METEO.DI 2.0

Andrea Acquaviva
Corso di Progetto di Sistemi a Sensore

AA 2018/2019

Indice

1 Introduzione

Nella fine del 2011 Giuseppe Fotino e Michele Tellarini hanno progettato e realizzato una stazione meteorologica basata su di un modulo esterno dotato di microcontrollore e sensori ed di un server interno per l'accumulo e la restituzione dei dati. Tale stazione, installata in via Comelico 39, è rimasta in funzione fino al 30 maggio 2016, fino a quando la mancanza di dissipazione alla CPU del server ne ha compromesso l'attività. In questo periodo ha collezionato circa 10MB di dati meteorologici suddivisi in due parti: i primi sono dati relativi ai primi 5 anni di attività, che forniscono un dato aggregato al giorno; i secondi sono i dati relativi al periodo dal 9/02/2015 al 30/05/2016, risultato di un'aggregazione ogni 5 minuti. Questo progetto d'esame intende riportare in attività tale stazione meteo nel nuovo dipartimento di via Celoria 18. Il nuovo contesto pone sfide non affrontate prima, in primis il rendere la stazione wireless dal punto di vista della connettività con il server.

1.1 Obiettivi

Il progetto si propone anzitutto di ripristinare la vecchia stazione meteo, verificandone il corretto funzionamento in ogni sua componente hardware e software. In seconda istanza viene inserita una board capace da una lato di comunicare con il controllore della vecchia stazione e dall'altro di gestire la connettività WiFi per il collegamento ad un server remoto. Tale server deve ricevere i dati, memorizzarli e restituirli pubblicamente su una pagina web, creando un servizio semplice, affidabile e sicuro.

2 Componenti

2.1 Stazione meteo 1.0

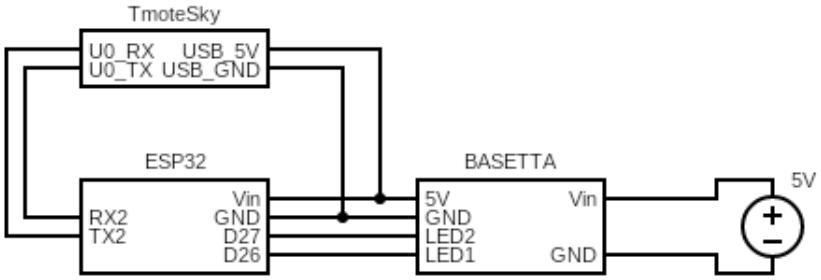
Della vecchia stazione meteo è stata mantenuta tutta la "centralina", ovvero il box composto dal modulo Tmote Sky e da una basetta di supporto per i sensori esterni. Questi ultimi sono stati riutilizzati in quanto ancora in buone condizioni. Durante i test effettuati per verificare il funzionamento della centralina è stato riscontrato che un'insufficiente corrente a disposizione della scheda porta all'invio di pacchetti corrotti, ovvero pacchetti il cui CRC (Cyclic Redundancy Check) calcolato è differente da quello a bordo del pacchetto (ulteriori informazioni nella sezione 3.5).

