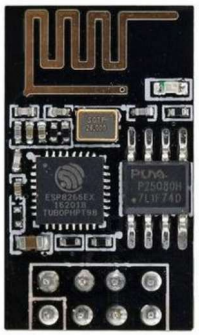
Módulo WiFi (ESP-8266)



Não podemos falar de Internet das Coisas com o Arduino sem efetivamente vermos como fazê-lo acessar a internet, o que possibilita desde a comunicação com outros dispositivos, leitura e gravação em bancos de dados online, uso de webservices, entre outros. Existem diversas formas possíveis para isso, desde o uso de um smartphone como ponte através de uma comunicação bluetooth, até o acesso direto através do uso de módulos que permitam a conexão cabeada (módulo ethernet) ou sem fio (módulo WiFi). O mais popular destes módulos WiFi é o ESP-8266, pelo seu baixo custo, facilidade de uso e alcance.

Principais métodos para trabalhar com o módulo ESP-8266

Para facilitar o uso do módulo, existe uma biblioteca própria que permite a criação de um objeto que representa uma conexão WiFi e através do qual podemos chamar métodos que permitem a autenticação, o envio e recebimento de dados e até mesmo a criação de um servidor HTTP que retorna um HTML como resposta a uma solicitação recebida em determinado endereço, permitindo, assim, a criação de uma página Web para visualizar informações e até mesmo enviar dados ou ordens ao Arduino. Esta biblioteca é a "WiFiEsp.h" e pode ser baixada no link: https://bit.ly/arduinoqi

A instanciação de um objeto desta classe que represente um servidor web (WiFiEspServer) é realizada como segue abaixo, informando a porta HTTP utilizada para estabelecer a conexão (sendo a 80 a porta padrão, modificada apenas em redes especiais e com redirecionamento para outras portas):

```
WiFiEspServer <nome_objeto>( <porta_http> );
```

Um WiFiEspServer necessita de uma conexão serial para estabelecer a comunicação e poder receber e enviar dados. Assim, precisamos da importação da biblioteca "SoftwareSerial.h" e da criação de um objeto deste tipo, informando em qual portas do Arduino estão conectados, respectivamente, os pinos RX e TX do módulo (ou de seu adaptador, como veremos adiante).

```
SoftwareSerial <nome_objeto>( <porta_RX>, <porta_TX> );
```

A biblioteca nos oferece outro objeto muito importante (já criado) chamado WiFi. Ele é responsável por diversas verificações e configuração de nossa conexão. O mesmo deve ser inicializado através de um método init, porém recebendo como parâmetro o endereço de alocação (&) de um objeto do tipo SoftwareSerial. Assim, sua inicialização seria como a demonstrada abaixo:

```
WiFi.init( &<nome_objeto_SoftwareSerial> );
```

Após a inicialização, o objeto WiFi deve receber um endereço IP para que consiga ter acesso à autenticação na rede de internet do usuário, respeitando a faixa de IPs desta rede. Para isso, utilizamos o método IPAddress que recebe por parâmetro os 4 números (4 octetos) que compõe o endereço.

```
<nome_objeto_WiFiEspServer>.config( IPAddress(<n1>,<n2>,<n3>,<n4>) );
```

Com endereço configurado, podemos tentar conectar em uma rede WiFi. Para auxiliar na manipulação (byte a byte), vamos criar duas variáveis auxiliares (arrays de caracteres) às quais atribuiremos o nome da rede (SSID) e a sua senha. Após, as utilizamos como parâmetro do método begin do objeto WiFi que estabelece a conexão.

```
char <variavel_nome_rede>[] = "nome_da_sua_rede";
char <variavel_senha_rede>[] = "senha_da_sua_rede";
WiFi.begin( <variavel_nome_rede>, <variavel_senha_rede> );
```

A chamada deste método begin do objeto WiFi sempre retorna um código de status que pode ser testado para verificar a situação da conexão. São eles:

- WL_NO_SHIELD: não detectado um shield WiFi (o ESP8266 ou outro qualquer)
- WL_IDLE_STATUS: shield detectado mas em estado IDLE (ainda não conectado)
- WL_CONNECTED: conexão estabelecida com sucesso
- WL_NOT_CONNECTED: conexão não estabelecida

