

Programmazione e sicurezza delle reti

VR443470

aprile 2023

Indice

1 Scrittura di applicazioni di rete mediante interfaccia socket	3
1.1 Host, processo e applicazione	3
1.2 Modalità di trasmissione in Internet	4
1.2.1 Applicazioni orientate al datagramma (UDP)	4
1.2.2 Applicazioni orientate alla connessione (TCP)	4
1.3 Schemi di applicazioni che utilizzano la rete	5
1.3.1 Modello client/server	5
1.4 Creazione dell'interfaccia Socket	5
1.5 Esempi di codice	6
1.5.1 Esecuzione degli esempi	6
1.5.2 Client UDP	6
1.5.3 Server UDP	7
1.5.4 Client_inc UDP	8
1.5.5 Server_inc UDP	9
1.5.6 Client TCP	10
1.5.7 Server TCP	11
1.5.8 Codice per la copia di un file	12
2 Dal Web ai Webservices	13
2.1 Protocollo HTTP/HTTPS	13
2.2 Hyper Text Markup Language (HTML) e Cascading Style Sheets (CSS)	15
2.2.1 HTML: tag per richiamare immagini	15
2.2.2 HTML: tag per il collegamento ipertestuale	16
2.2.3 Document Object Model (DOM)	16
2.3 Javascript	16
2.3.1 Javascript e Document Object Model (DOM)	17
2.4 Uniform Resource Locator (URL)	19
2.5 Passare dei dati al server web col metodo GET	20
2.6 Passare dei dati al server web col metodo POST	22
2.7 Common Gateway Interface (CGI)	24
2.8 Web socket	26

1 Scrittura di applicazioni di rete mediante interfaccia socket

1.1 Host, processo e applicazione

L'**host** (colui che ospita) è una **macchina** sempre identificata da un indirizzo IP a cui, opzionalmente, può essere associato un nome Internet.

Il **processo** è un **programma in esecuzione** sull'host, il quale trasmette/-riceve pacchetti verso/da altri processi su altri host attraverso la rete. Viene identificato tramite un numero di porta nell'intervallo 0 - 65535.

Un **applicazione** è una collaborazione tra un **insieme di processi** sparsi sulla rete per fare qualcosa di utile per l'utente, per esempio chat, e-mail, ecc.

Alcuni **esempi**:

- Eseguendo una ricerca su internet:
 - Il *web* è l'applicazione;
 - Mentre i browser (Chrome, Firefox, Edge, Safari) sono il processo di esecuzione;
 - L'host è il PC, tablet o smartphone su cui viene aperto il browser;
 - Apache o NGINX è il processo di esecuzione sulla macchina remota, anch'essa identificata come host.
- Aprendo un'applicazione come Telegram:
 - Telegram è l'applicazione;
 - Il processo di esecuzione è sempre l'app Telegram che è in esecuzione sul dispositivo attualmente in uso (PC, tablet, ecc.) che funge da host;
 - Il server di Telegram è il processo di esecuzione sulla macchina remota, anch'essa identificata come host.

1.2 Modalità di trasmissione in Internet

Su Internet la modalità di trasmissione è una sequenza di byte chiamata: pacchetto, Protocol Data Unit (PDU), Datagram. A seconda del livello del protocollo, ci sono nomi diversi.

1.2.1 Applicazioni orientate al datagramma (UDP)

Alcune **applicazioni sono orientate al datagramma**, quindi **ogni pacchetto scambiato tra gli host è indipendente dai precedenti e successivi**. Le **perdite** di pacchetti non vengono tenute in considerazione ed un **esempio** può essere la **trasmissione di temperature**: il ricevitore può non tener conto di alcune perdite poiché le informazioni ricevute non sono necessarie per il futuro.

1.2.2 Applicazioni orientate alla connessione (TCP)

Invece, alcune **applicazioni sono orientate alla connessione**. A differenza delle applicazioni orientate al datagramma, quelle orientate alla connessione devono tener conto delle perdite poiché solitamente le informazioni scambiate sono di dimensioni rilevanti (e.g. un'immagine). Di conseguenza, una perdita provocherebbe una lettura parziale o impossibile da parte del ricevitore.

Il **socket** si preoccupa di **numerare i pacchetti appartenenti alla stessa connessione** per rilevare eventuali pacchetti persi e poterli ritrasmettere.

Il sistema operativo introduce all'interno dei pacchetti un numero di sequenza così che possa rilevare eventuali pacchetti persi e ritrasmetterli.

