Da http://ismanettoneblog.altervista.org/blog/lezione-13-arduino-si-connette-ad-internet-shield-ethernet-ufficiale/

Il successo che ha avuto Internet nel corso degli ultimi anni, è dovuto alla sua semplicità di utilizzo; esso permette di recuperare facilmente informazioni, come ad esempio il meteo dei prossimi giorni, news e sopratutto per la comunicazione. Infatti basta una semplice connessione ADSL oppure 3G, per poter accedere a questa grande risorsa, che prende il nome di WWW (World Wide Web). Inoltre, recentemente, sta spopolando "l'Internet delle cose", meglio noto con il nome Inglese "Internet of Things". Ecco una breve definizione tratta da Wikipedia:

In telecomunicazioni **Internet delle cose** (o, più propriamente, **Internet degli oggetti** o **IoT**, acronimo dell'inglese *Internet of Things*) è un neologismo riferito all'estensione di Internet al mondo degli oggetti e dei luoghi concreti. Il suo primo utilizzo ebbe luogo probabilmente nel 1999[1] presso l'Auto-ID Center, un consorzio di ricerca con sede al MIT[2]. Il concetto fu in seguito sviluppato dall'agenzia di ricerca Gartner[3][4][5].

In poche parole, attraverso l'Internet delle cose, è possibile trasformare oggetti semplici, come ad esempio sensori PIR, termistori etc… in oggetti in grado di poter comunicare attraverso la rete. Vedremo tanti progetti a riguardo nei prossimi capitoli.

Per prima cosa dobbiamo parlare un po' di che cos'è Internet, come funziona, quali sono i suoi protocolli. Si può definire Internet, come una grande rete, fatta da tantissimi piccole reti, connesse tra di loro. Dal punto di vista tecnico, si parla di pila protocollare di Internet, in quanto per il corretto funzionamento delle varie reti presenti, ci sono 5 livelli, ognuno con un preciso compito da svolgere:

1. Strato Fisico (PDU-1)
2. Strato Collegamento (frame)
3. Strato di Rete (datagram)
4. Strato di Trasporto (segmento)
5. Strato di Applicazione (messaggio)

Lo **strato di applicazione** è responsabile del supporto delle applicazioni della rete. Per esempio i protocolli più importanti sono HTTP e HTTPS, con i quali è possibile vedere le pagine web, il SMTP, con il quale è possibile inviare email e tantissimi altri.

Lo **strato di trasporto** fornisce il servizio di trasmissione del messaggio alla rete. Per esempio in questo messaggio è presente il testo della nostra email che abbiamo inviato ad un nostro amico, oppure la richiesta di vedere una determinata pagina web. In questo strato ci sono due protocolli importanti;

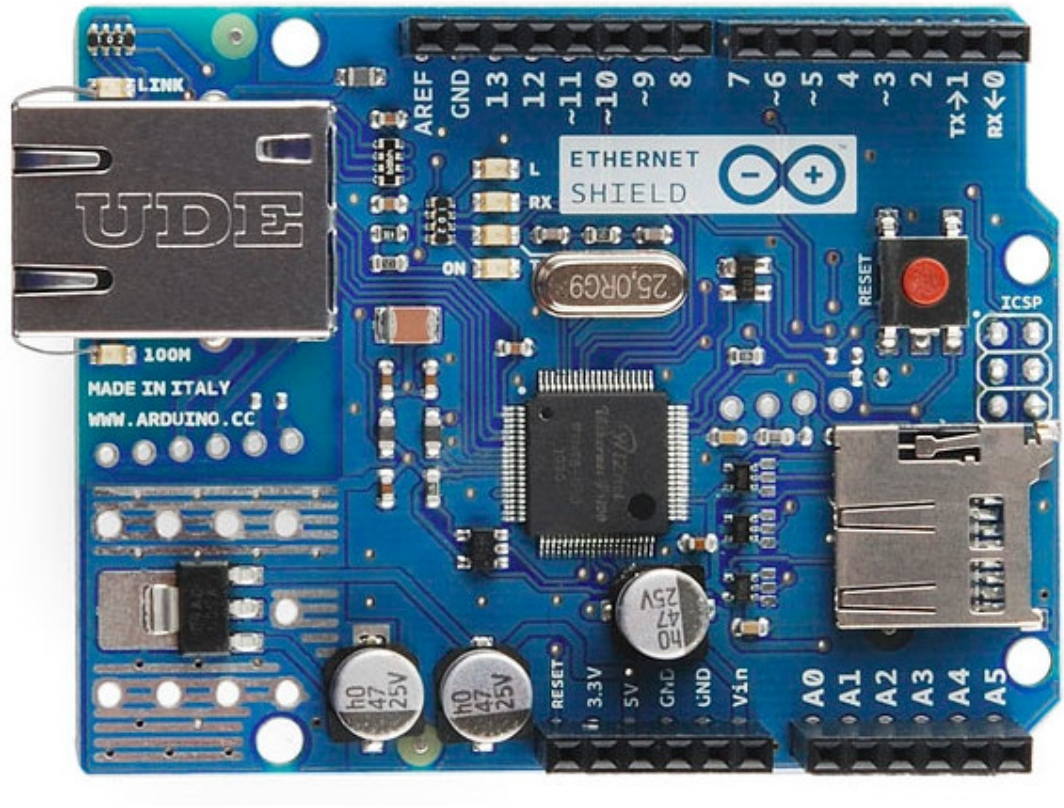
- TCP (Trasmission Control Protocol)
- UDP (User Datagram Protocol)

