Università degli Studi di Padova

**FACOLTÀ DI INGEGNERIA ELETTRONICA ED INFORMATICA**

**CORSO DI LAUREA IN INGEGNERIA ELETTRONICA**

# INTEGRAZIONE DEI PROTOCOLLI TCP/IP E ZIGBEE IN UN NETWORK CAPABLE APPLICATION PROCESSOR (NCAP) IEEE 1451

**LAUREANDO:**

Devis Biacco

**RELATORE:**

Chiar.mo Prof. Claudio Narduzzi

Padova, 18 Febbraio 2010

Anno Accademico 2009/2010


UNIVERSITÁ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

**TESI DI LAUREA**

# INTEGRAZIONE DEI PROTOCOLLI TCP/IP E ZIGBEE IN UN NETWORK CAPABLE APPLICATION PROCESSOR (NCAP) IEEE 1451

LAUREANDO: Devis Biacco

RELATORE: Prof. Claudio Narduzzi

**CORSO DI LAUREA IN INGEGNERIA ELETTRONICA**

Padova, 18 Febbraio 2010

Anno Accademico 2009/2010

*Alla mia famiglia*

*e ad Alessandra.*

**Sommario**

Lo scopo di questa tesi consiste nell'integrare all'interno di un sistema di misura basato su architettura IEEE 1451 il protocollo di comunicazione TCP/IP. Il lavoro svolto é introdotto da una prima sezione in cui vengono fornite le informazioni sullo standard 1451 preso in considerazione per il progetto. Nei capitoli successivi viene introdotto e spiegato il protocollo TCP/IP senza entrare nei dettagli. Segue una breve descrizione dello Stack TCP/IP della Microchip analizzando poi le scelte effettuate nella modifica, nell'intento di utilizzare meno spazio di memoria possibile all'interno dell'NCAP. Viene spiegato e analizzato brevemente come avviene la comunicazione Client/Server con un applicativo di Test. L'eventuale futuro inserimento di opportuni comandi "AD HOC" permetterá l'effettiva interrogazione da remoto del sistema di misura 1451.

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Introduzione

## Introduzione allo standard 1451

L'interfacciamento di trasduttori richiede, per garantire l'interoperabilitá a livello di rete per ogni nodo, che vi sia un protocollo di comunicazione comune per gestire i dati provenienti dagli stessi, facilitandone le operazioni di controllo e di configurazione. Senza una direttiva standard qualsiasi modifica agli elementi hardware o alla piattaforma di rete comporterebbe malfunzionamenti e notevoli costi in termini economici e temporali. Queste considerazioni hanno dato vita alla famiglia di standard IEEE 1451,un insieme di documenti elaborati negli ultimi dieci anni dall'Institute of Electric and Electronic Engineers, che prevede la definizione di un'interfaccia standard per una rete di sensori intelligenti. Questa famiglia di standard definisce un'architettura di base della rete, che consente di modificarne la configurazione in modalitá plug and play. Oggetto di questa tesi é lo sviluppo dell'interfaccia verso una rete Ethernet che permette l'accesso alle funzionalitá della rete di trasduttori.

# Capitolo 1

# Standard 1451

La direttiva 1451 considera un modello di dispositivo intelligente programmabile ed indipendente dalla rete a cui é collegato dotato di un interfaccia digitale e di un protocollo di comunicazione di accedere al trasduttore via microprocessore. Una volta definite le caratteristiche principali delle reti, l'IEEE ha definito in maniera piú dettagliata le caratteristiche dei singoli nodi, classificandoli in due distinte categorie:

- TIM(Transducer Interface Module): dispositivi a cui spetta la gestione del sensore/-trasduttore/attuatore , il condizionamento dell'informazione rilevata, la conversione di tali informazioni e trasmissione delle stesse; sono considerati dei nodi terminali di rete e si appoggiano ai nodi della seconda categoria: gli NCAP.

- NCAP(Network Capable Application Processor): dispositivi ai quali spetta la gestione della rete e l'inoltro di messaggi che circolano in essa. Tali dispositivi operano da Gateway tra gli utenti della rete ed i TIM , nel caso fosse previsto l' accesso da esterno.La famiglia di standard 1451 non definisce l'interfaccia fisica tra l' NCAP e la rete , ma i tipi di interfaccia che possono esistere tra l ' NCAP e i TIM.

I protocolli sono gli standard che specificano come avvengono i trasferimenti da una macchina ad un'altra. Essi specificano come sono rappresentati i dati, le tecniche per la rivelazione d'errore e il meccanismo di acknowledgement per i pacchetti trasmessi. Infine tali protocolli rendono invisibile all'utente l'hardware sottostante durante una qualsiasi sessione di lavoro. Per comprendere l'organizzazione dei protocolli della famiglia IEEE

1451 é utile osservare la Figura **??**. La trattazione riguarderá solamente gli aspetti piú rilevanti di ogni direttiva, ovvero quelli concernenti il progetto, poiché si tratta di standard complessi che interessano diverse tipologie di interfacce fisiche ed offrono talvolta funzionalitá molto avanzate, non sviluppate nel progetto.

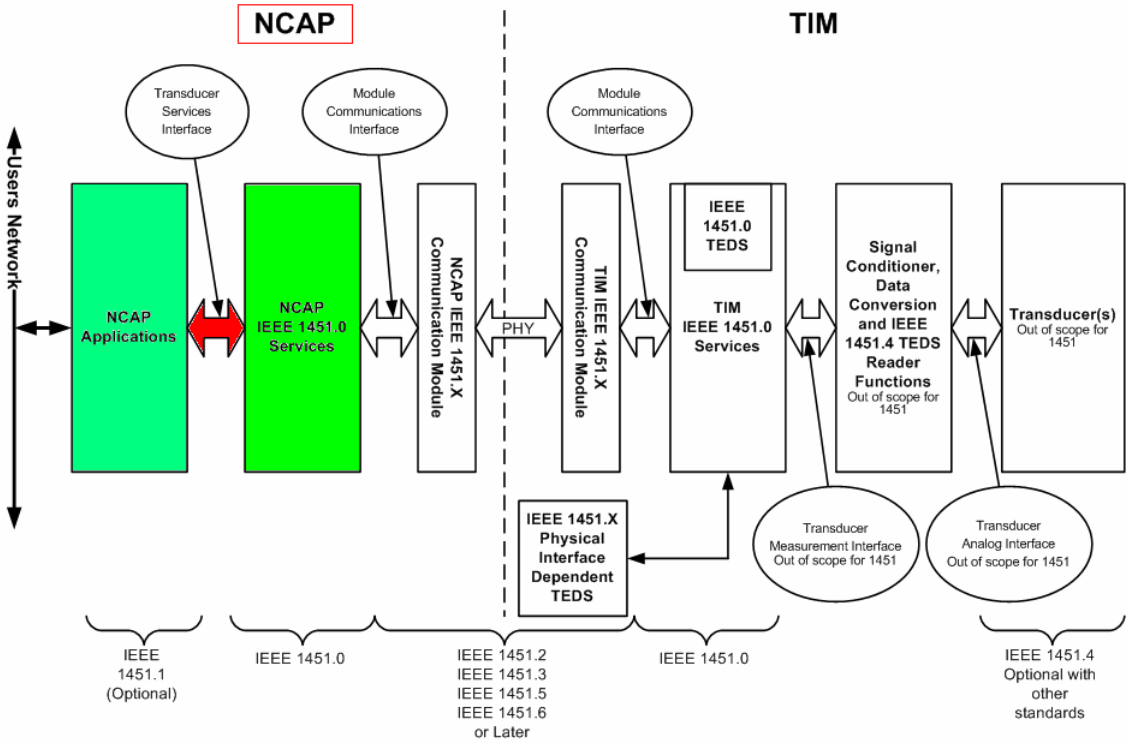


Figura 1.1: *Modello di riferimento*

A partire da sinistra, ovvero dall'NCAP, si incontra prima l'applicazione lato utente,la quale si interfaccia al livello 1451.0 tramite una Application API (AAPI). Il livello 1451.0 si interfaccia a sua volta con il sottostante 1451.5 tramite una Communication API (CAPI).I l 1451.5 invece é collegato ad un livello fisico da esso supportato (nel caso di questo progetto ZigBee ). Proseguendo sempre verso destra, in maniera speculare, si incontrano il livello fisico (ZigBee), il livello 1451.5 e il livello 1451.0 del WTIM, interfacciati tra di loro ancora tramite CAPI e AAPI rispettivamente. Infine, il livello 1451.0 del WTIM,tramite una Transducer Measurement API, é connesso al trasduttore (il quale puó implementare il protocollo 1451.4, della stessa famiglia).

## 1.1 1451.5

Lo standard IEEE 1451.5 introduce il concetto di Wireless Interface Transducer Module (WTIM) connesso ad un Network-Capable Application Processor (NCAP) Service Module attraverso un canale radio approvato tra IEEE 802.11, IEEE 802.15.4, IEEE Bluetooth e ZigBee. Secondo le direttive 1451.5, un WTIM é un dispositivo che comprende una Dot 5 Approved Radio (Dot5AR), il condizionamento del segnale, la conversione analogica/digitale e uno o piú trasduttori (sensori/attuatori). Poiché i WTIM possono avere interfacce Dot5AR diverse tra loro, l'NCAP dovr|'a avere almeno una Dot5AR per ogni tipo presente nei WTIM a cui deve essere associato. Questo standard si focalizza sull'interfaccia di comunicazione tra WTIM ed NCAP attraverso i protocolli Dot5AR, stabilendo di fatto i metodi e i formati di dati necessari a governare i trasduttori, per le operazioni di rete e per i TEDS . Lo standard in questione é basato unicamente su interfacce wireless, ma non specifica le caratteristiche fisiche e tecniche né dei trasduttori né del sistema wireless. La nascita di questo protocollo é dovuta alla volontá di uniformare le specifiche ed accomunare piú tecnologie sotto un unico standard aperto, riducendo al minimo il rischio di incompatibilitá e i costi di produzione. Le specifiche dell'IEEE 1451.5 riguardanti ZigBee indicano i requisiti che una rete di tale tipo deve avere affinché possa fungere da rete di trasporto per un sistema compatibile con IEEE 1451.

## 1.2 Protocollo HTTP per 1451

Il protocollo HTTP viene utilizzato per trasferire o trasmettere informazioni sul World Wide Web. HTTP é un protocollo Client/Server tramite il quale due processori possono comunicare su una connessione di tipo TCP/IP. Un Server HTTP é un programma che risiede in un processore il quale rimane in ascolto di eventuali richieste HTTP.Un Client HTTP apre una connessione TCP/IP verso il Server tramite un socket, trasmette una richiesta, e pio aspetta una risposta dal server. Per questo standard IEEE, il modello "client-server", usualmente chiamato HTTP, é simile al livello 1451.0 "NCAP-Utente_Finale". L'NCAP puó essere visto come il Server il quale "serve" i dati al network

connesso, e l'Utente_Finale come il Client il quale riceve i dati del sensore inviati al Server e invia dati e comandi al Server per comandare gli attuatori.

HTTP Client Request: il Client invia un messaggio di richiesta formattato in base alle regole dello standard HTTP (*HTTP Request*); questo messaggio contiene la variabile che il client richiede e l'informazione da fornire al server.

HTTP Server Reponse: il Server legge e interpreta la richiesta, eseguendo quanto richiesto e creando un messaggio HTTP di risposta, che trasmette al client; indica se la richiesta ha avuto successo e puó contenere il contenuto della variabile che il client ha richiesto.

L'HTTP definisce otto metodi i quali indicano la funzione che si desidera eseguire da un particolare dispositivo identificato. Questi metodi includono: GET, POST, HEAD, PUT, DELETE, TRACE, e OPTIONS. In questa API (Application Program Interface) si utilizzano solo i metodi GET e POST.

**GET** : recupera qualsiasi informazione si trovi nella stringa di richiesta URI(Uniform Resource Identifier); Questo é in genere il funzionamento di un modulo di richiesta(query). L'interazione é piu simile a una domanda, come una query, un'operazione di lettura o di ricerca. Puó essere utilizzato per recuperare i dati dal server. In questa API, GET puó essere utilizzato per leggere e scrivere i dati e ie TEDS del trasduttore.

**POST** : richiede che il server di origine accetti la variabile racchiusa nella richiesta URI come una nuova variabile; l'interazione cambia lo stato della variabile nel modo richiesto dall'utente, oppure tiene conto dei risultati delle interazioni; questo metodo puó essere utilizzato per dare o ricevere un comando da parte dell'utente: in particolare in quest'API puó essere utilizzato per leggere o scrivere sia i dati sia i TEDS.

## 1.3 IEEE 1451.0 HTTP API

Il TSI(Transducer Services Interface) é un API NCAP utilizzato da applicazioni di misura e di controllo per accedere al livello IEEE 1451.0. Contiene funzioni per leggere e scrivere i TransducerChannels, per leggere e scrivere i TEDS, e inviare comandi di configurazione, controllo e operazioni ai TIMs. Il TSI del livello 1451.0 contiene cinque interfacce: TransducerAccess, TransducerManager, TimDiscovery, TEDSManager, e AppCallback. Le prime quattro interfacce sono implementate dal livello 1451.0 e utilizzate dalle applicazioni di misura. Se l'applicazione ha bisogno di servizi opzionali avanzati, é necessario implementare l'interfaccia AppCallback ,la quale invoca il livello 1451.0. Le API IEEE 1451.0 HTTP sono di seguito elencate e spiegate:

**Discovery** : in questa interfaccia sono sviluppati metodi di applicazioni per ricercare moduli di comunicazione IEEE 1451.X disponibili, TIMs e TransducerChannels.

**TransducerAccess** : utilizzati quando un'applicazione richiede di avere accesso a uno dei TransducerChannels di un particolare trasduttore o attuatore.

**TransducerManager** : utilizzati con applicazioni che hanno bisogno di maggior controllo sull'accesso al TIM.

**TEDSManager** : utilizzati per leggere e scrivere i TEDS

**AppCallback** : utilizzati per applicazioni avanzate(é necessario implementare questa interfaccia)

Si concentrano principalmente sull'accesso ai dati e ai TEDS del trasduttore utilizzando il protocollo HTTP(la Figura 1.2 mostra come l'HTTP interagisce con l'NCAP nel quale risiede un HTTP Server)[1]. Gli utenti possono inviare una richiesta HTTP al server nell'NCAP e avere una risposta dal server stesso. L'intero processo viene di seguito descritto:

1. un utente(client) invia una richiesta HTTP al server nell'NCAP

[1]Nella Figura1.2 "S" significa Sensore e "A" significa Attuatore

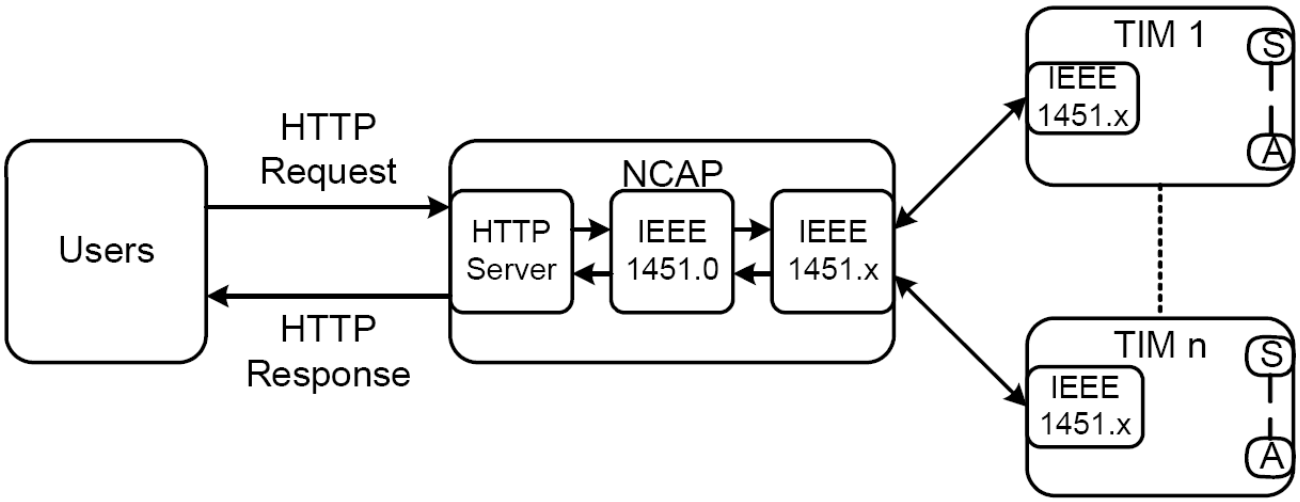


Figura 1.2: *Accesso HTTP nell'IEEE 1451.0 NCAP*

2. il server riceve una richiesta HTTP. la processa, e richiama la relativa IEEE 1451.0 API

3. l'IEEE 1451.0 API chiama l'IEEE 1451.X API per comunicare con l'IEEE 1451.X TIM e per ricevere i dati da esso

4. il server nell'NCAP riceve i dati dall'IEEE 1451.0 API e inoltra la risposta HTTP all'utente

## 1.4 Formato del messaggio HTTP di richiesta

Viene ora descritto come utilizzare il protocollo "HTTP" per inviare messaggi da un client remoto verso un NCAP, che in questo modello si comporta come un server, ritornando al client i dati del trasduttore(la Tabella1.1 spiega il formato del messaggio HTTP, la Tabella1.3 mostra dei possibili esempi di possibili argomenti e la Tabella1.4 mostra la lista delle API HTTP). Il messaggio HTTP da un client remoto viene trasmesso su una rete che si connette il client remoto con l'IEEE 1451.X nodo trasduttore. Il messaggio aderisce alla la sintassi URL HTTP secondo la seguente forma:

http://<host>:<port>/<path>?<parameters>

### 1.4.1 Formato del messaggio HTTP di risposta

Le specifiche o gli argomenti di output, ad esempio l'ArgumentArray. puó essere incluso nella richiesta HTTP come pure nella risposta alla richiesta stessa. Il formato di usci-

Tabella 1.1: Stack Configuration Definitions

| Campo | Definizione | Esempio |
|---|---|---|
| <host>: | la porzione host di questo stato include il nome del dominio del nodo target IEEE 1451 | <host> = ”192.168.1.91” |
| <port> | Il numero della porta é opzionale e di default é messo 80 se non viene specificato nella richiesta. Puó vantaggioso selezionare una porta inutilizzata # (not port #80) che diventa la porta principale dell'IEEE 1451. | <port>=”80” |
| <path> | il path indica l'indirizzo dell'IEEE1451 nel comando stesso | <path>=”1451/ TransducerAccess /ReadData” |
| <parameters> | i parametri associati con il comando | <parameters>=”timId=1 &channelId=2&timeout=14 &samplingMode=continuous&format=text” |

Tabella 1.3: Esempi di parametri

| tipo API | Nome | path |
|---|---|---|
| Discovery API | Discovery API | 1451/Discovery/TIMDiscovery |
| | TransducerDiscovery | 1451/Discovery/TransducerDiscovery |
| Transducer Access API | ReadData | 1451/TransducerAccess/ReadData |
| | StartReadData | 1451/TransducerAccess/StartReadData |
| | MeasurementUpdate | 1451/TransducerAccess/MeasurementUpdate |
| | WriteData | 1451/TransducerAccess/WriteData |
| | StartWriteData | 1451/TransducerAccess/StartWriteData |
| TEDS Manager API | ReadTeds | 1451/TEDSManager/ReadTeds |
| | ReadRawTeds | 1451/TEDSManager/ReadRawTeds |
| | WriteTeds | 1451/TEDSManager/WriteTeds |
| | WriteRawTeds | 1451/TEDSManager/WriteRawTeds |
| | UpdateTedsCache | 1451/TEDSManager/UpdateTedsCache |
| Transducer Manager API | SendCommand | 1451/TransducerManager/SendCommand |
| | StartCommand | 1451/TransducerManager/StartCommand |
| | CommandComplete | 1451/TransducerManager/CommandComplete |
| | Trigger | 1451/TransducerManager/Trigger |
| | StartTrigger | 1451/TransducerManager/StartTrigger |

Tabella 1.4: HTTP API

| Campo | Definizione | Esempio |
|---|---|---|
| timId | l'identificatore del TIM selezionato | 1 |
| channelId | l'identificatore del TransducerChannel del TIM selezionato | 2 |
| timeout | quanto aspettare per effettuare una lettura prima di generare un time-out error | 14 |
| samplingMode | modo di campionatura | Continuous |
| format | il formato di ritorno desiderato: "text","HTML" o "xml" | text |

# Capitolo 2

# Protocollo TCP/IP

Per TCP/IP(Transmission Control Protocol/Internet Protocol) non si intende solo il protocollo di trasmissione TCP ed il protocollo di rete IP, ma una famiglia di protocolli(vedi Figura 2.1) comprendente anche l'UDP, l'ICMP, l'ARP, il RARP ed altri, da cui peró sono escluse le applicazioni quali la posta elettronica (E-mail), il trasferimento di file (FTP) e l'emulazione di terminale remota (TELNET)(nella Figura2.1 viene visualizzata la suite dei protocolli).