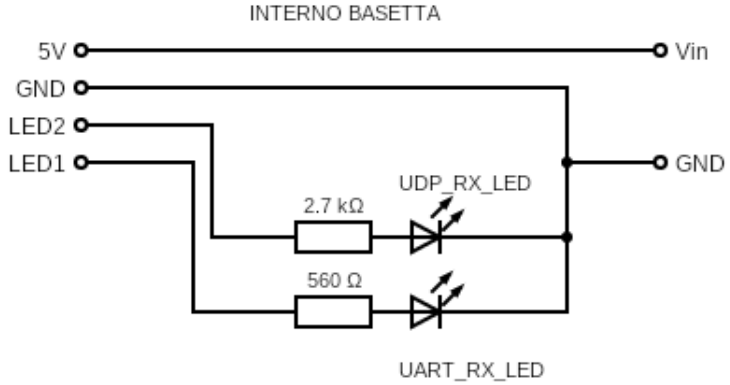
2.2 ESP32

É stata introdotta una board di tipo MCU (MicroController Unit), modello ESP32-WROOM-32 di Espressif, montata su di un kit di sviluppo. Questa MCU supporta connettività Wi-Fi 802.11 b/g/n, Bluetooth v4.2 BR/EDR e BLE, UART, SPI, SDIO, I2C. Il cuore della MCU è un microprocessore dotato di due core Xtensa® 32-bit LX6 operanti ad una frequenza massima di 240MHz, con supporto a istruzioni FP e DSP. La memoria disponibile è suddivisa in 448KB di ROM per i binari, 520KB di SRAM on-chip per dati/istruzioni. Sono inoltre disponibili 11MB di memoria flash. Indubbi vantaggi di questa scheda sono l'estrema compattezza in rapporto all'ampia disponibilità di soluzioni di connettività, il basso consumo e il buon supporto software grazie al Espressif IoT Development Framework (ESP-IDF).

L'ESP32 è stato collegato con il Tmote Sky, per permettere l'inoltro dei dati dei sensori al server remoto, attraverso interfaccia UART (Universal Asynchronous Receiver-Transmitter), in particolare la UART2 mappata sui

pin GPIO 16 e 17. Non è stato possibile usare la UART0, mappata sulla porta USB, perché già usata durante la programmazione del device e durante il boot [1]. Sono stati aggiunti sulla basetta due ulteriori LED di stato, collegati da una parte (lato anodo) ai pin GPIO 26 e 27 (frapponendovi rispettivamente una resistenza da 560Ω e da 2700Ω ciascuno) e dall'altra al pin di massa GND (lato catodo). Il primo LED è blu e il secondo è bianco.





2.3 Server

Il server è stato fornito dalla sala macchine del Dipartimento di Informatica, sotto forma di Virtual Private Server (VPS). Tale server è dotato di una CPU Intel Xeon quad-core da 2.8GHz, 8 GB di RAM e di uno spazio su disco di 98GB.

3 Software

3.1 Tmote Sky

Il codice binario originale in funzione sul Tmote Sky è stato modificato per consentire la comunicazione con l'ESP32. Il Tmote Sky in precedenza comunicava con il server attraverso l'interfaccia UART1, mappata sulla porta USB attraverso un convertitore seriale-USB FT232BM di FTDI. Questo modo di comunicare è valido solo in caso di comunicazione con un host USB, quale ad esempio un pc o server, in grado di enumerare[2] il device ad esso collegato; pertanto il Tmote Sky non può essere collegato via USB ad un device come l'ESP32. Per semplicità si è scelto di redirigere l'I/O dall'UART1 all'UART0, mappata direttamente sui pin 2 e 4 del connettore a 10pin messo a disposizione del Tmote. Tale modifica è stata realizzata su due file sorgenti del

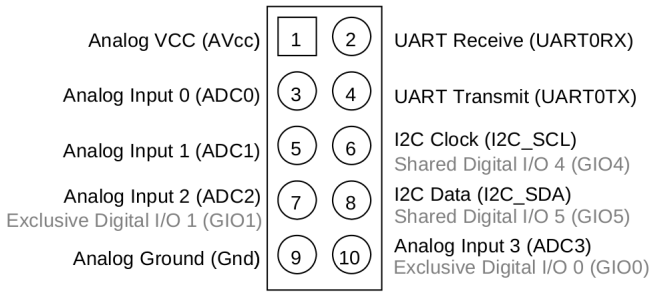


Figure 20 : Functionality of the 10-pin expansion connector (U2). Alternative pin uses are shown in gray.

sistema operativo TinyOS, essenziale nel mettere a disposizione le risorse del Tmote Sky al programmatore in modo semplificato. TinyOS è principalmente scritto in NesC, un dialetto del C basato sul concetto di evento. La sintassi del NesC è molto diversa dal C e prevede anzitutto la distinzione tra file di configurazione e programmi veri e propri. I file di configurazione effettuano il cosiddetto "wiring" delle componenti, ovvero risolvono le dipendenze dei componenti usati nei relativi programmi. Risalendo le dipendenze della funzione AMSend è stato trovato il file da modificare.