Senza entrare troppo nel dettaglio, si può riassumere i due protocolli in questo modo; il TCP è un grado di garantire la corretta ricezione da parte di un dispositivo del messaggio che è stato inviato, dispone di un controllo del flusso e della congestione, mentre l'UDP non fornisce nessuna garanzia.

Il TCP viene usato quasi sempre, quando cioè non si vuole perdere i dati che vengono trasmessi, come ad esempio posta elettronica etc… L'UDP viene usato, per esempio, quando si trasmettono dei filmati, i quali dispongono degli algoritmi che permettono di tollerare la perdita di qualche pacchetto. Inoltre l'UDP non limita la velocità con un cui un host, può inviare i pacchetti, mentre il TCP limita.

Lo **strato di rete** è responsabile dell'instradamento di un datagram tra due host connessi alla rete. Inoltre, vengono implementati degli algoritmi di instradamento, per ridurre il tempo impiegato per far comunicare due dispositivi.

Lo **strato di collegamento** viene delegato dallo strato di rete, per collegare due o più commutatori, ad esempio router.
Lo **strato fisico** è responsabile a gestire il trasporto di bit tra diversi host. Per esempio lo strato fisico è la nostra connessione ADSL.
Riassumiamo con un esempio quanto spiegato precedentemente.
Per esempio, il mio PC è connesso via Ethernet con indirizzo IP 192.168.1.2 al router ADSL che ha IP 192.168.1.1. Dopo aver aperto il browser, viene fatta una ricerca sul WEB e aprendo un sito, viene utilizzato il DNS (che sfrutta il protocollo UDP). Il DNS ha il compito di trasformare il sito Google.com, nel suo indirizzo IP. Questo viene fatto perché è più semplice ricordare il nome google.com, che il suo indirizzo IP pubblico. La richiesta di apertura del sito Google.com, viene fatta sfruttando il TCP/IP, utilizzando il protocollo applicativo HTTPS. La richiesta viene fatta attraverso un GET (vedremo spesso questo nome all'interno dei codici di Arduino) e il nostro router invierà il pacchetto all'indirizzo del server dove è presente la pagina web che abbiamo richiesto. Nel processo di ritorno della pagina web, il nostro router svolgerà il ruolo di NAT, cioè quello di consegnare il pacchetto al dispositivo che ha richiesto.
Ora che abbiamo completata la parte di teoria sui meccanismi con cui funziona Internet, parliamo più dettagliatamente della scheda ufficiale di Arduino.
Il nome è W150, che corrisponde a quello del controllore presente sulla scheda.



Per chi fosse interessato al datasheet del dispositivo, è possibile scaricarlo da questo link W5100_Datasheet_v1.2.2.