Il TCP/IP, al contrario di molti standard, diventati in seguito protocolli, é nato prima come protocollo e successivamente é stato affinato per farlo diventare uno standard, effettuando prove sul campo; é in qualche modo uno standard cooperativo.

## 2.1 Il modello a strati

Il software TCP/IP é organizzato concettualmente in quattro livelli piú un quinto, costituito dal supporto fisico vero e proprio (nella Figura2.2 vengono riportati in uno schema a blocchi):

**Application Layer** : A livello piú alto, l'utente invoca i programmi applicativi che permettono di accedere ai servizi disponibili attraverso internet; tale livello riguarda tutte le possibili opzioni, chiamate e necessitá dei vari programmi. In pratica gestisce l'interattivitá tra l'utente e la macchina. Un programma applicativo interagisce con uno dei

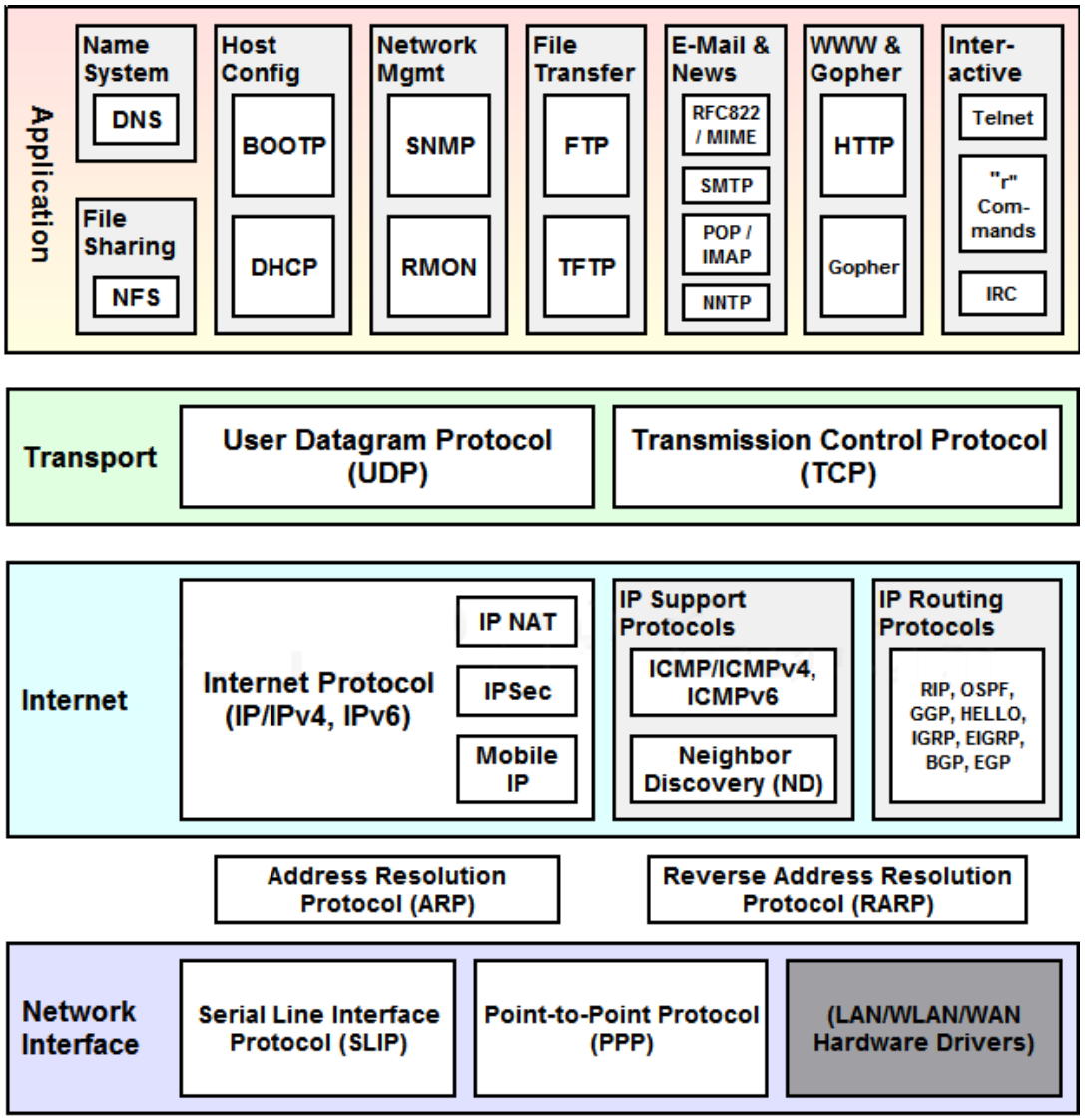


Figura 2.1: *Suite Protocolli TCP/IP*

protocolli di livello trasporto per inviare o ricevere dati e li passa al livello trasporto nella forma richiesta.

**Transport Layer** : Lo scopo primario del livello trasporto é consentire la connessione in rete fra due utenti ovvero permettere la comunicazione tra un livello applicativo ed un altro; una comunicazione di questo tipo é spesso detta end-to-end. Il software di tale livello divide il flusso di dati in pacchetti (di solito di circa 500 byte), che vengono passati insieme all'indirizzo di destinazione allo strato sottostante. Il livello di trasporto deve accettare dati da molti utenti contemporaneamente e, viceversa, deve smistare i pacchetti che gli arrivano da sotto ai vari specifici programmi; deve quindi usare dei codici appositi per indicare le cosiddette porte. Le routine di trasporto pacchetti aggiungono,

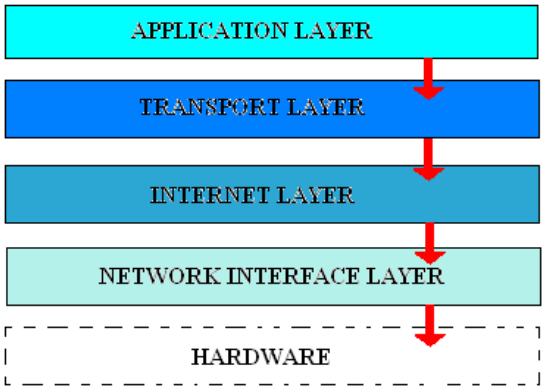


Figura 2.2: *Livelli Protocollo TCP/IP*

ad ogni pacchetto, alcuni bit in piú, i quali servono per codificare i programmi sorgente e destinazione. Il livello di trasporto puó regolare il flusso di informazioni e puó, nel caso del TCP (a differenza invece dell'UDP), fornire un trasporto affidabile assicurando che i dati giungano a destinazione senza errori ed in sequenza mediante un meccanismo di acknowledgement e ritrasmissione.

**Internet Layer (IP)** : Questo livello gestisce la comunicazione tra una macchina ed un'altra; accetta una richiesta di inoltro di un pacchetto da un livello di trasporto insieme all'identificazione della macchina alla quale il pacchetto deve essere inviato. |'E il livello piú caratteristico della Internet, detto appunto IP (Internet Protocol) che crea il datagramma di base della rete, sostanzialmente, riceve e trasferisce senza garanzie i pacchetti, che gli arrivano dal livello superiore, verso la macchina destinataria. Esso accetta i pacchetti TCP, li spezzetta se necessario e li incapsula nei datagramma di base IP, riempie gli header necessari ed usa l'algoritmo di routing per decidere a chi deve mandare questo pacchetto, in particolare se si tratta di un caso di routing diretto o di routing indiretto. Il livello Internet gestisce anche i datagrammi in ingresso, verifica la loro validitá ed usa l'algoritmo di routing per decidere se il datagramma deve essere inoltrato o processato localmente; in quest'ultimo caso il software elimina l'header del datagramma e sceglie quale protocollo di trasporto gestirá il pacchetto. In tale fase non solo si svolge la funzione di instradamento, ma si verifica anche la validitá dei pacchetti ricevuti. Inoltre questo livello gestisce integralmente i messaggi ICMP in ingresso ed uscita.

**Network Interface Layer** : Il quarto ed ultimo strato é costituito da una interfaccia di rete che accetta il datagramma IP e lo trasmette, previo incapsularlo in appositi frame, sull'hardware di rete (il cavo) tramite, ad esempio, un transceiver. Se necessario,come

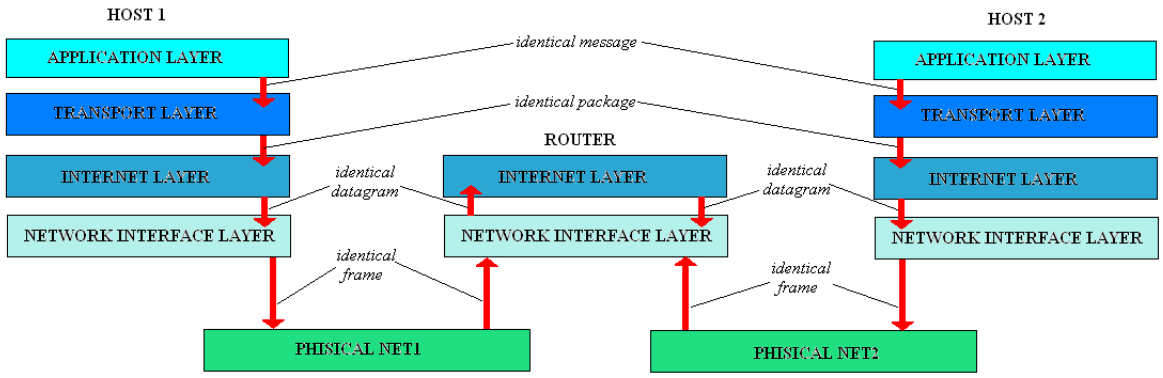


Figura 2.3: *Livelli Protocollo TCP/IP con Router intermedio*

illustrato in Figura 2.3, il pacchetto puó attraversare altre macchine intermedie (router) prima di giungere a destinazione, ma in queste penetra solo i due strati piú bassi dell'interfaccia di rete e del datagramma base IP. Uno dei vantaggi piú significativi di questa separazione concettuale é che diventa possibile, entro certi termini, sostituire una parte senza disturbare necessariamente le altre, cosicché ricerca e sviluppo possono procedere concorrentemente su ognuno dei tre livelli.

## 2.2 Confronto tra TCP/IP e il Modello ISO/OSI(ex: X.25)

La prima differenza tra i due modelli sta nel numero di strati, in particolare, sette per l'OSI e cinque(*quattro + Hardware*) per internet. Esistono due sottili ma importanti differenze fra lo schema a strati dell'internet e quello del X.25 che é il piú famoso protocollo aderente alla normative ISO. La prima riguarda l'affidabilitá del servizio di trasporto dati e la seconda la localizzazione dell'autoritá e controllo.

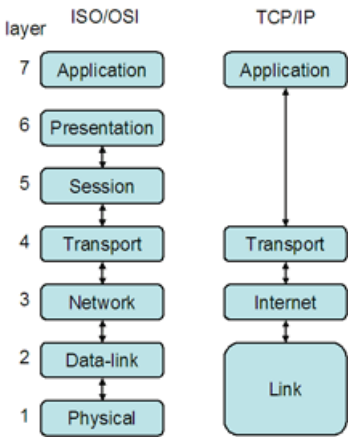


Figura 2.4: *Livelli dei due Protocolli a confronto*

Nel modello X.25 il software del protocollo verifica l'integritá dei dati ad ognuno dei primi quattro livelli (escluso ovviamente il livello fisico di rete). In particolare, i livelli due e tre, cioé del link e del network, includono oltre al checksum un meccanismo di timeout and retransmission, mentre il livello quattro del transport realizza l'affidabilit'a finale detta end-to-end. Questo crea dei problemi, poiché ogni operazione di checksum mette in rete un acknowledgement ed ogni volta che si eccede il timeout viene duplicato un pacchetto, tutte queste operazioni sono ripetute ad ogni nodo attraversato anche se di passaggio (nel senso che vi si penetra solo al livello minimo, cioé di link). Al contrario, nell'internet l'affidabilitá é solamente un problema end-to-end, infatti il livello di trasporto, e quindi la destinazione e la sorgente, é l'unico a gestire le ritrasmissioni e gli acknowledgement; i nodi di passaggio sono pressoché trasparenti anche se in effetti, hanno la capacitá di buttar via i pacchetti se sono corrotti o se i buffer sono pieni.

## 2.3 Internet Protocol Adresses

Affinché un sistema di comunicazione sia universale é necessario utilizzare un metodo di identificazione di ogni computer connesso ad esso (*host*). Il TCP/IP assegna ad ogni host, come identificatore universale, un indirizzo binario a 32 bits detto Internet Address o IP Address, usando una struttura analoga a quella degli indirizzi fisici di rete. Per rendere questi indirizzi piú comprensibili, essi sono suddivisi in quattro gruppi di bits con i rispettivi valori scritti in decimale e separati da punti (Dotted Decimal Notation).

Concettualmente, ciascun indirizzo IP é una coppia netid-hostid, dove il netid identifica la rete dove é connesso l'host, mentre l'hostid identifica lo stesso host su quella rete, come mostrato in figura2.5: Gli indirizzi IP sono divisi in cinque classi, di cui tre primarie (A,

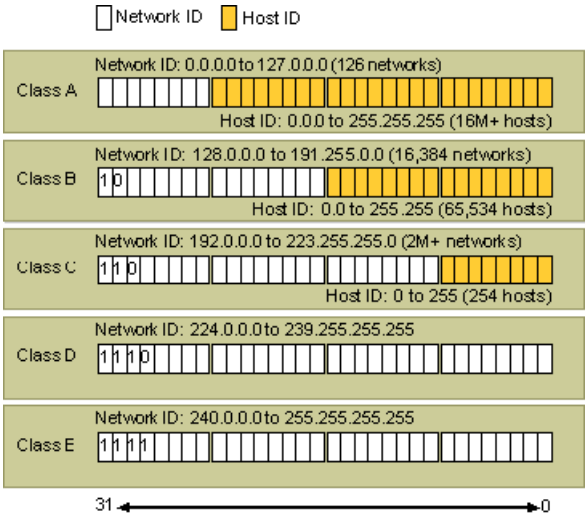


Figura 2.5: *Gruppi Indirizzi IP*

B, C) distinguibili dai tre bit di ordine piú alto.

**Classe A** : usata per reti con piú di 2 alla 16 (65536) hosts, dedica 7 bits per la netid e 24 per la hostid.

**Classe B** : usata per reti con un numero di hosts compreso tra $2^8$ (256) e $2^(16)$ (65536) hosts, dedica 14 bits per la netid e 16 per la hostid.

**Classe C** : usata per reti con meno di $2^8$ (256) hosts, dedica 21 bits per la netid e 8 per la hostid.

**Classe D** : usata per la particolare distribuzione dei dati, detta multicasting.

**Classe E** : destinata ad usi futuri.

Nella tabella di figura2.5 sono riportati i range della Dotted Decimal Notation corrispondenti in ciascuna classe degli indirizzi IP; alcuni valori sono riservati per scopi specifici (127.0.0.0 é riservato al local host)

Materialmente le reti sono interconnesse tramite computer molto veloci detti router che instradano i pacchetti leggendo l'indirizzo internet. Considerando che un router possa

essere connesso a due o piú reti fisiche, é necessario assegnare altrettanti indirizzi IP per identificare ciascuna rete: tali computers sono definiti multi-homed hosts. Poiché gli indirizzi IP codificano entrambi la rete e l'host ad essa connessa, essi non possono specificare un particolare computer, bensí una *connessione* ad una rete. Perció un router che connette $n$ reti necessita di altrettanti indirizzi IP. L'operazione di routing é molto efficace perché, in realtá, viene svolta leggendo solo la parte di indirizzo relativo alla rete. Uno dei vantaggi della struttura di indirizzamento Internet é che la sua forma puó specificare un indirizzo per un particolare host, una rete o tutti gli hosts su una rete (*Broadcast*). Lo svantaggio é che, se una macchina ha piú indirizzi, la conoscenza di un indirizzo di rete potrebbe essere non sufficiente per raggiungerla, se tale rete non é disponibile.

## 2.4 ARP(*Addresses Resolution Protocol*)

Gli indirizzi IP sono assegnati indipendentemente dagli indirizzi fisici di una macchina. I router utilizzano gli indirizzi internet, ma é bene sottolineare come due macchine qualsiasi possono comunicare solo se conoscono gli indirizzi fisici di rete; sorge quindi il problema di associare agli indirizzi IP quelli reali a livello fisico di ogni rete. Per risolvere tale problema esiste l'ARP (Address Resolution Protocol): una macchina, a partire dall'indirizzo IP, usa un messaggio broadcast (ARP request) per trovare l'indirizzo fisico di un'altra macchina. Tra tutte le macchine che ricevono l'ARP request, quella a cui corrisponde l'indirizzo IP risponde inviando il proprio indirizzo fisico alla macchina che ha inoltrato la richiesta.

## 2.5 RARP(*Reverse Addresses Resolution Protocol*)

Quando, per vari motivi, una macchina non conosce il proprio indirizzo IP, procede con il Reverse Address Resolution Protocol (RARP) emettendo un ARP-broadcast request ed indicando sé stesso come destinazione finale; sará compito di alcuni server di rete autorizzati a fornire il servizio RARP, rispondere fornendo l'indirizzo richiesto.

## 2.6 IP(*Internet Protocol*)

Concettualmente l'internet prevede tre tipi di servizi, come mostrato in Figura**??**, che hanno tra di loro una chiara gerarchia di dipendenze. A livello piú basso, un servizio di

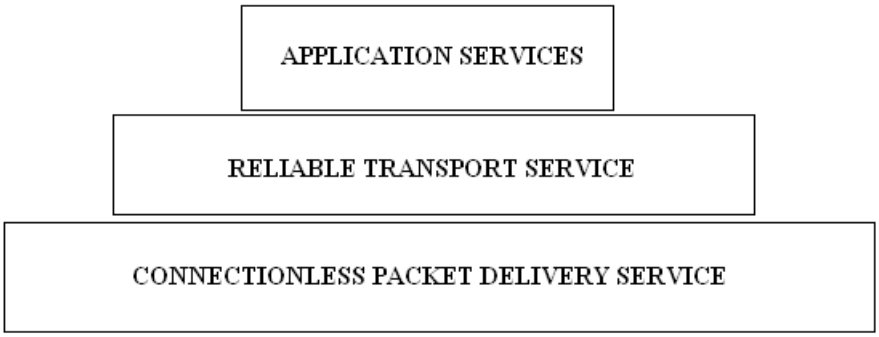


Figura 2.6: *Gruppi Indirizzi IP*

recapito *connectionless* rappresenta la base su cui si poggia ogni cosa; infatti il protocollo IP é fondamentale perché, anche se di per sé realizza una trasmissione non sicura, fornisce il supporto necessario per tutti gli altri protocolli affidabili (*reliable transport service*) quali il TCP e quindi per gli stessi applicativi (*application services*) quali l'FTP ed il TELNET. Tecnicamente, il protocollo IP é definito:

- **Unreliable**: é un servizio senza garanzie; i pacchetti possono essere persi, venire duplicati o consegnati fuori ordine senza per questo avvisare né l'utente sorgente né quello destinazione. Va comunque sottolineato che la perdita di un pacchetto avviene solamente in casi abbastanza eccezionali, quali la congestione totale di un componente o, addirittura, la caduta della rete.

