Programmi presentati durante il seminario hands-on su IoT

di Augusto Ciuffoletti per il Corso di Telematica - Unformatica Umanistica - Università di Pisa

30 aprile 2019

Questo documento raccoglie i programmi proposti durante il laboratorio *hands on* su Internet Of Things (IoT) che ho preparato per il corso di Telematica (prossimamente Protocolli e Servizi di Rete) durante l'AA 2018/19.

I programmi puntano sull'argomento del corso, quindi privilegiano gli aspetti di rete alla sensoristica. Il vantaggio è che per realizzare le attività è sufficiente collegare il solo modulo ESP8266 al PC, senza aggiungere altri componenti.

L'ordine degli esercizi non è progressivo nella complessità dei programmi, ma rispetto ad un persorso bottom-up nella gerarchi dei protocolli: quindi si parte da "network" e si arriva a "application". Il primo degli esercizi si colloca fuori schema e serve a fare un check iniziale.

I programmi, nell'ordine:

- blink: l'"hallo world" dei SoC
- IoT_WiFI: scansione degli AP
- IoT_AP_ServerTCP: creazione di un Access Point con un server HTTP in ascolto
- IoT_Join: collegamento ad un Access Point esterno
- IoT_dweet: feed di un server IoT

Tutti i programmi, e questo documento, sono nel repository git su `https://bitbucket.org/labreti/lab_iot/src`

1 Prima di cominciare: blink (sketch `IoT_0`)

```
/*
 * Questo programma e' il primo della serie per il tutorial IoT
 * del corso di telematica.
 * E' il "solito" blink, con una piccola sorpresa nel loop
 */
#define LED 2 // Il LED sulla scheda e' collegato al pin 2

// Questa funzione viene eseguita una sola volta all'accensione
void setup() {
  // definisce la modalita' del pin 2 (LED) come output
  pinMode(LED,OUTPUT);
  // quando il pin viene messo a livello basso il LED si accende
  digitalWrite(LED,LOW);
  // per 5 secondi
  delay(5000); // ritardo in millesecondi
}

// Questa funzione viene ripetuta indefinitamente
// Terminato il setup il LED si accende 1 volta al secondo
void loop() {
  digitalWrite(LED,!(digitalRead(LED)));  // inverto lo stato del LED
  delay(500);                             // Attendo mezzo secondo
}
```

Listing 1: Blink

2 Osserviamo la rete (sketch `IoT_WiFi`)

```
/*
 * Ora usiamo l'interfaccia WiFi del modulo per visualizzare alcuni
 * dati dagli AP WiFi raggiungibili.
 * Per fare cio' utilizziamo una funzione di libreria specifica per
 * l'ESP8266 che costruisce e rende accessibili un array contenente
 * tali dati. Ogni elemento dell'array viene poi letto e stampato.
 */
#include "ESP8266WiFi.h"
#define LED 2 // Il LED sulla scheda e' collegato al pin 2

void setup() {
  Serial.begin(115200);    // Questo consente di visualizzare sul PC
  pinMode(LED,OUTPUT);     // Questo LED e' acceso durante la ricerca
  digitalWrite(LED,HIGH); // Inizialmente spento (HIGH -> spento)
  WiFi.mode(WIFI_STA);     // L'interfaccia e' configurata come stazione
}

void loop() {
  char line[80];                        // buffer delle righe di stampa
  digitalWrite(LED,LOW);                // LED acceso
  int n = WiFi.scanNetworks();          // Scansione
  while (! WiFi.scanComplete() ) { }; // Attendo che abbia terminato
  digitalWrite(LED,HIGH);               // LED spento
  if (n == 0) {                         // n = numero di AP trovati
    Serial.println("no networks found");
  } else {
// Stampa dei dati raccolti,
    sprintf(line,"%d networks found\n", n);
    Serial.print(line);
// Ciclo, per ogni AP trovato (0..n-1)
    for (int i = 0; i < n; ++i) {
      // Preparazione della riga di buffer
      sprintf(line,"%02d: %30s | sig=%02dDb | ch=%02d\n",
        i+1,                    // posizione (1..n) dell'AP
        WiFi.SSID(i).c_str(), // SSID dell'i-esimo AP trovato
        WiFi.RSSI(i),         // potenza dell'i-esimo AP trovato
        WiFi.channel(i));     // canale dell'i-esimo AP trovato
      // Stampa della riga di buffer
      Serial.print(line);
    }
  }
  Serial.println();
  delay(5000);  // Attendo 5 secondi prima di ripetere
}
```