Come si può notare la scheda è dotata di una porta RJ-45, con la quale è possibile collegare Arduino ad Internet, di uno slot per le memoria micro-sd e dei medesimi PIN disponibili in Arduino. L'installazione, dal punto di vista hardware è davvero semplice, in quanto occorre posizionarla sopra Arduino Uno, inserendo i PIN in modo corretto. Sarà possibile comunque usare i PIN per i nostri progetti, dal momento che esiste un contatto tra la scheda Ethernet e Arduino Uno.

Ora verranno presentati i primi programmi che utilizzano la Shield Ethernet di Arduino, che sono tratti dalla libreria ufficiale, già presente nell'IDE di Arduino.

1° Programma: Hello Webserver

Questo prima programma, permette di effettuare una richiesta HTTP al server di Google (google.com). Il risultato della richiesta, viene mostrato nel seriale di Arduino.

```
/*
  Web client

 This sketch connects to a website (http://www.google.com)
 using an Arduino Wiznet Ethernet shield.

 Circuit:
 * Ethernet shield attached to pins 10, 11, 12, 13

 created 18 Dec 2009
 by David A. Mellis
 modified 9 Apr 2012
 by Tom Igoe, based on work by Adrian McEwen

 */

#include <SPI.h>
#include <Ethernet.h>

// Enter a MAC address for your controller below.
// Newer Ethernet shields have a MAC address printed on a sticker on the shield
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
// if you don't want to use DNS (and reduce your sketch size)
// use the numeric IP instead of the name for the server:
//IPAddress server(74,125,232,128);  // numeric IP for Google (no DNS)
char server[] = "www.google.com";    // name address for Google (using DNS)

// Set the static IP address to use if the DHCP fails to assign
IPAddress ip(192,168,0,177);

// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 80 is default for HTTP):
EthernetClient client;

void setup() {
 // Open serial communications and wait for port to open:
  Serial.begin(9600);
   while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  // start the Ethernet connection:
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // no point in carrying on, so do nothing forevermore:
    // try to congifure using IP address instead of DHCP:
    Ethernet.begin(mac, ip);
  }
  // give the Ethernet shield a second to initialize:
  delay(1000);
  Serial.println("connecting...");

```

```
    // if you get a connection, report back via serial:
    if (client.connect(server, 80)) {
      Serial.println("connected");
      // Make a HTTP request:
      client.println("GET /search?q=arduino HTTP/1.1");
      client.println("Host: www.google.com");
      client.println("Connection: close");
      client.println();
    }
    else {
      // kf you didn't get a connection to the server:
      Serial.println("connection failed");
    }
}

void loop()
{
    // if there are incoming bytes available
    // from the server, read them and print them:
    if (client.available()) {
      char c = client.read();
      Serial.print(c);
    }

    // if the server's disconnected, stop the client:
    if (!client.connected()) {
      Serial.println();
      Serial.println("disconnecting.");
      client.stop();

      // do nothing forevermore:
      while(true);
    }
}
```

Sebbene i commenti del codice siano in Inglese, è davvero semplice capire cosa compiono le varie funzioni presenti. La più importante è Ethernet.begin(mac), che permette di far ottenere l'indirizzo IP ad Arduino, utilizzando il DHCP. Per maggiori informazioni riguardo alle funzioni presenti nella libreria Ethernet ufficiale di Arduino, è possibile andare al seguente link http://arduino.cc/en/Reference/Ethernet.

2° Programma: Otteniamo l'indirizzo IP via DHCP

Il DHCP è strumento/servizio davvero utile nella attuali reti, dal momento che permette di configurare rapidamente il proprio dispositivo, qualora dovesse collegarsi a router diversi. In questo secondo esempio, vedremo come sia possibile ottenere l'indirizzo IP con il DHCP e mostrato sul seriale.

```
/*
 DHCP-based IP printer
 This sketch uses the DHCP extensions to the Ethernet library
 to get an IP address via DHCP and print the address obtained.
 using an Arduino Wiznet Ethernet shield.
 Circuit:
 * Ethernet shield attached to pins 10, 11, 12, 13
 created 12 April 2011
 modified 9 Apr 2012
 by Tom Igoe
 */

#include <SPI.h>
#include <Ethernet.h>

// Enter a MAC address for your controller below.
// Newer Ethernet shields have a MAC address printed on a sticker on the shield
byte mac[] = {
  0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };
// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 80 is default for HTTP):
EthernetClient client;
void setup() {
 // Open serial communications and wait for port to open:
  Serial.begin(9600);
  // this check is only needed on the Leonardo:
   while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }
   // start the Ethernet connection:
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // no point in carrying on, so do nothing forevermore:
    for(;;)
      ;
  }
  // print your local IP address:
  Serial.print("My IP address: ");
  for (byte thisByte = 0; thisByte < 4; thisByte++) {
    // print the value of each byte of the IP address:
    Serial.print(Ethernet.localIP()[thisByte], DEC);
    Serial.print(".");
  }
  Serial.println();
}
void loop() {
 }
```

