Il protocollo TCP

Il protocollo **TCP** (**T**ransmission **C**ontrol **P**rotocol) è un protocollo che consente la trasmissione o il trasferimento di pacchetti di dati in rete e su Internet.
Esso è orientato alla connessione e allo Stream, cioè la connessione deve essere sempre attiva fino a quando l'ultimo byte viene trasmesso sul flusso (stream) di rete. Esso è affidabile perché garantisce la consegna di tutti i pacchetti da trasmette (salvo disconnessioni) e qualora un pacchetto non arrivi a destinazione, viene ritrasmesso dal mittente.

I dati inviati tramite connessioni **TCP** sono divisi in segmenti numerati indipendentemente. Ogni segmento contiene la destinazione di origine e la sezione dati che viene inserita in un'intestazione.
Il protocollo di controllo della trasmissione è quello responsabile del riordino dei segmenti nella giusta sequenza al suo arrivo nel terminale ricevente. **TCP** è responsabile di tenere traccia di questi segmenti, mentre l'IP gestisce l'effettiva consegna dei dati. Esso comprende un controllo degli errori integrato che garantisce la ricezione di ogni segmento richiesto.

TCP include anche il controllo degli errori, che assicura che ogni pacchetto sia consegnato come richiesto. Il trasferimento di dati come file e pagine web su Internet utilizza il protocollo **TCP**.
Il controllo del trasferimento affidabile dei dati è la funzione principale del **TCP**.

A causa della complessità della rete e degli apparati connessi, alcuni pacchetti potrebbero essere persi o arrivare al destinatario in ordine sparso. A tal proposito la ritrasmissione dei pacchetti persi o "l'ordinamento" della loro esatta sequenza, viene effettuata proprio dal protocollo **TCP**.

Nella seguente tabella, la struttura del segmento TCP:

<table>
<tr><th colspan="10">Struttura del segmento TCP</th></tr>
<tr><td colspan="5">Num. porta sorgente</td><td colspan="5">Num. porta destinazione</td></tr>
<tr><td colspan="10">Numero di sequenza</td></tr>
<tr><td colspan="10">Numero di riscontro</td></tr>
<tr><td>Lung. intest.</td><td>Non usato</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="2">Finestra di ricezione</td></tr>
<tr><td colspan="6">CHECKSUM</td><td colspan="4">Puntatore a dati urgenti</td></tr>
<tr><td colspan="10">OPZIONI</td></tr>
<tr><td colspan="10">DATI</td></tr>
</table>

