Relazione Tecnica - Programmazione di Reti

Progetto: Web Server in Python + Sito Web Statico
Studenti: Yue Shen (0001117228)
ChengZhou Hu (0001132306)
Corso di laurea: Ingegneria e Scienze Informatiche
Anno accademico: 2024/2025

---

1. Introduzione

Il progetto prevede la realizzazione di un Web Server minimale utilizzando il linguaggio Python e la libreria standard *socket,* con l'obiettivo di comprendere i meccanismi di base del protocollo HTTP e la gestione delle richieste client-server. Il server sviluppato è in grado di rispondere a richieste GET sulla porta localhost:8080 ed è stato progettato per servire contenuti statici scritti in HTML e CSS, simulando un semplice web-server locale.

2. Obiettivi e Requisiti

Requisiti Minimi:

- Rispondere su *localhost:8080*
- Gestione delle richieste HTTP GET
- Restituire risposta *200 OK* se il file richiesto esiste
- Gestione degli errori con *404 Not Found, 403 Forbidden, 405 Method Not Allowed*
- Servire almeno tre pagine HTML statiche

Estensioni Opzionali Implementate:

- Riconoscimento e gestione dei MIME types
- Logging delle richieste del file *log.txt*
- Layout responsive e animazioni nelle pagine web

3. Struttura del Progetto

La struttura prevede una directory *www/* da cui vengono letti i file HTML, CSS o immagini richieste. Il server verifica anche che il path non punti a directory esterne alla root. Tutte le richieste vengono loggate a console.

- **server.py:** codice sorgente del Web Server Pythone.
- **www/:** cartella root del sito web statico.
- **log.txt:** registra tutte le richieste ricevute.

```
Progetto_di_Reti-main/
│
├── server.py             # Server HTTP minimale in Python
├── log.txt               # Log delle richieste ricevute
├── README.md             # Documentazione del progetto
│
├── www/                  # Root directory dei contenuti serviti
│   ├── index.html        # Homepage
│   ├── game.html         # Pagina del gioco
│   ├── secret.html       # Easter Egg
│   ├── 403.html          # Pagina di errore 403 (forbidden)
```

```
│   ├── 404.html            # Pagina di errore 404 (not found)
│   ├── 405.html            # Pagina di errore 405 (metodo non consentito)
│   │
│   ├── css/
│   │   ├── errore.css      # Stile per le pagine di errore
│   │   ├── gameStyle.css   # Stile per la pagina del gioco
│   │   └── indexStyle.css  # Nuovo stile per la homepage
│   │
│   ├── js/
│   │   └── gameJS.js       # Logica JavaScript per il gioco
│   │
│   └── images/
│       ├── fase1.jpg
│       ├── fase2.jpg
│       ├── fase3.jpg
│       ├── fase4.jpg
│       └── warning.gif     # Immagini usate nel sito
```

4. **Funzionamento del Web Server**

Il server utilizza la libreria *socket* per creare un socket TCP, in ascolto su `localhost:8080.` Per ogni connessione:

1. Riceve la richiesta HTTP dal client
2. Analizza il metodo e il percorso richiesto.
3. Verifica l'esistenza del file all'interno della cartella *www/*
4. Restituisce una risposta *200 OK* con il contenuto del file se esiste, oppure *404 Not Found* con la pagina *404.html* se il file è assente
5. Determina il tipo di contenuto (MIME type) e registra i dettagli della richiesta nel file *log.txt*

5. **Gestione dei MIME types**

Tramite la funzione *get_mime_type()* che utilizza la libreria mimetypes, il server riconosce il tipo di contenuto del file richiesto (HTML, CSS, immagini, JS) e invia l'intestazione Content-Type corretta. Questo garantisce che il browser possa interpretare e rendere correttamente i file ricevuti.

6. **Logging delle Richieste**

Ogni richiesta ricevuta viene registrata nel file *log.txt* nel formato seguente:
`[YYYY-MM-DD HH:MM:SS] IP METODO PATH`
[data ora] path richiesto e codice di risposta
Questo consente di tenere traccia delle operazioni svolte dal server durante la sua esecuzione.

7. **Gestione degli Errori**

- **404 Not Found:** per file inesistenti
- **403 Forbidden:** se si tenta di accedere fuori dalla directory *www/*
- **405 Method Not Allowed:** per metodi HTTP diversi da GET

Queste pagine sono personalizzate e includono messaggi visivi e animazioni (GIF)

8. Il Sito Web Statico

Il sito servito dal server è composto da tre pagine HTML principali:

- ***index.html*:** homepage del sito con un pulsante per accedere alla pagina di gioco
- ***game.html*:** pagina con layout dinamico, immagini e script JavaScript
- ***secret.html:*** pagina *easter egg*
- pagine di errore personalizzate

Le pagine HTML sono stilizzate tramite file CSS dedicati. Sono presenti immagini (fase1.jpg, fase2.jpg, etc.) e un file JS (gameJS.js) per effetti dinamici.

