Il protocollo MODBUS

Nicolò Toscani

28 Febbraio 2017

Indice

1 Introduzione **3**

2 Descrizione generale **4**
2.1 Protocollo 4
2.2 Modello dei dati 5
2.3 Codici funzione 7

3 Modbus TCP/IP **8**
3.1 Il modello Client - Server 8
3.2 Architettura di comunicazione 9
3.3 Header MBAP 9
3.4 Descrizione funzionale 10
3.4.1 Communication Application Layer 11
3.4.2 TCP Management Layer 12
3.4.3 TCP/IP Stack Layer 13
3.5 Gestione delle connessioni TCP 13
3.5.1 Errori di connessione 14
3.6 Utilizzo dello stack TCP/IP 15
3.7 MODBUS Client 17
3.7.1 MODBUS Request 18
3.7.2 MODBUS Confirmation 18
3.8 MODBUS Server 19
3.8.1 Verifica della PDU 19
3.8.2 Costruzione della risposta MODBUS 20

4 Lavoro svolto **21**
4.1 ModBus Client 21
4.2 Strumenti utilizzati 22
4.3 Lettura registri 22
4.4 Scrittura registri 22
4.5 Analisi messaggi scambiati 22

5 Sviluppi futuri **27**

1 Introduzione

Modbus è un protocollo di comunicazione applicativo che si posiziona al livello 7 del modello ISO/OSI, basato sullo scambio di messaggi.
Il protocollo trova una notevole applicazione nel controllo di apparecchiature in ambito industriale, permettendo di stabilire comunicazioni tra controllori (PLC), sensori, attuatori, interfacce uomo-macchina (HMI) e sistemi di supervisione (SCADA).
Permette di realizzare una comunicazione di tipo *client-server* tra dispositivi diversi connessi su tipi di reti e bus differenti.
Modbus è un protocollo di tipo *request/response* che offre diversi servizi specificati da diversi codici funzione.
Esistono 3 varianti principali del protocollo:

- ***Modbus seriale*** permette di stabilire comunicazioni seriali asincrone su interfacce *RS-232* e *RS-485*.
- ***Modbus TCP/IP*** permette di stabilire comunicazioni su reti *Ethernet*.
- ***Modbus Plus*** destinato a reti ad alte prestazioni, basato sul passaggio di token.

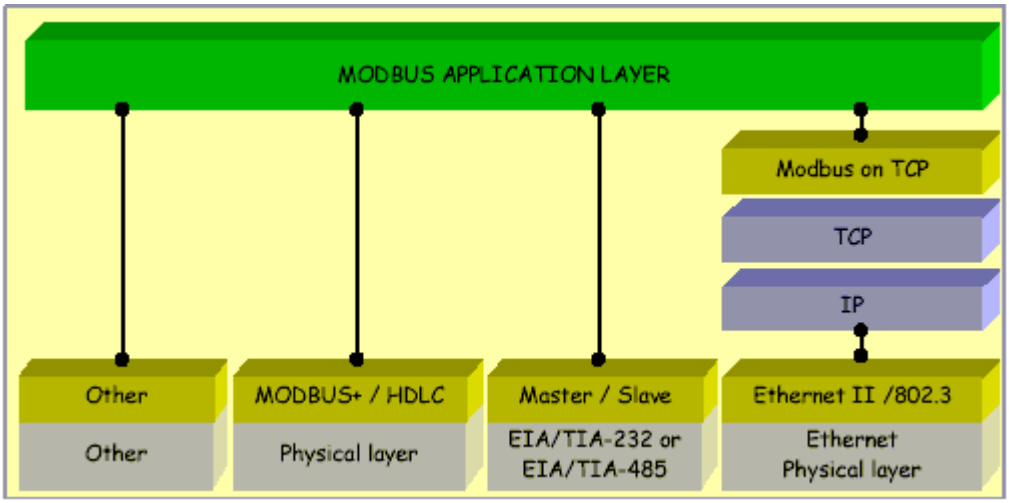

Figura 1: Comunicazione *Modbus*

Dispositivi collegati ad una rete come PLC, HMI, controllori di movimento, dispositivi di I/O,etc. possono utilizzare *Modbus* per avviare un operazione remota, come ad esempio la richiesta di lettura di un determinato parametro di produzione.
Utilizzando un gateway è possibile mettere in comunicazione diversi tipi di bus o reti.

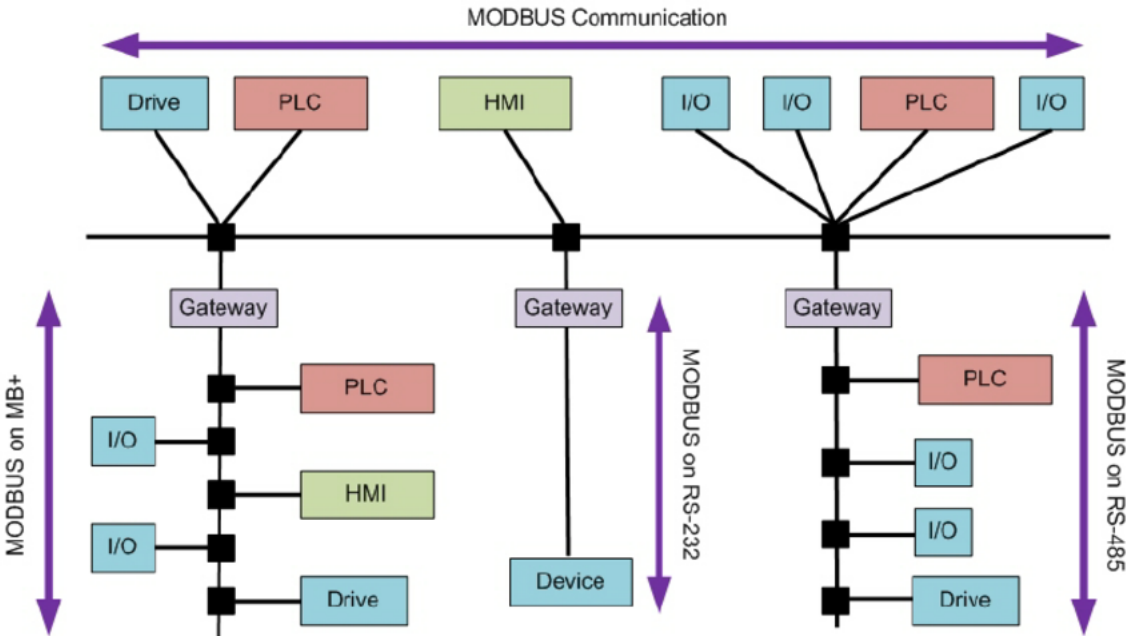

Figura 2: Esempio di un architettura di rete *Modbus*

In questa breve relazione saranno inizialmente descritte le specifiche generali del protocollo.
Successivamente verrà trattata la versione che sfrutta la suite dei protocolli TCP/IP.
Nella parte finale verrà descritta l'implementazione di un *client* che avrà il compito di inviare messaggi ad un dispositivo *server* e riceverne risposta.

2 Descrizione generale

2.1 Protocollo

Il protocollo *Modbus* definisce una ***Protocol Data Unit (PDU)*** indipendentemente dai livelli di comunicazione sottostanti, introducendo su specifici bus e sulle reti alcuni campi aggiuntivi, definiti nella ***Application Data Unit (ADU)***.

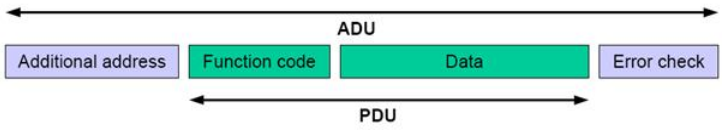

Figura 3: Frame generale *Modbus*

- ***Function code:*** indica al server l'azione da svolgere.
  - Dimensione campo 1 byte.
  - Valori decimali 1 - 255.
- ***Data:*** contiene informazioni aggiuntive che il server utilizza insieme al codice funzione.
  Ad esempio, nel caso di un messaggio inviato dal *client* al *server*, questo campo può includere:
  - Indirizzo registro.
  - Quantità di elementi da leggere.
  - Numero di byte nel campo data.