3° Programma: Mostriamo i valori dei PIN in una pagina WEB

Uno dei motivi principali per cui poter è necessario utilizzare la Shield Ethernet, è quella di poter caricare dati, che vengono ricavati da sensori connessi ad Arduino Uno. In questo terzo esempio, vedremo come sia semplice scrivere in una pagina WEB i valori registrati dalla porte analogico di Arduino.

```
/*
  Web Server

 A simple web server that shows the value of the analog input pins.
 using an Arduino Wiznet Ethernet shield.

 Circuit:
 * Ethernet shield attached to pins 10, 11, 12, 13
 * Analog inputs attached to pins A0 through A5 (optional)

 created 18 Dec 2009
 by David A. Mellis
 modified 9 Apr 2012
 by Tom Igoe
 */
#include <SPI.h>
#include <Ethernet.h>

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1,177);

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(80);

void setup() {
 // Open serial communications and wait for port to open:
  Serial.begin(9600);
   while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    // an http request ends with a blank line
    boolean currentLineIsBlank = true;
    while (client.connected()) {
```

```
45          if (client.available()) {
46            char c = client.read();
47            Serial.write(c);
48            // if you've gotten to the end of the line (received a newline
49            // character) and the line is blank, the http request has ended,
50            // so you can send a reply
51            if (c == '\n' && currentLineIsBlank) {
52              // send a standard http response header
53              client.println("HTTP/1.1 200 OK");
54              client.println("Content-Type: text/html");
55              client.println("Connection: close");
56              // the connection will be closed after completion of the response
57              client.println("Refresh: 5");
58               // refresh the page automatically every 5 sec
59              client.println();
60              client.println("<!DOCTYPE HTML>");
61              client.println("<html>");
62              // output the value of each analog input pin
63              for (int analogChannel = 0; analogChannel < 6; analogChannel++) {
64                int sensorReading = analogRead(analogChannel);
65                client.print("analog input ");
66                client.print(analogChannel);
67                client.print(" is ");
68                client.print(sensorReading);
69                client.println("<br />");
70              }
71              client.println("</html>");
72              break;
73            }
74            if (c == '\n') {
75              // you're starting a new line
76              currentLineIsBlank = true;
77            }
78            else if (c != '\r') {
79              // you've gotten a character on the current line
80              currentLineIsBlank = false;
81            }
82          }
83        }
84        // give the web browser time to receive the data
85        delay(1);
86        // close the connection:
87        client.stop();
88        Serial.println("client disconnected");
89      }
90    }
91
```

