TFTP

TFTP (*Trivial File Transfer Protocol*) è un semplice protocollo di trasferimento dei file descritto nello RFC 1350, ed è stato implementato al di sopra di UDP. Il protocollo è stato progettato per essere piccolo e semplice da implementare, di fatti non presenta tutte le caratteristiche di FTP, può solo leggere e scrivere file da e verso un server remoto; non ha alcuna forma di autenticazione e non permette di ottenere la lista di file e directory, come invece fa FTP. I dati scambiati sono quantificati in byte da 8 bit e prevede tre diversi formati di trasferimento dei file, di fatto implementando un livello di presentazione.

Il protocollo applicativo implementa una serie di caratteristiche che sono deducibili anche solo osservando il formato dei sui messaggi.

TFTP è un **protocollo connesso**, infatti la connessione viene richiesta dal client inviando il seguente messaggio nel quale i primi due byte costituiscono lo header e indicano se il cliente vuole effettuare un'operazione di lettura (`01`) o di scrittura (`02`). Il campo successivo, di lunghezza variabile (in byte), indica il nome del file su cui vuole agire il client; per indicare la fine del nome viene usato un byte posto a `0` (stringa zero terminata). Il nome del file è di fatto una forma di **indirizzamento** a livello applicativo in quanto individua la risorsa su cui si vuole agire. Dopo il nome del file viene indicata la modalità di trasferimento e, anche in questo caso la lunghezza del campo è variabile e quindi la terminazione della stringa viene effettuata tramite un byte posto a zero.

```
          2 bytes     string    1 byte      string    1 byte
          ------------------------------------------------
RRQ/    | 01/02 |   Filename  |    0   |     Mode     |    0   |
WRQ     ------------------------------------------------
```

Il server se accetta la richiesta di connessione si comporterà nel modo seguente:

- se il client ha richiesto una connessione per leggere un file (`RRQ`) il server invierà il primo blocco di dati con un messaggio che ha il seguente formato:

  ```
          2 bytes    2 bytes        n bytes
          ---------------------------------
  DATA  |   03    |   Block #  |    Data     |
          ---------------------------------
  ```

  L'operazione di invio dei dati è identificata nello header dal codice `03`, salvato in due byte, i blocchi dei dati sono numerati nello header con numeri progressivi partendo dal valore 01 ed utilizzando due byte, mentre i dati che costituiscono il payload, occupando esattamente 512 byte;

- se invece il client ha richiesto una connessione per scrivere un file (`WRQ`) il server invierà un messaggio di conferma cha ha il seguente formato:

```
        2 bytes    2 bytes
        -------------------
ACK   |  04   |   Block #  |
        --------------------
```

  Ogni **invio dei dati è confermato** e la conferma è individuata dal codice `04` nello header del messaggio (due byte), seguito dal numero di blocco che viene confermato; nel caso di conferma per l'apertura di connessione in scrittura il numero del blocco assume il valore `00`.

La **chiusura della connessione** avviene nel momento in cui viene spedito un messaggio dati contenente un payload di lunghezza inferiore a 512 byte.

Qualunque sia l'operazione richiesta dal client, il trasferimento dei dati, da e verso il server, avviene sempre usando il messaggio dati e la conferma indicati in precedenza. È importante però sottolineare che il protocollo prevede la trasmissione di messaggi con un payload non superiori a 512 byte, che implica di fatto la **frammentazione e il riassemblaggio** del file.

Inoltre prima di poter effettuare la trasmissione di un nuovo messaggio deve essere stata ricevuta la conferma di ricezione del messaggio precedente, quindi il protocollo **esegue un controllo di flusso** di tipo stop-and-wait.

Il protocollo effettua anche un **controllo errori** inviando un messaggio con il seguente formato.

```
        2 bytes  2 bytes        string    1 byte
        ----------------------------------------
ERROR |  05   |  ErrorCode |   ErrMsg   |   0  |
        ----------------------------------------
```

Un messaggio di errore non è mai confermato, quindi di fatto è una pura cortesia in quanto potrebbe andare perso durante la trasmissione. Nello header viene posto il codice `05` nei primi due byte, seguono due byte che riportano uno dei codici di errore elencati di seguito, e infine una stringa di messaggio di errore di lunghezza variabile e terminata da un byte posto a `0`; il messaggio di errore è destinato ad un utente umano, quindi dovrà essere comprensibile.

`Error Codes`

Value	Meaning
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.

```
3         Disk full or allocation exceeded.
4         Illegal TFTP operation.
5         Unknown transfer ID.
6         File already exists.
7         No such user.
```

Da quanto visto finora **il protocollo effettua l'incapsulamento** dei messaggi in quanto in ogni tipologia di messaggio **è presente uno header** che indica, tramite un codice, lo scopo del messaggio stesso. Questo codice è ovviamente comprensibile solo alle due entità che implementano il protocollo TFTP, e quindi solo alle due entità a livello applicativo.

Il protocollo non prevede la gestione del **multiplexing** e del **demultiplexing** in quanto non offre un servizio ad un protocollo di livello superiore per il quale effettuare aggregazione e disaggregazione dei dati trasmessi.

Il protocollo non prevede **servizi di trasmissione**, come la **priorità dei pacchetti**, la **garanzia di un certo livello di qualità del servizio** e la **gestione della sicurezza**.