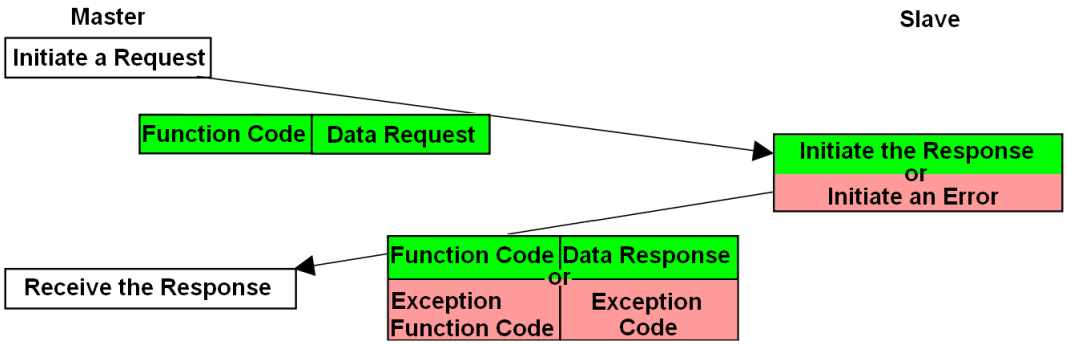

Figura 4: Transazione *Modbus*

Se non sono presenti errori di richiesta, il campo ***Data*** del messaggio di risposta dal *server* al *client* contiene i dati richiesti.
Al contrario, se sono presenti errori nella funzione di richiesta, il campo ***Data*** contiene il codice dell'eccezione che ha determinato l'impossibilità di rispondere generata dal *server*, mentre il campo ***Function Code*** restituisce un codice di errore.

2.2 Modello dei dati

Nel protocollo *Modbus* il ***registro*** è la più piccola entità dotata di indirizzo e le funzioni sono definite in modo da operare su questa unità.
Sopra a questi *registri* vengono definiti 4 tipi di dati primari:

- ***Discrete Inputs:*** Fornisce un sistema di I/O
  - Dimensione: 1 bit
  - Modalità: sola lettura

- ***Coils:*** Modificabile dal programma
  - Dimensione: 1 bit
  - Modalità: lettura-scrittura
- ***Input Registers:*** Fornisce un sistema di I/O
  - Dimensione: word 16 bit
  - Modalità: sola lettura
- ***Holding Registers:*** Modificabile dal programma
  - Dimensione: word 16 bit
  - Modalità: lettura-scrittura

Il protocollo definisce due tipi di dati primari:

- il tipo *discreto*
- il tipo *registro*

Il tipo di dato *discreto* rappresenta un valore bit utilizzato per indirizzare gli *output coils* e i *digital input* del dispositivo *server*, mentre un tipo di dato *registro* rappresenta valori interi a 16 bit.
Il protocollo utilizza una rappresentazione *big endian* per la memorizzazione di indirizzi e dati.
Ogni indirizzo dei registri può variare da 0 a 65535.

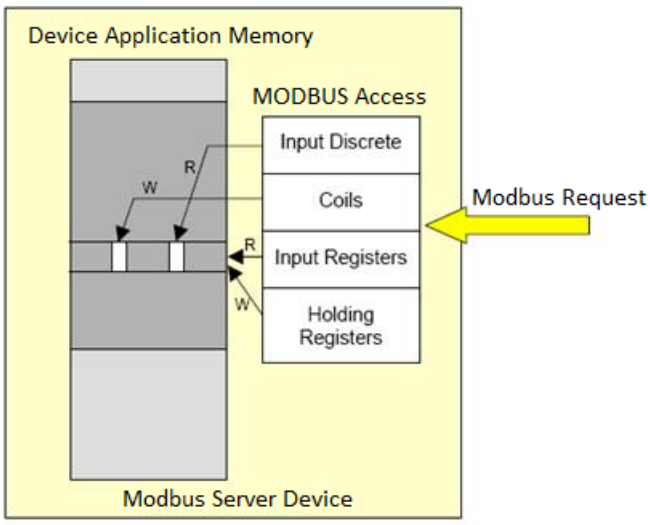

Figura 5: Esempio mappa registri in un dispositivo *server*

2.3 Codici funzione

Quando un messaggio viene spedito da un *client* ad un *server* il campo funzione specifica quale azione è richiesta al *server* (leggere o scrivere registri,caricare o verificare il programma dello slave, etc.). Le funzioni eseguibili dal *server* si dividono in 3 categorie:

- **Public Function Codes:** funzioni publiche convalidate dalla comunità di sviluppatori e utilizzatori *Modbus*.
- **User-Defined Function Codes:** funzioni definite dall'utente.
- **Reserved Function Codes:** funzioni con utilizzo riservato ad alcune compagnie non accessibili al publico utilizzo.

Modbus Function Code	Name	Function	Corresponding Address Type
01	Read coil status	Reads the bit data (N bits)	0x
02	Read input discrete value	Reads the bit data	1x
03	Read multiple registers	Reads the integer type/character type/status word /floating-point data (N words)	4x
04	Read input registers	Reads the integer type/status word/ floating-point type data	3x
05	Write single coil	Writes the bit data (one bit)	0x
06	Write single register	Writes the integer type/character type/status word /floating-point data (one word)	4x
15	Write multiple coils	Writes the bit data (N bits)	0x
16	Write multiple registers	Writes the integer type/character type/status word /floating-point data (N words)	4x

Figura 6: Esempi di codici funzione del protocollo

Un dispositivo *client* che vuole spedire un messaggio ad un dispositivo *server* collegato alla rete, per prima cosa specificherà il numero del nodo a cui inviare il messaggio, inserendo la funzione richiesta, i dati da spedire, ed un campo per il controllo degli errori.
Il messaggio di risposta dal *server* verso il *client* conterrà un codice con il campo di riferimento dell'azione eseguita, i dati richiesti ed un campo di controllo degli errori.
Nel caso si verificasse un errore nel codice funzione della richiesta, verrano restituite al *client* codice errore e tipo di eccezzione generata.

3 Modbus TCP/IP

La versione del protocollo **Modbus TCP/IP** sfrutta la suite dei protocolli *TCP/IP*. Questa versione del protocollo è una variante della versione seriale RTU che permette l'invio di messaggi su reti *Internet* e *Intranet*.
Un qualsiasi computer connesso in rete può comportarsi da *client* o da *server* scambiando messaggi in rete.
Questa variante del protocollo ha visto un notevole sviluppo negli ultimi anni, grazie al fatto di potersi appoggiare sulla tecnologia *Ethernet*, diventata uno degli standard per il networking industriale.
Le prestazioni dei trasferimenti dati attraverso una rete Internet non consentono la realizzazione di sistemi deterministici. Per questo viene utilizzato soprattutto per la parte di supervisione, manutenzione e scambio dati con i sistemi informativi aziendali.
Il protocollo *Modbus TCP/IP* è orientato alla connesione, utilizza la codifica binaria dei dati e il meccanismo di rilevamento TCP/IP per errori di trasmissione.

3.1 Il modello Client - Server

Il protocollo *Modbus TCP/IP* fornisce un servizio per lo scambio di messaggi tra dispositivi *client* e *server* connessi ad una rete *Ethernet TCP/IP*.
Per realizzare questo tipo di comunicazione vengono utilizzati 4 tipi differenti di messaggi:

- *Modbus Request:* messaggio spedito sulla rete dal dispositivo *client* per iniziare la comunicazione.
- *Modbus Indication:* messaggio di *Request* ricevuto dal dispositivo *server*.
- *Modbus Response:* messaggio di risposta alla *Request* inviato dal *server*.
- *Modbus Confirmation:* messaggio di *Response* ricevuto dal dispositivo *client*.

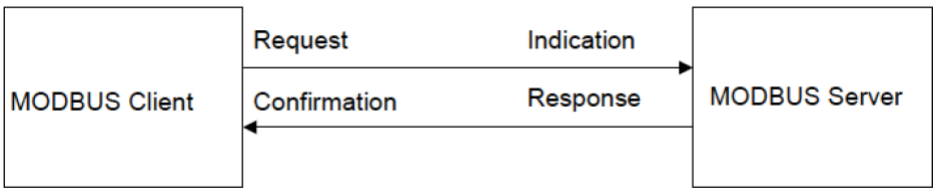

Figura 7: Messaggi scambiati in una comunicazione *Modbus TCP/IP*

3.2 Architettura di comunicazione

Un sistema di comunicazione che si bassa su *Modbus TCP/IP* può includere diversi tipi di dispositivi:

- Dispositivi *Modbus TCP/IP client* e *server* connessi ad una rete *TCP/IP*
- Dispositivi di interconnesione come bridge, router o gateway per consentire la comunicazione tra reti *TCP/IP* e reti *seriali*

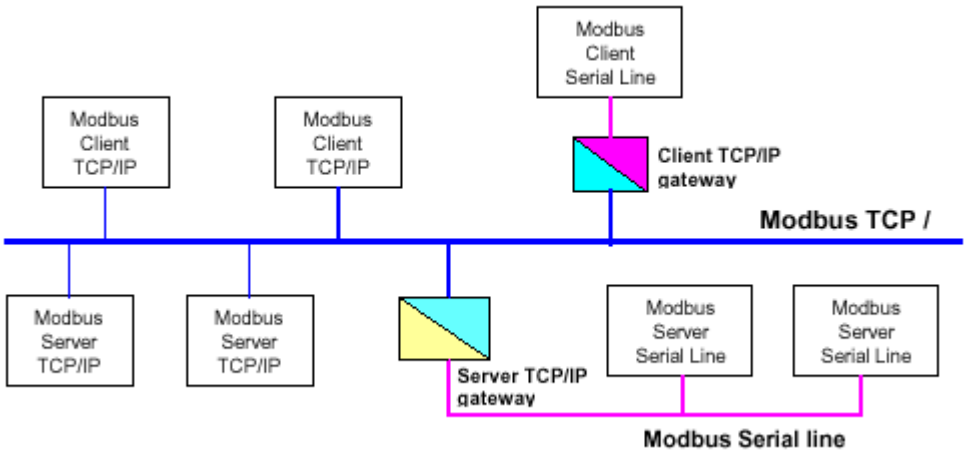

Figura 8: Architettura di comunicazione *Modbus TCP/IP*

3.3 Header MBAP

Un *header* dedicato è utilizzato su *Modbus TCP/IP* per identificare la *Modbus Application Data Unit (ADU)*.
Questo header è chiamato *MBAP (Modbus Application Protocol Header)*.

