The Network Time Protocol (NTP)

Pieter P

Network Time Protocol

There are many applications where you want to know the time. In a normal Arduino project, you would have to get a RTC module, set the right time, sacrifice some Arduino pins for communication ... And when the RTC battery runs out, you have to replace it.

On the ESP8266, all you need is an Internet connection: you can just ask a time server what time it is. To do this, the Network Time Protocol (**NTP**) is used.

In the previous examples (HTTP, WebSockets) we've only used TCP connections, but NTP is based on **UDP**. There are a couple of differences, but it's really easy to use, thanks to the great libraries that come with the ESP8266 Arduino Core.

The main difference between TCP and UDP is that TCP needs a connection to send messages: First a handshake is sent by the client, the server responds, and a connection is established, and the client can send its messages. After the client has received the response of the server, the connection is closed (except when using WebSockets). To send a new message, the client has to open a new connection to the server first. This introduces latency and overhead.

UDP doesn't use a connection, a client can just send a message to the server directly, and the server can just send a response message back to the client when it has finished processing. There is, however, no guarantee that the messages will arrive at their destination, and there's no way to know whether they arrived or not (without sending an acknowledgement, of course). This means that we can't halt the program to wait for a response, because the request or response packet could have been lost on the Internet, and the ESP8266 will enter an infinite loop.

Instead of waiting for a response, we just send multiple requests, with a fixed interval between two requests, and just regularly check if a response has been received.

Getting the time

Let's take a look at an example that uses UDP to request the time from a NTP server.

Libraries, constants and globals

```
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>
#include <WiFiUdp.h>

ESP8266WiFiMulti wifiMulti;      // Create an instance of the ESP8266WiFiMulti class, called 'wifiMulti'

WiFiUDP UDP;                     // Create an instance of the WiFiUDP class to send and receive

IPAddress timeServerIP;          // time.nist.gov NTP server address
const char* NTPServerName = "time.nist.gov";

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

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

To use UDP, we have to include the WiFiUdp library, and create a UDP object. We'll also need to allocate memory for a buffer to store the UDP packets. For NTP, we need a buffer of 48 bytes long.
To know where to send the UDP packets to, we need the hostname of the NTP server, this is time.nist.gov.

Setup

```
void setup() {
  Serial.begin(115200);          // Start the Serial communication to send messages to the computer
  delay(10);
  Serial.println("\r\n");

  startWiFi();                   // Try to connect to some given access points. Then wait for a connection

  startUDP();

  if(!WiFi.hostByName(NTPServerName, timeServerIP)) { // Get the IP address of the NTP server
    Serial.println("DNS lookup failed. Rebooting.");
    Serial.flush();
    ESP.reset();
  }
  Serial.print("Time server IP:\t");
  Serial.println(timeServerIP);

  Serial.println("\r\nSending NTP request ...");
  sendNTPpacket(timeServerIP);
}
```

In the setup, we just start our Serial and Wi-Fi, as usual, and we start UDP as well. We'll look at the implementation of this function later.

We need the IP address of the NTP server, so we perform a DNS lookup with the server's hostname. There's not much we can do without the IP address of the time server, so if the lookup fails, reboot the ESP.
If we do get an IP, send the first NTP request, and enter the loop.

Loop

```
unsigned long intervalNTP = 60000; // Request NTP time every minute
unsigned long prevNTP = 0;
unsigned long lastNTPResponse = millis();
uint32_t timeUNIX = 0;

unsigned long prevActualTime = 0;

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - prevNTP > intervalNTP) { // If a minute has passed since last NTP request
    prevNTP = currentMillis;
    Serial.println("\r\nSending NTP request ...");
    sendNTPpacket(timeServerIP);               // Send an NTP request
  }

  uint32_t time = getTime();                   // Check if an NTP response has arrived and get the (UNIX) time
  if (time) {                                  // If a new timestamp has been received
    timeUNIX = time;
    Serial.print("NTP response:\t");
    Serial.println(timeUNIX);
    lastNTPResponse = currentMillis;
  } else if ((currentMillis - lastNTPResponse) > 3600000) {
    Serial.println("More than 1 hour since last NTP response. Rebooting.");
    Serial.flush();
    ESP.reset();
  }

  uint32_t actualTime = timeUNIX + (currentMillis - lastNTPResponse)/1000;
  if (actualTime != prevActualTime && timeUNIX != 0) { // If a second has passed since last print
    prevActualTime = actualTime;
    Serial.printf("\rUTC time:\t%d:%d:%d   ", getHours(actualTime), getMinutes(actualTime), getSeconds(actualTime));
  }
}
```

The first part of the loop sends a new NTP request to the time server every minute. This is based on Blink Without Delay.
Then we call the getTime function to check if we've got a new response from the server. If this is the case, we update the timeUNIX variable with the new timestamp from the server.
If we don't get any responses for an hour, then there's something wrong, so we reboot the ESP.
The last part prints the actual time. The actual time is just the last NTP time plus the time since we received that NTP message.

Setup functions

Nothing special here, just a function to connect to Wi-Fi, and a new function to start listening for UDP messages on port 123.