- AppConfig.nc importa SerialActiveMessage.nc
- SerialActiveMessage.nc importa SerialDispatcher.nc
- SerialDispatcher.nc importa PlatformSerialC.nc

- PlatformSerialC.nc importa Msp430Uart1C.nc

Modificando quest'ultima importazione con il file Msp430Uart0C.nc è stato possibile comunicare con il Tmote Sky attraverso la UART0.

Per poter generare e caricare sul Tmote un nuovo binario con le suddette modifiche si è dovuti ricorrere ad un installazione specifica di Ubuntu 12.04 LTS, dovendo utilizzare del software (TinyOS) sviluppato per quella versione, dopo aver verificato l'impossibilità a procedere con sistemi operativi più recenti.

3.2 ESP32

L'ESP32 può essere programmato con l'ausilio di due framework: il già citato ESP-IDF o arduino-esp32. Quest'ultimo permette di programmare nello stile tipico degli sketches di Arduino, con le funzioni setup() e loop(). La scelta è ricaduta sul primo per il supporto nativo che offre nei confronti dell'ESP32. Il codice realizzato si preoccupa di:

- inizializzare lo stack di rete necessario per la comunicazione con il server
- effettuare la connessione ad una specifica rete Wi-Fi
- autenticarsi presso il server remoto
- inoltrare i pacchetti ricevuti via rete su UART
- inoltrare i pacchetti ricevuti via UART sulla rete
- gestire i casi limite per un funzionamento 24/7

Il codice si struttura in 7 funzioni, gestite dal sistema operativo real time FreeRTOS, e da un main di setup. Gestire le funzioni come task di FreeRTOS semplifica la gestione della memoria e degli eventi, oltre a raggruppare nella stessa chiamata di funzione alcuni parametri relativi allo scheduler (priorità, task's handle, etc).

3.2.1 UART

Per comunicare attraverso la UART è stato previsto un task FreeRTOS che non termina. Esso alloca memoria per una quantità massima di bytes da ricevere, fa blinkare un LED bianco posto sulla basetta, crea e invia un pacchetto

UDP contenente i bytes ricevuti. Per evitare episodi di inconsistenza questi bytes vengono copiati in una nuova porzione di memoria prima di essere inviati al server remoto tramite protocollo UDP. Le due primitive ESP-IDF usate per l'invio e la ricezioni di dati sulla UART sono:

```
int uart_read_bytes(uart_port_tuart_num, uint8_t *buf,
                    uint32_t length,
                    TickType_t ticks_to_wait);
int uart_write_bytes(uart_port_tuart_num,
                    const char *src, size_t size);
```

3.2.2 WiFi

Il codice relativo alla connessione Wi-Fi consente di stabilire una connessione con un access point sulla banda di 2.4GHz, sia attraverso autenticazione WPA/WPA2 che WPA Enterprise con PEAP. Questo consente all'ESP32 di connettersi sia a reti Wi-Fi di tipo domestico che a reti più complesse come quella di ateneo, semplicemente configurando i relativi parametri e selezionando l'opportuno valore della costante CONN_SELECTOR. É infatti già presente in memoria il certificato necessario alla seconda modalità di autenticazione. Le credenziali di accesso possono devono essere scritte nello specifico header file credentials.h situato in /include. L'ESP32 effettua un tentativo di connessione ogni 3s e una volta che la connessione è avvenuta viene settato CONNECTED_BIT ad 1 ed acceso il led blu sull'ESP32. In caso di mancata connessione o di disconnessione dopo 30s un altro task (reboot_task) riavvia l'ESP32. É stato scelto di implementare il riavvio dell'ESP32 in quanto è un operazione veloce ($\ll$ 5s) e ottima dal punto di vista del reset degli stati e delle variabili coinvolte nella procedura di autenticazione alla rete WiFi.