9. Esecuzione del Server

Per eseguire il server è sufficiente lanciare il file server.py. Il sito è accessibile all'indirizzo: http://localhost:8080.

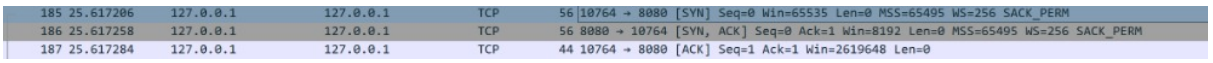
Per l'analisi del traffico, è possibile utilizzare strumenti come Wireshark per osservare il 3-way handshake TCP, le richieste GET e le relative risposte HTTP.

10. Esecuzione in Wireshark

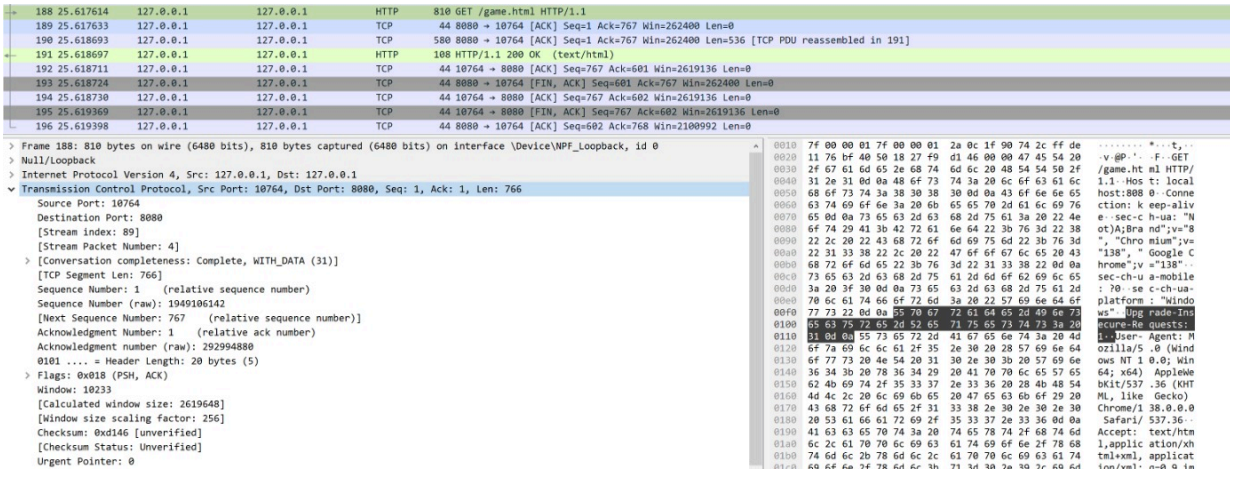
Per verificare il corretto funzionamento del Web Server, è stata dunque effettuata un'analisi del traffico locale tramite *Wireshark*, monitorando le richieste e risposte durante l'accesso alla risorsa.

3-way handshake TCP: si osservano le tre fasi del collegamento TCP tra client e server:

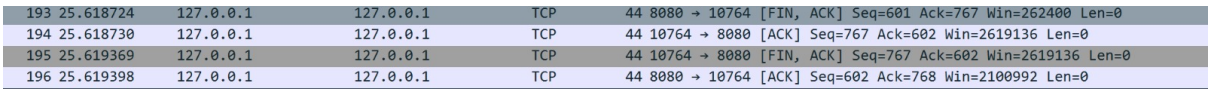
1. *185* - SYN dal client (porta 10764) al server (porta 8080)
2. *186* - SYN-ACK dal server al client
3. *187* - ACK dal client al server



La richiesta get e risposta 200: il client invia una richiesta HTTP GET (*riga 188*) e otteniamo una risposta HTTP/1.1 200 OK, seguito dal contenuto HTML della pagina (text/html) (*riga 191*).



Chiusura della connessione: la connessione viene chiusa poi correttamente con pacchetti FIN, ACK (*righe 193-196*), rispettando il meccanismo TCP di terminazione della sessione.



11. Considerazioni Finali

Il progetto ha permesso di comprendere i concetti fondamentali del protocollo HTTP, la gestione delle connessioni tramite socket in Python e la gestione dei contenuti web statici. Abbiamo implementato un server semplice ma completo, in grado di gestire le principali funzionalità richieste da un web server base e arricchito con funzionalità aggiuntive quali il logging e la gestione dei MIME types. Arricchendo così le nostre competenze pratiche sul funzionamento delle reti e della comunicazione client-server.