Listing 2: Osserviamo la rete

3 Creiamo un Access Point con un server HTTP I (sketch IoT_AP_ServerTCP)

```
/*
 * L'interfaccia WiFi si comporta ora da Access Point, e
 * mette a disposizione un server TCP. Il server riproduce
 * cio' che gli viene spedito sul monitor serial
 */
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#define LED 2
// Credenziali per l'Access Point (meglio cambiarle)
#define ESSID "WEMOS_mio"   // ESSID dell'AP
#define PASSWORD "password" // almeno 8 caratteri

// Creazione dell'oggetto server sulla porta 80 (HTTP)
WiFiServer TCPServer(80);

void setup() {
  delay(1000);
  Serial.begin(115200);         // Abilita il monitor
  pinMode(LED,OUTPUT);
// Configurazione dell'Access Point con essid e password
// Viene configurato anche un server DHCP nella rete 192.168.4.0/18
  if (! WiFi.softAP(ESSID,PASSWORD)) {
    Serial.println("Creazione dell'AP fallita");
    while (true) {}
  }
// Lettura e stampa sul monitor seriale dell'indirizzo IP
  Serial.print("\nIndirizzo IP dell'Access Point: ");
  Serial.println(WiFi.softAPIP());
// Attesa: connettersi all'AP ora
  digitalWrite(LED,LOW);
  delay(20000);
  digitalWrite(LED,HIGH);
//  Avvio del server TCP
  TCPServer.begin();
}
```

Listing 3: Creiamo un Access Point

Questo primo frammento comprende la funzione di setup che costruisce l'Access Point. Nel prossimo invece vediamo il server TCP.

Attenzione: le due funzionalità sono abbastanza pesanti, quindi il *join* può essere problematico e fallire spesso. Per semplificarlo ho inserito una istruzione di delay che lascia il sistema libero di gestire meglio le richieste di connessione, prima di avviare il server TCP. Durante questo intervallo il LED resta acceso: se allo spegnimento del LED non siete riusciti a fare la *join* premete il tasto di reset.

4 Creiamo un Access Point con un server HTTP II (sketch IoT_AP_ServerTCP)

```
 * Il loop gira a vuoto se non ci sono richieste in attesa. Quando un cliente
 * chiede la connessione il server riceve e visualizza (senza analizzarla)
 * la stringa inviata, e spedisce in risposta una semplice pagina HTML
 * utilizzando il protocollo HTTP
 */
void loop() {
  char line[80];
  WiFiClient client;  // Creazione oggetto cliente
  // Se non ci sono richieste chiude il loop
  if ( ! ( client = TCPServer.available() ) ) return;
  digitalWrite(LED,LOW);              // Accende il LED durante l'attivita'
  Serial.print("Cliente connesso: "); // Visualizza i dati del socket lato cliente
  Serial.print(client.remoteIP());
  Serial.print(":");
  Serial.println(client.remotePort());
  /*
   * Lettura e visualizzazione di tutti i caratteri inviati
   * dal cliente (una REQUEST HTTP)
   */
  while (client.available()) Serial.write(client.read());
  // Invio della RESPONSE
  client.print("HTTP/1.1 200 OK\r\n");          // startline
  client.print("\r\n");                         // riga vuota
  client.print("<html>\r\n");                   // TAG di apertura
  client.print("Hallo from you lazy AP!\r\n");  // Contenuto
  client.print("</html>\r\n");                  // TAG di chiusura
  client.stop();            // Chiusura della connessione
  digitalWrite(LED,HIGH);   // Spegnimento del LED
}
```