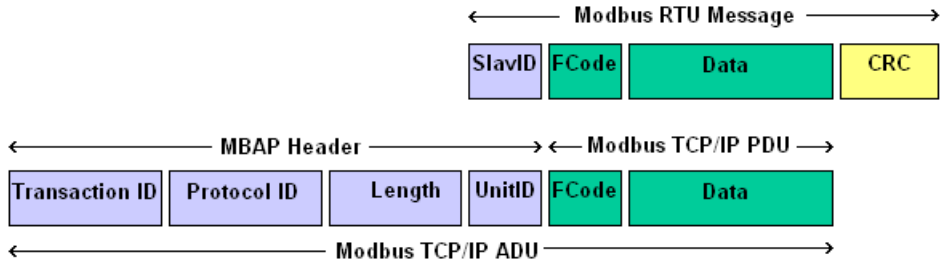

Figura 9: Frame di richiesta/risposta su TCP/IP

L'header file è composto da:

- **Transaction Identifier:** utilizzato per il controllo di parità della transazione, il server ricopia questo parametro nella response.
- **Protocol Indentifier:** il protocollo *Modbus* viene identificato dal valore decimale 0.
- **Length:** contatore dei byte dei campi successivi, incluso *Unit identifier* e i campi dei dati.
- **Unit Identifier:** utilizzato per l'instradamento intra sistema del messaggio.Tipicamente viene utilizzato per comunicare su una linea *Modbus seriale* attraverso un gateway tra una rete *Modbus seriale* e una rete *Ethernet TCP/IP*.
  Viene impostato dal client nel messaggio di *request* e deve essere ricopiato nel messaggio di *response* del server.

Fields	Length	Description -	Client	Server
Transaction Identifier	2 Bytes	Identification of a MODBUS Request / Response transaction.	Initialized by the client	Recopied by the server from the received request
Protocol Identifier	2 Bytes	0 = MODBUS protocol	Initialized by the client	Recopied by the server from the received request
Length	2 Bytes	Number of following bytes	Initialized by the client (request)	Initialized by the server (Response)
Unit Identifier	1 Byte	Identification of a remote slave connected on a serial line or on other buses.	Initialized by the client	Recopied by the server from the received request

Figura 10: Campi *MBAP* di un messaggio *Modbus TCP/IP*

3.4 Descrizione funzionale

Il modello architetturale si presenta come modello generale applicabile a qualsiasi dispositivo *client* o *server*.
Alcuni dispositivi possono solamente operare come *client* o come *server*.

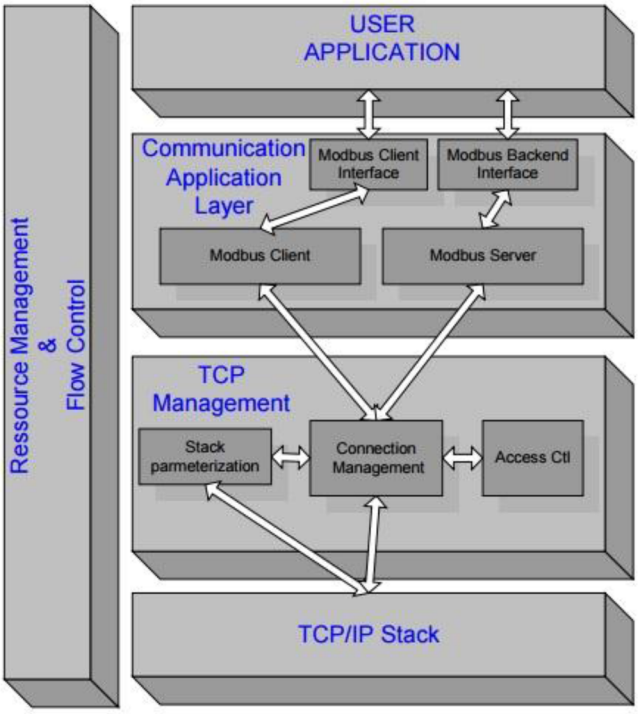

Figura 11: Modello architetturale del protocollo *Modbus TCP/IP*

3.4.1 Communication Application Layer

Un dispositivo *Modbus* può fornire un interfaccia *Modbus client* e/o *server* ed un ulteriore interfaccia di backend che permette all'utente di accedere indirettamente agli oggetti applicativi.
Questa interfaccia è composta da quattro aree:

- Ingressi discreti.
- Uscite discrete.
- Ingressi analogici.
- Uscite analogiche.

I dati dell'applicazione vengono mappati utilizzando questa interfaccia.

Primary tables	Object type	Type of access	Comments
Discretes Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system
Holding Registers	16-bit word	Read-Write	This type of data can be alterable by an application program.

Figura 12: Aree che compongono l'interfaccia utente

Nel livello applicativo possiamo avere quattro elementi distinti:

- ***MODBUS Client:*** consente all'applicazione utente di controllare lo scambio di messaggi con un dispositivo *server*. Il *MODBUS Client* costruisce una *request* partendo da una richiesta spedita dall'applicazione utente all'interfaccia Modbus Client.
  Il *MODBUS Client* utilizza successivamente una transazione per gestire i tempi e l'esecuzione dei processi per soddisfare una *request*, fornendo una conferma al *client stesso*.

- ***MODBUS Client Interface:*** fornisce un interfaccia che permette all'applicazione utente di costruire la *request* richiesta utilizzando servizi *Modbus* differenti.

- ***MODBUS Server:*** quando riceve una *request* questo modulo attiva una azione locale per leggere, scrivere o eseguire un ulteriore azione. L'esecuzione di queste azione è effettuata in modo totalmente trasparente al programmatore.
  Il *server Modbus* è in ascolto sulla porta *TCP* 502, svolge le operazioni indicate dalla *request* e costruisce una *response* pertinente alla richiesta.

- ***MODBUS Backend Interface:*** interfaccia dal *server* all'applicazione utente in cui vengono definiti gli oggetti applicativi.

3.4.2 TCP Management Layer

La principale funzione del servizio a scambio di messaggi è quella di gestire una comunicazione, in modo da stabilire un inizio, un termine della comunicazione e gestire un flusso di dati sulle connessioni *TCP*.

- ***Connection Management:*** La comunicazione tra *client* e *server Modbus* richiede l'utilizzo di un modulo di gestione della connessione *TCP*.
  Per le comunicazioni tramite il protocollo *Modbus TCP/IP* viene dedicata la porta **502**. A causa di appplicazioni che richiedono che questa porta venga modificata, sia il *client* che il *server* devono avere la possibilità di settare un numero di porta differente.

- ***Access Control Module:*** in certi contesti critici l'accessibilità a dati interni dei dispositivi deve essere revocata per determinati *host*.
  Con questo modulo è possibile controllare gli accessi a determinati oggetti e realizzare applicazioni sicure.
  Utilizzando una lista di indirizzi *IP* il modulo controlla ogni connessione in ingresso autorizzando o meno l'accesso alle risorse.

- ***Stack Parametrization:*** utilizzando determinati parametri del layer *TCP/IP* è possibile modificare il comportamento del sistema.
  Ad esempio è possibile modificare parametri per la gestione del flusso dei dati, la dimensione del buffer di ricezione, dei timer di *TCP* e l'impostazione degli indirizzi *IP* come la sottorete ed i gateway.