3.2.3 UDP

La comunicazione tra ESP32 e server è affidata al protocollo UDP, scelto per la semplicità dell'approccio connection-less e per le caratteristiche del meccanismo di acquisizione dati. Quest'ultimo tollera bene la perdita di un pacchetto, in quanto viene spedito al server un dato ogni 5s che poi viene aggregato in una finestra di 5 minuti. Come per la UART anche qui vi è un task, sempre attivo, il cui compito è ricevere i pacchetto UDP destinati all'ESP32 (send_thread). Questo task alloca un buffer per la ricezione ed ogni volta che un pacchetto è ricevuto lo scrive sulla UART ed effettua un blink del

LED blu sulla basetta. Per l'invio di un pacchetto UDP è stata predisposta una funzione (send_thread) che riceve un puntatore ad un pacchetto UDP da inviare alla coppia indirizzo-porta del server remoto.

3.2.4 Autenticazione

Quando sono pronte sia le strutture per ricevere i dati sulla UART che quelle per la comunicazione UDP viene iniziato lo scambio di messaggi che autentica l'ESP32 presso il server. La procedura di autenticazione si è resa necessaria per contrastare possibili tentativi di spoofing. Con la tecnica spiegata in seguito ogni dispositivo che tenterà di emulare il comportamento dell'ESP32 inviando dati al server non verrà considerato dal server. Un dispositivo è autenticato presso il server quando viene effettuato il salvataggio del suo indirizzo IP dopo essersi dimostrato fidato. Il processo di autenticazione lato ESP32 avviene ogni 5min e prevede tre passi:

- calcolo di una stringa di 30 caratteri con una funzione hash bitwise molto leggera scritta da Justin Sobel[3], inizializzata con uno specifico seme (identico per l'implementazione su ESP32 e server).
- aggiunta del prefisso esadecimale "3C" al payload UDP per segnalare la volontà di autenticarsi. Esso si differenzia dal prefisso "7E" che contraddistingue i pacchetti inviati dal Tmote.
- invio della stringa così ottenuta al server tramite UDP.

3.3 Logstation

Per la ricezione e la memorizzazione dei dati è stato modificato lato server il programma già in funzione nella precedente installazione. La scelta è stata motivata dalla volontà di mantenere il sistema operativo TinyOS lato Tmote, e quindi dalla necessità di effettuare agilmente operazioni sui pacchetti da esso inoltrati al server. Il server, scritto in Java, dispone infatti di una libreria apposita per la decodifica del pacchetto, nonché del controllo della sua integrità e dell'estrazione dei suoi campi fondamentali. La sfida principale è stata quella di modificare il codice, nato per la ricezione di pacchetti su di una porta seriale, in modo da ricevere pacchetti da una socket UDP. Tale operazione ha reso necessaria la modifica delle librerie Java di TinyOS in più punti e la creazione di nuove classi. Da ultimo sono state effettuate alcune modifiche per adeguare il codice alla nuova struttura del database.

3.3.1 Server UDP

Anzitutto è stato predisposto un server UDP, inserito nella classe UdpServer del package network_bridge. Tale server viene inizializzato con i parametri contenuti nel file di configurazione LogStation.config, che deve essere obbligatoriamente presente al momento dell'esecuzione (nello stesso path dell'eseguibile del server). Il server attende in primis il pacchetto di autenticazione inviato dall'ESP32 come descritto in 3.2.4. Quando questo viene ricevuto è chiamato il metodo checkSender della classe Authentication, per verificare l'attendibilità del mittente. Se questo è attendibile viene salvato il suo indirizzo IP e il server comincia a ricevere pacchetti dall'ESP32, da depositare in uno specifica coda di lettura. Tale coda è stata implementata con la classe Java ByteQueue; la coda ha associato un oggetto di tipo Object chiamato readLock per poter notificare l'aggiunta di un elemento tramite i classici metodi di sincronizzazione wait() e notify().