- **Vantaggi:**

- L'utente scrive/legge su un archivio remoto con la stessa naturalezza di quando scrive/legge su un archivio locale come se la rete in mezzo non ci fosse.

- **Svantaggi:**

- Gli host mittente e destinatario eseguono un lavoro più complesso con il sistema operativo;
 - Ritardo di ritrasmissione nel caso in cui i pacchetti vengano persi.

1.3 Schemi di applicazioni che utilizzano la rete

Le applicazioni di rete sono insiemi di processi su host diversi che si scambiano messaggi attraverso la rete. **Esistono degli schemi base che regolano lo scambio di messaggi:**

- Modello client/server;
 - Modello Publisher/Subscriber (Pub/Sub)
-

1.3.1 Modello client/server

Il **modello client/server** è quello più utilizzato e funziona nel seguente modo (attenzione all'ordine!):

1. Il *client* esegue una **richiesta** inviando dei dati al *server*;
2. Il *server* riceve i dati del *client*, processa i dati e invia la **risposta** al *client*. Infine, si mette in attesa di altre richieste.

I dati inviati dal *client* possono essere delle trasmissioni di dati o delle richieste di dati. In ogni caso, **il ruolo è determinato dall'ordine dei messaggi e non dal contenuto**. Si noti che il *client* e *server* sono processi e non host. Infatti, l'insieme di un client e un server costituisce l'applicazione di rete.

Un **esempio** di applicazione client/server è il **sensore di temperatura corporea** che funge da client e invia al server una temperatura. Le risposte del server sono “OK” per confermare l'avvenuta ricezione.

Un altro **esempio** di applicazione client/server è il display che funge da cliente e chiede al server una temperatura. Le risposte del server in questo caso saranno i dati richiesti.

1.4 Creazione dell'interfaccia Socket

Il programma, prima di utilizzare la rete, deve essere in grado di creare un oggetto di tipo **socket**. Esso è identificato principalmente da tre parametri:

- Indirizzo IP locale;
- Porta locale, la quale è un intero senza segno di 16 bit (quindi da 0 a 65'535). Nel modello client/server:
 - Il **server** deve decidere esplicitamente il numero di porta affinché i client possano saperlo (da 0 a 1023 le porte sono chiamate “porte note” perché sono utilizzate per protocolli famosi come HTTP);
 - Il **client** può decidere se scegliere il numero di porta esplicitamente oppure delegare la scelta al sistema operativo.
- Modalità di trasmissione: UDP o TCP.

1.5 Esempi di codice

1.5.1 Esecuzione degli esempi

1. Aprire due terminali
2. Compilare il server e successivamente eseguirlo con il comando:

```
1 -gcc network.c serverUDP.c -o serverUDP -lpthread
```

3. Compilare il client e successivamente eseguirlo con il comando:

```
1 -gcc network.c clientUDP.c -o clientUDP -lpthread
```

1.5.2 Client UDP

Il client UDP ha il seguente codice:

```
1 #include "network.h"
2
3 int main(void) {
4     socketif_t socket;
5     char request[]="Ciao sono il client!\n";
6     char response[MTU];
7     char hostAddress[MAXADDRESSLEN];
8     int port;
9
10    socket = createUDPIface(20000);
11    printf("[CLIENT] Spedisco messaggio al server\n");
12    printf("[CLIENT] Contenuto: %s\n", request);
13    UDPSend(socket, request, strlen(request), "127.0.0.1", 35000);
14    UDPReceive(socket, response, MTU, hostAddress, &port);
15    printf("[CLIENT] Ho ricevuto un messaggio da host/porta %s/%d\n",
16          hostAddress, port);
17    printf("[CLIENT] Contenuto: %s\n", response);
```

- (4-8) dichiarazione delle variabili tra cui la variabile socket;
- (10) inizializzazione dell’interfaccia socket sulla porta 20’000;
- (13) invio, tramite UDP, il messaggio “Ciao sono il client!” al destinatario avente indirizzo IP “127.0.0.1” (*localhost*) e porta 35’000;
- (14) attesta dell’arrivo della risposta del server;
- (15-16) scrittura sul terminale della porta, dell’indirizzo del mittente e del messaggio.