```
void startWiFi() { // Try to connect to some given access points. Then wait for a connection
  wifiMulti.addAP("ssid_from_AP_1", "your_password_for_AP_1");   // add Wi-Fi networks you want to connect to
  wifiMulti.addAP("ssid_from_AP_2", "your_password_for_AP_2");
  wifiMulti.addAP("ssid_from_AP_3", "your_password_for_AP_3");

  Serial.println("Connecting");
  while (wifiMulti.run() != WL_CONNECTED) {  // Wait for the Wi-Fi to connect
    delay(250);
    Serial.print('.');
  }
  Serial.println("\r\n");
  Serial.print("Connected to ");
  Serial.println(WiFi.SSID());             // Tell us what network we're connected to
  Serial.print("IP address:\t");
  Serial.print(WiFi.localIP());            // Send the IP address of the ESP8266 to the computer
  Serial.println("\r\n");
}

void startUDP() {
  Serial.println("Starting UDP");
  UDP.begin(123);                          // Start listening for UDP messages on port 123
  Serial.print("Local port:\t");
  Serial.println(UDP.localPort());
  Serial.println();
}
```

Helper functions

```
uint32_t getTime() {
  if (UDP.parsePacket() == 0) { // If there's no response (yet)
    return 0;
  }
  UDP.read(NTPBuffer, NTP_PACKET_SIZE); // read the packet into the buffer
  // Combine the 4 timestamp bytes into one 32-bit number
  uint32_t NTPTime = (NTPBuffer[40] << 24) | (NTPBuffer[41] << 16) | (NTPBuffer[42] << 8) | NTPBuffer[43];
  // Convert NTP time to a UNIX timestamp:
  // Unix time starts on Jan 1 1970. That's 2208988800 seconds in NTP time:
  const uint32_t seventyYears = 2208988800UL;
  // subtract seventy years:
  uint32_t UNIXTime = NTPTime - seventyYears;
```

```
  return UNIXTime;
}

void sendNTPpacket(IPAddress& address) {
  memset(NTPBuffer, 0, NTP_PACKET_SIZE);  // set all bytes in the buffer to 0
  // Initialize values needed to form NTP request
  NTPBuffer[0] = 0b11100011;   // LI, Version, Mode
  // send a packet requesting a timestamp:
  UDP.beginPacket(address, 123); // NTP requests are to port 123
  UDP.write(NTPBuffer, NTP_PACKET_SIZE);
  UDP.endPacket();
}

inline int getSeconds(uint32_t UNIXTime) {
  return UNIXTime % 60;
}

inline int getMinutes(uint32_t UNIXTime) {
  return UNIXTime / 60 % 60;
}

inline int getHours(uint32_t UNIXTime) {
  return UNIXTime / 3600 % 24;
}
```

In the getTime function, we first try to parse the UDP packet. If there's no packet available, the function just returns 0. If there is a UDP packet available however, read it into the buffer. The NTP timestamp is 32 bits or 4 bytes wide, so we combine these bytes into one long number. This number is the number of seconds since Jan 1, 1900, 00:00:00, but most applications use UNIX time, the number of seconds since Jan 1, 1970, 00:00:00 (UNIX epoch). To convert from NTP time to UNIX time, we just subtract 70 years worth of seconds.

To request the time from the NTP server, you have to send a certain sequence of 48 bytes. We don't need any fancy features, so just set the first byte to request the time, and leave all other 47 bytes zero.
To actually send the packet, you have to start the packet, specifying the IP address of the server, and the NTP port number, port 123. Then just write the buffer to the packet, and send it with endPacket.

The last three functions are just some simple math to convert seconds to hours, minutes and seconds.

Using the example

Enter your Wi-Fi credentials on lines 79-81, and hit upload. If you have a working Internet connection, you should get an output that looks like this:

```
Connecting
.........

Connected to Wi-Fi SSID
IP address:     192.168.1.2

Starting UDP
Local port:     123

Time server IP: 216.229.0.179

Sending NTP request ...
NTP response:   1488378061
UTC time:       14:21:53
Sending NTP request ...
NTP response:   1488378114
UTC time:       14:22:53
Sending NTP request ...
NTP response:   1488378174
UTC time:       14:23:53
Sending NTP request ...
NTP response:   1488378234
UTC time:       14:24:53
Sending NTP request ...
NTP response:   1488378294
UTC time:       14:25:53
...
```

You should see the time update every second, and `Sending NTP request ...` should show up every minute.
If you don't have an Internet connection, the DNS lookup of the time server will fail:

```
Connecting
.........

Connected to Wi-Fi SSID
IP address:     192.168.1.2

Starting UDP
Local port:     123

DNS lookup failed. Rebooting.

 ets Jan  8 2013,rst cause:2, boot mode:(3,6)
```

If your connection is not reliable, or if there's heavy traffic, you might encounter some dropped packets:

```
Sending NTP request ...
NTP response:   1488378780
UTC time:       14:33:54
```

```
Sending NTP request ...
UTC time:       14:34:54
Sending NTP request ...
NTP response:   1488378895
UTC time:       14:35:0
```

As you can see, the ESP never received a response to the second NTP request. That's not really an issue, as long as at least some packets make it through.

Local time and daylight savings

An NTP server returns the UTC time. If you want local time, you have to compensate for your time zone and daylight savings. For example, if you want CET (Central European Time), you have to add 3600 to the UNIX time during winter, (3600 s = 1 h), and 7200 during summer (DST).