3.3.2 Dalla porta seriale alla rete wireless

I metodi che creano il "ponte" tra la rete e la gestione della seriale della precedente logstation sono:

```
public static void serialToDatagram(byte[] payload)
public synchronized static byte datagramToSerial()
```

Il primo si occupa di redirigere i pacchetti destinati alla seriale all'ESP32, tramite il metodo send della socket UDP. Il secondo restituisce il primo byte del buffer solo se il buffer di lettura non è vuoto e in precedenza è arrivato un pacchetto non ancora letto interamente. Altrimenti il thread chiamante viene messo in wait, attendendo la ricezione di un nuovo pacchetto. La lettura di un byte per volta è resa necessaria da una funzione di TinyOS che analizza i pacchetti, ne identifica il prefisso e verifica il checksum.

Le modifiche alla libreria Java di TinyOS coinvolgono tre sorgenti: Packetizer, NativeSerial e TOSCommJNI. Nel primo nella funzione readFramedPacket() la chiamata alla funzione readByte() viene sostituita con la chiamata a datagramToSerial(). E' stato inoltre incrementato il valore di ACK_TIMEOUT da 1000ms a 3500ms, per andare incontro al maggior RTT (Round Trip Time) di un pacchetto TinyOS su rete wireless rispetto al cavo USB.

Nella classe NativeSerial è stato modificato il metodo write() sostituendo la chiamata al metodo nativo di scrittura sulla seriale (della classe TOSCommJNI) con la chiamata a serialToDatagram().

Nella classe TOSCommJNI sono stati sostituite alcune chiamate a metodi nativi con metodi stub, per rimuovere la dipendenza dalla porta seriale.

É stato quindi preferito un approccio che ha evitato l'eliminazione di intere classi ed ha preferito la loro modifica, l'introduzione di altrettante e di metodi stub. Questa modalità, oltre ad aver contribuito a portare in poco tempo un risultato concreto e funzionante, ha anche permesso di conservare tutte quelle funzionalità apprezzate della libreria Java di TinyOS lasciando intatte le molte dipendenze tra le classi esistenti, dipendenze difficili da isolare.

Si segnala inoltre che durante lo sviluppo è stato particolarmente utile aprire il canale seriale e leggere i messaggi scambiati tra la logstation e il Tmote Sky prima di procedere alla modifica del codice. Questo è stato possibile facendo uso di un adattatore USB-TTL, dotato di convertitore CP-2102.

3.3.3 Memorizzazione del dato

La memorizzazione del dato meterologico è stato affidato al RDBMS MariaDB, fork di MySQL inserito nello stack ufficiale di Red Hat Enterprise Linux, Slackware e Arch Linux. Sono state apportate alcune modifiche al salvataggio dei dati, in particolare:

- è stata eliminata la tabella history (conteneva un dato aggregato giornaliero) e la relativa procedura di aggregazione annuale
- è stato introdotto il salvataggio della raffica di vento (wind gust) nella tabella record (dato aggregato ogni 5min)

3.4 Sito web

Il server ospita anche un nuovo sito web che ha l'obiettivo di restituire in maniera semplice e fruibile i dati presenti nel DBMS. Il sito è strutturato in

tre pagine web: home/dashboard, dati storici e informazioni. La prima permette di leggere l'ultimo dato giunto sul server, che in condizioni di normale funzionamento è disponibile ogni 5sec. Nella seconda pagina vengono raggruppati grafici dei principali parametri della base di dati, visualizzabili su base giornaliera, settimanale, mensile, annuale o con un intervallo personalizzato. In fondo alla pagina è possibile scaricare in formato CSV i dati prodotti nella precedente installazione della stazione meteo o quelli dell'installazione corrente. I grafici sono stati realizzati sfruttando le Google Charts, strumento di visualizzazione di dati potente e flessibile. É stato così sufficiente preparare i dati con una funzione PHP, richiedendoli prima al DBMS e convertendoli in JSON, gestendo poi le charts in Javascript. Ulteriori semplificazioni sono state introdotte dall'uso di classi CSS create dal W3C e dal preprocessore LESS per CSS per la creazione di una bussola.