1.5.3 Server UDP

Il server UDP ha il seguente codice:

```
1 #include "network.h"
2
3 int main(void) {
4     socketif_t socket;
5     char response[]="Sono il server: ho ricevuto correttamente il
6     tuo messaggio!\n";
7     char request[MTU];
8     char hostAddress[MAXADDRESSLEN];
9     int port;
10
11     socket = createUDPIface(35000);
12     printf("[SERVER] Sono in attesa di richieste da qualche client\
13     n");
14     UDPReceive(socket, request, MTU, hostAddress, &port);
15     printf("[SERVER] Ho ricevuto un messaggio da host/porta %s/%d\n",
16     hostAddress, port);
17     printf("[SERVER] Contenuto: %s\n", request);
18     UDPSend(socket, response, strlen(response), hostAddress, port);
19 }
```

- (4-8) dichiarazione delle variabili tra cui la variabile socket;
- (10) inizializzazione dell'interfaccia socket sulla porta 35'000;
- (12) attesta dell'arrivo di qualche messaggio da parte di qualche client;
- (13-14) ricezione di un messaggio da parte di un client e stampa sul terminale dell'indirizzo, della porta e del messaggio del mittente;
- (15) invio della risposta del server al client.

1.5.4 Client_inc UDP

Il client UDP (paragrafo 1.5.2) può essere riscritto nel seguente modo:

```
1 #include "network.h"
2
3 int main(void) {
4     int request;
5     int response;
6     socketif_t socket;
7     char hostAddress[MAXADDRESSLEN];
8     int port;
9
10    socket = createUDPInterface(20000);
11    printf("Inserisci un numero intero:\n");
12    scanf("%d", &request);
13    UDPSend(socket, &request, sizeof(request), "127.0.0.1", 35000);
14    UDPReceive(socket, &response, sizeof(response), hostAddress, &
15    port);
16    printf("[CLIENT] Ho ricevuto un messaggio da host/porta %s/%d\n",
17    hostAddress, port);
18    printf("[CLIENT] Contenuto: %d\n", response);
```

- (4-8) dichiarazione delle variabili tra cui la variabile socket;
- (10) inizializzazione dell’interfaccia socket sulla porta 20’000;
- (11-12) inserimento di un numero intero da parte dell’utente;
- (13) invio, tramite UDP, del numero inserito dall’utente al destinatario aventure indirizzo IP “127.0.0.1” (*localhost*) e porta 35’000;
- (14) attesta dell’arrivo della risposta del server;
- (15-16) scrittura sul terminale della porta, dell’indirizzo del mittente e del messaggio.

1.5.5 Server_inc UDP

Il server UDP (paragrafo 1.5.3) può essere riscritto nel seguente modo:

```
1 #include "network.h"
2
3 int main(void) {
4     int request;
5     int response;
6     socketif_t socket;
7     char hostAddress[MAXADDRESSLEN];
8     int port;
9
10    socket = createUDPInterface(35000);
11    printf("[SERVER] Sono in attesa di richieste da qualche client\
n");
12    UDPReceive(socket, &request, sizeof(request), hostAddress, &
13    port);
14    printf("[SERVER] Ho ricevuto un messaggio da host/porta %s/%d\n",
15    hostAddress, port);
16    printf("[SERVER] Contenuto: %d\n", request);
17    response = request + 1;
18    UDPSend(socket, &response, sizeof(response), hostAddress, port)
19    ;
20 }
```

- (4-8) dichiarazione delle variabili tra cui la variabile socket;
- (10) inizializzazione dell'interfaccia socket sulla porta 35'000;
- (12) attesta dell'arrivo di qualche messaggio da parte di qualche client;
- (13-14) ricezione di un messaggio da parte di un client e stampa sul terminale dell'indirizzo, della porta e del messaggio del mittente;
- (15) incremento di uno del valore intero ottenuto;
- (15) invio della risposta del server al client con il valore incrementato.

1.5.6 Client TCP

Il client TCP ha il seguente codice:

```
1 #include "network.h"
2
3 int main(void) {
4     connection_t connection;
5     int request, response;
6
7     printf("[CLIENT] Creo una connessione logica col server\n");
8     connection = createTCPConnection("localhost", 35000);
9     if (connection < 0) {
10         printf("[CLIENT] Errore nella connessione al server: %i\n",
11               connection);
12     }
13     else
14     {
15         printf("[CLIENT] Inserisci un numero intero:\n");
16         scanf("%d", &request);
17         printf("[CLIENT] Invio richiesta con numero al server\n");
18         TCPSend(connection, &request, sizeof(request));
19         TCPReceive(connection, &response, sizeof(response));
20         printf("[CLIENT] Ho ricevuto la seguente risposta dal
21               server: %d\n", response);
22         closeConnection(connection);
23     }
24 }
```

- (4-5) dichiarazione delle variabili tra cui la variabile `connection` per gestire la connessione;
- (7-8) creazione di una connessione TCP con il server utilizzando `localhost` e la porta 35'000.
- (9-10) controllo del valore della connessione per verificare se c'è stato un errore. In tal caso, la connessione termina con la stampa dell'errore su terminale;
- (12-15) in caso di connessione riuscita, il client richiede l'inserimento di un valore intero all'utente;
- (16-17) invio del valore intero al server;
- (18) attesa di una risposta da parte del server;
- (19-20) al momento della ricezione della risposta, il client stampa la risposta del server e chiude la connessione.