- **Connectionless**: é un servizio non orientato alla connessione: ogni pacchetto viene trattato indipendentemente da tutti gli altri. Una sequenza di pacchetti spediti da un computer ad un altro potrebbero seguire cammini diversi per giungere a destinazione.

- **Best-effort**: il protocollo IP effettua un serio tentativo nell'inoltro dei pacchetti. In sostanza l'IP definisce l'unitá base del datagramma e la forma esatta di tutti i dati che passano attraverso l'Internet, gestisce le funzioni di routing (specificando il

percorso sul quale verranno spediti i dati) ed include un insieme di regole che caratterizzano le modalitá in base alle quali gli hosts ed i routers processano i pacchetti, la generazione dei messaggi di errore e le condizioni sotto le quali un pacchetto é scartato.

## 2.7 Il Datagramma IP

Il datagramma IP (IP Datagram) é l'unitá di trasferimento base del TCP/IP. Esso presenta una forte analogia con i frames di una rete reale; infatti si compone di una *header area* e di una *data area*. L'header (di almeno 20 byte) contiene l'indirizzo di destinazione (Destination IP Address) e l'indirizzo di partenza (Source IP Address), anche perché ricordiamo che il livello IP svolge funzioni di routing fondamentali per l'architettura dell'intera rete internet proprio su questi indirizzi. La differenza con l'header del frame fisico reale é che, mentre quest'ultimo contiene un indirizzo fisico, quello del datagramma contiene un indirizzo IP. La figura 2.7 mostra la struttura del datagramma IP: Piú in dettaglio, l'hea-

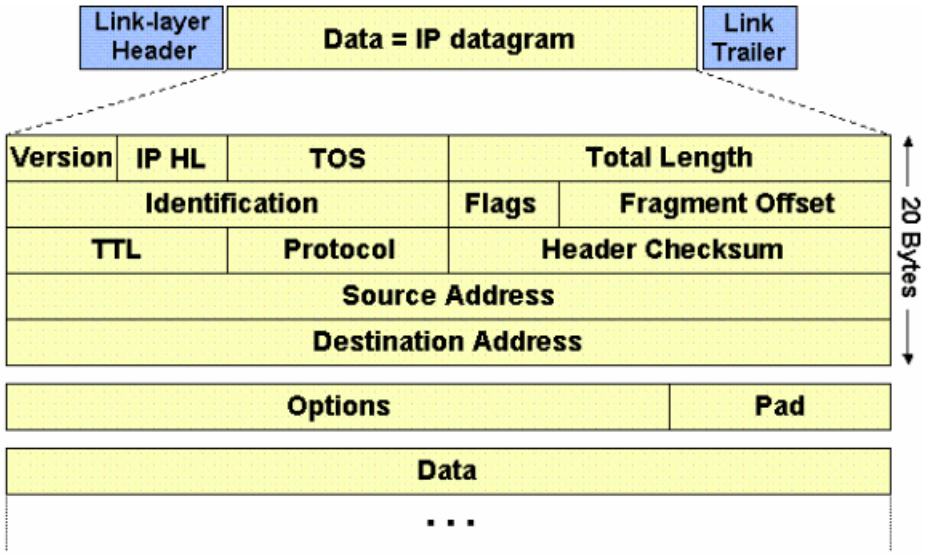


Figura 2.7: *Struttura Datagramma IP*

der contiene anche un *type* field, che identifica il contenuto del datagramma, un *checksum* field, che assicura l'integritá dei valori contenuti in esso ed il *Time To Live*, che specifica quanto tempo (in secondi) puó sopravvivere un datagramma IP nella rete (questo per evitare che girino sempre nella rete internet causando ovvi problemi). Un altro campo

molto importante é il *protocol*; infatti un datagramma IP, puó contenere al suo interno un pacchetto TCP, un UDP, un ICMP o un VMTP e quindi é necessario procedere con un demultiplex logico basato appunto su tale campo. Fra le altre informazioni, l'header del datagramma contiene il controllo della frammentazione, delle precedenze e degli errori veri e propri. Nel caso ideale, un intero datagramma IP viene incapsulato in un frame fisico, per rendere efficiente la trasmissione attraverso una rete fisica reale. Tuttavia, ogni rete locale consente una dimensione minima e massima (indicata con Maximum Transfer Unit MTU) per i pacchetti che la attraversano, ed é quindi ovvio che, siccome i datagrammi IP devono essere incapsulati in un frame, non possono prescindere da queste dimensioni, per altro fortemente variabili a seconda del tipo di rete. Allora per tutelarsi dalla disomogeneitá delle reti, invece di strutturare il datagramma in modo eccessivamente aderente ai vincoli fisici dei vari prodotti, si é deciso di scegliere una dimensione conveniente dei datagrammi IP ed escogitare poi un metodo (segmentation and reassembly) di dividerli in piccoli pezzi detti frammenti per poter essere accettati da una rete con qualsivoglia piccolo MTU e, ovviamente, riassemblati in uscita.

## 2.8 Routing

In un sistema packets switching quale il TCP/IP, il routing rappresenta il processo di scelta del percorso su cui inoltrare i pacchetti ed il router é un computer che effettua tale instradamento. Da sottolineare che il routing IP avviene in ambiente software, mentre in realtá il routing fisico é a livello di MAC. Idealmente il routing IP dovrebbe esaminare la connessione alla rete, la lunghezza del datagramma e quando selezionare il percorso migliore. Quando un programma applicativo su un host tenta di instaurare una comunicazione con un host remoto, sia l'host locale che i routers partecipano all'instradamento dei datagrammi IP fino alla loro destinazione. Si puó parlare di due tipi di routing:

Diretto : se l'host locale ed il remoto appartengono alla stessa rete fisica (es. una singola Ethernet); in questo caso non sará necessario l'impiego di routers.

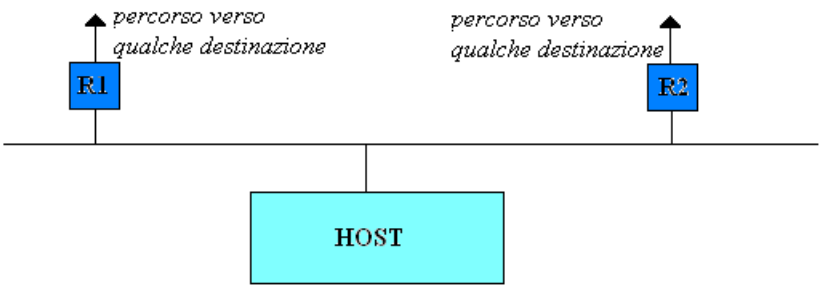


Figura 2.8: *Host*

Indiretto : se l'utente destinazione é connesso ad una rete fisica diversa ed é necessario instradare il datagramma sorgente attraverso un router.

Per sapere se una destinazione appartiene alla propria rete, l'utente sorgente estrae dall'indirizzo IP di destinazione la parte relativa alla rete, la cosiddetta netid, e la confronta con la propria: se differisce, evidentemente la destinazione del datagramma é esterna. Nel primo caso(Diretto), il Network Interface Layer dell'utente sorgente incapsula il datagramma IP in un frame fisico, associa l'indirizzo IP al relativo fisico ed usa l'hardware della rete (il cavo) per trasferirlo. Un possibile meccanismo per conoscere l'indirizzo fisico corrispondente é quello di utilizzare il protocollo ARP. Il caso del routing indiretto é piú difficoltoso, perché l'utente sorgente deve identificare il router a cui inviare il datagramma; se nella rete locale c'é un solo router che permette la connessione con altre reti, il routing diventa piú semplice in quanto, appena l'host locale ha capito che la destinazione non appartiene alla propria rete, indirizza il datagramma direttamente al router; in generale, se é connesso a piú routers, lo indirizzerá al piú vicino. Una volta che il frame contenente il datagramma ha raggiunto il router, il software del Network Interface Layer estrae il datagramma incapsulato e le routine di routing, e sulla base degli indirizzi internet, il software IP seleziona il prossimo router a cui inviare il datagramma, incapsulato in un nuovo frame. I routers in Internet formano una struttura interconnessa in continuo contatto fra loro nella quale il datagramma scorre finché non raggiunge quel particolare router che gli permette di giungere direttamente a destinazione. L'algoritmo che svolge queste funzioni (*IP routing algorithm*) utilizza su ogni macchina una tabella di routing (*IP routing table*), che contiene informazioni circa alcune possibili destinazioni e su come raggiungerle. Se

comprendesse tutte le destinazioni possibili diventerebbe troppo ingombrante e sarebbe impossibile tenerla aggiornata. Allora di solito ci si limita a mantenere le informazioni degli utenti sulla stessa rete o quelle piú frequentemente usate lasciando un indirizzo di default per tutti gli altri. Poiché i router operano in base solo alla parte di indirizzo IP relativo alla rete di destinazione l'instradamento é molto efficiente e le tabelle necessarie abbastanza piccole: tipicamente una tabella di routing contiene una coppia N-R, dove N é l'indirizzo IP della rete di destinazione ed R (next-hop) é quello del successivo router che permette di raggiungere la rete N. Chiaramente tutte le R presenti nella tabella di un router indicheranno al piú i routers connessi direttamente ad esso (come mostrato nell'esempio in figura2.9) che presenta quattro reti e tre routers (Q, R, S):

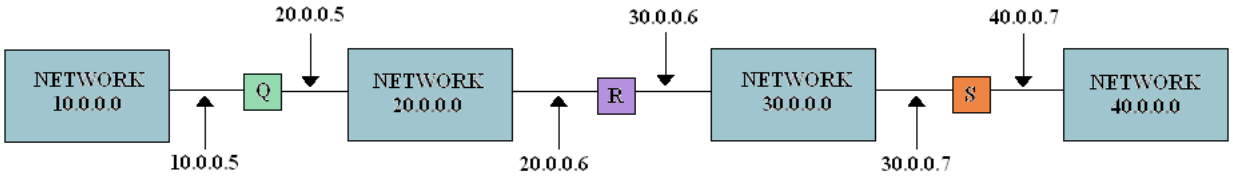


Figura 2.9: *Routing*

## 2.9 ICMP (*Internet Control Message Protocol*)

Si tratta di un meccanismo attraverso il quale i routers e gli utenti comunicano per sondare eventuali problemi o comportamenti anomali verificatisi in rete. Il protocollo IP, di per sé, non contiene nessuno strumento per poter riscontrare, da parte della stazione sorgente e nemmeno della stazione di destinazione, la perdita di un pacchetto o il collasso di una rete. L'ICMP consente una comunicazione straordinaria tra routers ed hosts permettendo lo scambio di segnali di errore o di controllo attraverso le interfacce software dell'Internet, senza peró arrivare su fino al livello degli applicativi; esso é una parte necessaria ed integrante dell'IP ed é contenuto nell'area dati di un datagramma IP. L'ICMP include

messaggi di *source quench*, che ritardano il rate di trasmissione, messaggi di *redirect*, che richiedono ad un host di cambiare la propria tabella di routing, e messaggi di *echo request/reply*, che l'host puó usare per determinare se la destinazione puó essere raggiunta (*ping*). Un messaggio ICMP(vedi figura2.10) ha tre campi di lunghezza fissa alla testa del messaggio: il *type* field (8 bits), che identifica il messaggio, il *code* field (8 bits), che contiene informazioni circa il tipo del messaggio, ed il *checksum* field (16 bits). I restanti campi del formato ICMP variano in base al tipo di messaggio.

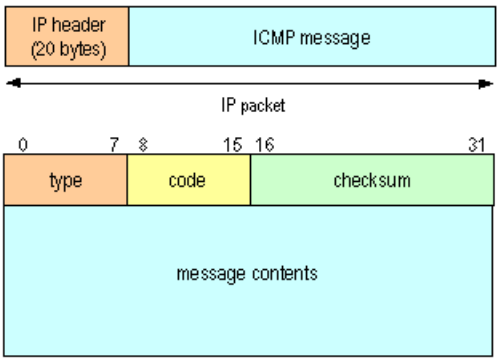


Figura 2.10: *Struttura di un messaggio ICMP*

## 2.10 UDP(*User Datagram Protocol*)

Nel complesso del protocollo TCP/IP, l'User Datagram Protocol (UDP) fornisce un servizio di recapito dei datagrammi connectionless ed é inaffidabile, usando l'IP per trasportare messaggi da una macchina ad un'altra; prevede delle porte di protocollo usate per distinguere tra piú programmi in esecuzione (o processi) su una singola macchina. É un protocollo di trasporto che si colloca sopra l'Internet Protocol Layer (IP): É simile all'IP, ma oltre ai dati spediti, ciascun messaggio UDP contiene sia il numero di porta di destinazione che quello di origine, rendendo possibile al software UDP di destinazione di recapitare il messaggio al corretto ricevente (programma oppure utente), ed a quest'ultimo di inviare una replica. L'inaffidabilitá é quella propria dell'IP, in quanto l'UDP non prevede nessun protocollo per il controllo dell'errore, a differenza del TCP: non usa acknowledgement per assicurare al mittente che i messaggi siano arrivati, non dispone

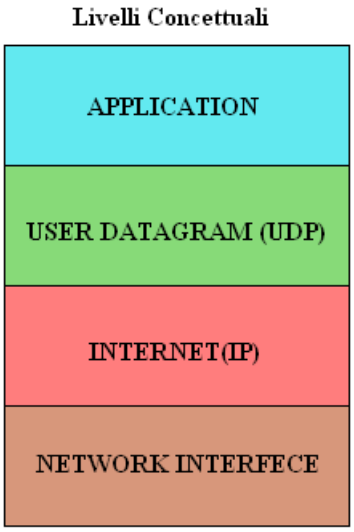


Figura 2.11: *Livelli UDP*

le sequenze di datagrammi in ordine e non fornisce una retroazione per il controllo del rate del flusso di informazioni tra macchine. Perció i messaggi UDP possono essere persi, duplicati oppure arrivare fuori dall'ordine; inoltre i datagrammi possono arrivare piú velocemente di quanto il ricevente sia in grado di processarli. Il protocollo UDP lavora bene in una rete locale, ma potrebbe fallire se usato in una rete di maggiori dimensioni. Esso realizza un datagramma piú utile di quello IP per i collegamenti fra utenti; infatti, per fare un esempio, su tale base é stato costruito l'applicativo Network File System (NFS) largamente diffuso nelle reti locali. Ciascun messaggio UDP é detto user datagram ed é costituito di due parti: *UDP header* ed *UDP data area*, che verranno incapsulate nell'IP datagram. Come mostra la figura2.12, l'header del'UDP é simile a quello del TCP, ma risulta essere piú corto non avendo nessun numero di sequenza; é diviso in quattro campi di 16 bits, che specificano il numero di porta da cui il messaggio é stato spedito (opzionale), quello della porta di destinazione (usata per demultiplexare i datagrammi tra i processi che aspettano di riceverli), la lunghezza del messaggio ed il checksum.

### 2.10.1 Le porte del protocollo UDP

I sistemi operativi della maggior parte dei computers, permettono a piú programmi applicativi di essere in esecuzione contemporaneamente (*processes, tasks, application pro-*

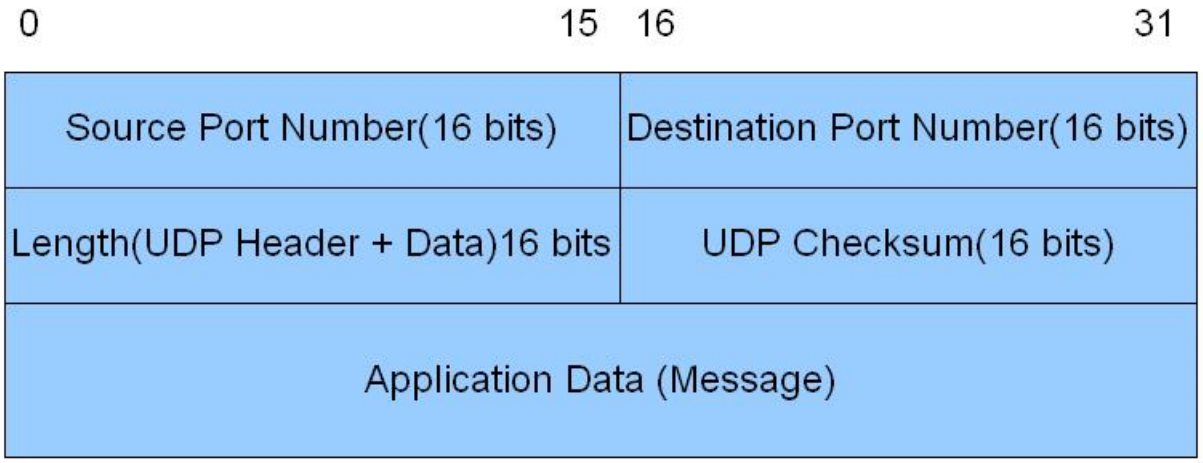


Figura 2.12: *Formato messaggio UDP*

*grams*); tali sistemi sono detti Sistemi Multitasking. Potrebbe sembrare naturale che un processo su una particolare macchina sia l'ultima destinazione di un messaggio, ma é piú opportuno immaginare che ciascuna macchina contenga un set di punti di destinazione astratti, detti protocol-ports, identificati ciascuno da un intero positivo. Il sistema operativo locale fornisce un meccanismo di interfaccia che i processi usano per specificare una porta o accedere ad essa. In generale, le porte sono bufferizzate, in modo che i dati arrivati prima che il processo sia pronto, non vengano perduti. Per permettere il buffering, il software di protocollo del sistema operativo posiziona in una coda (finita) i pacchetti che arrivano per una particolare porta, finché il processo non li estrae. Per comunicare con una porta esterna, il mittente deve conoscere sia l'indirizzo IP che il numero di porta di protocollo della macchina di destinazione. Ciascun messaggio deve contenere sia il numero della Destination Port della macchina da cui il messaggio é spedito che il numero della Source Port della macchina mittente, a cui la replica dovrá essere indirizzata, rendendo possibile, per ogni processo, il colloquio tra mittente e destinatario. Ci sono due fondamentali approcci per l'assegnazione delle porte, usando:

- **Central Authority**: due computers che devono interoperare tra di loro, si accordano per permettere ad un'autoritá centrale di assegnare i numeri di porta (Well-Known Ports) che necessitano e di pubblicare la lista di tutte le assegnazioni (Universal Assignment) il software che gestisce le porte sará realizzato in base a tale

lista.

- **Dynamic Binding**: in questo approccio le porte non sono universalmente conosciute; infatti, se un programma necessita di una porta, é il software di rete ad assegnargliela. Per sapere la porta corrente assegnata su un altro computer, é necessario inviargli una richiesta del numero di porta assegnata al servizio di interesse.