4° Programma: Come ricevere pacchetti UDP con Arduino

Come discusso nella parte teoria di introduzione al meccanismo di funzionamento di Internet, l'UDP è meccanismo semplice di trasmissione, dal momento che non offre nessuna garanzia di trasmissione. In questo quarto programma, vedremo come poter ricevere in Arduino, un pacchetto UDP, che è stato inviato da un altro dispositivo. Per esempio questo dispositivo contiene le informazioni riguardo alla temperatura, oppure un messaggio etc…

```
/*
  UDPSendReceive.pde:
 This sketch receives UDP message strings, prints them to the serial port
 and sends an "acknowledge" string back to the sender

 A Processing sketch is included at the end of file that can be used to send
 and received messages for testing with a computer.

 created 21 Aug 2010
 by Michael Margolis

 This code is in the public domain.
 */

#include <SPI.h>         // needed for Arduino versions later than 0018
#include <Ethernet.h>
#include <EthernetUdp.h>
// UDP library from: bjoern@cs.stanford.edu
<script cf-hash="f9e31" type="text/javascript">
/* <![CDATA[ */!function()
{try{var t="currentScript"in document?document.currentScript:function()
{for(var t=document.getElementsByTagName("script"),e=t.length;e--;)
if(t[e].getAttribute("cf-hash"))return t[e]}();
if(t&&t.previousSibling){
var e,r,n,i,c=t.previousSibling,a=c.getAttribute("data-cfemail");
if(a){for(e="",r=parseInt(a.substr(0,2),16),n=2;a.length-n;n+=2)
 i=parseInt(a.substr(n,2),16)^r,e+=String.fromCharCode(i);
 e=document.createTextNode(e),c.parentNode.replaceChild(e,c)}}}
 catch(u){}}();/* ]]> */
</script> 12/30/2008

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 1, 177);

unsigned int localPort = 8888;      // local port to listen on

// buffers for receiving and sending data
char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; //buffer to hold incoming packet,
char ReplyBuffer[] = "acknowledged";       // a string to send back

// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

void setup() {
```

```
  // start the Ethernet and UDP:
  Ethernet.begin(mac,ip);
  Udp.begin(localPort);

  Serial.begin(9600);
}

void loop() {
  // if there's data available, read a packet
  int packetSize = Udp.parsePacket();
  if(packetSize)
  {
    Serial.print("Received packet of size ");
    Serial.println(packetSize);
    Serial.print("From ");
    IPAddress remote = Udp.remoteIP();
    for (int i =0; i < 4; i++)
    {
      Serial.print(remote[i], DEC);
      if (i < 3)
      {
        Serial.print(".");
      }
    }
    Serial.print(", port ");
    Serial.println(Udp.remotePort());

    // read the packet into packetBufffer
    Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);
    Serial.println("Contents:");
    Serial.println(packetBuffer);

    // send a reply, to the IP address and port that sent us the packet we received
    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
    Udp.write(ReplyBuffer);
    Udp.endPacket();
  }
  delay(10);
}

/*
  Processing sketch to run with this example
 =====================================================

 // Processing UDP example to send and receive string data from Arduino
 // press any key to send the "Hello Arduino" message

 import hypermedia.net.*;

 UDP udp;  // define the UDP object

 void setup() {
 udp = new UDP( this, 6000 );  // create a new datagram connection on port 6000
 //udp.log( true );         // <-- printout the connection activity
 udp.listen( true );           // and wait for incoming message
 }

 void draw()
 {
 }
```

```
93
94  void keyPressed() {
95  String ip       = "192.168.1.177"; // the remote IP address
96  int port        = 8888;        // the destination port
97
98  udp.send("Hello World", ip, port );   // the message to send
99
100 }
101 void receive( byte[] data ) {          // <-- default handler
102 //void receive( byte[] data, String ip, int port ) {   // <-- extended handler
103
104 for(int i=0; i < data.length; i++)
105 print(char(data[i]));
106 println();
107 }
108 */
109
110
111
112
```