3.4.3 TCP/IP Stack Layer

Lo stack *TCP/IP* può essere parametrizzato in modo da poter agire sul controllo del flusso, sulla gestione degli indirizzi e la gestione delle connessioni definendo determinati vincoli specifici per ogni dispositivo o sistema.
Generalmente per gestire le connessioni *TCP* vengo utilizzate le socket BSD.

3.5 Gestione delle connessioni TCP

Una comunicazione *Modbus* richiede di stabilire una connessione *TCP* tra il *client* e il *server*.
La creazione della connessione può essere attivata in modo esplicito dall'applicazione utente o automaticamente dal modulo di gestione delle connessioni TCP.

- *Gestione diretta delle connessioni:* l'applicazione utente gestisce direttamente la connessione TCP. Questa modalità di connessione viene effettuata sia per il dispositico *client* che per il *server* utilizzando un socket BSD.

- *Gestione automatica delle connessioni:* la gestione della comunicazione è totalmente trasparente all'applicazione utente.
  Il modulo di gestione delle connessioni gestisce le connessioni *client - server*.
  Una connessione viene aperta quando un dispositivo riceve un pacchetto dal *client*. La connessione viene chiusa se si invia un pacchetto di chiusura della rete dal dispositivo che ha aperto la comunicazione.
  Quando viene ricevuta una richiesta di connessione il modulo di controllo degli accessi permette di adottare delle politiche di accesso dei client.

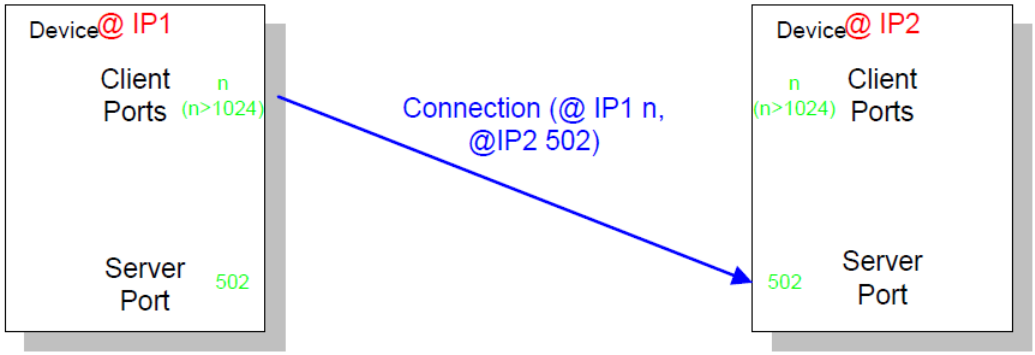

Figura 13: Connessione *Modbus TCP/IP*

Il servizio di messaggistica deve fornire una socket di ascolto sulla porta *502* che permette di accettare un nuovo collegamento e iniziare lo scambio di messaggi con i dispositivi.
Il *server* metterà a disposizione sulla porta *502* una socket per accettare una connessione da un *client*.
Il *client* aprirà quindi una connessione verso la porta *502* del *server*. Una volta stabilita la connessione avviene il trasferimento dei dati.
La connessione deve rimanere aperta durante tutta la trasmissione.
Quando il trasferimento dei dati è terminato, il *client* richiede al *server* la chiusura della connessione.

3.5.1 Errori di connessione

Si prensentano situazioni che possono avere delle conseguenze sulla connessione TCP.
Una connessione può essere chiusa solo da un lato, senza che il dispositivo con cui stavamo comunicando venga avvisato. Questa connessione viene chiamata *half-open*.

- *Interruzione della comunicazione tra due host:* può essere causata dalla disconnessione del cavo di collegamento *Ethernet* da un lato dei due dispositivi. Se nessun pacchetto viene inviato durante la disconnessione, l'interruzione non sarà visibile se dura meno del valore del timer *Keep Alive*. In caso contario viene restituito un errore a livello TCP in grado di ripristinare la connessione.
  Se alcuni pacchetti vengono inviati prima e dopo la disconnessione vengono attivati gli algoritmi di ritrassmissione di *TCP (Jacobson's, Karn's e l'algoritmo esponenzione di backoff)*.

- *Crash e riavvio del server:* dopo un crash e un riavvio del *server* si ha una connessione *"half-open"* lato *client*.

Se non viene inviato nessun pacchetto la connessione rimane aperta fino allo scadere del timer *Keep Alive* che ripristina la connessione. Se alcuni pacchetti vengono inviati, il *server* riceve dei dati che appartengono ad una connessione che non esiste più. Viene inviato un messaggio per chiudere la connessione lato *client* rimasta aperta.

- *Crash e riavvio del client:* dopo un crash e un riavvio del *client* si ha una connessione *"half-open"* lato *server*.
  Se non viene inviato nessun pacchetto la connessione rimane aperta fino allo scadere del timer *Keep Alive* che ripristina la connessione.
  Se il *client* apre una nuova connessione con le stesse caratteristiche della precedente (stessi indirizzi IP e stesse porte per sorgente e destinazione) prima dello scadere del timer *Keep Alive*, la connessione avrà un esito negativo.
  Se il *client* apre una nuova connessione con caratteristiche diverse rispetto alla connessione precedente, la connessione avrà esito positivo.

3.6 Utilizzo dello stack TCP/IP

Lo *stack TCP/IP* fornisce un interfaccia per la gestione delle connessioni, invio e ricezione dei dati e per effettuare alcune parametrizzazioni.
L'interfaccia dello stack è basata sulle *BSD - Berkeley Software Distribution*.

- *socket():* crea una socket senza indirizzo IP e numero di porta.
- *bind():* crea un'associazione tra socket e numero di porta.
- *connect():* crea collegamento all'indirizzo e il numero di porta specificato.
- *accept():* accetta una connessione sul socket specificato.
- *listen():* attende la richiesta di connessione sulla socket specificata.
- *send():* invia i dati sulla socket specificata.
- *recv():* riceve i dati dalla socket specificata.
- *close():* chiude connessione sulla socket specificata.

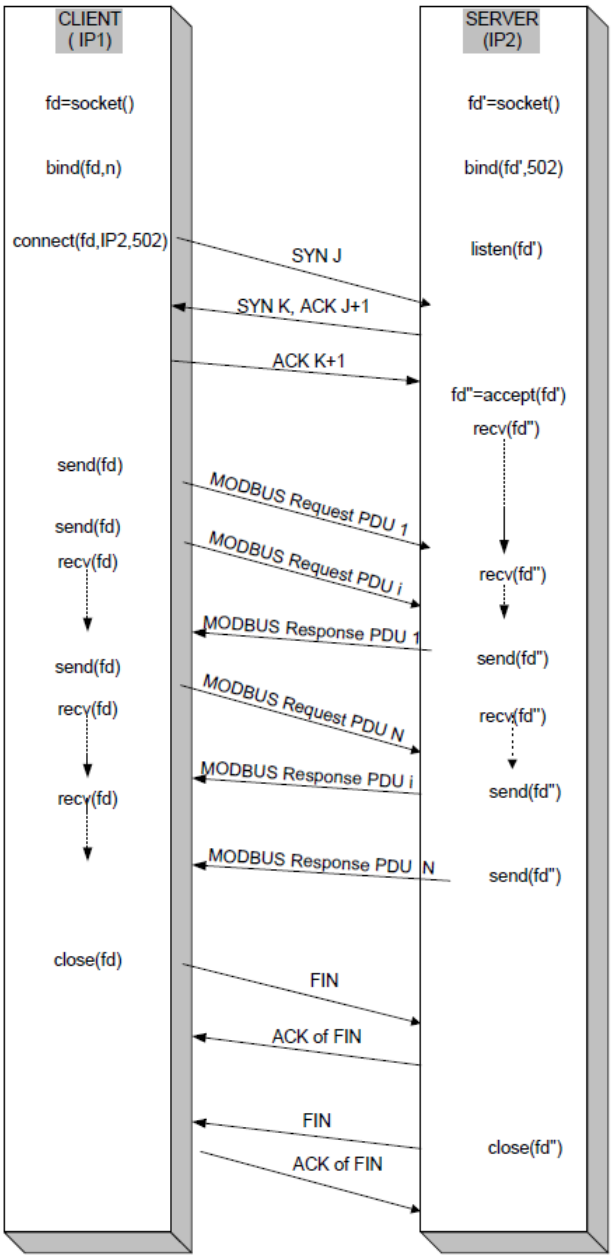

Figura 14: Sessione *Modbus TCP/IP*

Alcuni parametri dello *stack TCP/IP* possono essere modificati per adattare il comportamento del sistema. I parametri modificabili sono:

- *SO-RCVBUF, SO-SNDBUF:* permettono di impostare le dimensioni del buffer di ricezione e invio del socket.
- *TCP-NODELAY:* consente di inviare piccoli pacchetti sulla rete, e non di riunire questi in un unico pacchetto da spedire. (Algoritmo di Nagle)
- *SO-REUSEADDR:* consente il riutilizzo della stesso indirizzo per la comunicazione mentre un lato della comunicazione viene interrotta.
- *SO-KEEPALIVE:* consente di attivare il rilevamento della perdita di una stazione.
- *Time-out sulle connessioni:* consente di impostare il tempo per stabilire una nuova connessione.
- *Time-out di ritrasmissione:* consente di impostare il timer di ritrasmissione di un pacchetto in caso di perdita.
- *Indirizzo IP locale*
- *Maschera di sottorete*
- *Gateway di default*

3.7 MODBUS Client

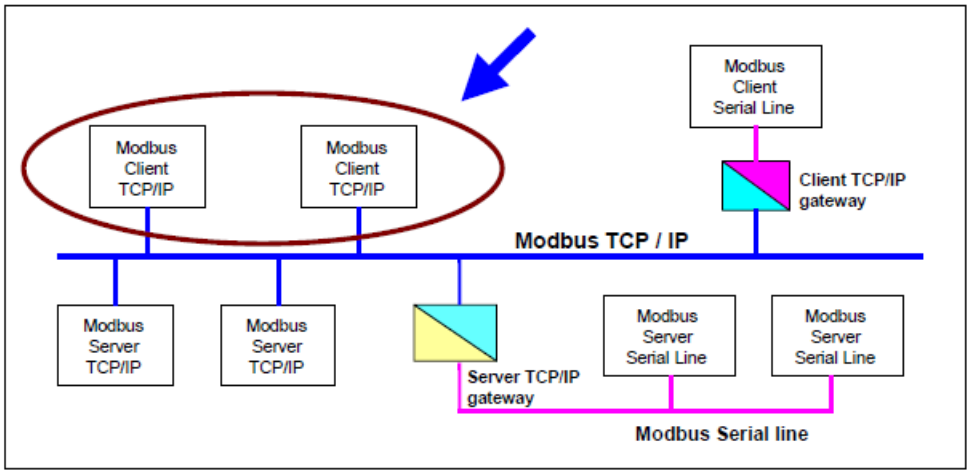

Figura 15: Dispositivi *Modbus TCP/IP client*

Un *client* Modbus può ricevere tre eventi:

- Una nuova richiesta dall'applicazione utente di inviare una richiesta.
  In questo caso una richiesta deve essere codificata e inviata sulla rete utilizzando il servizio del componente di gestione *TCP*.

- Una risposta dal gestore *TCP*.
  In questo caso il *client* deve analizzare il contenuto della risposta e inviare una conferma all'applicazione utente.

- La scadenza di un timer di una mancata risposta.
  In questo caso può essere effettuato un nuovo tentativo oppure viene inviato un messaggio di errore all'applicazione utente.

3.7.1 MODBUS Request

Il *client* a seguito di una richiesta dall'applicazione utente, deve costruire la *richiesta Modbus* da inviare al *gestore TCP*.

- Viene instanziata una transazione per permettere al *client* di memorizzare tutte le informazioni richieste per recuperare le risposte dal *server*.

- La richiesta viene codificata. Vengono composti i campi da inserire nella *PDU* ed i campi dell'header *MBAP* vengono riempiti.

- La richiesta viene passata al modulo di gestione *TCP*.

3.7.2 MODBUS Confirmation

Quando il *client* riceve il frame di risposta dal *server*, il campo dell'identificazione di transizione della risposta viene confrontato con quello inviato dal *client*.
Se il confronto fallisce la risposta viene scartata. Se invece i due valori corrispondono la risposta viene analizzata.
Il parsing della risposta consiste nella verifica dell'header *MBAP* e nella *PDU* di risposta.
Dopo aver verificato che il valore del campo *"Protocol Identifier"* contiene il valore zero, si ottiene la lunghezza della risposta contenuta nel campo *Length*.
Per quanto riguarda la verifica della *PDU* viene verificato il codice funzione nella risposta dal *server*. Se il codice funzione è lo stesso inviato dal *client* nella richiesta, avviene un riscontro positivo.
Nel caso il campo del *Function Code* di risposta contenga un codice di eccezzione, questa viene notificata all'applicazione utente.

3.8 MODBUS Server

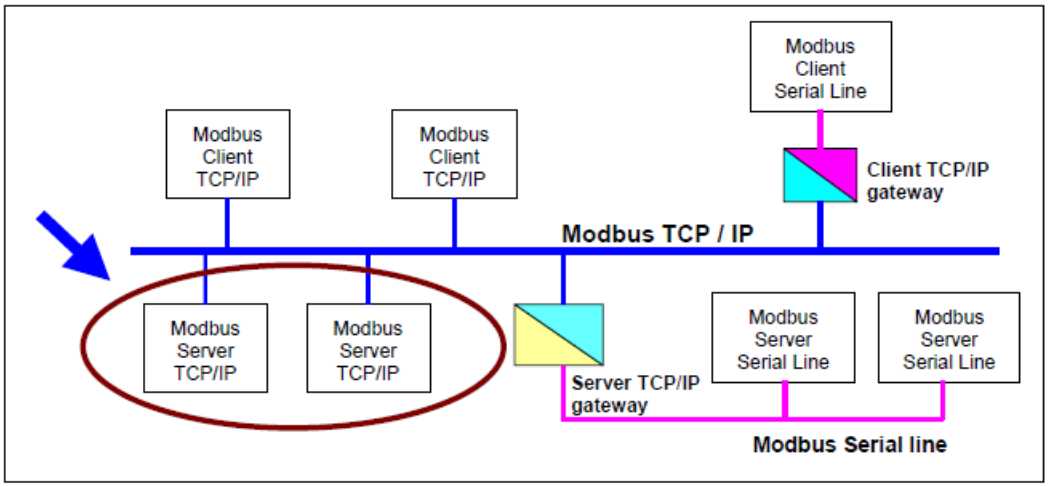

Figura 16: Dispositivi *Modbus TCP/IP server*

Il ruolo di un *server Modbus* è quello di fornire alle applicazioni *client* l'accesso ad oggetti e servizi messi a disposizione.
Il *server* deve mappare gli oggetti dell'applicazione utente in oggetti *Modbus* leggibili e scrivibili in modo da fornire i servizi richiesti.
Viene analizzata la richiesta ricevuta e fornisce una risposta al *client*.
Il *server* può servire più richieste contemporaneamente.

3.8.1 Verifica della PDU

Una volta ricevuto il messaggio, il *server* deve analizzarne il contenuto, per verificarne la correttezza e fornire una risposta.
Per prima cosa si procede all'analisi dell'header *MBAP*, analizzando il contenuto del campo *Protocol Identifier*, verificando se il suo valore è zero. Se il confronto fallisce il messaggio viene scartato. Altrimenti si procede con la creazione di una transazione *Modbus*. Nella transazione creata vengono inseriti:

- Identificatori della connessione *TCP*.
- Valore del campo *Transaction ID*.
- Valore del campo *Unit Identifier*.

Successivamente viene analizzato il *PDU* del messaggio. Una volta controllato il codice funzione, con esito positivo, il *server* recupera le informazioni richieste

dall'applicazione utente e costruisce il messaggio di risposta da restituire al *client* con le relative informazioni.

3.8.2 Costruzione della risposta MODBUS

A seconda del risultato dell'elaborazione della richiesta ricevuta dal *client*, possono essere costruiti due tipi differenti di risposte:

- Risposta positiva: il campo *Function Code* contiene lo stesso valore della richiesta.
- Eccezzione: il campo *Function Code* contiene il valore della richiesta + 80 mentre il campo dati contiene il il codice che indica la causa dell'errore.

Exception Code	MODBUS name	Comments
01	Illegal Function Code	The function code is unknown by the server
02	Illegal Data Address	Dependant on the request
03	Illegal Data Value	Dependant on the request
04	Server Failure	The server failed during the execution
05	Acknowledge	The server accepted the service invocation but the service requires a relatively long time to execute. The server therefore returns only an acknowledgement of the service invocation receipt.
06	Server Busy	The server was unable to accept the MB Request PDU. The client application has the responsibility of deciding if and when to re-send the request.
0A	Gateway problem	Gateway paths not available.
0B	Gateway problem	The targeted device failed to respond. The gateway generates this exception

Figura 17: Codici eccezione restituiti dal *server Modbus TCP/IP*

La *PDU* di risposta deve essere preceduta dall'intestazione *MBAP* che viene costruita utilizzando i valori memorizzati nella transazione creata durante l'analisi della richiesta.
Vengono inseriti i valori nei campi:

- Unit Identifier: contiene il valore contenuto nel campo della richiesta.
- Length: il server calcola la dimensione del *PDU* sommato alla dimensione in byte del campo *Unit Identifier*.
- Protocol Identifier: viene impostato a zero.
- Transaction Identifier: viene inserito lo stesso valore della richiesta.