I progettisti del TCP/IP usano un approccio ibrido che assegna alcuni numeri di porta a priori (Low values) e lascia altri disponibili per siti locali o programmi applicativi (High values). La tabella seguente2.13 contiene alcune tra le piú significative UDP well-known ports:

| Port | Protocol |
|---|---|
| 7 | Echo |
| 9 | Discard |
| 11 | Users |
| 13 | Daytime |
| 17 | Quote |
| 19 | Chargen |
| 53 | Nameserver |
| 67 | Bootps |
| 68 | Bootpc |
| 69 | TFTP |
| 111 | RPC |
| 123 | NTP |

Figura 2.13: *UDP well-known ports*

## 2.11 TCP (*Transmission Control Protocol*)

Il Transmission Control Protocol (TCP), si assume la responsabilitá di instaurare un collegamento tra due utenti, di rendere affidabile il trasferimento di dati e comandi tra essi ed infine di chiudere la connessione. Esso é capace di trasferire un flusso continuo di dati fra due utenti in entrambe le direzioni (full-duplex), decidendo quando bloccare o continuare le operazioni a suo piacimento. Poiché il TCP fa veramente poche assunzioni riguardo l'hardware sottostante, é possibile implementarlo sia su una singola rete come una ethernet sia su un complesso variegato quale l'Internet. Tale protocollo, come l'UDP, si colloca,

nel modello a strati, sopra l'Internet Protocol Layer (IP), che gestisce il trasferimento e l'instradamento del singolo pacchetto fino a destinazione, ma, come ulteriore funzionalitá, tiene una traccia di ció che é stato trasmesso ed eventualmente ritrasmette quella parte di informazione che é andata perduta lungo il tragitto. Come l'UDP, il TCP permette a piú programmi applicativi su una stessa macchina di comunicare contemporaneamente, e demultiplexa il traffico dei pacchetti in ingresso a tali programmi; usa i numeri di porta per identificare la destinazione finale all'interno di una macchina. La fondamentale differenza con l'UDP é che il TCP garantisce un servizio di trasporto affidabile (Reliable Delivery Service), ponendo rimedio alle cause di inaffidabilitá proprie dell'IP (duplicazione e perdita di dati, caduta di rete, ritardi, pacchetti ricevuti fuori ordine, etc.), anche se ció comporta una implementazione piú complessa. L'importanza dell'affidabilitá del flusso permessa da tale protocollo é il motivo per cui il complesso del protocollo TCP/IP ha tale nome.

L'affidabilitá di questo servizio é caratterizzata da cinque proprietá:

- Stream Orientation: quando due programmi applicativi trasferiscono dati (stream of bits), il flusso nella macchina di destinazione passa al ricevente esattamente come é stato originato nella macchina sorgente.

- Virtual Circuit Connection: dal punto di vista del programmatore e dell'utente, il servizio che il TCP fornisce é analogo a fornire una connessione dedicata.

- Buffered Trasfer: i routers interessati dal trasferimento sono provvisti di buffers per rendere piú efficiente il trasferimento e minimizzare il traffico di rete.

- Unstructured Stream: il TCP/IP stream service non adotta un flusso di dati strutturato; ovvero non c'é modo di distinguere i records che costituiscono il flusso dati.

- Full-duplex Connection: la connessione fornita dal TCP/IP stream service permette un trasferimento di flusso contemporaneo ed indipendente in entrambe le direzioni, senza apparente interazione.

Se un qualunque messaggio é troppo grande per un singolo pacchetto TCP (gli standard consigliano una dimensione di 576 byte compreso l'header del IP) si procede a dividerlo in segmenti di lunghezza fissa e poi, arrivato a destinazione, si controlla che siano in ordine e si riassemblano, in modo che tale operazione risulti del tutto invisibile ai due utenti. Poiché queste funzionalitá sono necessarie per molte applicazioni, sono state messe tutte insieme in questo protocollo piuttosto che inserirle, come parte del programma, in ogni applicativo che ne ha bisogno. L'affidabilitá é garantita da una tecnica di fondamentale importanza nota come acknowledgement with retransmission (riscontro con ritrasmissione). Tale tecnica prevede che il destinatario invii un messaggio di acknowledgement (ACK) al mittente, una volta ricevuto un pacchetto. Il mittente mantiene una copia di ciascun pacchetto spedito e la rimuove dal buffer di trasmissione solo dopo aver ricevuto l'ACK relativo ad essa. Nella configurazione piú banale e meno efficiente l'utente sorgente, dopo aver trasmesso un pacchetto, aspetta di ricevere il suo ACK prima di spedire il successivo; inoltre fa anche partire un cronometro per il timeout, allo scadere del quale, se non ha ricevuto risposta, ritrasmette quello stesso pacchetto: Un semplice protocollo del

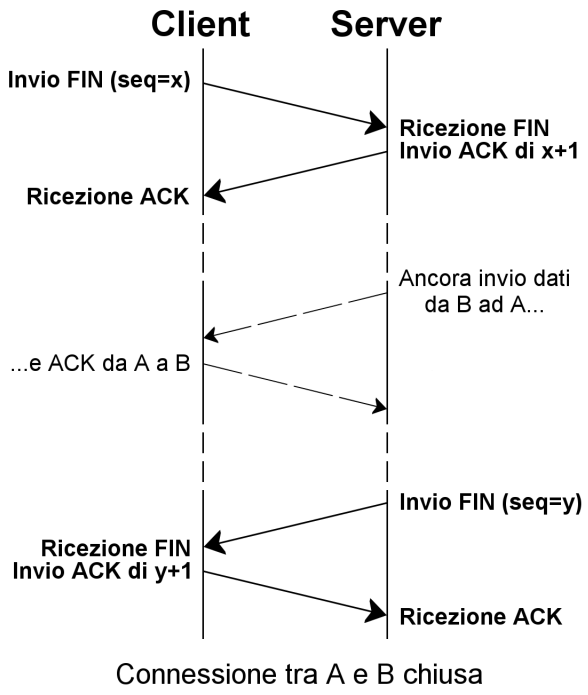


Figura 2.14: *Messaggio TCP*

tipo stop and wait come questo abbassa notevolmente le prestazioni della rete, sprecando

gran parte della banda disponibile nell'attesa dell'ACK relativo al pacchetto precedente; infatti un canale full-duplex é utilizzato come se fosse un half-duplex. Tale problema é accentuato se i pacchetti devono attraversare lungo il cammino molti componenti quali bridge, router, repeater che, ovviamente, introducono un ritardo fisso di elaborazione, piú una componente dovuta al traffico in rete.

### 2.11.1 Le porte del protocollo TCP

Le porte del TCP sono molto piú complesse rispetto a quelle dell'UDP, perché un dato numero di porta non corrisponde ad un singolo oggetto. Infatti nel TCP gli oggetti da identificare sono delle connessioni di circuito virtuali tra due programmi applicativi, e non delle particolari porte. Il TCP usa la connessione, e non la porta di protocollo, come sua fondamentale astrazione; le connessioni sono identificate da una coppia di end points, ognuno dei quali é costituito da due interi host,port, dove l'host é l'indirizzo IP dell'host e port é il numero di porta TCP su quell'host (per esempio: l'end point 128.10.2.3,25 specifica la porta 25 sulla macchina di indirizzo 128.10.2.3). Poiché il TCP identifica una connessione con una coppia di valori, uno dato numero di porta puó essere condiviso da piú connessioni su una stessa macchina, senza che si crei ambiguitá. Perció la macchina identificata da 128.10.2.3,53 puó comunicare simultaneamente con le macchine identificate da 128.2.254.139,1184 e 128.9.0.32,1184. Si possono cosí creare servizi concorrenti con connessioni multiple simultanee, senza dover riservare un numero di porta locale per ogni connessione. Per esempio, alcuni sistemi forniscono un accesso concorrente al loro servizio di posta elettronica, permettendo a piú utenti di spedire un e-mail contemporaneamente. Ci sono due fondamentali approcci per l'assegnazione delle porte, usando:

- Central Authority: due computers che devono interoperare tra di loro, si accordano per permettere ad un'autoritá centrale di assegnare i numeri di porta (Well-known ports) che necessitano e di pubblicare la lista di tutte le assegnazioni (Universal assignment) il software che gestisce le porte sará realizzato in base a tale lista.

- Dynamic Binding: in questo approccio le porte non sono universalmente conosciute; infatti, se un programma necessita di una porta, é il software di rete ad assegnar-

gliela. Per sapere la porta corrente assegnata su un altro computer, é necessario inviargli una richiesta del numero di porta assegnata al servizio di interesse.

I progettisti del TCP/IP usano un approccio ibrido che assegna alcuni numeri di porta a priori (Low values) e lascia altri disponibili per siti locali o programmi applicativi (High values). La tabella seguente2.15 contiene alcune tra le piú significative TCP well-known ports:

**Popular Applications and Their Well-Known Port Numbers**

| Port Number | Protocol | Application |
|---|---|---|
| 20 | TCP | FTP data |
| 21 | TCP | FTP control |
| 22 | TCP | SSH |
| 23 | TCP | Telnet |
| 25 | TCP | SMTP |
| 53 | UDP, TCP | DNS |
| 67, 68 | UDP | DHCP |
| 69 | UDP | TFTP |
| 80 | TCP | HTTP (WWW) |
| 110 | TCP | POP3 |
| 161 | UDP | SNMP |
| 443 | TCP | SSL |
| 16,384–32,767 | UDP | RTP-based Voice (VoIP) and Video |

Figura 2.15: *TCP well-known ports*

## 2.12 Il controllo della Congestione

Esistono almeno due notevoli problemi col TCP, relativi alla congestione della rete ed al meccanismo di timeout and retransmission. La congestione é una condizione di ritardo critico causata da un sovraccarico dei datagrammi in uno o piú switching points (es. router). Quando avviene una congestione, il ritardo aumenta ed i routers iniziano ad accodare datagrammi, finché non sono in grado di instradarli. Nel peggiore dei casi, il numero dei datagrammi che arrivano ad un router congestionato cresce (esponenzialmente nel tempo) fino a che esso non raggiunge la sua massima capacitá e comincia a

perdere datagrammi. Dal punto di vista degli hosts, la congestione é semplicemente un aumento di ritardo. Inoltre, poiché la maggior parte dei protocolli usa un meccanismo di timeout and retransmission, essi rispondono al ritardo ritrasmettendo i datagrammi, aggravando cosí la congestione. Un aumento di traffico produce un aumento di ritardo, che provoca a sua volta un aumento del traffico, e cosí via, finché la rete non puó essere piú usata: tale condizione é detta *Congestion Collapse* (Collasso dovuto alla congestione). Non esiste un meccanismo esplicito per risolvere il controllo della congestione, anche se un'attenta implementazione del TCP/IP puó permette di individuare ed affrontare meglio la situazione. Esistono due modi per affrontare il problema della congestione di rete: recuperare la funzionalitá una volta che la congestione ha avuto luogo (*Recovery*) oppure evitarla (*Avoidance*). Per evitare il collasso della rete, il TCP puó utilizzare la tecnica del Multiplicative Decrease Congestion Avoidance. Il TCP/IP mantiene un secondo limite, oltre dimensione della finestra del ricevente, detto *congestion window limit*; in ogni istante il TCP assume come dimensione della finestra di trasmissione, la minima tra le due. In condizioni normali, le due finestre sono uguali, ma in condizioni di congestione, la congestion window riduce il traffico che il TCP immette in rete, dimezzando la propria dimensione ogni volta che si perde un segmento (fino ad un minimo di uno). Il rate di trasmissione é ridotto in modo esponenziale ed il valore del timeout viene raddoppiato per ogni perdita. Se, una volta superata la congestione, si dovesse invertire la tecnica del Multiolicative Decrase, raddoppiando la congestion window, si avrebbe un sistema instabile che oscillerebbe ampiamente tra assenza di traffico e congestione. Per ripristinare le condizioni di normale funzionamento, una volta che é avvenuto il collasso, il TCP puó invece adottare una tecnica di Slow Start Recovery. Non appena inizia il traffico su una nuova connessione o aumenta dopo un periodo di congestione, la congestion window ha la dimensione di un singolo segmento ed ogni volta che arriva un ACK, viene incrementata di uno. In questo modo, dopo aver trasmesso il primo segmento ed aver ricevuto il suo ACK, la finestra di congestione viene raddoppiata; una volta inviati i due segmenti, per ogni ACK ricevuto la congestion window sará incrementata di una unitá, cosí il TCP potrá spedire quattro segmenti, e cosí via, fino a raggiungere il limite imposto dalla finestra del

ricevente. Per evitare che la dimensione della finestra si incrementi troppo velocemente e causi congestione addizionale, il TCP impone una ulteriore restrizione. Una volta che la finestra di congestione raggiunge la metá del suo valore originale, il TPC entra in una fase di congestion avoidance e rallenta il rate di incremento; in questo caso la dimensione della finestra sará incrementata di una sola unitá dopo che tutti i segmenti della finestra hanno ricevuto ACK. La combinazione delle due tecniche di Recovery ed Avoidance migliora drasticamente le prestazioni del TCP senza bisogno dell'aggiunta di ulteriori strumenti per il controllo della congestione.

## 2.13 SMTP (*Simple Mail Transfer Protocol*)

L'SMTP é un protocollo creato per gestire in maniera semplice e veloce l'invio di messaggi di posta elettronica (e-mail). Esso si basa su una connessione (indifferentemente TCP o UDP), tra un host client ed un server il cui demone software é in ascolto sulla porta 25. La comunicazione tra le due macchine, avviene attraverso alcuni semplici scambi informazioni. Una volta stabilita la connessione, infatti:

- il client si presenta, inviando una stringa del tipo: HELLO Devis, nella quale indica il suo nome. Il server risponde con un 250 +OK che é il codice utilizzato dai server SMTP per segnalare l'avvenuta ricezione e l'elaborazione del messaggio;
- il client specifica che si tratta di una e-mail e dovrebbe ora fornire il proprio indirizzo di posta. I nuovi software di ricezione ignorano di pari passo questo campo. Essendo comunque obbligatorio, é necessario inserire la stringa: MAIL FROM:<>;
- il client specifica quindi l'indirizzo del destinatario della e-mail. Una stringa di esempio puó essere: RCPT TO:<F_Biacco@hotmail.it>;
- é terminata la fase di autenticazione e puó quindi cominciare quella di scrittura e trasmissione del messaggio. Prima di tutto é necessario il comando DATA;
- ora si puó indicare l'indirizzo del mittente. Ad esempio: FROM:devis@tkd.it;
- é quindi necessario ripetere l'indirizzo del destinatario: TO:F_Biacco@hotmail.it;

- il messaggio dovrebbe contenere un soggetto; per specificarlo: SUBJECT: soggetto prova;

- ora serve una riga vuota, per indicare al server che l'intestazione della e-mail é conclusa e che ora inizia il corpo del messaggio. Il testo puó essere inviato tutto insieme o anche riga per riga. Il server provvede ad inoltrare la e-mail quando riceve la stringa ".";

- il client si scollega: "QUIT".

## 2.14 Pop3 (*Post Office Protocol* v.3)

Il POP3, descritto in almeno 20 RFC diverse (1081, 1225 e 1460 solo per citare le pié significative) é il protocollo complementare dell'SMTP. Se infatti l'SMTP si occupa della spedizione delle e-mail, il POP3 fa l'esatto contrario, ovvero fornisce una serie di comandi per la fase di ricezione. É per merito dei server POP3 che é possibile ricevere le e-mail on demand. Una volta inoltrato dal server SMTP del mittente, infatti, un messaggio viene memorizzato dal server POP3 del destinatario, che solitamente é una macchina sempre accesa e destinata prevalentemente a ricevere la posta in arrivo. L'utente puó collegarsi in un qualsiasi momento con il proprio server POP3 e trasferire in locale tutti i messaggi destinati a lui, tenendone o meno una copia sul server. Il demone POP3 é solitamente in ascolto sulla porta 110 TCP, alla quale il client deve accedere per poter controllare la sua mailbox. I procedimenti principali utilizzati dal protocollo per avviare il trasferimento sono i seguenti:

- il client si identifica, inserendo il proprio username. La stringa da inviare deve essere ad esempio: USER alessandra@tkd.it. Il server risponde con un **OK** ed un codice che indica la corretta ricezione ed elaborazione della stringa trasmessa;

- una volta fornito il proprio username, l'utente deve farsi riconoscere mediante una password. La stringa da inviare é del tipo PASS 5info2. Il server risponde inviando il solito OK, seguito peró dal numero di messaggi presenti nella mailbox;

- il client puó quindi iniziare la ricezione dei messaggi, con la stringa RETR_numero_messaggio. Ad esempio: RETR_1 provoca l'invio dal server al client del primo messaggio (in ordine temporale basato sulla spedizione) presente nella casella di posta;

- nonostante venga trasmesso al legittimo destinatario, il messaggio rimane memorizzato anche nell'hard disk del server. Per eliminarlo il client puó usare la forma: DELE numeromessaggio;

- per chiudere la connessione, come nel caso dell'SMTP, é sufficiente l'invio della stringa: QUIT.

Il protocollo POP3 é stato ormai sostituito dal piú recente IMAP4. Questo nuovo protocollo non ha comunque introdotto sostanziali migliorie al POP3, al punto che, se fosse possibile pesare il numero di software che sfruttano uno solo dei due protocolli, l'ago della bilancia continuerebbe a pendere inesorabilmente verso il POP3.

## 2.15 FTP (File Transfer Protocol)

L'FTP é un protocollo che garantisce il trasferimento di file tra due host, in maniera affidabile ed efficiente. Esso si appoggia ai protocolli TCP e Telnet per le connessioni, aggiungendo una interfaccia piú complessa, ma decisamente piú affidabile.

## 2.16 HTTP (*Hyper Text Transfer Protocol*)

L'HTTP é uno dei protocolli piú recenti (l'RFC 1945 di Tim Berners-Lee é datata 1996), che ha notevolmente agevolato l'espandersi a livello mondiale di Internet. La sua funzione principale é quella di interfacciamento con il WWW (*World Wide Web*) e di conseguenza con l'HTML (*Hyper-Text Markup Language*), il linguaggio che sta alla base dei documenti ipertestuali. Proprio per agevolare l'ipertestualitá delle pagine, con link che permettono di passare da una pagina all'altra tramite un semplice clic del mouse, l'HTTP é stato progettato come un protocollo state-less (senza memoria). Ció significa che il procedimento di ricezione dati non é vincolato a quelli appena ricevuti. Quando richiediamo la

visualizzazione di una pagina HTML, infatti, il nostro browser non fa altro che connettersi al server dove essa é memorizzata, richiederla attraverso semplici stringhe ASCII di testo (chiamate method), riceverla e chiudere immediatamente la connessione. L'unico inconveniente di questo protocollo si riscontra quando l'utente ha problemi a connettersi al server HTTP. Puó infatti capitare di riuscire a vedere qualche pagina, senza peró essere in grado di visualizzare quelle successive.

## 2.17 DNS (*Domain Name Service*)

Il DNS é un particolare servizio che consente la risoluzione del nome di un host nell'indirizzo IP corrispondente. In ambiente Windows NT questa funzione viene assolta da un server locale (server DNS); nell'ambito di Internet, invece, il server DNS non é locale, ma messo a disposizione in remoto dal provider che fornisce l'accesso al web. Sostanzialmente, il server DNS é un database, contenente un elenco di nomi di host e corrispettivi indirizzi IP. L'HTTP si appoggia pesantemente al DNS. Ció permette di collegarsi ad un server web utilizzando un nome facile da ricordare (ad esempio http://www.tkdacademy.it) invece che il corrispettivo IP (ad esempio 188.234.321.45).

# Capitolo 3

# Microchip TCP/IP Stack

Lo Stack TCP/IP (Transmission Control Protocol/Internet Protocol) Microchip é una suite di programmi che fornisce servizi alle applicazioni standard basate sul protocollo TCP/IP (HTTP Server, Mail Client, etc.). Lo Stack TCP/IP Microchip é implementato in una struttura modulare che rispecchia la struttura a livelli dello standard TCP/IP descritto al capitolo 2. Chi utilizza lo Stack non ha bisogno di sapere tutte le varie dipendenze delle funzioni implementate nello Stack TCP/IP per poterlo usare [1].