Este mesmo status também pode ser testado mesmo antes ou após a chamada do método begin (ou seja, fora do seu retorno), através do método status do WiFi.

```
WiFi.status()
```

Com a conexão estabelecida e o Web Server ativo, podemos criar um WiFiEspClient, que será o responsável por responder às solicitações dos clientes através da porta 80 (requisições HTTP).

```
WiFiEspClient <nome_objeto_WiFiEspCliente> =
  <nome_objeto_WiFiEspServer>.available();
```

Após criado, podemos testar o objeto WiFiEspClient dentro de um if pois sempre que um novo cliente tenta conectar ele nos devolve um true.

```
if( <nome_objeto_WiFiEspClient> ) ...
```

Também podemos verificar se a conexão do cliente já aconteceu e está ativa, momento a partir do qual podemos receber ou enviar informações para ele. Para isso, utilizamos o método connected.

```
<nome_objeto_WiFiEspClient>.connected()
```

Além disso, outra verificação importante é se estamos recebendo alguma coisa (requisição HTTP) do nosso cliente.

```
<nome_objeto_WiFiEspClient>.available()
```

Para o recebimento dessas informações através da serial da conexão estabelecida, podemos criar um objeto do tipo RingBuffer (como uma variável global) para receber e armazenar os bytes recebidos. Como parâmetro, passamos o tamanho desejado para nosso buffer.

```
RingBuffer <nome_objeto_RingBuffer>(<tamanho_buffer>);
```

Este objeto necessita de uma inicialização simples através de um método init.

```
<nome_objeto_RingBuffer>.init;
```

Caso se perceba o recebimento de uma requisição do cliente, podemos pegar byte a byte dela através do método read do cliente e armazenar no buffer com o seu método push, para depois testar o que foi recebido.

```
<nome_objeto_RingBuffer>.push(<nome_objeto_WiFiEspClient>.read())
```

Geralmente essa leitura byte a byte é realizada dentro de um laço que é executado enquanto existir requisição do cliente (available). Sabemos que o recebimento encerrou quando a requisição HTTP trouxer a sequência "\r\n\r\n" (duas quebras de linha). Assim, conseguimos testar com um condicional se esta sequência está no final do texto armazenado dentro do buffer, com o método endsWith.

```
if( <nome_objeto_RingBuffer>.endsWith("\r\n\r\n") ) ...
```

Quando uma requisição HTTP é recebida, nosso objeto WiFiEspClient pode enviar de volta ao cliente uma resposta, que pode ser desde uma simples String ou até mesmo uma página web completa que será exibida através de um browser. Esse envio é possível através do método println.

```
<nome_objeto_WiFiEspClient>.println("texto_a_enviar");
```

Após enviar a resposta, devemos encerrar a conexão do cliente para ficar esperando uma nova requisição. Para isso, utilizamos o método stop.

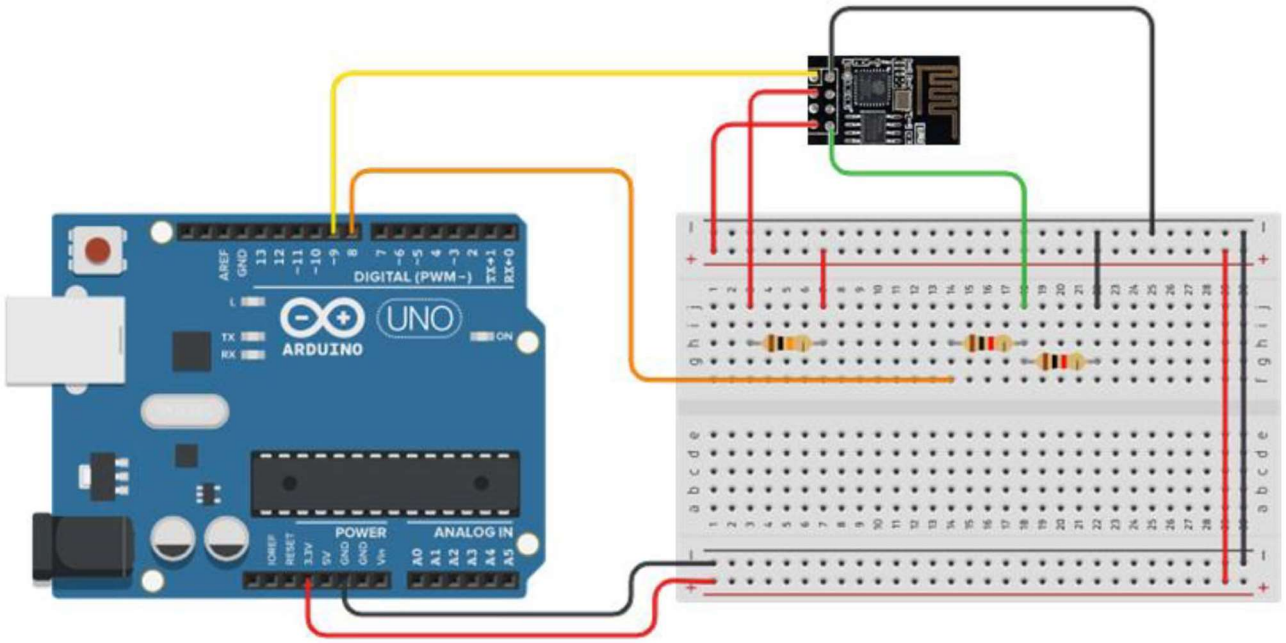
```
<nome_objeto_WiFiEspClient>.stop();
```