4 Lavoro svolto

4.1 ModBus Client

L'obbiettivo principale del progetto è lo sviluppo di un *client* minimale che sfrutti le funzioni messe a disposizione dal protocollo *Modbus TCP/IP*.
Per lo sviluppo è stata utilizzata una dei numerose librerie *open source* disponibili in rete che implementano lo scambio dei messaggi da *client* e *server*. In questo esempio verrà utilizzata *EasyModbusTCP.Java*
L'applicativo invia le richieste in base alle funzioni del protocollo e riceve un riscontro da parte del *server*, mostrando i risultati ottenuti o eventuali errori.
Il progetto si pone come base per un più complesso applicativo di supervisione come uno *SCADA* o un *HMI* realizzabile in *JAVA*.

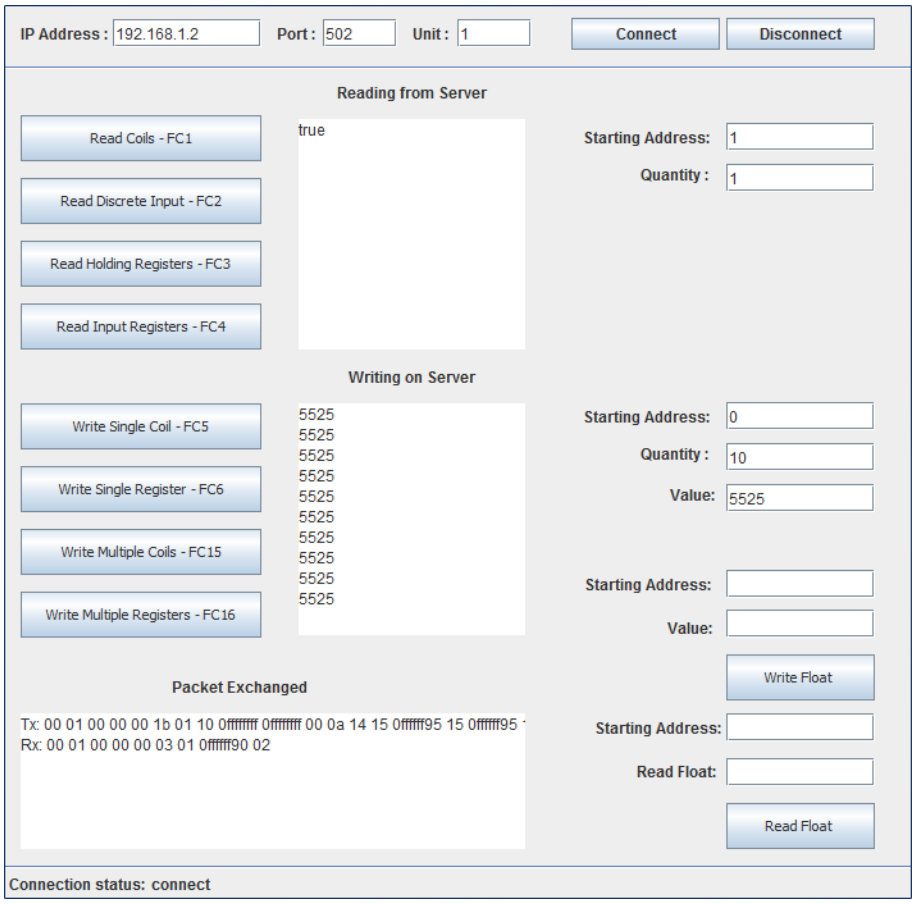

Figura 18: Interfaccia utente

4.2 Strumenti utilizzati

L'intero lavoro è stato svolto in ambiente Windows, utilizzando un simulatore *server* sulla stessa macchina su cui era avviato il *client*.
Successivamente è stato utilizzato un reale dispositivo *server* presente in rete.
Eclipse Neon. Lo strumento di sviluppo per JAVA. Utilizzato nella fase di implementazione e di testing.
Wireshark. Software per l'analisi dei pacchetti in transito sull'interfaccia di rete. Utilizzato per verificare la corretta composizione dei pacchetti inviati e ricevuti.
EasyModbusTCP.Java. Libreria JAVA che implementa lo scambio dei messaggi tramite protocollo *Modbus*.
Disponibile al seguente link: `www.easymodbustcp.net`.
EasyModbusServerSimulator. Simulatore per server *Modbus* che implementa la mappatura dei vari registri del dispositivo.
Disponibile al seguente link: `www.easymodbustcp.net`.

4.3 Lettura registri

L'applicazione permette di leggere il valore dei registri del *server*, utilizzando le funzioni di lettura messe a disposizione dalla libreria.

- boolean[] ReadCoils(int startingAddress, int quantity) - FC1
- boolean[] ReadDiscreteInputs(int startingAddress, int quantity) - FC2
- int[] ReadHoldingRegisters(int startingAddress, int quantity) - FC3
- int[] ReadInputRegisters(int startingAddress, int quantity) - FC4

4.4 Scrittura registri

L'applicazione permette di scrivere un valore nei registri del *server*, utilizzando le funzioni di scrittura messa a disposizione dalla libreria.

- void WriteSingleCoil(int startingAddress, boolean value) - FC5
- void WriteSingleRegister(int startingAddress, int value) - FC6
- void WriteMultipleCoils(int startingAddress, boolean[] values) - FC15
- void WriteMultipleRegisters(int startingAddress, int[] values) - FC16

4.5 Analisi messaggi scambiati

Nelle immagini che seguono, applicando un filtro di ricerca al protocollo in esame, è possibile osservare il contenuto dei vari pacchetti scambiati tra il dispositivo *client* e il dispositivo *server*.