Spiegazione campi

Num. porta sorgente	Identifica il *numero di porta sull'host mittente* associato alla connessione TCP
Num. porta destinazione	Identifica il *numero di porta sull'host destinatario* associato alla connessione TCP
Numero di sequenza	Numero di sequenza, indica lo scostamento (espresso in byte) dell'inizio del segmento TCP all'interno del flusso completo, a partire dall'Initial Sequence Number (*ISN*), deciso all'apertura della connessione
Numero di riscontro	Numero di riscontro, ha significato solo se il flag ACK è impostato a 1, e conferma la ricezione di una parte del flusso di dati nella direzione opposta, indicando il valore del prossimo *Sequence number* che l'host mittente del segmento TCP si aspetta di ricevere
Lung. intestazione	Indica la lunghezza (in dword da 32 bit) dell'header del segmento TCP; tale lunghezza può variare da 5 dword (20 byte) a 15 dword (60 byte) a seconda della presenza e della lunghezza del campo facoltativo *Options*.
URG	Se impostato a 1 indica che nel flusso sono presenti *dati urgenti* alla posizione (offset) indicata dal campo *Urgent pointer*. *Urgent Pointer* punta alla fine dei dati urgenti
ACK	Se impostato a 1 indica che il campo *Acknowledgment number* è valido
PSH	Se impostato a 1 indica che i dati in arrivo non devono essere bufferizzati ma passati subito ai livelli superiori dell'applicazione
RST	Se impostato a 1 indica che la connessione non è valida; viene utilizzato in caso di grave errore; a volte utilizzato insieme al flag ACK per la chiusura di una connessione
SYN	Se impostato a 1 indica che l'host mittente del segmento vuole *aprire una connessione TCP* con l'host destinatario e specifica nel campo *Sequence number* il valore dell'Initial Sequence Number (*ISN*); ha lo scopo di sincronizzare i numeri di sequenza dei due host. L'host che ha inviato il SYN deve attendere dall'host remoto un pacchetto SYN/ACK.
FIN	Se impostato a 1 indica che l'host mittente del segmento vuole *chiudere la connessione TCP* aperta con l'host destinatario. Il mittente attende la conferma dal ricevente (con un FIN-ACK). A questo punto la connessione è ritenuta chiusa per metà: l'host che ha inviato FIN non potrà più inviare dati, mentre l'altro host ha il canale di comunicazione ancora disponibile. Quando anche l'altro host invierà il pacchetto con FIN impostato la connessione, dopo il relativo FIN-ACK, sarà considerata completamente chiusa.
Finestra di ricezione	Indica la dimensione della *finestra di ricezione* dell'host mittente, cioè il numero di byte che il mittente è in grado di accettare a partire da quello specificato dall'acknowledgment number.
CHECKSUM	Campo di controllo utilizzato per la verifica della validità del segmento. È ottenuto facendo il complemento a 1 della somma complemento a uno a 16 bit dell'intero header TCP (con il campo checksum messo a zero), dell'intero payload, con l'aggiunta di uno pseudo header composto da: indirizzo IP sorgente(32bit),indirizzo IP destinazione(32bit), un byte di zeri, un byte che indica il protocollo e due byte che indicano la lunghezza del pacchetto TCP (header + dati).
Puntatore a dati urgenti	Puntatore a dato urgente, ha significato solo se il flag URG è impostato a 1 ed indica lo scostamento in byte a partire dal *Sequence number* del byte di dati urgenti all'interno del flusso.
OPZIONI	Opzioni (facoltative) per usi del protocollo avanzati.
DATI	Rappresenta il carico utile o *payload* da trasmettere cioè la PDU proveniente dal livello superiore.

Primo esempio di trasmissione/ricezione via TCP

Detto questo, ora scriviamo un piccolo programma che trasmette una stringa sul flusso di rete utilizzando il protocollo TCP ad un programma ricevente.

Il primo programma si chiama "**TCP_send**" con il quale trasmettiamo una stringa letta da console ad un altro programmino che la riceve. Quest'ultimo si chiama "**TCP_receive**" e la visualizza.

Questo è il codice del primo:

```
using System;
using System.Net.Sockets;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;

namespace TCP_Send
{
    class Program
    {
        static void Main(string[] args)
        {
            string messaggio = "";
            Console.WriteLine("Inserire il messaggio da spedire:");
            TcpClient tcpClient = new TcpClient("localhost", 3306);
            NetworkStream netstream = tcpClient.GetStream();
            messaggio = Console.ReadLine();
            int dati = messaggio.Length;
            for (int i = 0; i < dati; i++)
            {
                netstream.WriteByte((byte)messaggio[i]);
            }
            netstream.Close();
            tcpClient.Close();
            Console.ReadKey();
        }
    }
}
```

Spiegazione del codice

`TcpClient tcpClient = new TcpClient("localhost", 3306);`	Crea un oggetto di tipo TcpClient che fornisce la connessione client per il TCP. Al costruttore vengono passati due parametri che rappresentano rispettivamente l'indirizzo di destinazione e il numero di porta. L'indirizzo di destinazione è di tipo string e, in questo esempio, rappresenta il PC stesso (***localhost***) dove viene eseguito il codice.
`NetworkStream netstream = tcpClient.GetStream();`	Crea un oggetto che rappresenta il flusso di rete (NetworkStream) per la trasmissione del messaggio. Esso è creato richiamando il metodo GetStream() dall'oggetto tcpClient che ritorna appunto un oggetto di tipo NetworkStream.
`messaggio = Console.ReadLine();`	Memorizzo il messaggio digitato alla console
`int dati = messaggio.Length;`	Memorizzo la lunghezza del messaggio digitato

`for (int i = 0; i < dati; i++)`
`{`
`netstream.WriteByte((byte)messaggio[i]);`
`}`
	Attraverso un ciclo, scrivo (trasmetto) un carattere alla volta nel flusso dopo averli convertiti in byte. Il messaggio è una stringa ma allo stesso tempo è un array di caratteri che posso leggere attraverso un ciclo for.