## 3.1 Architettura dello Stack

Molte implementazioni del TCP/IP seguono un'Architettura Software riferita al modello di riferimento TCP/IP descritto nel capitolo precedente. Lo Stack basato su questo modello é scritto nel linguaggio di programmazione C ed é diviso in livelli multipli, dove i livelli sono accatastati uno sull'altro(da cui il nome TCP/IP Stack) e ogni livello accede ai servizi da uno o piú livelli direttamente inferiori. Una versione semplificata del modello su cui si basa lo Stack TCP/IP é mostrata in Figura2.2.

## 3.2 Livelli dello Stack

Per facilitare il processo di configurazione, lo Stack utilizza dei moduli scritti in C; per abilitare,disabilitare, in generale per settare un particolare parametro, l'utente modifica

[1] Per il nostro progetto infatti siamo interessati alla sola integrazione del Server HTTP e per questo ci basta andare a modificare le specifiche dello Stack in base alle nostre esigenze

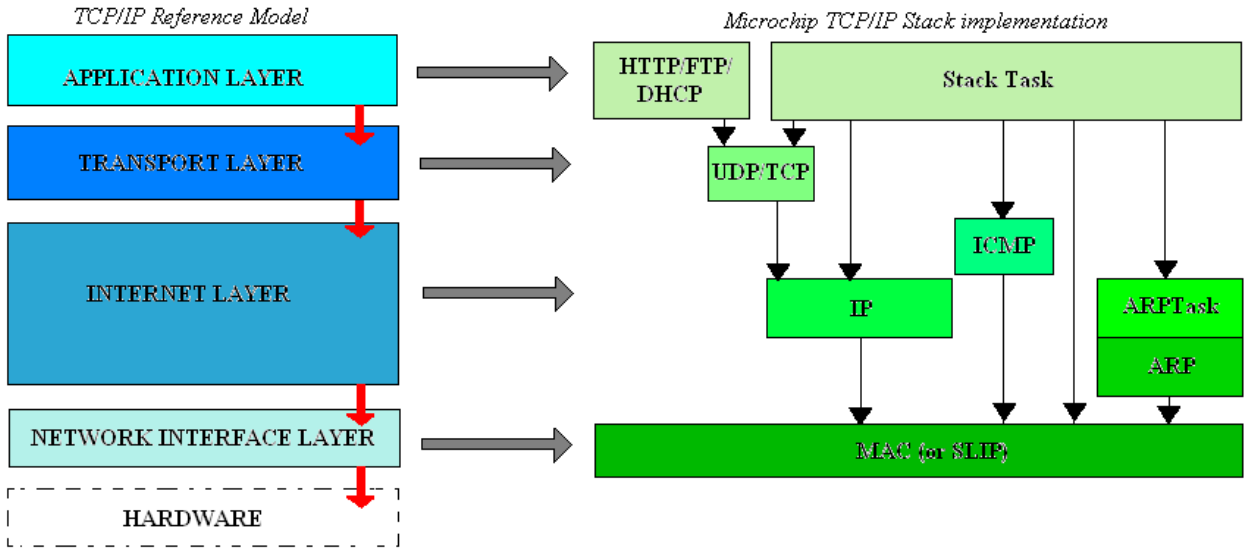


Figura 3.1: *Corrispondenze tra modello di riferimento e Stack implementato*

la specifica *define* all'interno di uno o piú di questi file. Molte di queste funzionalitá sono definite nell'Header *StackTsk.h*. Una volta che i file sono stati modificati, l'utente deve ricompilare il progetto per rendere effettive le modifiche. Le definizioni sono mostrate nella Tabella 3.1

## 3.3 Come usare lo Stack

Tra i file forniti da Microchip si trovano i file sorgente per lo Stack TCP/IP, il Server HTTP e il server FTP, il DCHP e IP Gleaning modules .

### 3.3.1 Files necessari per il progetto

Di seguito viene riportata una lista completa dei file necessari all'implementazione del Server HTTP nell'NCAP:

- HTTP.c
- TCP.c
- IP.c
- MAC.c oppure SLIP.c

Tabella 3.1: Stack Configuration Definitions

| Define | Values | Used By | Purpose |
|---|---|---|---|
| MPFS_USE_PGRM | N/A | MP File System(MPFS.c) | Uncomment this if program memory will be used for MPFS storage |
| MPFS_USE_EEPROM | N/A | MPFS.c | Uncomment this if external serial EEPROM will be used for MPFS storage |
| MPFS_RESERVE_BLOCK | 0-255 | MPFS.c | Number of bytes to reserve before MPFS storage starts |
| EEPROM_CONTROL | External Data EEPROM Control Code | MPFS.c | To address external data EEPROM |
| STACK_USE_ICMP | N/A | StackTsk.c | Comment this if ICMP is not required |
| STACK_USE_IP_GLEANING | N/A | StackTsk.c | Comment this if IP Gleaning is not required |
| STACK_USE_DHCP | N/A | DHCP.c, StackTsk.c | Comment this if DHCP is not required |
| STACK_USE_TCP | N/A | TCP.c, StackTsk.c | Comment this if TCP module is notrequired. This module will be automatically enabled if there is at least one high-level module requiring TCP. |
| TCP_NO_WAIT_FOR_ACK | N/A | TCP.c | TCP will wait for ACK before transmitting next packet |
| MAX_SOCKETS | 1-253 | TCP.c | To define the total number of sockets supported (limited by available RAM).Compile-time check is done to make sure that enough sockets are available for selected TCP applications. |
| MAC_TX_BUFFER_SIZE | 201-1500 | TCP.c, MAC.c | To define individual transmit buffer size |
| MAX_TX_BUFFER_COUNT | 1-255 | MAC.c | To define total number of transmit buffers. This number is limited by available MAC buffer size. |
| MAX_HTTP_CONNECTIONS | 1-255 | HTTP.c | To define maximum number of HTTP connections allowed at any time |
| MPFS_WRITE_PAGE_SIZE (MPFS.h) | 1-255 | MPFS.c | To define writable page size for current MPFS storage media |
| MAX_HTTP_ARGS (HTTP.c) | 1-31 | HTTP.c | To define maximum number of HTML form fields including HTML form name |
| MAX_HTML_CMD_LEN (HTTP.c) | 1-128 | HTTP.c | To define maximum length of HTML form URL string |

- Helpers.c
- Tick.c
- MPFS.c
- XEEPROM.c[2]

## 3.4 Il Server HTTP Microchip

Il Server HTTP incluso nell'applicazione coesiste con lo Stack TCP/IP Microchip e l'applicazione principale dell'utente. Il server stesso viene implementato nel file HTTP.c. Il Server non implementa tutte le fuzionalitá dell'HTTP:é infatti un server minimo appositamente scritto per sistemi Embedded. É possibile aggiungere o togliere funzionalitá a seconda delle esigenze[3]. Il Server HTTP incorpora le seguenti funzioni principali:

- Supporta connessioni HTTP multiple
- Contiene un semplice file system (MPFS)
- Supporta la memorizzazione di pagine Web nella sua memoria flash di programma interna o nella memoria seriale esterna EEPROM
- Include un programma per la creazione di immagini MPFS da una directory assegnata
- Supporta il metodo HTTP GET e POST(altri metodi possono essere facilmente aggiunti)
- Supporta una CGI (Common Gateway Interface) per invocare funzioni predefinite dal browser remoto
- Supporta la generazione dinamica del contenuto delle pagine web

[2]Nella prova di funzionamento di questo progetto la pagina Web viene caricata via web e salvata nella memoria EEPROM del microprocessore.

[3]Nel nostro progetto verranno ridotte al minimo

Il server consiste dei seguenti componenti piú importanti:

- MPFS Image Builder
- MPFS Access Library
- MPFS Download Routine (implementata dall'applicazione principale)
- HTTP Server Task

Il Server utilizza il file *index.htm* come la sua pagina Web di default. Se un Client remoto(browser) accede al Server HTTP attraverso il suo indirizzo IP o solo nome del dominio, *index.htm* é la pagina di default visualizzata. Questo necessita che tutte le applicazioni che utilizzano il Server contengano un file nominato *index.htm* come parte della loro immagine MPFS. Se dovesse essere necessario, il nome di questa pagina di default puó essere cambiato modificando la *HTTP_DEFAULT_FILE_STRING* nel file *http.c*[4]. É importante assicurarsi che il nome di nessuna pagina web contenga i seguenti caratteri: ' , " , < , > , # , % , [ , ] , { , } , | , \ , ^ , ∼.

Se un nome di file contiene qualcuno di questi caratteri, la corrispondente pagina web diventerá inaccessibile e non viene dato nessun avviso in fase di compilazione. Il Server Http contiene una lista di tipi di file che supporta. Utilizza questa informazione per - avvisare il browser remoto su come interpretare un file particolare, basato sull'estensione di tre lettere del file. Come default il Server Microchip supporta file con estensioni: .txt, .htm, .gif, .cgi, .jpg, .cla e .wav. Si puó modificare la tabella *httpFiles*, dove sono definiti i file sopracitati, aggiungendone o togliendone.

Per Maggiori informazioni sul Server HTTP della Microchip si rimanda all'Application Note AN833[4]. Ai fini della sua integrazione nell'EXPLORER16, nel capitolo seguente vengono esposte le scelte effettuate.

[4]Nel nostro progetto non abbiamo modificato il nome della pagina

# Capitolo 4

# Integrazione Stack Microchip nell'NCAP

Scopo del progetto é integrare all'interno dell'NCAP il protocollo TCP/IP, in particolare un Server HTTP, per farlo coesistere in un successivo sviluppo con uno Stack ZigBee. Si affronta il problema integrando il Server HTTP curando peró di utilizzare il minor spazio possibile nella memoria, in quanto si vuole lasciare il piú possibile spazio disponibile per l'integrazione futura dello Stack ZigBee, giá esistente e funzionante, e di eventuali altri moduli. Lo Stack della Microchip, nella sua implementazione in linguaggio C, presenta un file di configurazione *TCPIPConfig.h* che permette di configurarlo a seconda dell'utilizzo: in particolare in questo progetto si riducono al minimo le funzionalitá del Server, e quindi la memoria utilizzata, mantendo solamente quelle di base. La versione dello Stack utilizzata é la 5.20.

## 4.1 File di Configurazione

Il file TCPIPConfig.h contiene una sezione Application Options dove si possono effettuare le scelte di progettazione dello Stack. Vediamo le scelte nel dettaglio, con riferimento al listato di codice seguente, dove le righe precedute da "//" vanno interpretate come commento o istruzioni non utilizzate.

```
//
    ============================================================================

// Application Options
//
    ============================================================================

/* Application Level Module Selection
 * Uncomment or comment the following lines to enable or
 * disabled the following high-level application modules.
 */
#define STACK_USE_UART     // Application demo using UART for IP address display and stack configuration
//#define STACK_USE_UART2TCP_BRIDGE  // UART to TCP Bridge application example
//#define STACK_USE_IP_GLEANING
//#define STACK_USE_ICMP_SERVER   // Ping query and response capability
//#define STACK_USE_ICMP_CLIENT   // Ping transmission capability
//#define STACK_USE_HTTP_SERVER   // Old HTTP server
#define STACK_USE_HTTP2_SERVER   // New HTTP server with POST, Cookies, Authentication, etc.
//#define STACK_USE_SSL_SERVER   // SSL server socket support (Requires SW300052)
//#define STACK_USE_SSL_CLIENT   // SSL client socket support (Requires SW300052)
#define STACK_USE_AUTO_IP // Dynamic link-layer IP address automatic configuration protocol
#define STACK_USE_DHCP_CLIENT   // Dynamic Host Configuration Protocol client for obtaining IP address and
    other parameters
#define STACK_USE_DHCP_SERVER   // Single host DHCP server
//#define STACK_USE_FTP_SERVER   // File Transfer Protocol (old)
//#define STACK_USE_SMTP_CLIENT   // Simple Mail Transfer Protocol for sending email
//#define STACK_USE_SNMP_SERVER   // Simple Network Management Protocol v2C Community Agent
//#define STACK_USE_TFTP_CLIENT   // Trivial File Transfer Protocol client
//#define STACK_USE_GENERIC_TCP_CLIENT_EXAMPLE // HTTP Client example in GenericTCPClient.c
//#define STACK_USE_GENERIC_TCP_SERVER_EXAMPLE // ToUpper server example in GenericTCPServer.c
//#define STACK_USE_TELNET_SERVER   // Telnet server
//#define STACK_USE_ANNOUNCE    // Microchip Embedded Ethernet Device Discoverer server/client
//#define STACK_USE_DNS     // Domain Name Service Client for resolving hostname strings to IP addresses
//#define STACK_USE_NBNS     // NetBIOS Name Service Server for repsonding to NBNS hostname broadcast
    queries
#define STACK_USE_REBOOT_SERVER   // Module for resetting this PIC remotely. Primarily useful for a
    Bootloader.----(forse da commentare)
//#define STACK_USE_SNTP_CLIENT   // Simple Network Time Protocol for obtaining current date/time from
    Internet
//#define STACK_USE_UDP_PERFORMANCE_TEST // Module for testing UDP TX performance characteristics.
    NOTE: Enabling this will cause a huge amount of UDP broadcast packets to flood your network on the discard port
    . Use care when enabling this on production networks, especially with VPNs (could tunnel broadcast traffic across a
     limited bandwidth connection).
//#define STACK_USE_TCP_PERFORMANCE_TEST // Module for testing TCP TX performance characteristics
//#define STACK_USE_DYNAMICDNS_CLIENT  // Dynamic DNS client updater module
//#define STACK_USE_BERKELEY_API   // Berekely Sockets APIs are available
```

Listing 4.1: Modulo selezione livello applicazione, TCPIPConfig.h

### 4.1.1 STACK_USE_UART

**Caratteristica** UART Demo

**Descrizione** Applicazione Demo che utilizza la UART per visualizzare l'indirizzo IP e la versione dello Stack integrato[1]

**File Richiesti** UART.c, UART.h

**Utilizzo** Nel nostro progetto tramite l'utilizzo del display LCD in dotazione nella schada Explorer16, riusciamo a visualizzare l'indirizzo IP da utilizzare per connettersi al Server, necessaria in quanto si utilizza la generazione dell'indirizzo IP dinamico, che quindi varia ad ogni riaccensione dell'NCAP; la visualizzazione della versione dello Stack é puramente a titolo informativo.

### 4.1.2 STACK_USE_UART2TCP_BRIDGE

**Caratteristica** UART-to-TCP Bridge

**Descrizione** Esempio di applicazione Bridge tra UART e TCP

**File Richiesti** UART2TCPBridge.c, UART2TCPBridge.h

**Utilizzo** **Non utilizzata**

### 4.1.3 STACK_USE_IP_GLEANING

**Caratteristica** IP Gleaning

**Descrizione** Consente l'assegnazione di un indirizzo IP tramite la ricezione di un pacchetto ICMP con un indirizzo IP valido durante la modalitá di configurazione

**File Richiesti** Nessuno

**Utilizzo** **Non utilizzata**

[1] Nel nostro caso V5.20

### 4.1.4 STACK_USE_ICMP_SERVER e STACK_ USE_ICMP _CLIENT

**Caratteristica** L'Internet Control Message Protocol é usato per inviare messaggi di errore e richieste di stato(SERVER) e per trasmettere(CLIENT). Il modulo ICMP implementa il tipo di messaggio di Echo Reply (comunemente indicato come un ping) che possono essere utilizzati per determinare se un host specificato é raggiungibile attraverso una rete IP da un dispositivo che esegue il protocollo TCP / IP stack. Un server ICMP é supportato anche di rispondere ai ping da altri dispositivi.

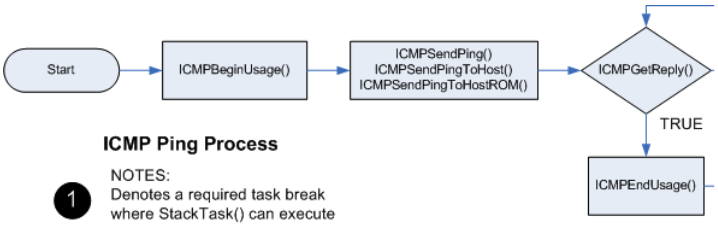


Figura 4.1: *ICMP Ping Process*

**Descrizione** Offre la possibilitá di interrogare e rispondere ai ping.

**File Richiesti** ICMP.c, ICMP.h

**Utilizzo** **Non utilizzata**

### 4.1.5 STACK_USE_HTTP_SERVER

**Caratteristica** HTTP

**Descrizione** Offre la possibilitá di usufruire delle funzionalitá del Server HTTP

**File Richiesti** HTTP.c, HTTP.h, TCP.c, TCP.h

**Utilizzo** **Non utilizzata**:Viene utilizzato l'HTTP2.

### 4.1.6 STACK_USE_HTTP2_SERVER

**Caratteristica** Il modulo web server HTTP2 e il suo modulo di file system associato MPFS2 consente alla Board di agire come un web server. Questo facilita un metodo

semplice per visualizzare le informazioni di stato e di controllare le applicazioni con qualsiasi browser web standard. Tre componenti principali sono necessarie per capire come il Server Web HTTP2 lavora: le pagine web, l'utility MPFS2 e i files sorgente *CustomHTTPApp.c* e H*TTPPrint.h*. Una visione d'insieme dell'intero progetto é mostrata nella Figura4.4 e spiegata in particolare.

**Descrizione** Fornisce funzionalitá di server HTTP con variabili dinamiche, POST, Cookie, autenticazione e altre funzionalitá.

**File Richiesti** HTTP2.c, HTTP2.h, TCP.c, TCP.h, CustomHTTPApp.c e HTTPPrint.h

**Utilizzo** É la nostra applicazione principale, utilizzata al posto dell'HTTP in quanto piú performante.

### 4.1.7 STACK_USE_SSL_SERVER e STACK_ USE_SSL_CLIENT

**Caratteristica** Il modulo di crittografia SSL aggiunge il supporto per lo strato TCP con l'attuazione del protocollo SSLv3. Questo protocollo é lo standard per le comunicazioni sicure attraverso Internet, e impedisce snooping o manomissione dei dati che viaggiano su una rete non sicura.

**Descrizione** Fornisce il supporto per socket server(client) SSL.

**File Richiesti** SSL.c, SSL.h, ARCFOUR.c, ARCFOUR.h, BigInt.c, BigInt.h, Random.c, Random.h, RSA.c, RSA.h

**Utilizzo** **Non utilizzata**

### 4.1.8 STACK_USE_AUTO_IP

**Caratteristica** Il modulo AutoIP permetterá all'applicazione di scegliere un indirizzo IP e assumerlo come proprio durante tutta la fase di accensione dell'NCAP.

**Descrizione** L'applicazione sceglie un indirizzo IP all'inizializzazione della rete e lo mantiene univoco.

**File Richiesti** AutoIP.c e AutoIP.h

**Utilizzo** **Utilizzata**:Aggiunta al progetto in quanto poco pesante nello Stack e di eventuale utilitá futura.

### 4.1.9 STACK_USE_DHCP_CLIENT e STACK_ USE_DHCP _SERVER

**Caratteristica** Il modulo client DHCP consentirá all'applicazione di ottenere dinamicamente un indirizzo IP dal server DHCP sulla stessa rete. In questo modo si resetta l'indirizzo IP, subnet mask, l'indirizzo del gateway, e alcuni altri parametri di configurazione nella struttura AppConfig.