Listing 4: Creiamo un Access Point

Il loop esegue un polling continuo del socket lato server in attesa che un cliente chieda l'apertura di una connessione. Il socket è aperto sulla porta 80, normalmene associata ad HTTP. Quando accade l'istruzione *if* fallisce e viene eseguito il resto della funzione di loop: durante la sua esecuzione il LED è acceso. La funzione legge tutti i caratteri provenienti dal socket e li stampa: potrebbe trattarsi di una REQUEST HTTP. Quindi il server risponde con una semplice RESPONSE HTTP.

Alcuni esperimenti possibili:

- usando il PC, dopo il join all'AP sul WEMOS, inviare dei ping all'indirizzo indicato
  `ping 192.168.4.1`
- se avete installato `nc` inviare il comando
  `echo "Hallo!" | nc 192.168.4.1 80`
- con un browser (anche dallo smartphone) visitate la URL
  `http://192.168.4.1`

5 Colleghiamoci ad un Access Point (sketch `IoT_Join`)

```
// Al posto di XXXXXX inserire le credenziali per il vostro AP o tethering
#define MYSSID "XXXXXX"
#define MYPASSWD "XXXXXX"
```

Listing 5: il file secret.h

```
/*
 * La scheda WEMOS si associa ad un AP, quello di casa o quello realizzato dal
    vostro
 * smartphone come hotspot (tethering). Una volta connesso continua a lampeggiare.
 * E' possibile utilizzare il codice del server del programma precedente per
 * avere lo stesso server HTTP/TCP.
 */
#include "secret.h"
#include <ESP8266WiFi.h>
#define LED 2

/*
 * Riutilizzeremo questa nei prossimi esercizi: realizza il join
 * all'AP le cui credenziali sono registrate nel file secret.h.
 */
void joinAP() {
  WiFi.mode(WIFI_STA);                          // Configura come stazione
  Serial.print("Mi sto connettendo");
  if ( ! WiFi.begin(ESSID, PASSWORD) ) {      // Verifica l'esistenza dell'AP
    Serial.println("ESSID inesistente: addio!");
    while ( true ) {};
  }
  while (WiFi.status() != WL_CONNECTED)       // Attende il successo del JOIN
  {
    delay(500);
    Serial.print(".");
  }
  Serial.print("\nConnesso come ");           // Visualizza il proprio IP
  Serial.println(WiFi.localIP());
}

// Qui viene richiamata la funzione di join
void setup() {
  pinMode(LED, OUTPUT);
  Serial.begin(115200);
  joinAP();
}

/*
 * Nessuna funzionalita' interessante nel loop, ma e' possibile sostituirlo
 * con quello dell'esercizio precedente, creando prima l'oggetto Server TCP
 * ed inizializzandolo nella funzione "setup"
 */
void loop() {
  digitalWrite(LED,!(digitalRead(LED)));
  delay(500);
}
```

Listing 6: Colleghiamoci ad un Access Point

6 Usiamo la API di dweet.io I (sketch `IoT_dweet`)

```
// Al posto di XXXXXX inserire le credenziali per il vostro AP o tethering
#define ESSID "XXXXXX"                  // identificatore associato all'AP
#define PASSWORD "XXXXXX"               // password di accesso all'AP
#define DWEET "so12pnjt89s"     // meglio che sia unico
```