Ligação eletrônica

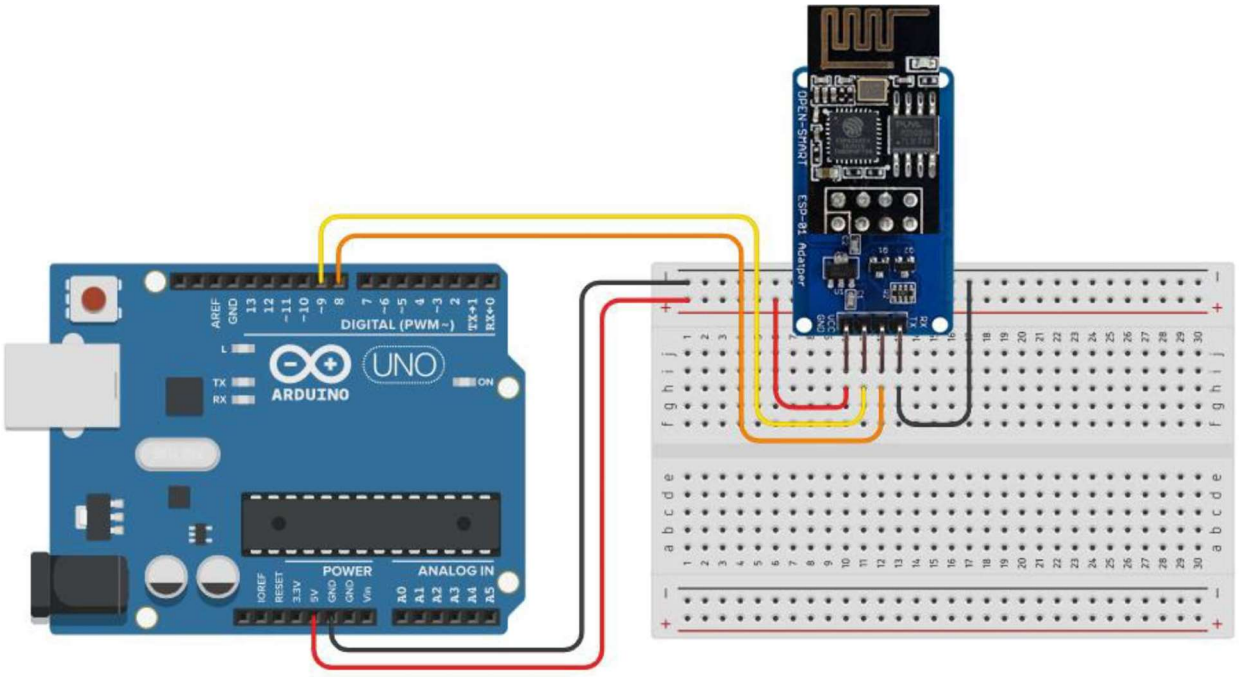
O módulo ESP-8266 (geralmente alimentado com 3.3V) pode ser utilizado de forma direta, ligando os seus pinos nas portas do Arduino (sendo que 5 dos 8 pinos necessitam de ligações e 2 deles obrigam a criação de um divisor de tensão, tornando o projeto eletrônico um pouco mais complexo). Porém, existem no mercado diversos adaptadores para o módulo (alimentados com 5V), que recebem seus 8 pinos (já com os divisores de tensão necessários internalizados no adaptador) e que permitem a comunicação com o Arduino de forma serial, necessitando assim da ligação de apenas um pino VCC, um GND,

