MIME - Multipurpose Internet Mail Extension Fino ad ora i body erano composti solo da caratteri ASCII a 7 bit. Si definisce quindi una codifica per trasferimento media e testo non-ASCII. Si aggiungono quindi dei parametri nelle intestazioni per dichiarare l'utilizzo del MIME nel body.

2.6 Protocolli POP3/IMAP

Fino ad ora abbiamo descritto il protocollo SMTP che pusha l'informazione fino all'ultimo mail server. Essendo però un tipo di comunicazione asincrona, sarà il client del destinatario a pullare il messaggio. Si definiscono quindi i protocolli POP e IMAP.

1. **POP3**: Il client apre una connessione TCP sulla porta 110, si autentica con utente e password e richiede l'elenco dei messaggi e li preleva uno alla volta.
2. **IMAP**: Più feature di POP3, permette la gestione logica in folder dei messaggi memorizzati sul server e permette di estrarre solo alcuni componenti dei messaggi.
3. **HTTP**: In alternativa, anche HTTP può effettuare richieste ai Mail Server tramite browser.