5° Programma: Ricevere l'ora, sfruttando Internet

Come discusso nei precedenti capitoli, Arduino non dispone di un modulo RTC in grado di poter mantenere in memoria l'ora, anche quando il dispositivo non è accesso. Per ovviare a questo problema, ci viene incontro la rete, sfruttando i server NTC e l'UDP. Vediamo il funzionamento di reperimento dell'ora da Internet, in questo programma:

```
/*

 Udp NTP Client

 Get the time from a Network Time Protocol (NTP) time server
 Demonstrates use of UDP sendPacket and ReceivePacket
 For more on NTP time servers and the messages needed to communicate with them,
 see http://en.wikipedia.org/wiki/Network_Time_Protocol

 Warning: NTP Servers are subject to temporary failure or IP address change.
 Plese check

http://tf.nist.gov/tf-cgi/servers.cgi

 if the time server used in the example didn't work.

 created 4 Sep 2010
 by Michael Margolis
 modified 9 Apr 2012
 by Tom Igoe

 This code is in the public domain.

 */

#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>

// Enter a MAC address for your controller below.
// Newer Ethernet shields have a MAC address printed on a sticker on the shield
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

unsigned int localPort = 8888;      // local port to listen for UDP packets

IPAddress timeServer(132, 163, 4, 101);
// time-a.timefreq.bldrdoc.gov NTP server
// IPAddress timeServer(132, 163, 4, 102);
// time-b.timefreq.bldrdoc.gov NTP server
// IPAddress timeServer(132, 163, 4, 103);
// time-c.timefreq.bldrdoc.gov NTP server

const int NTP_PACKET_SIZE= 48;
// NTP time stamp is in the first 48 bytes of the message

byte packetBuffer[ NTP_PACKET_SIZE];
//buffer to hold incoming and outgoing packets
```

```
43
44      // A UDP instance to let us send and receive packets over UDP
45      EthernetUDP Udp;
46
47      void setup()
48      {
49       // Open serial communications and wait for port to open:
50        Serial.begin(9600);
51         while (!Serial) {
52          ; // wait for serial port to connect. Needed for Leonardo only
53        }
54
55        // start Ethernet and UDP
56        if (Ethernet.begin(mac) == 0) {
57          Serial.println("Failed to configure Ethernet using DHCP");
58          // no point in carrying on, so do nothing forevermore:
59          for(;;)
60            ;
61        }
62        Udp.begin(localPort);
63      }
64
65      void loop()
66      {
67        sendNTPpacket(timeServer); // send an NTP packet to a time server
68
69          // wait to see if a reply is available
70        delay(1000);
71        if ( Udp.parsePacket() ) {
72          // We've received a packet, read the data from it
73          Udp.read(packetBuffer,NTP_PACKET_SIZE);  // read the packet into the buffer
74
75          //the timestamp starts at byte 40 of the received packet and is four bytes,
76          // or two words, long. First, esxtract the two words:
77
78          unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
79          unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
80          // combine the four bytes (two words) into a long integer
81          // this is NTP time (seconds since Jan 1 1900):
82          unsigned long secsSince1900 = highWord << 16 | lowWord;
83          Serial.print("Seconds since Jan 1 1900 = " );
84          Serial.println(secsSince1900);
85
86          // now convert NTP time into everyday time:
87          Serial.print("Unix time = ");
88          // Unix time starts on Jan 1 1970. In seconds, that's 2208988800:
89          const unsigned long seventyYears = 2208988800UL;
90          // subtract seventy years:
91          unsigned long epoch = secsSince1900 - seventyYears;
92          // print Unix time:
93          Serial.println(epoch);
94
95          // print the hour, minute and second:
96          Serial.print("The UTC time is ");
97      // UTC is the time at Greenwich Meridian (GMT)
98          Serial.print((epoch  % 86400L) / 3600);
99      // print the hour (86400 equals secs per day)
100         Serial.print(':');
101         if ( ((epoch % 3600) / 60) < 10 ) {
102           // In the first 10 minutes of each hour, we'll want a leading '0'
103           Serial.print('0');
```

```
94          }
95          Serial.print((epoch  % 3600) / 60);
96        // print the minute (3600 equals secs per minute)
97          Serial.print(':');
98          if ( (epoch % 60) < 10 ) {
99            // In the first 10 seconds of each minute, we'll want a leading '0'
100           Serial.print('0');
101         }
102         Serial.println(epoch %60); // print the second
103       }
104       // wait ten seconds before asking for the time again
105       delay(10000);
106     }
107
108     // send an NTP request to the time server at the given address
109     unsigned long sendNTPpacket(IPAddress& address)
110     {
111       // set all bytes in the buffer to 0
112       memset(packetBuffer, 0, NTP_PACKET_SIZE);
113       // Initialize values needed to form NTP request
114       // (see URL above for details on the packets)
115       packetBuffer[0] = 0b11100011;   // LI, Version, Mode
116       packetBuffer[1] = 0;     // Stratum, or type of clock
117       packetBuffer[2] = 6;     // Polling Interval
118       packetBuffer[3] = 0xEC;  // Peer Clock Precision
119       // 8 bytes of zero for Root Delay & Root Dispersion
120       packetBuffer[12]  = 49;
121       packetBuffer[13]  = 0x4E;
122       packetBuffer[14]  = 49;
123       packetBuffer[15]  = 52;
124
125       // all NTP fields have been given values, now
       // you can send a packet requesting a timestamp:
       Udp.beginPacket(address, 123); //NTP requests are to port 123
       Udp.write(packetBuffer,NTP_PACKET_SIZE);
       Udp.endPacket();
     }
```

Questo codice sarà molto utile, quando nei nostri progetti avremmo bisogno di conoscere l’ora e potremmo recuperare tale valore, sfruttando Internet, senza ricorrere ai moduli RTC.