1.5.7 Server TCP

Il server TCP ha il seguente codice:

```
1 #include "network.h"
2
3 int main(void) {
4     int request, response;
5     socketif_t socket;
6     connection_t connection;
7
8     socket = createTCPServer(35000);
9     if (socket < 0){
10         printf("[SERVER] Errore di creazione del socket: %i\n",
11               socket);
12     }
13     else
14     {
15         printf("[SERVER] Sono in attesa di richieste di connessione
16               da qualche client\n");
17         connection = acceptConnection(socket);
18         printf("[SERVER] Connessione instaurata\n");
19         TCPReceive(connection, &request, sizeof(request));
20         printf("[SERVER] Ho ricevuto la seguente richiesta dal
21               client: %d\n", request);
22         response = request + 1;
23         printf("[SERVER] Invio la risposta al client\n");
24         TCPSend(connection, &response, sizeof(response));
25         closeConnection(connection);
26     }
27 }
```

- (4-6) dichiarazione delle variabili tra cui la variabile `connection` per gestire la connessione e `socket` per gestire i dati;
- (8) inizializzazione di un socket TCP utilizzando la porta 35'000.
- (9-10) controllo del valore del socket per verificare se c'è stato un errore. In tal caso, la creazione termina con la stampa dell'errore su terminale;
- (12-14) in caso di creazione del socket riuscita, il server attende la connessione da parte di qualche client;
- (15-17) attesa di una richiesta da parte di qualche client. Nel momento in cui viene ricevuta una richiesta, il server la accetta e instaura la connessione e attende la ricezione dei dati;
- (18-19) all'arrivo dei dati da parte del client, il server esegue un incremento di uno del valore ricevuto dal client;
- (20-22) il server invia il valore al client e infine chiude la connessione.

1.5.8 Codice per la copia di un file

Il codice per la copia di un file è strutturato nel seguente modo:

```
1 #include <stdio.h>
2 #include <stdlib.h> // For exit()
3
4 int main()
5 {
6     FILE *fptr1, *fptr2;
7     char filename[100], c;
8
9     printf("Enter the filename to open for reading \n");
10    scanf("%s", filename);
11
12    // Open one file for reading
13    fptr1 = fopen(filename, "r");
14    if (fptr1 == NULL)
15    {
16        printf("Cannot open file %s \n", filename);
17        exit(0);
18    }
19
20    printf("Enter the filename to open for writing \n");
21    scanf("%s", filename);
22
23    // Open another file for writing
24    fptr2 = fopen(filename, "w");
25    if (fptr2 == NULL)
26    {
27        printf("Cannot open file %s \n", filename);
28        exit(0);
29    }
30
31    // Read contents from file
32    c = fgetc(fptr1);
33    while (c != EOF)
34    {
35        fputc(c, fptr2);
36        c = fgetc(fptr1);
37    }
38
39    printf("\nContents copied to %s", filename);
40
41    fclose(fptr1);
42    fclose(fptr2);
43    return 0;
44 }
```

- (6-29) dichiarazione dei puntatori ai file e tentativi di apertura dei due file richiesti dall'utente;
- (32-37) viene eseguita la lettura dal primo file e salvata nella variabile `c`. A questo punto, finché viene letto un carattere valido, ovvero che non sia la fine del file (*End Of File*, EOF), il contenuto della variabile `c` viene inserito nel secondo file;
- (39-43) al termine del processo di copia, viene stampato il file nel quale sono stati copiati i valori e chiusi i rispettivi file descriptors.

2 Dal Web ai Webservices

2.1 Protocollo HTTP/HTTPS

Il **protocollo HTTP** venne inventato per fruire dei contenuti in rete (*World Wide Web*). Tuttavia, al giorno d'oggi viene usato per l'invocazione di funzionalità remote, tecnica chiamata **Webservice**.