3.5 Installazione

Il server fornito dalla sala macchine è dotato di un sistema CentOS Linux (release 7.6.1810) già dotato di webserver Apache, PHP, PHPMyAdmin e MariaDB. Per rendere operativo e sicuro il server è stato necessario il seguente setup:

- il database è stato inizializzato con le istruzioni riportate in new_db.sql. É sufficiente eseguire tali istruzioni dal box di query di PHPMyAdmin a database vuoto
- il firewall è stato configurato con il seguente comando:

  ```
  firewall-cmd --zone=public --permanent --add-port=50000/udp
  ```

  In questo modo sono consentiti i pacchetti in entrata provenienti dall'ESP32 (porta UDP 50000)
- i file necessari per il funzionamento del sito web devono essere spostati in /var/www/meteo.di.unimi.it/html
- l'eseguibile jar della Logstation può essere posizionato ovunque, ma deve essere accompagnato (nello stesso path) dal file di configurazione logstation.config
- in logstation.config e nel file sorgente dell'ESP32 va riportato l'IP statico assegnato al server

- la logstation viene gestita per mezzo di un servizio systemd dedicato, per poter gestire agilmente il riavvio in caso di fail e l'autostart. Tale servizio è descritto dal file /etc/systemd/system/weather.service e può essere lanciato dopo aver dato i seguenti comandi:

```
systemctl daemon-reload
systemctl start weather.service
```

- è necessario alimentare l'ESP32 e il Tmote Sky con un alimentatore 5V che supporti un output massimo di almeno 1A per supportare le richieste di picco delle due board.

4 Conclusioni e sviluppi futuri

Durante la realizzazione del progetto sono state affrontate molte sfide, dalla riprogrammazione del Tmote Sky allo sviluppo della comunicazione tra Tmote e server, intermediata dall'ESP32. Inizialmente sono stati fatti anche numerosi tentativi di comunicazione tra le due board tramite USB, interfaccia rivelatasi non idonea allo scopo (vedi 3.1). Un ultima sfida è derivata dal fenomeno di corruzione di pacchetti lato server, rivelatosi dovuto ad un insufficiente alimentazione del Tmote.

All'inizio del progetto è stata valutata la possibilità di integrare nel sistema anche un sensore di polverosità ambientale Sharp GP2Y1010AU0F, già gestito da un microcontrollore Texas Instruments MSP-EXP432P401R LaunchPad Rev.B e dotato di software per la lettura del dato. Purtroppo è stato riscontrato il malfunzionamento del sistema, incapace di acquisire dati dalla scheda; nonostante numerosi tentativi non è stato possibile riprogrammare il microcontrollore, a causa dell'impossibilità di risolvere alcune dipendenze obbligatorie nella compilazione dei sorgenti. Questo deficit è probabilmente dovuto alle molteplici modifiche che si sono susseguite alla SDK per MSP432 (ulteriori informazioni sono disponibili al capitolo 3 del documento Moving From Evaluation to Production With SimpleLink™ MSP432P401x Microcontroller [4]). Tuttavia esistono due strade viabili: riscrivere il codice del microcontrollore con la SDK attuale o gestire il sensore di polverosità con un altro controllore, ad esempio direttamente con l'ESP32. Infine un ulteriore miglioria potrebbe interessare l'alimentazione del sistema, scollegando la stazione meteo dalla presa a muro di corrente elettrica, rendendola quindi energeticamente autonoma con un kit di batteria, pannello solare e regolatore di carica.

Bibliografia

[1] G6EJD. *How to use ESP32 hardware serial ports*. URL: `https://github.com/G6EJD/ESP32-Using-Hardware-Serial-Ports`.

[2] Vincenzo Villa. *I compiti dell'host e dei device USB*. URL: `https://www.vincenzov.net/tutorial/USB/host_e_device.htm`.

[3] Arash Partow. *General Purpose Hash Function Algorithms*. URL: `http://www.partow.net/programming/hashfunctions/#JSHashFunction`.

[4] Texas Instruments. *Moving From Evaluation to Production With SimpleLink^{TM} MSP432P401x Microcontrollers*. URL: http://www.ti.com/lit/an/slaa700a/slaa700a.pdf.