`netstream.Close();` `tcpClient.Close();`	Uscito dal ciclo chiudo lo stream e la connessione TCP

E questo è il codice del programma ricevitore che dovrà essere eseguito per primo:

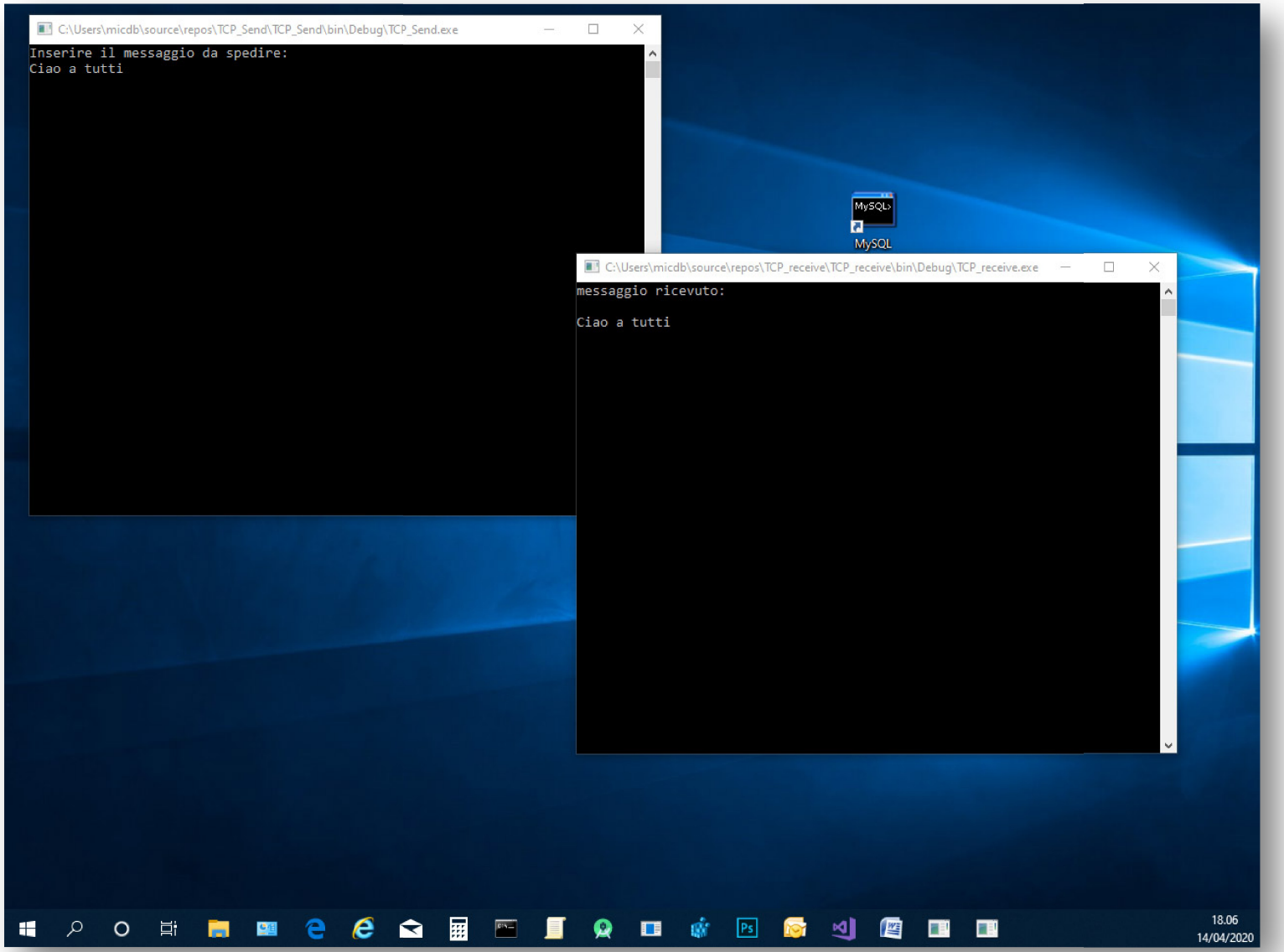
```
namespace TCP_receive
{
    class Program
    {
        public static void Main(string[] args)
        {
            Thread thread = new Thread(new ThreadStart(Listen));
            thread.Start();
            Console.ReadKey();
        }

        public static void Listen()
        {
            IPAddress locale = IPAddress.Parse("127.0.0.1");
            int porta = 3306;
            TcpListener tcpListener = new TcpListener(locale, porta);
            tcpListener.Start();
            TcpClient tcpClient = tcpListener.AcceptTcpClient();
            NetworkStream networkStream = tcpClient.GetStream();
            StreamReader streamReader = new StreamReader(networkStream);

            string messaggio = streamReader.ReadToEnd();
            Console.WriteLine("Messaggio ricevuto:\n");
            Console.WriteLine(messaggio);
            tcpClient.Close();
            tcpListener.Stop();
        }
    }
}
```