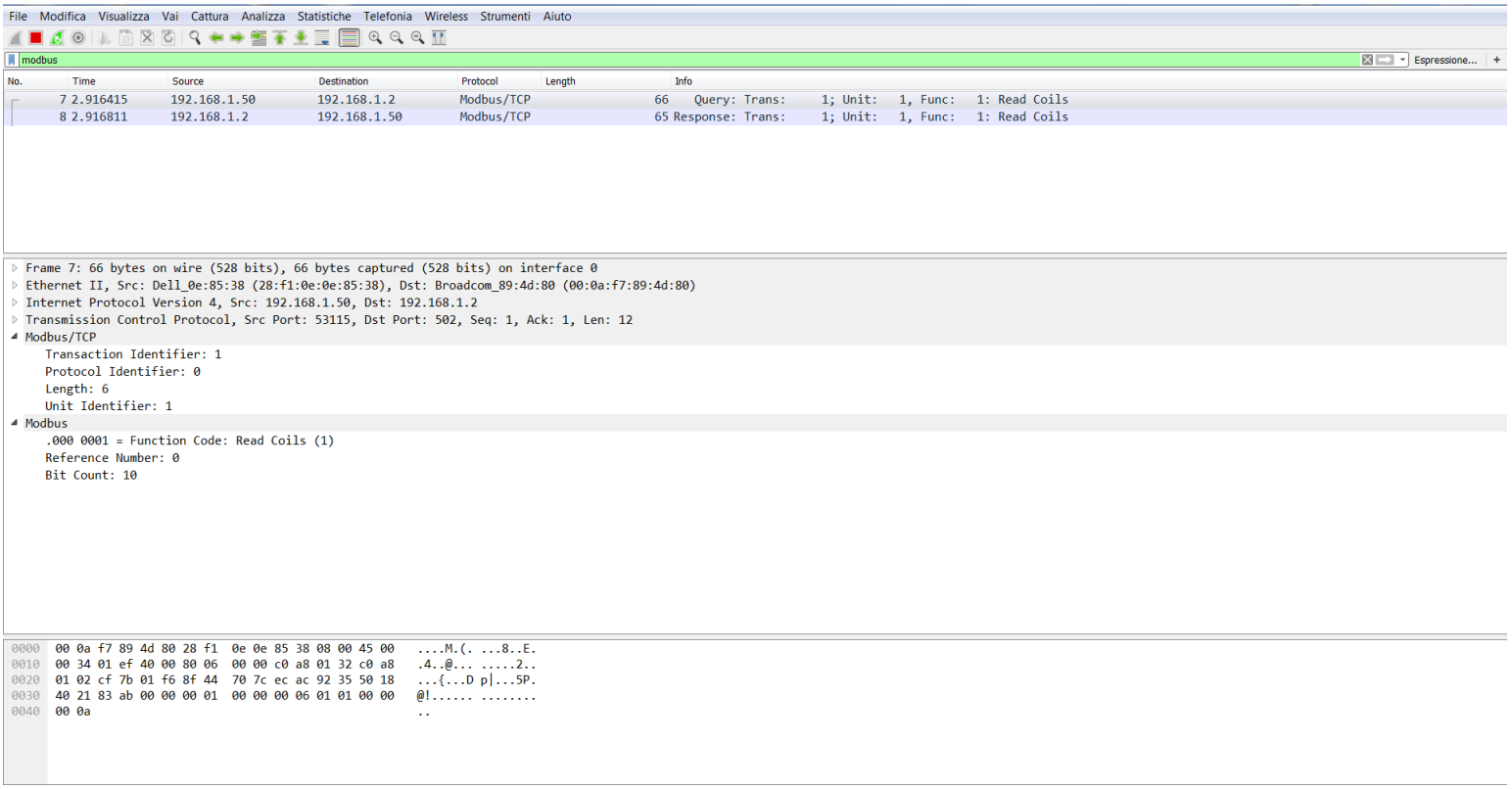

Figura 19: Richiesta di lettura *Coils*

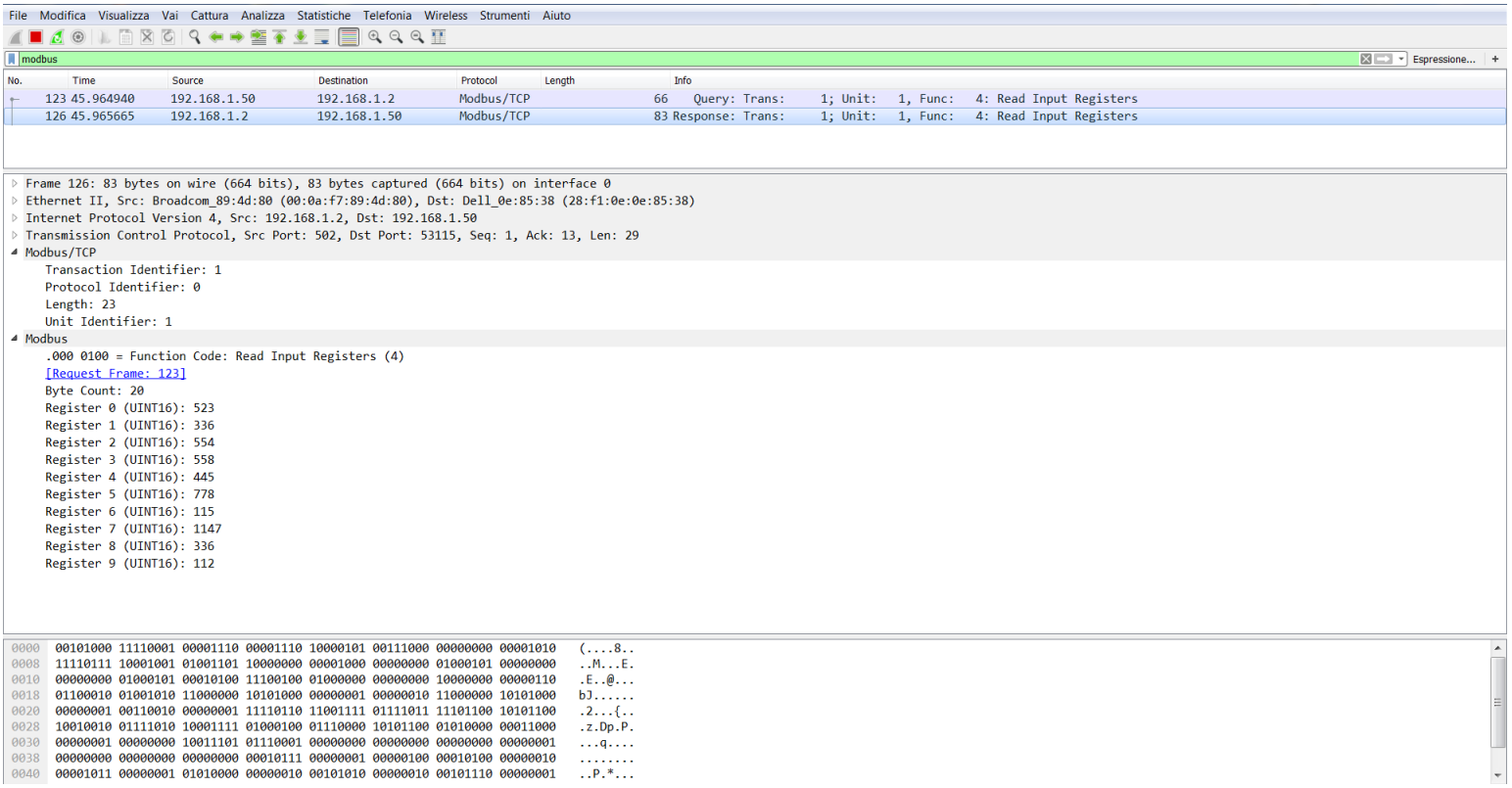

Figura 20: Risposta ad una richiesta di lettura *Input Registers*

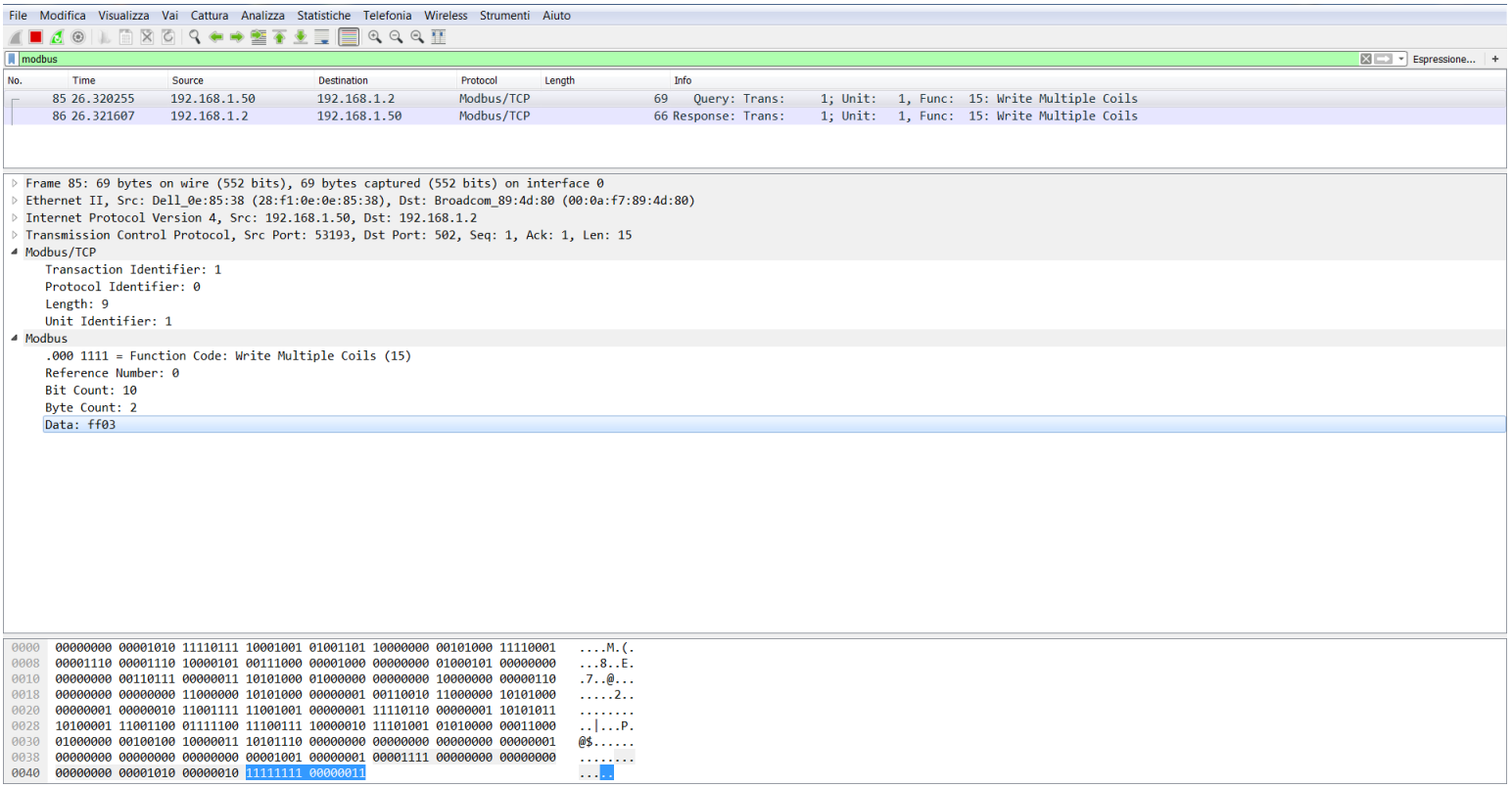

Figura 21: Richiesta di scrittura *Multiple Coils*

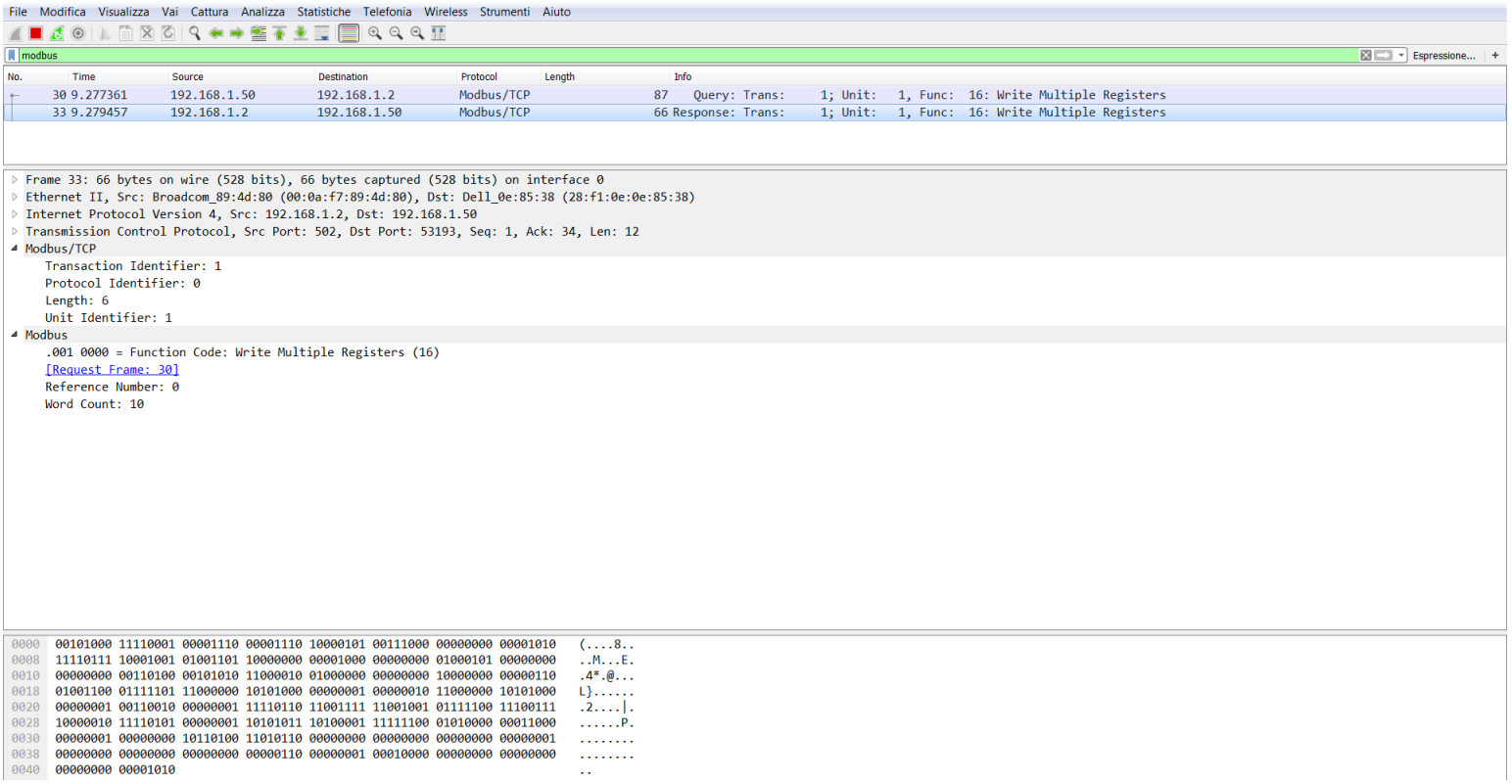

Figura 22: Risposta ad una richiesta di scrittura *Multiple Registers*

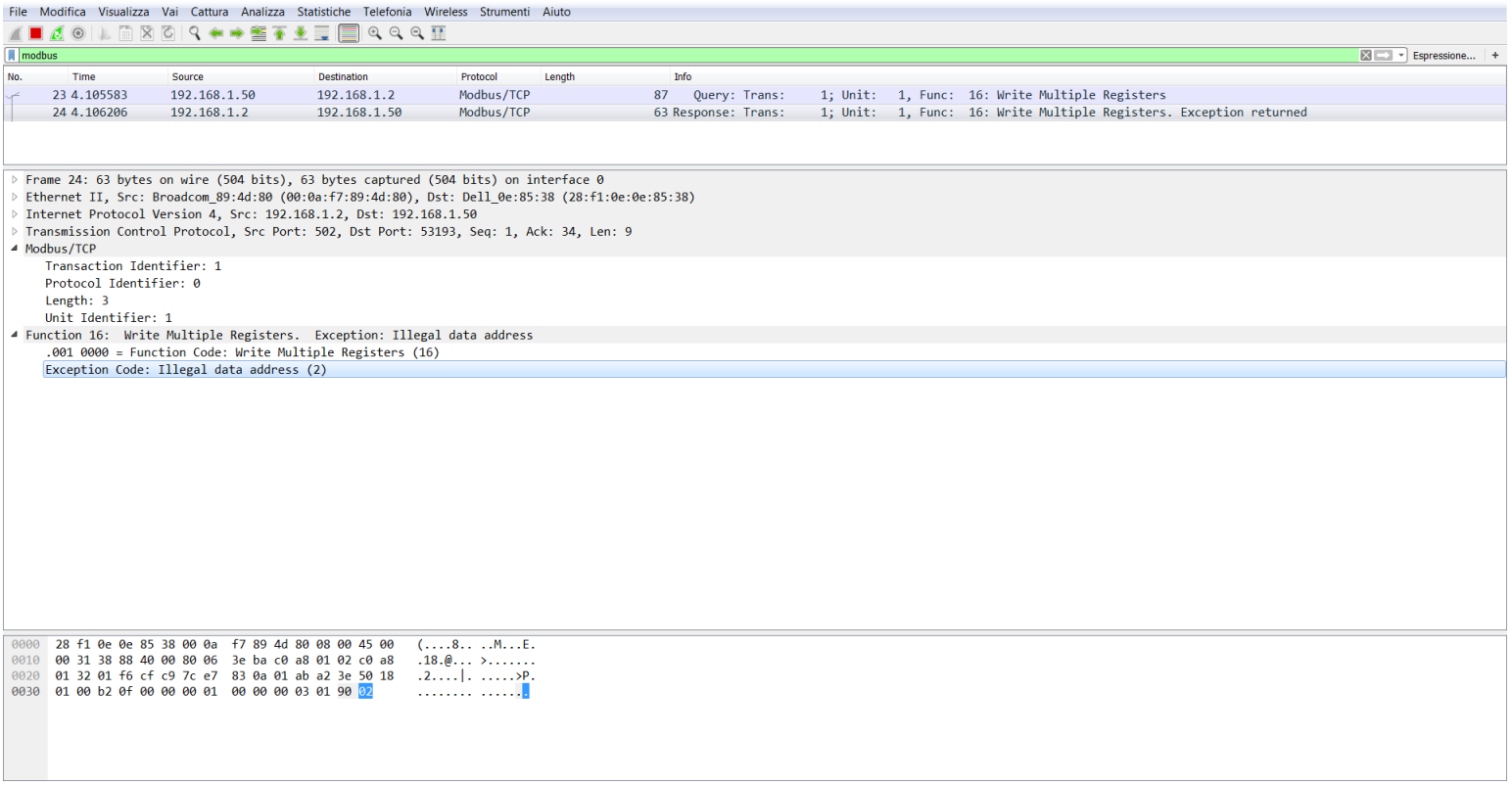

Figura 23: Richiesta di scrittura ad un indirizzo errato

5 Sviluppi futuri

Allo stato attuale il *client* può inviare richieste di scrittura e lettura ai registri in modo manuale, ovvero inserendo i parametri (indirizzo registri, quantità da elaborare) manualmente premendo un bottone per inviare la richiesta.
Alcuni miglioramenti da apportare potrebbero essere:

- Migliorare l'interfaccia grafica.
- Automatizzare la gestione delle richieste, in modo da interrogare e scrivere ciclicamente i valori all'interno dei registri, rendendo possibile l'utilizzo dell'applicativo come vero supervisore.
- Aggiungere la gestione di valori a 64 bit.

Riferimenti bibliografici

[1] modbus.org, MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3 , 26 Aprile 2012;

[2] modbus.org, MODBUS MESSAGING ON TCP/IP IMPLEMENTATION GUIDE V1.0b , 24 Ottobre 2006;