**Descrizione** Permette di ottenere un indirizzo IP dinamico.

**File Richiesti** DHCP.c, DHCPs.c, e DHCP.h

**Utilizzo** :Per assegnare un indirizzo IP dinamico; non indispensabile, ma mantenuta perché non occupa molto spazio nel progetto

### 4.1.10 STACK_USE_FTP_SERVER

**Caratteristica** FTP

**Descrizione** Fornisce la capacitá di caricare le immagini in remoto MPFS ai server HTTP

**File Richiesti** FTP.c, FTP.h, TCP.c, TCP.h

**Utilizzo** **Non utilizzata**:Utilizzato HTTP2

### 4.1.11 STACK_USE_SMTP_CLIENT e STACK_ USE_SNMP _SERVER

**Caratteristica** Il modulo client SMTP in stack TCP/IP consente alle applicazioni di inviare e-mail a qualsiasi destinatario nel mondo. Tali messaggi possono includere informazioni di stato o avvisi importanti. Utilizzando il servizio gateway dall'e-mail all'SMS fornito dalla maggior parte dei telefoni cellulari, questi messaggi possono anche essere ricevuti direttamente dai telefoni cellulari[2]. Utilizzando il client SMTP, esso richiede l'accesso a un server di posta locale (come ad esempio mail.tuodominio.com)per un funzionamento affidabile.

**Descrizione** Offre la possibilitá di inviare e-mail(Client).

**File Richiesti** SMTP.c, SMTP.h, TCP.c, TCP.h, Helpers.c, Helpers.h

**Utilizzo** **Non utilizzata**:Da considerare nelle implementazioni future per un progetto completo

### 4.1.12 STACK_USE_TFTP_CLIENT

**Caratteristica** Il Trivial File Transfer Protocol fornisce upload inaffidabili e servizi di download di applicazioni connesse alla UDP-based server TFTP.

**Descrizione** Fornisce un inaffidabile servizio di upload/download file.

**File Richiesti** TFTPc.c, TFTPc.h, TCP.c, TCP.h

**Utilizzo** **Non utilizzata**:Utilizzato MPFS2.

### 4.1.13 STACK_USE_GENERIC_TCP_CLIENT_ EXAMPLE

**Caratteristica** Esempio di HTTP Client

**Descrizione** Esempio di HTTP Client

[2]Puó essere implementato in futuro per ricevere via SMS avvisi di variazioni di temperatura, pressione, allarme, etc. a seconda del campo d'impiego del sistema di misura progettato

**File Richiesti** GenericTCPClient.c

**Utilizzo** **Non utilizzata**

### 4.1.14 STACK_USE_GENERIC_TCP_SERVER_EXAMPLE

**Caratteristica** Esempio di HTTP Server

**Descrizione** Esempio di HTTP Server

**File Richiesti** GenericTCPServer.c

**Utilizzo** **Non utilizzata**

### 4.1.15 STACK_USE_TELNET_SERVER

**Caratteristica** Telnet fornisce una comunicazione bidirezionale, interattiva tra due nodi su Internet o su una LAN(Local Area Network).Il codice Telnet incluso con lo stack TCP/IP Microchip é una dimostrazione della struttura di una applicazione Telnet. Questa demo inizia con l'ascolto di una connessione Telnet. Quando un client tenta di fare un accesso, la demo richiede al client un nome utente e una password, e se quella corretta viene fornito, saranno mandati in uscita e periodicamente aggiornati i diversi valori ottenuti dalla scheda demo.

**Descrizione** Fornisce i servizi Telnet.

**File Richiesti** Telnet.c, Telnet.h, TCP.c, TCP.h

**Utilizzo** **Non utilizzata**

### 4.1.16 STACK_USE_ANNOUNCE

**Caratteristica** Questo modulo facilita la scoperta di un dispositivo in reti di trasmissione DHCP abilitate nella trasmissione di un messaggio UDP sulla porta 30303 ogni volta che cambia l'indirizzo IP locale.

**Descrizione** Offre la scoperta dell'Hostname del dispositivo e dell'indirizzo IP in una sottorete locale Ethernet.

**File Richiesti** Announce.c, Announce.h

**Utilizzo** **Non utilizzata**

### 4.1.17 STACK_USE_DNS

**Caratteristica** Il Domain Name Service Associates associa agli host (come www.tkdacademy.it) agli indirizzi IP (come 10.0.54.2). Il modulo Client DNS fornisce funzionalitá DNS allo stack.

**Descrizione** Fornisce la capacitá di tradurre i nomi host in indirizzi IP.

**File Richiesti** DNS.c, DNS.h, UDP.c, UDP.h

**Utilizzo** **Non utilizzata**

### 4.1.18 STACK_USE_NBNS

**Caratteristica** Il protocollo NetBIOS Name Service associa gli host con gli indirizzi IP, in modo simile a DNS, ma sulla stessa sottorete IP. In pratica, questo permette l'assegnazione degli hostnames alle schede di accesso sulla stessa sottorete. Per esempio, nella sezione Demo TCPIP App,é programmato con il nome mchpboard é quindi possibile accedere direttamente invece che con il suo indirizzo IP.

**Descrizione** Fornisce la capacitá di associare i nomi host agli indirizzi IP sulla stessa sottorete.

**File Richiesti** NBNS.c, NBNS.h, UDP.c, UDP.h

**Utilizzo** **Non utilizzata**

### 4.1.19 STACK_USE_REBOOT_SERVER

**Caratteristica** Il modulo di Reboot(riavvio) puó consentire a un utente remoto di riavviare il microcontrollore PIC su cui é in esecuzione il TCP/IP stack. Questa funzione é utilizzata principalmente per applicazioni bootloader, in cui il microcontrollore deve essere resettato per accedere alla sezione codice bootloader. Questo modulo esegue un compito di ascoltare su una porta UDP specifica per un pacchetto, e poi si riavvia se ne riceve uno.

**Descrizione** Consente al PIC di essere reimpostata in modalitá remota.

**File Richiesti** Nessuno

**Utilizzo** **Utilizzata**

### 4.1.20 STACK_USE_SNTP_CLIENT

**Caratteristica** Il modulo SNTP implementa il Simple Network Time Protocol.

**Descrizione** Offre la possibilitá di ottenere la data e l'ora da internet.

**File Richiesti** SNTP.c, SNTP.h, UDP.c, UDP.h

**Utilizzo** **Non utilizzata**:Nel futuro puó rivelarsi una utilitá nel monitoraggio automatizzato.

### 4.1.21 STACK_USE_UDP_PERFORMANCE_TEST

**Caratteristica** I moduli TCP e UDP Performance Test forniscono un metodo per ottenere informazioni sulle prestazioni dello stack. Essi possono essere utilizzati per testare lo Stack su varie piattaforme hardware, e sono anche utili per determinare in che modo l'applicazione personalizzata ha modificato le prestazioni dello Stack.

**Descrizione** Test UDP prestazioni. Monitorizza una rete locale per i pacchetti UDP con un packet sniffer. Questo test trasmette 1024 pacchetti.

**File Richiesti** UDPPerformanceTest.c, UDPPerformanceTest.h

**Utilizzo** **Non utilizzata**

### 4.1.22 STACK_USE_TCP_PERFORMANCE_TEST

**Caratteristica** I moduli TCP e UDP Performance Test forniscono un metodo per ottenere informazioni sulle prestazioni dello stack. Essi possono essere utilizzati per testare o stack su varie piattaforme hardware, e sono anche utili per determinare in che modo l'applicazione personalizzata ha modificato le prestazioni dello stack.

**Descrizione** Test TCP performance. Collega una scheda di demo a un PC tramite UART, esegue il codice, e monitorizza la velocitá sul terminale PC.

**File Richiesti** TCPPerformanceTest.c, TCPPerformanceTest.h

**Utilizzo** **Non utilizzata**

### 4.1.23 STACK_USE_DYNAMICDNS_CLIENT

**Caratteristica** Il modulo Dynamic DNS Client fornisce un metodo per aggiornare un indirizzo IP dinamico ad un servizio DDNS. Questi servizi possono essere utilizzati per fornire DNS mapping hostname a dispositivi che dietro router, firewall, e/o reti che assegnano dinamicamente gli indirizzi IP.

**Descrizione** Fornisce la capacitá di risolvere i nomi host in indirizzi IP che cambiano spesso.

**File Richiesti**

**Utilizzo** **Non utilizzata**

### 4.1.24 STACK_USE_BERKELEY_API

**Caratteristica** La distribuzione Berkeley Socket (BSD) API fornisce un wrapper BSD al nativo Microchip TCP/IP Stack API. Utilizzando questa interfaccia, i program-

```
//
    ============================================================================

// Data Storage Options
//
    ============================================================================


/* MPFS Configuration
 * MPFS is automatically included when required for other
 * applications. If your custom application requires it
 * otherwise, uncomment the appropriate selection.
 */
//#define STACK_USE_MPFS
#define STACK_USE_MPFS2

/* MPFS Storage Location
 * If html pages are stored in internal program memory,
 * comment both MPFS_USE_EEPROM and MPFS_USE_SPI_FLASH, then
 * include an MPFS image (.c or .s file) in the project.
 * If html pages are stored in external memory, uncomment the
 * appropriate definition.
 *
 * Supported serial flash parts include the SST25VFxxxB series.
 */
#define MPFS_USE_EEPROM
//#define MPFS_USE_SPI_FLASH
```

Listing 4.2: Data Storage Options

matori che hanno familiaritá con i sockets BSD possono velocemente sviluppare rapidamente applicazioni utilizzando il Microchip TCP/IP Stack.

**Descrizione** Fornisce un livello di astrazione Berkeley Sockets API

**File Richiesti** BerkeleyAPI.c, BerkeleyAPI.h

**Utilizzo** **Non utilizzata**

## 4.2 Data Storage Options

In Questa sezione si impostano le preferenze di memorizzazione in particolare delle pagine web.

### 4.2.1 MPFS Configuration

L'MPFS é automaticamente incluso per le varie applicazioni. Se l'applicazione personalizzata non lo richiede si decommenta la scelta corrispondente.

**STACK_USE_MPFS**

**Caratteristica** Fornisce servizi MPFS per applicazioni standard. Questa funzionalitá viene attivata/richiesta automaticamente da protocolli basati su stack che richiedono MPFS.

**File Richiesti** Nessuno

**Utilizzo** **Non utilizzata**:Per questo progetto é stato scelto di utilizzare l'MPFS2)

**STACK_USE_MPFS2**

**Caratteristica** :Il modulo file system MPFS2 fornisce un read-only file system che puó essere memorizzato in una EEPROM esterna, esterna Flash seriale, o la memoria flash di programma interna. Questo file system serve come base per il modulo HTTP2 server web, ma viene utilizzato anche dal modulo SNMP ed é disponibile per altre applicazioni che richiedono capacitá di memorizzazione di sola lettura.Il modulo MPFS2 include un programma di utilitá MPFS2 che gira sul PC. Questo programma crea immagini MPFS2 in vari formati per l'utilizzo del vari formati di archiviazione supportati.

**File Richiesti** Nessuno

**Utilizzo** **Utilizzata**:Per questo progetto é stato scelto di utilizzare l'MPFS2

### 4.2.2 MPFS Storage Location

Se le pagine html sono memorizzate nella memoria interna del programma, si commenta sia MPFS_USE_EEPROM sia MPFS_USE_SPI_FLASH, e si include nel progetto un'immagine MPFS (. C o. S file). Se le pagine html vengono memorizzate nella memoria esterna, si rimuove la definizione del caso.

**MPFS_USE_EEPROM**

**Caratteristica** :La EEPROM puó essere utilizzata per memorizzare le immagini MPFS/MPFS2 delle pagine web e le strutture di applicazioni personalizzate

**File Richiesti** Nessuno

**Utilizzo Utilizzata**:Per questo progetto é stato scelto di utilizzare l'EEPROM per memorizzare la pagina web di prova

**MPFS_USE_SPI_FLASH**

**Caratteristica** : la memorizzazione per le immagini e le strutture MPFS personalizzato é disponibile anche su dispositivi flash seriali

**File Richiesti** : nessuno

**Utilizzo Non utilizzata**: per questo progetto é stato scelto di utilizzare l'EEPROM.

## 4.3 Livello ottimizzazione spazio memoria

Per far fronte al poco spazio a disposizione per l'integrazione dello Stack, si agisce nelle opzioni di progetto, in particolare in Build Options(MPLAB C30), dove si trova la finestra di Figura4.2. Il grafico Speed-CodeSize permette di scegliere diversi criteri di ottimizzazione, dove velocitá di esecuzione e spazio occupato viaggiano in maniera inversamente proporzionale l'uno rispetto all'altro. Nel progetto si é privilegiato il minor spazio occupato a discapito della velocitá di esecuzione. La scelta del livello di Ottimizzazione (-0s) comporta un buon guadagno di spazio in memoria. Lo spazio in memoria occupato dalle altre aggiunte é descritto in dettaglio nella tabella della Memory Usage del Microchip TCP/IP Stack Help[8]; con le semplificazioni effettuate si é ottenuto un'occupazione in memoria di 49140 Bytes su 132090 Bytes totali a disposizione ( viene mostrato nella Figura4.3). In questo modo viene lasciato lo spazio sufficiente per la futura integrazione dello Stack ZigBee.

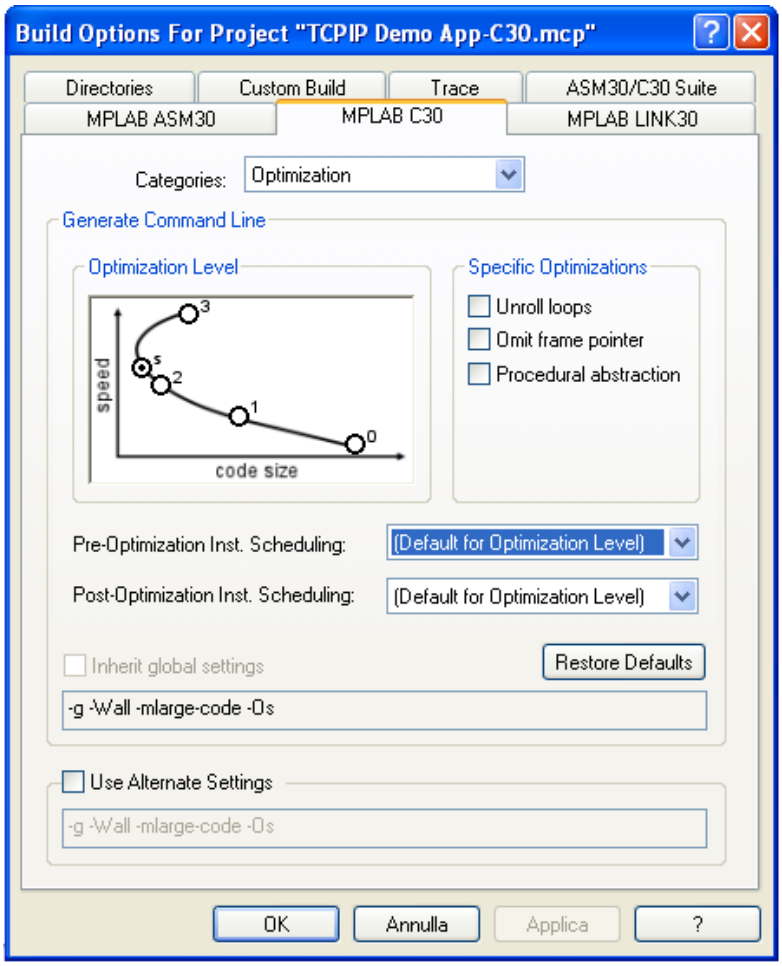


Figura 4.2: *Finestra di Ottimizzazione*

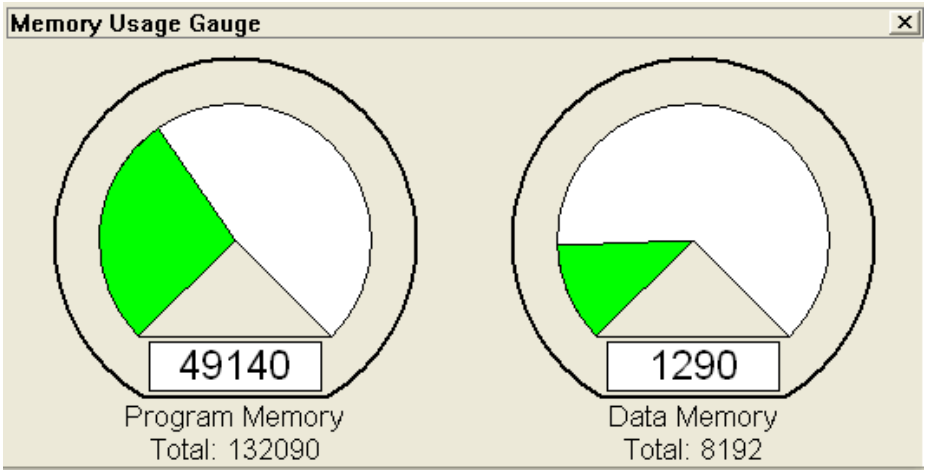


Figura 4.3: *Ottimizzazione finale*

## 4.4 Il Server HTTP2

Grazie al Server HTTP é possibile comunicare con la Board semplicemente con un web-browser, da un qualunque sistema operativo. Il server é in grado di prelevare un file dalla

Tabella 4.1: Spazio in Memoria richiesto

| Modulo | Program Memory Size(Bytes) |
|---|---|
| endfoot Codice Richiesto dallo Stack | 9,192 |
| DHCPClient | +4,107 |
| DHCPServer | +3,225 |
| HTTP2Server/MPFS2 | +20,133 |

memoria EEPROM ed inviarlo a destinazione. Inoltre una richiesta puó interagire con il software essenzialmente in due modi: in lettura, richiedendo dati tramite una pagina CGI(Common Gateway Interface), ed in scrittura ovvero inviando dei comandi. Il modulo web Server HTTP2 e il suo modulo di file system MPFS2 associato consente alla Board di comportarsi come un web Server. Questo facilita il modo di visualizzare informazioni di stato e di controllo. Tre componenti principali sono necessari per capire come funziona il web Server HTTP2:

- le pagine web
- l'utility MPFS2
- i files sorgenti CustomHTTPApp.c e HTTPPrint.h

Sempre nel file *TCPIPConfig.h* si possono impostare le opzioni di utilizzo del Server HTTP2: L'abilitazione di molte opzioni é legata direttamente alle impostazioni dello Stack descritte precedentemente; ad esempio, é qui che viene scelto il metodo di upload dei file(*mpfsupload*) e il file di default a cui fare riferimento (*index.htm*).

## 4.5 Il Server

Una visione d'insieme del progetto é mostrata nella Figura4.4 di seguito:

### Pagine Web

Questa componente include tutte le pagine HTML e le immagini associate, i fogli di stile CSS, e gli JavaScript necessari alla visualizzazione delle pagine web. Un'applicazione di

```
// -- HTTP2 Server options ------------------------------------------------

  // Maximum numbers of simultaneous HTTP connections allowed.
  // Each connection consumes 2 bytes of RAM and a TCP socket
  #define MAX_HTTP_CONNECTIONS (2u)
  // Indicate what file to serve when no specific one is requested
  #define HTTP_DEFAULT_FILE "index.htm"
  #define HTTPS_DEFAULT_FILE "index.htm"
  #define HTTP_DEFAULT_LEN (10u) // For buffer overrun protection.
                       // Set to longest length of above two strings.
  // Configure MPFS over HTTP updating
  // Comment this line to disable updating via HTTP
  #define HTTP_MPFS_UPLOAD "mpfsupload"
  //#define HTTP_MPFS_UPLOAD_REQUIRES_AUTH // Require password for MPFS uploads
    // Certain firewall and router combinations cause the MPFS2 Utility to fail
    // when uploading. If this happens, comment out this definition.
  // Define which HTTP modules to use
  // If not using a specific module, comment it to save resources
  #define HTTP_USE_POST    // Enable POST support
  #define HTTP_USE_COOKIES   // Enable cookie support
  #define HTTP_USE_AUTHENTICATION  // Enable basic authentication support
  //#define HTTP_NO_AUTH_WITHOUT_SSL // Uncomment to require SSL before requesting a password
  #define HTTP_SSL_ONLY_CHAR 0xFF // Files beginning with this character will only be served over HTTPS
                       // Set to 0x00 to require for all files
                       // Set to 0xff to require for no files
  #define STACK_USE_HTTP_APP_RECONFIG // Use the AppConfig web page in the Demo App (~2.5kb ROM,
      ~0b RAM)
  #define STACK_USE_HTTP_MD5_DEMO  // Use the MD5 Demo web page (~5kb ROM, ~160b RAM)
  //#define STACK_USE_HTTP_EMAIL_DEMO // Use the e-mail demo web page
```

Listing 4.3: Opzioni Server HTTP2

esempio che utilizza tutti questi componenti é stata modificata dalla versione di prova della Microchip, semplificandola per l'utilizzo nel progetto. Nel semplificarla sono stati mantenute le caratteristiche di utilizzo dei LED, dei Pulsanti , del Potenziometro e dello schermo LCD, il tutto a titolo di esempio.