Il protocollo si divide in più **fasi**:

1. Apertura di una connessione TCP;
2. Nel caso del protocollo HTTPS, avviene l'autenticazione del server e negoziazione di una chiave di cifratura;
3. Invio di messaggi di *request* e *response*;
4. Chiusura della connessione TCP.

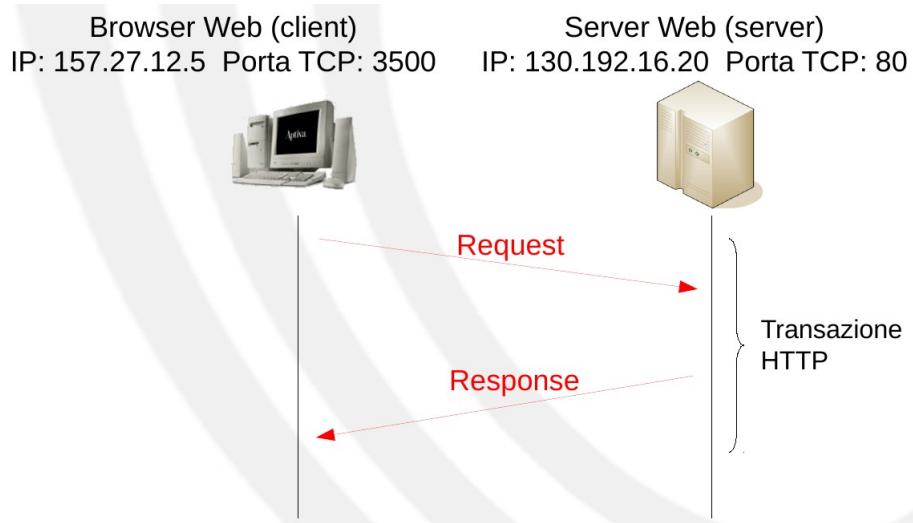


Figura 1: Esempio di scambio di messaggi nel protocollo HTTP.

Nel **protocollo HTTPS** i messaggi che passano nella connessione TCP, sono gli stessi del protocollo HTTP con l'aggiunta di una **cifratura dei dati in transito** e di una **autenticazione del server mediante certificato digitale**. Inoltre, il server lavora sulla porta 443 e non sulla porta classica 80.

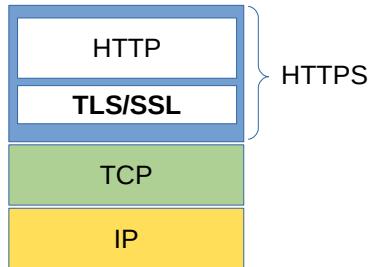


Figura 2: Protocollo HTTPS.

Un **esempio** di richiesta:

```

    GET /it/i-nostri-servizi/servizi-per-studenti HTTP/1.1
    User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.8)
    Gecko/20100214 Ubuntu/9.10 (karmic) Firefox/3.5.8
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
    Accept-Language: en-us,en;q=0.5
    Accept-Encoding: gzip,deflate
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
    Keep-Alive: 300
    Connection: keep-alive
    !!!Riga vuota!!!
  
```

Figura 3: Messaggio di richiesta.

Un **esempio** di risposta:

```

    HTTP/1.1 200 OK
    Date: Mon, 17 May 2022 16:10:48 GMT
    Server: Apache
    Last-Modified: Mon, 29 Mar 2022 13:57:17 GMT
    Keep-Alive: timeout=15, max=100
    Connection: Keep-Alive
    Transfer-Encoding: chunked
    Content-Type: text/html
    !!!Riga vuota!!!
    <html>
    ...
    </html>
  
```

Le parentesi graffe a destra riuniscono le righe da "HTTP/1.1 200 OK" a "Content-Type: text/html" e sono etichettate come "Intestazione della risposta". Le parentesi graffe a destra riuniscono le righe da "!!!Riga vuota!!!" a "</html>" e sono etichettate come "Corpo della risposta".

Figura 4: Messaggio di risposta.

2.2 Hyper Text Markup Language (HTML) e Cascading Style Sheets (CSS)

HTML è un linguaggio testuale di descrizione di una pagina, in particolare è la specializzazione del generico XML (*eXtensible Markup Language*). HTML si basa sui “tag” annidati, i quali eventualmente contengono attributi.

Questo linguaggio, spesso viene utilizzato con un altro linguaggio chiamato **CSS**.