um TX e um RX. Os dois tipos de ligação (com ou sem adaptador) funcionam de forma eficiente, sendo escolhidas de acordo com a necessidade. A seguir, os dois esquemas são apresentados.

Esquema eletrônico com acesso direto (resistores do divisor de tensão de 1KΩ e resistor da porta positiva de 10KΩ):

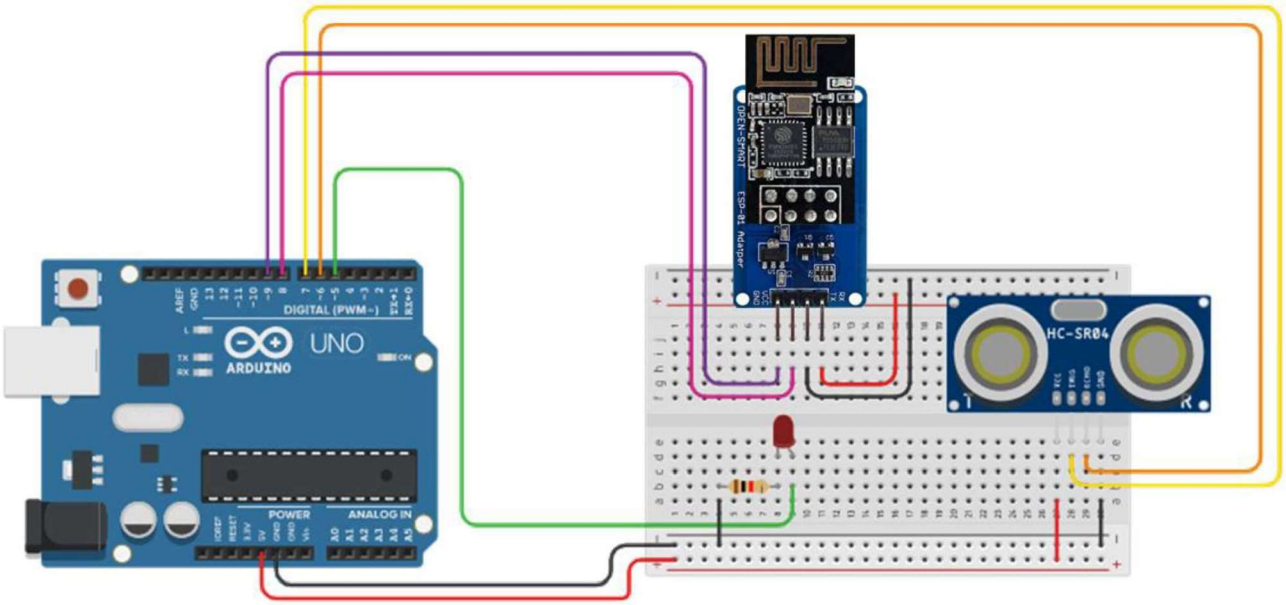


Esquema eletrônico com uso de adaptador:



Exemplo prático: comunicação e controle com webpage através do WiFi

Para um exemplo prático, vamos adaptar o projeto anterior para que com o uso de requisição HTTP o usuário possa acessar uma página web e através dela ver a distância medida pelo ultrassônico do Arduino, além de controlar um led através de um botão. O esquemático eletrônico será o mesmo, respeitando as mesmas portas, apenas removendo o módulo bluetooth e colocando em seu lugar um módulo ESP-8266 integrado a um adaptador. Deste modo, teremos:



Vamos então programar o nosso Arduino. Para um melhor entendimento, vamos quebrar o código em partes para facilitar a explicação.