### Utility MPFS2

Questo programma/utility della Microchip, impacchetta le pagine web in un formato che puó essere memorizzato in maniera molto efficiente nella memoria esterna non-volatile ,

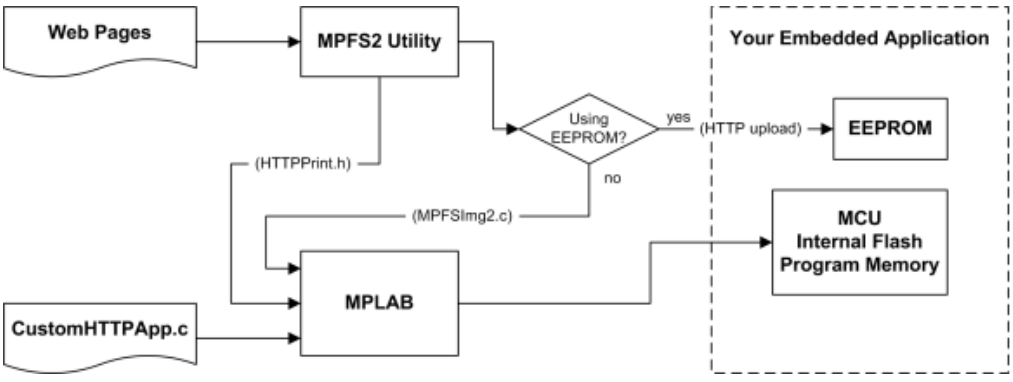


Figura 4.4: *Visione Progetto*

oppure nella memoria flash interna. Il programma esegue un compito molto importante, perché indicizza anche le variabili dinamiche trovate nelle pagine web e le memorizza nel file *HTTPPrint.h* aggiornandolo. Questo file é l'anello di connessione vero e proprio tra utente e Board: infatti fa riconoscere le variabili dinamiche modificate nelle pagine web allo Stack integrato nella Board. Se viene utilizzata la memoria esterna EEPROM, come nel nostro caso, l'utility MPFS2 fornisce in output un file BIN e puó eseguire l'ipload del file direttamente nella Board, oppure tramite il browser web entrando nella sottodirectory http://*IP_Address*/mpfsupload. Nel Progetto si é utilizzato l'upload via web: questa scelta é stata fatta per prediligere maggiormente l'interazione Client/Server. Se invece il dato viene memorizzato nella memoria di programma flash, l'MPFS2 genera un file sorgente C da includere nel progetto. Questa ultima scelta richiederebbe la ricompilazione dell'intero progetto tramite l'MPLAB IDE ogni volta che viene modificata la pagina web e riprogrammato il PIC24. Quando le variabili dinamiche vengono aggiunte o rimosse dall'applicazione web, l'utility MPFS2 aggiorna il file HTTPPrint.h. Quando ció avviene, il progetto deve essere ricompilato obbligatoriamente nel MPLAB IDE per assicurarsi che i cambiamenti vengano messi in atto, in particolare i nuovi indici variabile.

**CustomHTTPApp.c**

Questo file implementa l'applicazione web di esempio. Descrive l'output per le variabili dinamiche, (richiamando i metodi che si trovano nel file HTTPPrint.h definiti con la sintassi *HTTPPrint_NomeVariabile*), copia i dati passati tramite i moduli (nei metodi *HTTPExecuteGete HTTPExecutePost*) e valida le credenziali di autorizzazione (nel metodo *HTTPAuthenticate*). Le funzionalitá specifiche riguardo a queste funzioni sono descritte con le pagine web dell'applicazione di origine della Microchip nella cartella WebPages2.

**HTTPPrint.h**

Questo file viene generato automaticamente dall'utility MPFS2. Indicizza tutte le variabili dinamiche fornendo in tal modo il collante importantissimo tra le variabili che

risiedono nelle pagine web e le loro funzioni *HTTPPrint_NomeVariabile* definite nel file CustomHTTPApp.c. Questo file non deve essere modificato, perché costituisce la mappa dei collegamenti funzioni-variabili.

### 4.5.1 Variabili Dinamiche HTTP2

Le variabili dinamiche vengono create nella pagina web semplicemente includendo il nome della variabile racchiuse entro un paio di tilde(∼). Quando viene eseguito l'MPFS2, questo indicizza automaticamente le variabili dinamiche nel file HTTPPrint.h. Quest'ultimo é l'indice di tutte le variabili presenti nelle pagine web create e ha il compito di indicare all'applicazione (nel nostro caso *CustomHTTPApp.c*) quale funzione invocare: ad esempio la variabile dinamica MyVariable viene indicizzata con la funzione *HTTPPrint_MyVariable*. Possono essere passati anche parametri alle variabili dinamiche semplicemente inserendo tra parentesi tonde un valore e postponendole dopo il nome e prima della seconda tilde.Ad esempio ∼led(2)∼ stamperá il valore del secondo led (il numero 2) della Board; in particolare il valore tra parentesi tonde é il valore in ingresso della funzione creata nel file HTTPPrint.h e definita nel file CustomHTTPApp.c: il valore numerico viene passato come valore di tipo WORD alla funzione[3]. In maniera differente ancora avviene la comunicazione con l'LCD, in particolare il protocollo HTTP opera con un Buffer statico per la trasmissione.

## 4.6 Metodi GET e POST

Molte applicazioni hanno bisogno di accettare i dati da un utente. Una soluzione comune é quella di presentare un modulo all'utente in una pagina web, poi attraverso questo modulo vengono passati i valori alla Board. I moduli web di richiesta vengono passati utilizzando uno di due metodi(GET e POST), e il web server HTTP2 li implementa entrambi.

[3]Nel progetto il parametro, che va a definire ad esempio quale led deve essere acceso, viene definito nel file mchp.cgi

### 4.6.1 GET

Il metodo GET é concepito per chiedere informazioni ad un server, inviando pochi parametri tramite URL attraverso la stringa di query: aggiunge i dati alla fine dell'indirizzo web (query) di seguito al marcatore "?" nella barra di indirizzo del browser (es:http://169.142.1.1/ forms.htm?led1=0&led2=1&led3=0). I dati passati tramite il metodo GET sono automaticamente decodificati e memorizzati nell'array *curHTTP.data*. Dal momento che vengono memorizzati in memoria, questi dati sono limitati nello spazio messo a disposizione dalla dimensioni di *curHTTP.data*, che per default é posto a 100 bytes. In generale risulta facile processare i dati ricevuti in questa maniera. La funzione *HTTPExecuteGet* é implementata per poter processare questi dati e per eseguire ogni azione necessaria. Le funzioni *HTTPGetArg* e *HTTPGetROMArg* forniscono il metodo per tradurre e recuperare i valori passati per essere infine processati. La sequenza di azioni eseguite viene illustrata di seguito tramite un esempio:

1. l'utente seleziona i LED1 e i LED2 e al server viene passata la seguente stringa: GET /leds.htm?led1=1&led3=1

2. il server HTTP analizza la stringa e memorizza la stringa led1\01\0led3\01\0\0 nell'array *curHTTP.data*

3. viene ora chiamato il metodo *HTTPExecuteGet* che processa i dati ricevuti; il processo é suddiviso in alcuni passaggi:

   (a) viene chiamata la funzione *MPFSGetFilename* per verificare quale modulo é stato passato(questo passaggio puó venire omesso se l'applicazione prevede un solo modulo)

   (b) viene posto uno stato di default

   (c) la chiamata deve cercare per ogni argomento possibile che puó aspettarsi, compara il valore e imposta i pin dei LED

### 4.6.2 POST

Il metodo POST trasmette i dati dopo che tutte le richieste sono state inviate. Questi dati non sono visibili nella barra di indirizzo del web browser; puó essere visualizzato solo tramite un tool di cattura dei pacchetti. Tuttavia utilizza lo stesso metodo di codifica URL del metodo GET. É concepito per inviare al server molte informazioni, senza limite sulla quantitá di dati e sul loro tipo ed appunto in modo invisibile da URL[4]. In realtá il metodo POST sostituisce il metodo GET qualora la richiesta effettuata tramite GET superi il massimo dei caratteri consentiti dal server e le informazioni da inviare non siano solo di tipo alfanumerico, ma non é il nostro caso.

## 4.7 Pagina HTML

Tutti i files necessari alla creazione del file .bin da caricare nella Board sono raccolti nella cartella WebPages3: dove si trovano i files *.htm*(intex e forms), *.inc*(header e footer), *.css*(*Cascading Style Sheets* per lo stile della pagina). In quest'ultimo file *mchp.css* sono presenti le differenti impostazioni per i vari elementi della pagina, identificati con delle etichette-strutture, richiamate poi nelle pagine *.htm* e *.inc*. La pagina HTML é sostanzialmente divisa in due parti: un index.htm e un forms.htm. Entrambi i file son strutturati, a livello html, in tre parti: un header, un footer che rimangono sempre invariati e scritti in files separati (*header.inc* e *footer.inc*), e da un corpo principale che varia invece a seconda della pagina. Il file header richiama il file *mchp.css* che contiene le caratteristiche stilistiche del documento; a seguire richiama il file *mchp.js*, dove viene definita la funzionalitá AJAX , in particolare newAJAXCommand e poolAJAX per il refresh automatico delle pagine; nel campo $<body>$ si inserisce un'immagine della Microchip a decorazione e un menú di scelta interfacciato con le due pagine index.htm e forms.htm. Il file footer é la base della struttura, dove si é inserito una semplice riga di testo.

[4]Negli sviluppi futuri del progetto il metodo POST non ha senso di essere utilizzato, in quanto interessati al solo richiedere informazioni al Server, e non inviarne

# Capitolo 5

# Prova di Comunicazione NCAP-Remoto

Si collega l'EXPLORER16 al PC dopo aver programmato il microprocessore tramite l'ICD2 con il file esadecimale delo Stack TCP/IP opportunamente modificato prodotto dal MPLAB IDE. Nella realizzazione del progetto si é voluto verificare la corretta integrazione dello Stack modificato nella Board, mettendo alla prova gran parte delle funzionalitá della scheda di sviluppo. In particolar modo il Display LCD, i 4 pulsanti (RD6 S3, RD7 S6, RD7 S5, RD13 S4),gli 8 led(D10, D9, D8, D7, D6, D5, D4, D3) e il Potenziometro da 10 KΩ.[1]

## 5.1 Accensione del Kit

Innanzitutto si collega il modulo Ethernet alla Board in uno dei due slot liberi. Si collega poi la scheda all'alimentazione tramite il trasformatore a 9V avendo cura di rispettare la polaritá indicata; in seguito, una volta stabilita la connessione, appare sull'LCD della Board5.1, la versione dello Stack installato e l'indirizzo IP al quale collegarsi tramite un qualsiasi Web Browser.

[1]Nella Board utilizzata il LED D10 risulta danneggiato, mantenendo una costante luce debole; ció non ha comunque prodotto particolari problemi nell'utilizzo della Board, in particolare nell'effettuare chiamate di cambiamento di stato del LED, lo stesso semplicemente non rispondeva

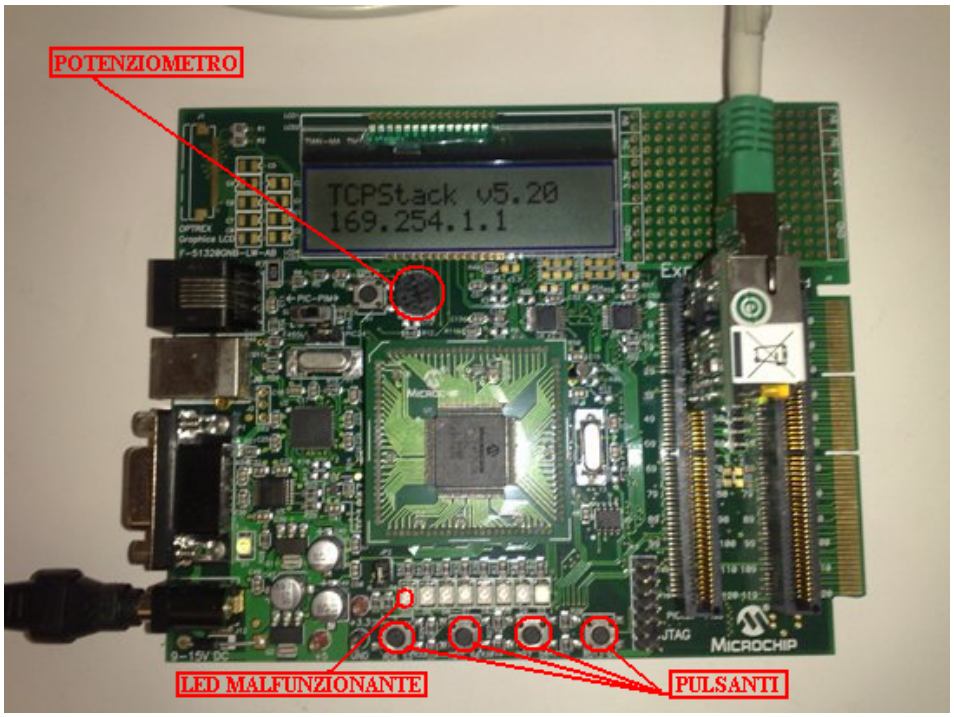


Figura 5.1: *Visualizzazione Versione dello Stack e indirizzo IP assegnato*

## 5.2 MPFS2

L'MPFS2 é un software incluso nel pacchetto Microchip che incorpora l'intero Server in un unico file .bin da caricare via internet con l'apposita applicazione. Una volta scritte le pagine web da mettere on-line, vengono salvate in una cartella assieme ai file .gif (ed eventualmente altri files necessari alla corretta visualizzazione delle pagine). Nel nostro progetto i file sono stati salvati nella cartella WebPages3, la quale viene indicata all'MPFS2 assieme a una directory di uscita, diversa da quella di partenza, dove il programma salva il file da caricare nella EEPROM della Board, nel nostro caso chiamato MPFSImg2.bin (come evidenziato in Figura5.2). Il programma, se non sono avvenuti errori, esegue correttamente creando il file richiesto( finestra mostrata in Figura5.3). Una volta creato il file, si apre con un browser l'indirizzo del Server, che nel nostro caso in esempio é http://169.254.1.1; il server non contiene nessuna pagina web visibile, dal momento che non áncora stato caricato nessun file. Per questo progetto é stata scelto l'upload da internet tramite l'utility "mpfsupload" implementata nello stack e raggiungibile digitando l'indirizzo http://169.254.1.1/mpfsupload: appare a questo punto la pagina che permette di caricare il file( come da Figura5.4): Nel campo, utilizzando il comando "sfoglia", si

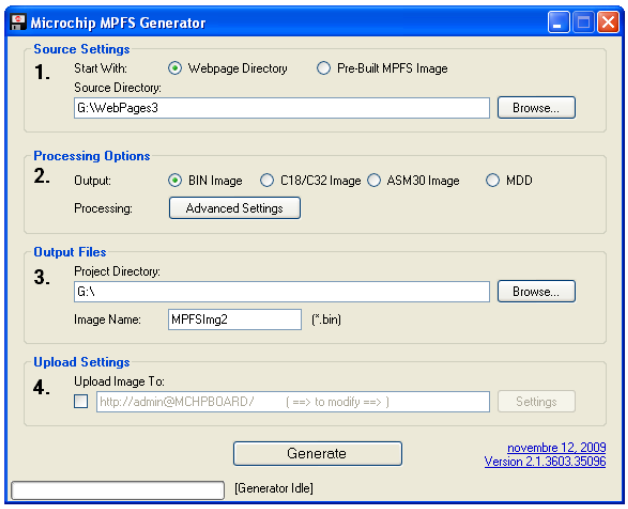


Figura 5.2: *Utilizzo dell'MPFS2*

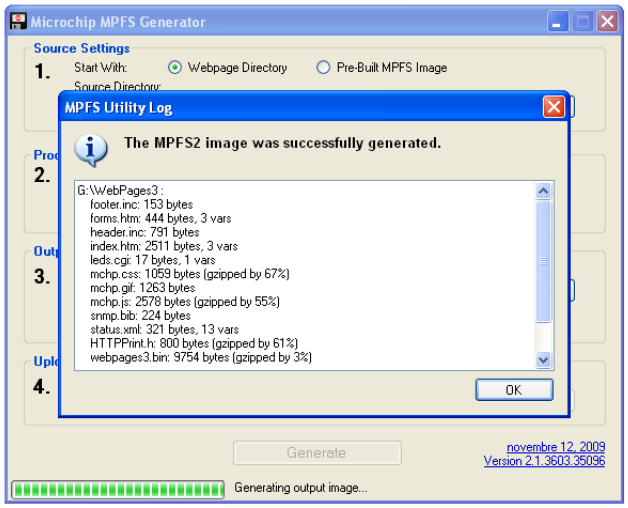


Figura 5.3: *Operazione eseguita con successo*

inserisce il percorso dove si trova il file MPFSImg2.bin. Nel nostro caso specifico abbiamo caricato il file chiamato webpages3.bin ( come mostrato in Figura5.5) ottenendo il messaggio di upload avvenuto con successo( mostrato nella Figura5.6): Ora le pagine web sono state caricate correttamente nell'EEPROM della Board ed é possibile aprirle da browser, in particolare é possibile, con questa versione di prova, interagire con i componenti dell'EXPLORER16.

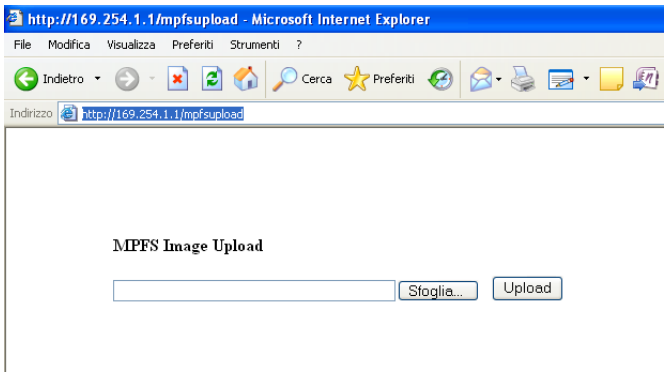


Figura 5.4: *pagina mpfs upload*


Figura 5.5: *Upload pagine*

## 5.3 Overview

Aprendo l'indirizzo *http://169.254.1.1/index.htm* si apre la schermata principale(come mostrato in Figua5.7), in particolare la schermata Overview, dove appaiono i campi con cui interagire: rispettivamente , dall'alto verso il basso, i Leds,i Pulsanti e il Potenziometro . Cliccando nei campi dei Led ( visualizzati nella Figura5.8), il Led in corrispondenza si accende o si spegne( come si puó notarte nella Figura5.9), ad eccezione dell'ultimo il quale, da scelta progettuale, lampeggia ripetutamente. Cliccando invece i pulsanti della Board, nella pagina web corrispondentemente le icone a forma di freccia variano direzione. Il campo potenziometro, ruotando manualmente il relativo componente installato nella Board, visualizza nella pagina web un intervallo di valori che vanno da 0 a 1023.