Esempi di codice HTML:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <link rel="stylesheet" href="styles.css">
5 </head>
6 <body>
7
8 <h1>This is a heading</h1>
9 <p>This is a paragraph.</p>
10
11 </body>
12 </html>
```

E CSS:

```
1 body {
2   background-color: powderblue;
3 }
4 h1 {
5   color: blue;
6 }
7 p {
8   color: red;
9 }
```

2.2.1 HTML: tag per richiamare immagini

Viene utilizzato “img src” per richiamare le immagini:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h2>I trulli di Alberobello</h2>
6 
7
8 </body>
9 </html>
```

2.2.2 HTML: tag per il collegamento ipertestuale

Viene utilizzato “href” per il collegamento ipertestuale:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h1>HTML Links</h1>
6
7 <p><a href="https://www.w3schools.com/">Visit W3Schools.com!</a></p>
8
9 </body>
10 </html>
```

2.2.3 Document Object Model (DOM)

Il **Document Object Model (DOM)** è una forma di rappresentazione dei documenti (pagina) strutturati come modello orientato agli oggetti.

2.3 Javascript

Javascript è un linguaggio di programmazione multi paradigma orientato agli eventi, utilizzato sia nella programmazione lato client web che lato server. È facile trovarlo all'interno di codice HTML anche grazie al suo tag riconoscibile: `<script>`.

Tuttavia, è difficile trovare del codice Javascript pure scritto nelle pagine HTML. Solitamente vengono create vere e proprie librerie così da rendere il codice più leggibile e mantenibile.

Un **esempio** di codice Javascript:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h2>Use JavaScript to Change Text</h2>
6 <p>This example writes "Hello JavaScript!" into an HTML element
   with id="demo":</p>
7
8 <p id="demo"></p>
9
10 <script>
11   document.getElementById("demo").innerHTML = "Hello
12   JavaScript!";
13 </script>
14 </body>
15 </html>
```

Javascript trova il suo grande utilizzo con gli **eventi causati dall'utente**, per esempio con la pressione di un bottone all'interno di una pagina web:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <button onclick="document.getElementById('demo').innerHTML=Date()">
6 Che ora e'?
7 </button>
8
9 <p id="demo"></p>
10
11 </body>
12 </html>
```

2.3.1 Javascript e Document Object Model (DOM)

Il *Document Object Model* consente di trasformare una pagina web da **documento statico** a **Graphical User Interface (GUI)**, cioè interattivo.

Infatti, grazie al codice Javascript contenuto nella pagina HTML ed eseguito dal browser, l'utente può modificare lo stato della pagina web a seconda di determinate azioni. Quindi, la pagina web si automodifica e assume le sembianze di una applicazione web, chiamata in gergo *web application*.

Alcuni **esempi** di codice interattivo:

- Per avere un riquadro con la pagina web ANSA:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <iframe id="area" height="2000" width="1000"></iframe>
6
7
8 <script>
9   document.getElementById("area").src = "https://www.ansa.it/
  sito/notizie/topnews/index.shtml";
10 </script>
11
12 </body>
13 </html>
```

- Per avere un timer che alla fine del tempo stampa “Hello”:

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h1>The Window Object</h1>
6 <h2>The setInterval() Method</h2>
7
8 <p id="demo"></p>
9
10 <script>
11 setInterval(displayHello, 1000);
12
13 function displayHello() {
14   document.getElementById("demo").innerHTML += "Hello";
15 }
16 </script>
17
18 </body>
19 </html>
```

- Per avere lo stesso effetto del punto precedente ma utilizzando una **funzione**:

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h1>The Window Object</h1>
6 <h2>The setInterval() Method</h2>
7
8 <p id="demo"></p>
9
10 <script>
11 setInterval(function() {document.getElementById("demo").
12   innerHTML += "Hello"}, 1000);
13 </script>
14
15 </body>
16 </html>
```

2.4 Uniform Resource Locator (URL)

Chiamato anche Universal Resource Locator, l'**URL** consente di identificare in maniera univoca una risorsa HTTP in qualsiasi parte della rete mondiale.

È **strutturato** in tre parti:

- Il protocollo utilizzato a livello di applicazione, di trasporto e la porta utilizzata, per esempio HTTP con protocollo TCP e porta 80;
- Nome/IP dell'host che eroga tale risorsa;
- Nome della risorsa con il percorso logico completo.