Código parte 1) Includes, variáveis e objetos iniciais

```
#include <Ultrasonic.h>
#include <SoftwareSerial.h>
#include <WiFiEsp.h>

//Variáveis auxiliares
char nomeRede[] = "Sentido da Vida Network";
char senhaRede[] = "sentido=42";
int statusLed = LOW;
int distancia = 0;

//Objetos das bibliotecas
Ultrasonic ultrassom(7,6);
SoftwareSerial serialWiFi(8,9);
WiFiEspServer servidor(80);
RingBuffer bufferAux(8);
```

Inicialmente realizamos o include das 3 bibliotecas necessárias ao projeto: "Ultrasonic.h" (para o uso do sensor ultrassônico), "SoftwareSerial.h" (para a criação de uma comunicação serial utilizada pelo WiFi para comunicação) e WiFiEsp (para criação dos objetos que implementam um servidor e um cliente web). Após, criamos algumas variáveis auxiliares que conterão o nome e senha da rede e o status inicial do dispositivo WiFi (IDLE até que ele seja conectado posteriormente). Também é necessária uma variável para representar o estado do led (iniciando com LOW) e uma variável para guardar a distância lida pelo ultrassônico (iniciando com 0). Além das variáveis, criamos também alguns objetos: ultrassom (informando as portas onde estão ligados os pinos trigger e echo do sensor ultrassônico), serialWiFi (informando as portas TX e RX do módulo ESP8266 e que será utilizado pelo WiFi para comunicação), servidor (informando a porta 80 para implementar o objeto WiFiEspServer e aceitar requisições) e bufferAux (informando 8 bytes de tamanho para implementar o objeto BufferRing que armazenará as informações da requisição recebida pelo WiFi).

Código parte 2) Setup e inicializações iniciais

```
void setup(){
  pinMode(5, OUTPUT);
  digitalWrite(5, LOW);
  Serial.begin(9600);
  serialWiFi.begin(9600);
  WiFi.init(&serialWiFi);
  WiFi.config(IPAddress(192,168,0,100));
}
```

No setup, definimos a porta 5 (do led) como saída e iniciamos com ela desligada. O objeto Serial assim como o serialWiFi (também um serial), precisam ser iniciados com o método begin e a sua velocidade (aqui, 9600, que é o valor padrão). Já o objeto WiFi, ao ser iniciado com o init, recebe por parâmetro o endereço de uma conexão serial (no nosso caso, o endereço de serialWiFi). Após, o seu endereço IP é configurado para 192.168.0.100.

Código parte 3) Testando a conexão

```
void loop() {

  if(WiFi.status() == WL_NO_SHIELD){
    return;
  }
  if(status != WL_CONNECTED){
    status = WiFi.begin(nomeRede, senha);
    return;
  } else {
    servidor.begin();
  }
```

Ao executar o loop, testamos se o status da conexão é WL_NO_SHIELD pois se for, o Arduino não encontrou nenhum módulo WiFi ou o mesmo parou de funcionar. Caso isso aconteça, abandonamos o laço e tentamos de novo até que um módulo seja detectado. Após, testamos se o status da conexão é WL_CONNECTED que só é possível após a conexão e autenticação serem realizadas com sucesso. Enquanto isso não acontecer, tentamos novamente conectar utilizando o nome (SSID) e a senha da rede (armazenadas previamente nas variáveis nomeRede e senhaRede). Quando a conexão finalmente ocorrer, inicializamos o servidor (objeto do tipo WiFiEspServer) e seguimos adiante no loop.

Código parte 4) Implementando o cliente para receber requisições HTTP

```
    statusLed = ultrassom.Ranging(CM);

    WiFiEspClient cliente = servidor.available();

    if(cliente) {
      bufferAux.init();
      while(cliente.connected()){
        if(cliente.available()){
          bufferAux.push(cliente.read());

          if(bufferAux.endsWith("\r\n\r\n")) {
            respostaRequisicao(cliente);
            break;
          }
          if(bufferAux.endsWith("GET /L")){
            digitalWrite(5, HIGH);
            statusLed = 1;
          }
          else{
            if (bufferAux.endsWith("GET /D")) {
              digitalWrite(5, LOW);
              statusLed = 0;
            }
          }
        }
      }
      cliente.stop();
    }
  } //fim do loop
```