Figura 5.6: *Success Upload*

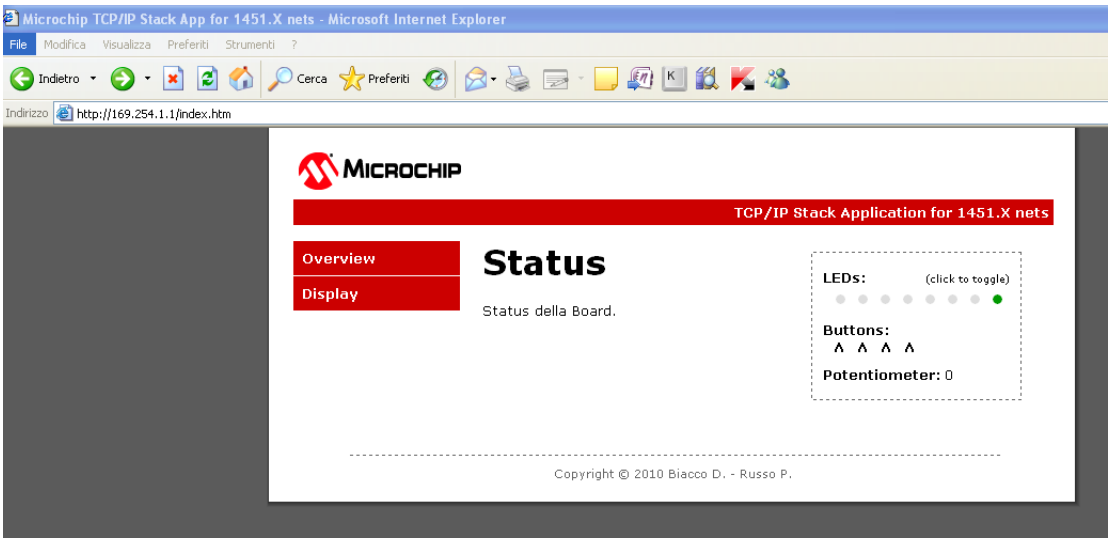


Figura 5.7: *Index.htm*

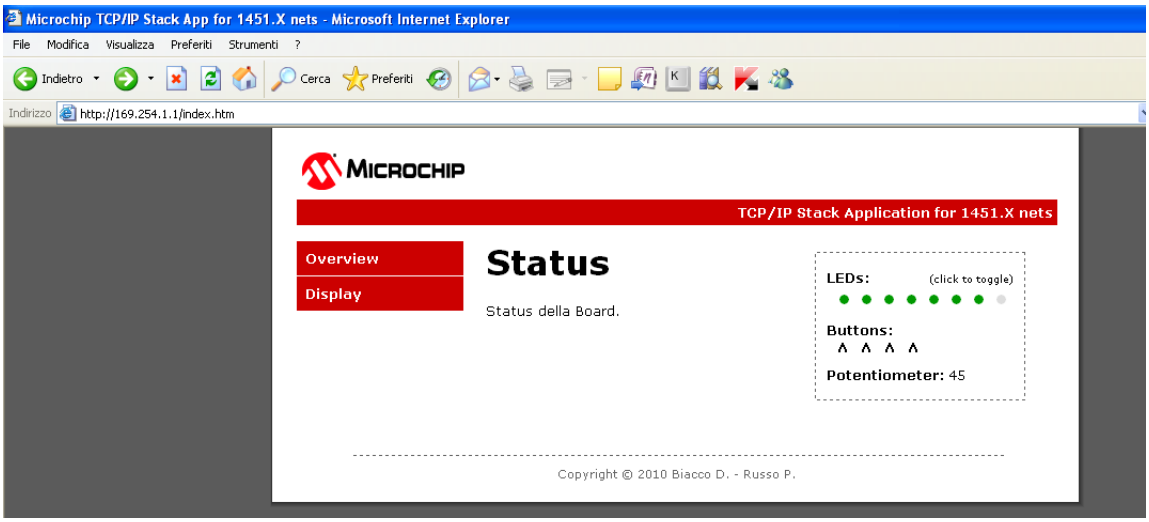


Figura 5.8: *Prova di accensione Led*

## 5.4 Display

Aprendo *http://169.254.1.1/forms.htm* é possibile dialogare con lo schermo LCD: scrivendo una breve stringa di testo nel campo apposito della pagina web(come visualizzato in figura5.10), essa viene acquisita con il metodo POST dal Server che la salva nella memoria della Board per essere poi visualizzata nello schermo LCD (nella Figura5.11viene visualizzata una stringa di prova).

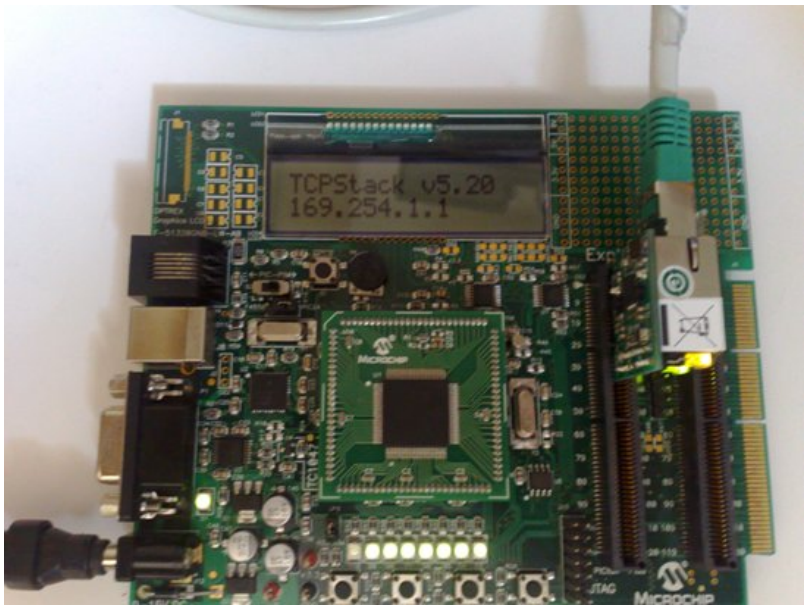


Figura 5.9: *Led accesi sulla Board*

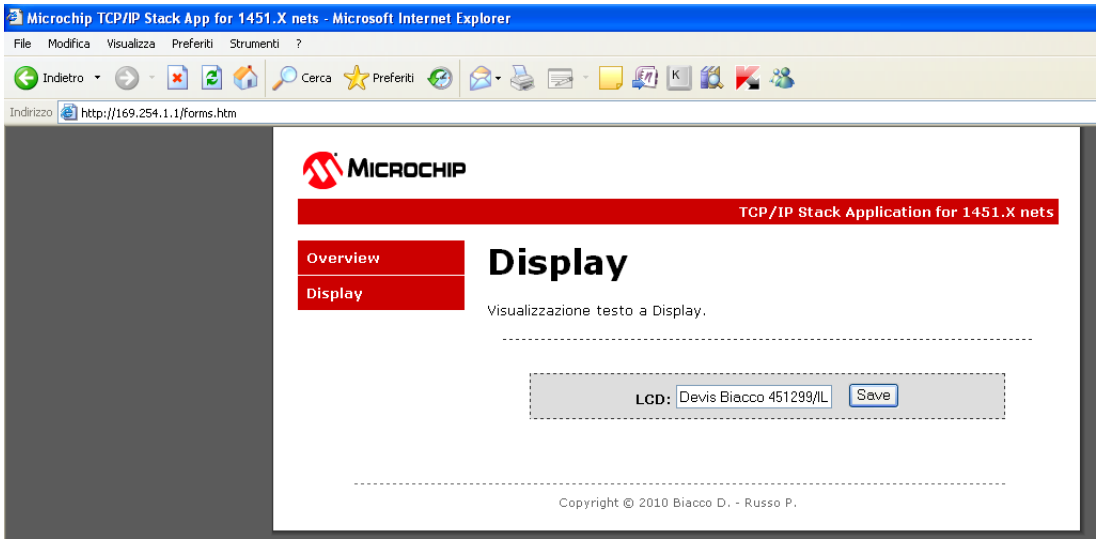


Figura 5.10: *Prova di scrittura su LCD*


Figura 5.11: *Visualizzazione su LCD*

# Capitolo 6

# Conclusioni

Scopo del progetto era integrare il Protocollo TCP/IP all'interno del microprocessore dell'EXPLORER16, utilizzando il minor spazio di memoria, e verificarne il funzionamento. Per integrare il Protocollo si é utilizzato lo Stack TCP/IP della Microchip. Dopo un'attento studio dell'Application Note 833 (AN833) fornito assieme allo Stack, l'integrazione é stata inizialmente affrontata eliminando ad uno ad uno file meno utili, o quantomeno inessenziali per il corretto funzionamento dello Stack; in un secondo momento, dopo numerose difficoltá dovute alle interdipendenze esistenti tra i vari file dello Stack, si é iniziato a vedere lo Stack dalla giusta ottica, analizzando correttamente il file di configurazione incluso nello Stack, il *TCPIPConfig.h*, nel quale, con dovute accortezze, si é potuto integrare lo Stack senza particolari problemi. Particolari problemi sono stati trovati nella comprensione e nell'utilizzo del Server HTTP2, a livello di interazione Client/Server, superati con l'ausilio di un'applicazione Custom, sempre messa a disposizione da Microchip e opportunamente modificata. Il progetto di integrare il protocollo TCP/IP, in particolare il Server HTTP, nell'NCAP utilizzando la minima memoria necessaria, é andato a buon fine, sia per quanto riguarda le funzionalitá base, sia appunto per quanto riguarda il problema dello spazio utilizzato, lasciando ben quasi 82000 byte di memoria programma. Lo spazio in memoria rimasto potrá essere utilizzato in futuro, ad esempio per l'integrazione del protocollo ZigBee per la comunicazione Wireless.

# Appendice A

# Materiale a disposizione

Per lo svolgimento del progetto sono stati utilizzati hardware e software che verranno di seguito elencati.

## A.1 Hardware

### A.1.1 EXPLORER 16 development board

La Scheda di sviluppo(visualizzata in FiguraA.1) é composta dai seguenti elementi hardware principali:

- un microcontrollore PIC serie 24 (PIC 24FJ128GA010) su zoccolo da 100 pin.
- 4 pulsanti connessi alle porte I/O del microcontrollore;
- 8 LED connessi alle porte di I/O del microcontrollore;
- memoria EEPROM da 256 kByte con interfaccia seriale ;
- sensore di temperatura;
- display LCD, 2 righe da 16 caratteri;
- potenziometro analogico;
- un connettore RJ-11 per la programmazione del firmware;

- un connettore RS-232 per la programmazione del firmware per l'outputseriale;
- un connettore Jack femmina per l'alimentazione esterna.

Il PIC 24 in questione é costituito da :

- 128 kByte di memoria programmata
- 8 kByte di memoria RAM sincrona;
- 5 timer a 16 bit;
- 5 interfacce I/O di tipo capture/compare , con uscita PWM;
- 2 interfacce UART;
- 2 serial peripheral interface (SPI);
- 2 interfacce I2C;
- convertitore A/D a 10 bit, 16 canali;
- 2 comparatori;

Il segnale di clock primario per il microcontrollore é fornito da un oscillatore a 8 MHZ.

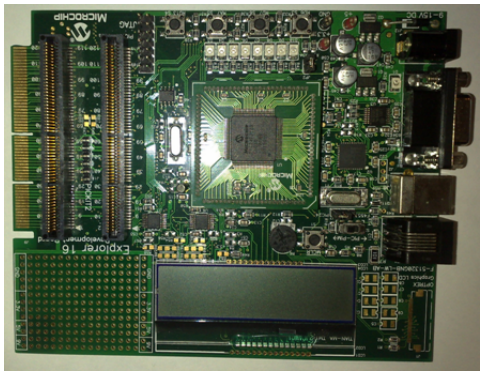

Figura A.1: *EXPLORER 16 development board*

### Connettori di espansione PICTail Plus

La scheda EXPLORER 16 é equipaggiata con due connettori PICTail Plus da 120 pin, divisi in tre sezioni da 30 - 30 - 56 pin rispettivamente,che permettono la connessione di moduli aggiuntivi. I due connettori sono connessi in parallelo e forniscono una facile via di accesso a gran parte dei pin del processore. Le due sezioni da 30 pin riportano connessioni equivalenti: ad esempio,i pin 1, 3, 5 e 7 del primo segmento di 30 pin riportano le connessioni al modulo SPI1 e gli stessi pin sul secondo segmento riportano le corrispondenti connessioni al modulo SPI2.

### PICTail Daughter Boards

Le soluzioni di Microchip permettono di estendere le capacitá del processore in uso in maniera estremamente economica e pratica: in questo modo si possono aggiungere interfacce di rete, interfacce radio, video e molto altro. Si tratta di piccole schede da inserire nell'apposito connettore di espansione della scheda di sviluppo la quale fornisce tutte le linee di connessione e l'energia. Per questo progetto é stata utilizzata una ENC28J60,(Ethernet PICtail Plus Daughter Board) connessa all'EXPLORER16.

### Ethernet PICtail Plus Daughter Board

L'Ethernet PICtail Plus Daughter Board (visualizzato in FiguraA.2) consente uno sviluppo di applicazioni di controllo Ethernet poco costoso e molto efficace. La scheda é progettata per la flessibilitá e puó essere collegato all'Explorer 16 di Microchip (DM240001). La scheda di sviluppo viene assemblata con l'ENC28J60 controller Ethernet di Microchip a 28-pin, che si interfaccia con il connettore RJ-45 femmina.

Figura A.2: *Ethernet PICtail Plus Daughter Board*

### A.1.2 materiali di connessione

- Cavo di rete cross per comunicazione PC-EXPLORER16(visualizzato in FiguraA.3)
- Alimentatore 9V per EXPLORER16(visualizzato in FiguraA.4)
- Cavo USB per connessione Programmatore-PC
- RJ-11 a 6 fili per connessione Programmatore-EXPLORER16

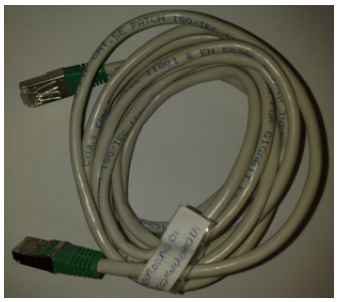

Figura A.3: *Cavo Cross Ethernet*

Figura A.4: *Alimentatore da 1.5V - 3V - 4.5V - 5V - 6V - 7.5V - 9V - 12V*

### A.1.3 Microchip MPLAB ICD 2

Questo dispositivo (visualizzato in FiguraA.5) é un programmatore per microprocessori PIC, che riunisce le caratteristiche di un In-Circuit Debugger (ICD) a basso costo e di un ICSP(In-Circuit Serial Programmer). Esso fornisce diverse funzionalitá come l'esecuzione del codice in tempo reale e passopasso, il monitoraggio e la modifica di variabili e registri, il debug direttamente nel circuito di prova, il monitoraggio della tensione di alimentazione del circuito, led di diagnostica (Power, Busy, Error ), un'interfaccia utente con Microchip MPLAB IDE e una connessione RS-232 o USB per il collegamento con il pc. Questo dispositivo permette di programmare il PIC 24FJ128GA010 attraverso l'interfaccia RJ-11 a 6 fili dell' EXPLORER 16 development board.


Figura A.5: *Microchip MPLAB ICD 2*

## A.2 Software

### A.2.1 Microchip MPLAB IDE

L'MPLAB Integrated Development Environment (IDE) é l'ambiente di sviluppo fornito da Microchip per la creazione di applicazioni basate su microcontrollori PIC. Esso permette di seguire tutte le fasi di sviluppo dell'applicazione. Contiene:

- un editor per la stesura del codice
- un project manager per gestire i file sorgenti e le impostazioni
- un compilatore per convertire il codice sorgente in linguaggio macchina
- un dispositivo per la simulazione e la gestione di un programmatore hardware che trasferisce il codice macchina al microcontrollore (il Microchip MPLAB ICD 2 )

### A.2.2 Microchip MPLAB C-30 Compiler

L'MPLAB C-30 Compiler é un add-on per l'ambiente di sviluppo integrato MPLAB sopra descritto che consente la compilazione di programmi scritti in linguaggio ANSI C per microprocessori della seria Microchip PIC24.

### A.2.3 Microchip TCP/IP Stack

Lo Stack TCP/IP (Transmission Control Protocol/Internet Protocol) Microchip é una suite di programmi che fornisce servizi alle applicazioni standard basate sul protocollo TCP/IP (HTTP Server, Mail Client, etc.). Lo Stack TCP/IP Microchip é implementato in una struttura modulare che rispecchia la struttura a livelli dello standard TCP/IP descritto al capitolo 2. Chi utilizza lo Stack non ha bisogno di sapere tutte le varie dipendenze delle funzioni implementate nello Stack TCP/IP per poterlo usare. Lo Stack basato su questo modello é diviso in livelli multipli, dove i livelli sono accatastati uno sull'altro(da cui il nome TCP/IP Stack) e ogni livello accede ai servizi da uno o piú livelli direttamente inferiori.

# Bibliografia


[1] http://ieeexplore.ieee.org.

[2] http://www.microchip.com.

[3] Giada Giorgi. Microsoft powerpoint slide1451.ppt. 2009.

[4] Nilesh Rajbharti Microchip Technology Inc. An833 - the microchip tcp/ip stack. 2008.

[5] IEEE Instrumentation and Measurement Society. Ieee std. 1451.0. 2007.

[6] IEEE Instrumentation and Measurement Society. Ieee std. 1451.5. 2007.

[7] Da Rold Mario. *Implementazione su microcontrollore PIC 24 di un nodo IEEE 1451 con interfaccia ZigBee*. PhD thesis, Universitá di Padova, 2009.

[8] Inc. Microchip Technology. Microchip tcp/ip stack 5.20 help. 2009.

[9] K.S.Siyan T.Parker. *TCP/IP tutto e oltre*. 2003.

# Ringraziamenti

*Grazie* ai miei genitori, Paolo e Imelda per avermi supportato nel proseguimento degli studi permettendomi di conseguire questo importante traguardo.

*Grazie* a mia sorella Vania per la pazienza portata con me e per essere sempre pronta ad aiutami.

*Grazie* ai miei zii Rita e Silvano per essere sempre stati disponibili nei momenti di bisogno.

*Grazie* a tutta la mia famiglia per il sostegno durante questo difficile percorso universitario.

*Grazie* ad Alessandra per aver creduto in me e il sostegno dato per finire questo importante percorso.

*Grazie* al Prof. Narduzzi per avermi dato la possibilitá di sviluppare questo appassionante progetto.

*Grazie* a Pierpaolo, compagno di laboratorio e degli ultimi esami sostenuti.

*Grazie* a tutti i miei amici, Nicola, Simone, Antonino, Luca, Poldo, Alessio, Lele, Rodrigo, Mattia, Andrea e tutti quelli che non nomino ma che sanno di esserci o di esserci stati.

*Grazie* al Taekwon-Do, mia passione e mio sfogo, e a tutti gli amici conosciuti con esso.

*Grazie* anche a B., che mi ha accompagnato nello studio dell'ultimo esame.

*Grazie* a tutti quelli che non hanno mai creduto in me, perché anche grazie a loro ce lh́o fatta.