Listing 7: il file secret.h

```
/*
 * Dweet.io e' un servizio di cloud orientato ad applicazioni IoT. E'
 * possibile inviare dati (numerici o stringhe) che vengono memorizzati
 * e sono successivamente disponibili tramite il servizio stesso.
 * Per questa funzionalita' di usa una API REST offerta dal servizio. Il
 * nostro dispositivo realizza un client HTTP che comunica al nostro
 * dweet (si chiama cosi' una istanza del servizio) gli interi positivi. Il nome
    del
 * dweet va inserito nel file secret.h: scegliete un nome che sia
 * unico. Il dweet e' aperto a tutti, in lettura e scrittura!
 */
#include "secret.h"
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

#define LED 2         // Il LED sulla scheda e' collegato al pin 2

HTTPClient dweet;     // L'oggetto HTTPClient per comunicare con dweet.io
int n=0;              // L'intero che sara' incrementato

void joinAP() {       // Funzione di join gia' vista nell'esercizio precedente
  WiFi.mode(WIFI_STA);
  Serial.print("Mi sto connettendo");
  if ( ! WiFi.begin(MYSSID, MYPASSWD) ) {
    Serial.println("SSID inesistente: addio!");
    while ( true ) {};
  }
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Connesso come ");
  Serial.println(WiFi.localIP());
}

void setup() {        // Setup gia' visto in precedenza
  pinMode(LED, OUTPUT);
  Serial.begin(115200);
  joinAP();
}
```

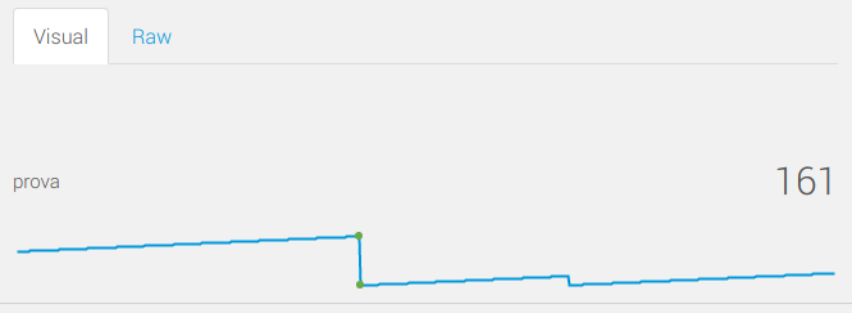
Listing 8: Usiamo la API di dweet.io

7 Usiamo la API di dweet.io II (sketch `IoT_dweet`)

```
/*
 * Nel loop viene inviata la REQUEST HTTP al server dweet.io. La parte
 * informativa della REQUEST, che e' di tipo GET, sta nella URL, nella
 * parte detta di query. Dopo aver inviato la GET si attende la risposta
 * e poi si ripete il loop, incrementanto il valore inviato.
 */
void loop() {
  char url[100];  // Buffer per la URL di dweet.io
  char body[100]; // Buffer body HTTP delle REQUEST
  /*
   *  Costruzione della URL
   *  L'ultimo elemento del path e' il nome del dweet di destinazione
   *  Dopo il ? segue la query che riporta l'identificatore di un campo
   *  del dweet e il valore assegnato. Nel nostro caso abbiamo un unico
   *  campo "prova" ed il suo valore e' l'intero crescente
   */
  sprintf(url,"http://dweet.io/dweet/for/%s?prova=%d",DWEET,n++);
  Serial.print("URL: ");Serial.println(url);  // Visualizza la URL utilizzata
  digitalWrite(LED_BUILTIN,LOW);              // LED acceso durante l'attivita'
  // Apre la connessione HTTP con il server REST di dweet.io
  if (dweet.begin(url)) {                     // Apre la connessione
    int x=dweet.GET();                        // Esegue la GET, in x l'esito
    if(x == 200)
      Serial.println("Channel update successful.");
    else
      Serial.println("Problem updating channel. HTTP error code " + String(x));
    dweet.end();                              // Chiude la connessione
  }
  digitalWrite(LED_BUILTIN,HIGH);             // Spegne il LED
  delay(2000);                                // Ritardo prima del prossimo feed
}
```

Listing 9: Usiamo la API di dweet.io

Per verificare il funzionamento, con un browser visitate la pagina del vostro dweet, all'indirizzo
`https://dweet.io/follow/xxxxxxxx`
sostituendo a xxxxxxxx il nome che gli avete assegnato.



Cautela: tenete presente che i dweet sono pubblici, aperti in lettura e scrittura: quindi nei vostri esperimenti tutelate anche la vostra privacy ;-)