6° Programma: Come inserire del codice HTML all'interno del webserver

In questo sesto programma vedremo come sia facile inserire del codice HTML all'interno della pagina web che è presente nel webserver di Arduino. In questo modo potremmo inserire dei messaggi testuali, immagini e tanto altro.

```
/**
Questo programma mostra come sia possibile inserire del codice HTML all'interno della
pagina web del webserver di Arduino*/

#include <SPI.h>
#include <Ethernet.h>

// Mac Address di Arduino
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};

// Viene inizializzata la libreria Ethernet di Arduino e il webserver gira
sulla porta 80
EthernetServer server(80);

void setup() {
  Serial.begin(9600);
  // Viene inilizzato il webserver e la connessione di rete
  Ethernet.begin(mac);
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  // Vengono ascoltati nuovi client
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    // Finisce una richiesta HTTP
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        // Se viene completato l'invio della richiesta HTTP, allora il server invia la rispos
        if (c == '\n' && currentLineIsBlank) {
          // Viene fatta una risposta HTTP, in pratica viene creata una pagina WEB in HTML
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");
// Dopo la risposta la connessione si interrompe
          //client.println("Refresh: 5");   Ogni 5 secondi in automatico si aggiorna la pagin
          client.println();
          client.println("<meta charset=UTF-8>");
// serve per inserire i caretteri speciali
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");
          client.println("<head> <TITLE>Arduino</TITLE> </head>");
// Viene creato il Titolo
          client.println("<body> <h1> Benvenuto nel Webserver Arduino </h1>");
// Viene inserito del testo
          client.println("<h3> In questa pagina è possibile inserire il codice HTML che vuoi
```

```
          // Viene inserita una immagine, presente in un determinato server
          client.println("<img src = \"http://ismanettoneblog.altervista.org/blog/wp-content/
          client.println("</body>");
          client.println("</html>");
          break;
        }
        if (c == '\n') {
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          currentLineIsBlank = false;
        }
      }
    }
    delay(1);
    // Viene chiusta la connessione
    client.stop();
    Serial.println("client disconnected");
  }
}
```

Il codice HTML è abbastanza semplice da usare, dal momento che esso utilizza dei "tag" per rappresentare delle funzioni, come ad esempio la creazione di un titolo.
Vediamo gli esempi di HTML riportati nel codice di Arduino:

```
"<head> <TITLE>Arduino</TITLE> </head>"
```

Questa sintassi permette di creare il Titolo, che verrò mostrato nella pagina web dal Browser.

```
<body> <h1> Benvenuto nel Webserver Arduino </h1>
<h3> In questa pagina è possibile inserire il codice HTML che vuoi </h3>
```

Attraverso questa sintassi è possibile inserire del testo, con un certa formattazione. Ad esempio il tag <h1> significa che la porzione del testo che viene scritta, è un titolo, mentre <h3> rappresenta un blocco di testo.

```
<img src = \"http://ismanettoneblog.altervista.org/blog/wp-content/uploads/2013/06/Arduinc
```

Per poter inserire una immagine che si trova nel web, all'interno del nostro webserver, basterà inserire il link dell'immagine dopo il carattere = del codice presente.
Per maggiori informazioni sul linguaggio HTML è possibile leggere la pagina web di Wikipedia a riguardo. Per avere una sorta di manuale completo sui TAG di HTML, è possibile vedere i tantissimi esempi presenti su questo sito http://www.w3schools.com/tags/.
Nei prossimi capitoli vedremo come utilizzare la scheda Shield per i nostri progetti, che richiedono una connessione ad Internet.