2.5 Passare dei dati al server web col metodo GET

Il codice HTML per utilizzare il metodo GET è il seguente:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h2>The method Attribute</h2>
6
7 <p>This form will be submitted using the GET method:</p>
8
9 <form action="http://127.0.0.1/action" target="_blank" method="get">
10   <label for="fname">First name:</label><br>
11   <input type="text" id="fname" name="fname" value="John"><br>
12   <label for="lname">Last name:</label><br>
13   <input type="text" id="lname" name="lname" value="Doe"><br><br>
14   <input type="submit" value="Invia">
15 </form>
16
17 <p>After you submit, notice that the form values is visible in the
18   address bar of the new browser tab.</p>
19 </body>
20 </html>
```

La pagina web visualizzata è la seguente:

The method Attribute

This form will be submitted using the GET method:

First name:

Last name:

After you submit, notice that the form values is visible in the address bar of the new browser tab.

Una volta inserito nome e cognome, alla pressione del tasto, i dati saranno inviati al *localhost* aggiungendo come parametri nell'URL `fname=name` e `lname=name`, dove al posto di `name` vengono inseriti nome e cognome. Il link risulta: <http://127.0.0.1/action?fname=John&lname=Doe>.

→ Parametri (max 2048 caratteri)

```
GET /action_page.php?fname=John&lname=Doe HTTP/1.1
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.8)
Gecko/20100214 Ubuntu/9.10 (karmic) Firefox/3.5.8
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
!!!Riga vuota!!!
```

Figura 5: La richiesta GET HTTP.

2.6 Passare dei dati al server web col metodo POST

Il codice HTML per utilizzare il metodo POST è il seguente:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h2>The method Attribute</h2>
6
7 <p>This form will be submitted using the POST method:</p>
8
9 <form action="http://127.0.0.1/action" target="_blank" method="post">
10   <label for="fname">First name:</label><br>
11   <input type="text" id="fname" name="fname" value="John"><br>
12   <label for="lname">Last name:</label><br>
13   <input type="text" id="lname" name="lname" value="Doe"><br><br>
14   <input type="submit" value="Submit">
15 </form>
16
17 <p>Notice that the form values is NOT visible in the address bar of
18   the new browser tab.</p>
19 </body>
20 </html>
```

La pagina web visualizzata è la seguente:

The method Attribute

This form will be submitted using the POST method:

First name:

John

Last name:

Doe

Submit

Notice that the form values is NOT visible in the address bar of the new browser tab.

Una volta inserito nome e cognome, alla pressione del tasto, i dati saranno inviati al *localhost*. A differenza del metodo GET, i parametri non vengono specificati nell'URL, di conseguenza la sicurezza aumenta. Il link dunque risulta: <http://127.0.0.1/action>.

I valori vengono inseriti all'interno della richiesta HTTP.

```
POST /action_page.php HTTP/1.1
User-Agent: Mozilla/5.0
Accept: text/html
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
!!!Riga vuota!!!
fname=John&lname=Doe
!!!Riga vuota!!!
```

The diagram illustrates a POST HTTP request. It consists of two main parts: the header (Intestazione della richiesta) and the body (Corpo della richiesta). The header includes standard HTTP headers like User-Agent, Accept, and Content-Type, along with specific parameters for the form submission. The body contains the form data, including field names (fname and lname) and their values (John and Doe), separated by empty lines.

Figura 6: La richiesta POST HTTP.

2.7 Common Gateway Interface (CGI)

Il **Common Gateway Interface (CGI)** è una tecnologia utilizzata dai *web server* per interfacciarsi con applicazioni esterne generando contenuti web dinamici.

Ogni qualvolta che un *client* richiede al web server un URL corrispondente a un documento HTML, gli viene restituito un documento statico. Al contrario, se l'URL corrisponde a un programma CGI, il server lo esegue in tempo reale, generando dinamicamente informazioni per l'utente. Sostanzialmente è l'**esecuzione di un determinato programma sul server**.

Di conseguenza, il browser diventa il *client* di molte applicazioni di rete, per esempio la posta elettronica.

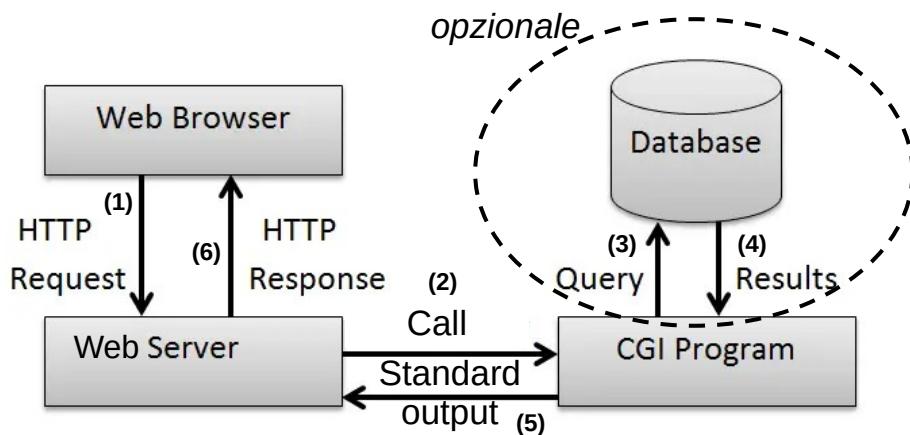


Figura 7: Fasi del CGI.