Primeiramente lemos o valor da distância do sensor ultrassônico e armazenamos na variável *distancia*. Em seguida, criamos um cliente web (objeto WiFiEspCliente) através de uma instância do servidor inicializado anteriormente. Colocando *cliente* em um if, ele nos retorna true quando perceber que um novo cliente está tentando estabelecer uma conexão. Então, iniciamos o buffer (com o método init). Enquanto o cliente estiver conectado, entramos em um laço testando se existe algum byte da requisição HTTP a receber (available). Se sim, lemos (read) byte a byte a cada repetição do laço e inserimos dentro do

buffer (push). Assim, cada byte depositado no buffer faz com que tenhamos um texto parcial dentro dele. Com isso, podemos testar qual o texto atual existente no final deste buffer com o método endsWith. Se terminar com "\r\n\r\n" é porque a requisição HTTP chegou ao fim e então enviamos uma resposta ao cliente (através de um objeto WiFiEspClient), que no nosso caso é uma página web montada com o método respostaRequisicao (explicado na próxima etapa). Se terminar com"GET /L" é porque recebemos um /L (de Ligado) através da requisição, que corresponde à uma ordem para ligar o Led. Se terminar com "GET /D" é porque recebemos um /D (de Desligado) através da requisição, que corresponde à uma ordem para desligar o Led. Ao final, interrompemos e fechamos a conexão para ficar à espera de uma nova requisição.

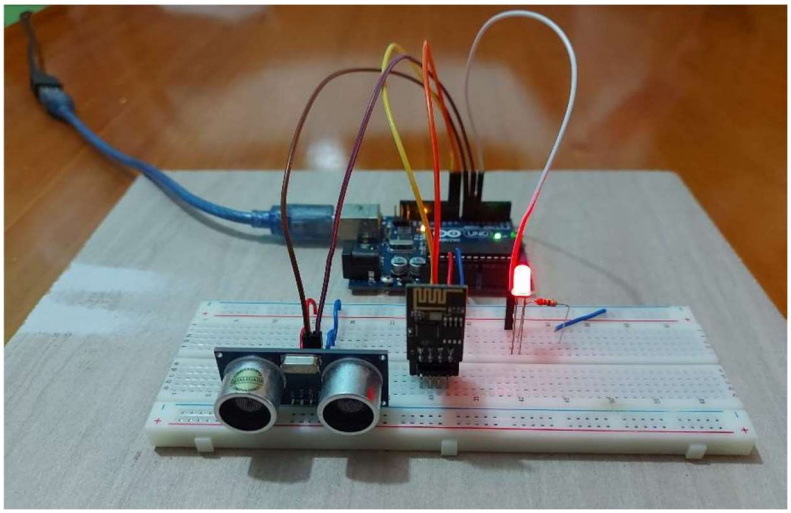
Código parte 5) Retornando uma página para a requisição

```
void respostaRequisicao(WiFiEspClient cliente){
  cliente.println("HTTP/1.1 200 OK");
  cliente.println("Content-Type: text/html");
  cliente.println("<!DOCTYPE HTML>");
  cliente.println("<html>");
  cliente.println("<head>");
  cliente.println("<title>Controle Web Arduino</title>");
  cliente.println("</head>");

  cliente.println("<body>");
  cliente.println("<div style='text-align: center'>");
  cliente.println("<p><font style='font-size:1.5em;'>
                   Informações da Placa Arduino</font></p>");

  cliente.println("<br><br>DISTÂNCIA");
  cliente.println("<p style='line-height:2;'><font color='green'>" +
                  (String)distancia + " cm</font></p>"

  cliente.println("<br><br>ESTADO ATUAL DO LED");

  if(statusLed == HIGH){
    cliente.println("<p style='line-height:2;'><font color='blue'>LIGADO</font></p>");
    cliente.println("<a href=\"/D\" style='color:white;background-color:red;
                     text-decoration:none;padding:10px;'>desligar</a>");
  } else {
    cliente.println("<p style='line-height:2;'><font color='red'>DESLIGADO</font></p>");
    cliente.println("<a href=\"/L\" style='color:white;background-color:blue;
                    text-decoration:none;padding:10px;'> ligar </a>");
  }

  cliente.println("</body>");
  cliente.println("</html>");
  delay(10);
}
```