Spiegazione del codice

Codice del metodo Main(..)	
`Thread thread = new Thread(new ThreadStart(Listen));`	Crea un thread che 'lancia' un metodo 'Listen' che resta il ascolto di eventuali connessioni. Il metodo viene eseguito in un thread diverso da quello principale. **N.B.:** Il parametro passato al costruttore, cioè `new ThreadStart(…)` tecnicamente viene definito 'delegato'
`thread.Start();`	Fa partire il codice del metodo 'Listen'

Codice del metodo Listen()	
`IPAddress locale = IPAddress.Parse("127.0.0.1");`	Crea un oggetto che acquisisce un indirizzo IP attraverso la conversione (*Parse*) di una stringa contenente proprio l'indirizzo IP. Anche in questo caso esso rappresenta l'indirizzo locale del PC su cui viene eseguito il codice.
`int porta = 3306;`	Memorizzo il numero di porta che combacia, ovviamente, con quello impostato nel programma TCP_send.
`TcpListener tcpListener = new TcpListener(locale, porta);`	Creo l'oggetto di tipo TcpListener che resta all'ascolto di una connessione su l'indirizzo IP e la porta passati come parametri al costruttore.
`tcpListener.Start();`	Inizia l'ascolto di eventuali richieste di connessione.
`TcpClient tcpClient = tcpListener.AcceptTcpClient();`	Accetta una richiesta di connessione da parte del client.
`NetworkStream networkStream = tcpClient.GetStream();`	Crea un oggetto che rappresenta il flusso di rete (NetworkStream) per la ricezione del messaggio. Esso è creato richiamando il metodo GetStream() dall'oggetto tcpClient che ritorna appunto un oggetto di tipo NetworkStream.
`StreamReader streamReader = new StreamReader(networkStream);`	Creo un oggetto di tipo StreamReader che legge i caratteri provenienti dall'oggetto NetworkStream passato come parametro nel costruttore.
`string messaggio = streamReader.ReadToEnd();`	Memorizzo il messaggio proveniente dal flusso di lettura del NetworkStream
`Console.WriteLine("Messaggio ricevuto:\n");` `Console.WriteLine(messaggio);`	Visualizzo il messaggio ricevuto
`tcpClient.Close();` `tcpListener.Stop();`	Chiudo gli oggetti relativi alla connessione

Di seguito una prova di esecuzione:


C:\Users\micdb\source\repos\TCP_Send\TCP_Send\bin\Debug\TCP_Send.exe
Inserire il messaggio da spedire:
Ciao a tutti
MySQL
C:\Users\micdb\source\repos\TCP_receive\TCP_receive\bin\Debug\TCP_receive.exe
messaggio ricevuto:
Ciao a tutti
18.06
14/04/2020