Degli **esempi** di esecuzione lato server di un programma sono un eseguibile come il client della posta elettronica, oppure un codice PHP, Java, NodeJS, ecc.

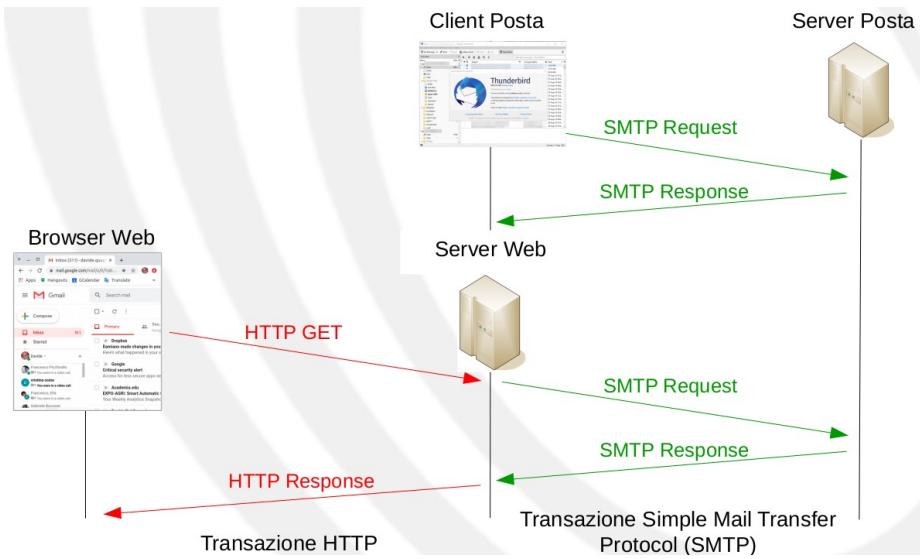


Figura 8: Esempio di CGI, la classica posta elettronica.

2.8 Web socket

La **web socket** è una tecnologia web che fornisce canali di comunicazione chiamati *full-duplex*, cioè bidirezionali, attraverso una singola connessione TCP. Viene utilizzato principalmente per realizzare applicazioni che forniscono contenuti e giochi in tempo reale. Questo perché **il protocollo consente maggiore interazione tra browser e server** grazie al alcune caratteristiche.

Innanzitutto è un **protocollo a livello di applicazione**, per cui è un **metodo alternativo a HTTP e HTTPS**. Ha la caratteristica fondamentale di **comunicazione simmetrica** tra *browser* e *web server*, ovverosia che **i processi possono “prendere l'iniziativa” e inviare dei dati alla controparte**. Inoltre, nasce da una sessione HTTP/HTTPS attraverso un'operazione chiamata **Protocol Upgrade**.

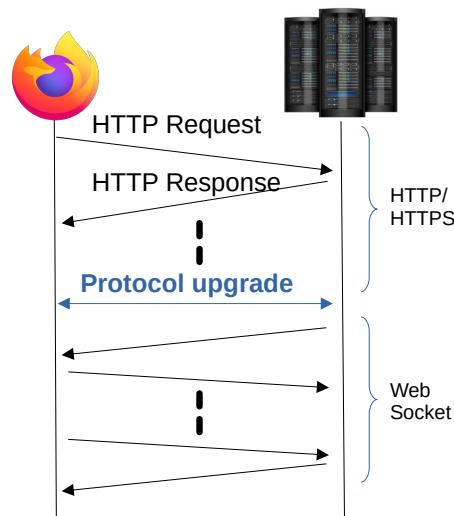


Figura 9: Esempio di web socket e Protocol Upgrade.

Nel messaggio HTTP, ci sono due campi che vengono modificati per indicare il Protocol Upgrade: `Upgrade` e `Connection`. Entrambi i campi vengono modificati con i rispettivi valori `websocket` e `Upgrade`.



Figura 10: Il Protocol Upgrade nel dettaglio.