O método respostaRequisicao, utiliza o objeto WiFiEspClient recebido por parâmetro para, através dele, enviar como resposta linhas que componham uma página HTML que, quando recebida pelo cliente, será mostrada pelo browser. Esta página é bem simples e possui alguns elementos centralizados dentro de uma div. Primeiramente, temos um título

("Informações da Placa Arduino") com letra maior do que as demais. Para exibir a distância lida do ultrassônico, temos um subtítulo ("DISTÂNCIA") seguido de um parágrafo com o valor da variável distância concatenado. Logo em seguida, posicionamos um subtítulo ("ESTADO ATUAL DO LED "). Abaixo deste subtítulo queremos montar dois elementos importantes: um texto que informe o estado atual do led (com letra azul quando ligado e vermelha quando desligado) e um botão para ligar ou desligar o led (com letra branca e fundo azul quando indicar a ação ligar ou vermelho quando indicar a ação desligar). Nós sabemos quando efetuar uma ou outra montagem através da análise da variável estadoLed. É importante perceber que nosso botão nada mais é do que uma âncora que quando montada tem seu href (link) configurado de modo a tentar abrir um endereço /D ou /L, o que acaba gerando uma requisição HTTP que contém, respectivamente "GET /D" ou "GET /L". É assim que conseguimos enviar o indicativo para ligar ou desligar o led via requisição HTTP ao clicar, sendo esta recebida pelo cliente e interpretada. Assim, quando o cliente acessar o endereço IP configurado no objeto WiFi (no nosso exemplo, 192.198.0.100), através do browser, a página será enviada como resposta, como abaixo:

Abaixo, uma foto do projeto montado com o led ligado após o Arduino receber uma requisição "GET /L" com o clique na âncora da página (botão "ligar"):



Adaptando o exemplo para redes cabeadas com a biblioteca Ethernet.h

Outra forma muito interessante de colocar nosso Arduino na internet é o uso de módulos ethernet, que permitam sua ligação à uma rede cabeada (cabos de par trançado). Apesar de ser um meio um pouco menos prático e que demanda a existência (ou a criação) de uma infraestrutura maior, pode ser uma alternativa interessante em algumas situações.

Diferentemente da tecnologia WiFi que possui alguns poucos módulos muito bem consolidados e disseminados, para redes ethernet existem uma profusão de soluções distintas, que variam de Shields (placas secundárias que são encaixadas sobre a placa Arduino estendendo suas funções, como na imagem à esquerda que mostra uma W5100) a módulos simples (como na imagem à direita que mostra uma ENC28J60). A grande quantidade, variedade e as múltiplas peculiaridades de dispositivos que agreguem ao Arduino a funcionalidade ethernet, faz com que tenhamos uma multiplicidade de opções, mas quanto à ligação eletrônica a grande maioria desses módulos utilizam comunicação SPI. Geralmente isso faz com que os pinos utilizados sejam os mesmos, variando quais são necessários, o que já pode provocar pequenas mudanças no esquemático eletrônico. Porém, todos os módulos (mesmo aqueles que já possuem uma biblioteca própria) são compatíveis no uso e na implementação com a biblioteca "Ethernet.h". Portanto, independente do módulo escolhido, do esquemático eletrônico e das peculiaridades presentes, conhecendo esta biblioteca conseguimos programar e utilizar qualquer um destes módulos ethernet em nossos projetos. Para encerrar este capítulo dedicado a módulos especiais e comunicação, vamos adaptar apenas o código do nosso exemplo anterior para ver as diferenças do uso de WiFi (com a biblioteca "WiFiEsp.h") e de redes ethernet (com a biblioteca "Ethernet.h") com o Arduino. O esquemático eletrônico não será apresentado pois deve ser seguido conforme o módulo escolhido. Nos limitaremos aqui ao código e à lógica de programação.

Inicialmente, vamos ver os principais métodos da biblioteca "Ethernet.h" e estabelecer uma comparação com a biblioteca "WiFiEsp.h", pois facilmente perceberemos que muitos objetos e métodos são equivalentes. Ambas as bibliotecas necessitam de um servidor web e de um cliente web, sendo o cliente uma instância identificada de um servidor após autenticado. Assim, podemos verificar que os métodos para a criação deste

cliente e deste servidor na "Ethernet.h" são análogos aos da biblioteca WiFiEsp vista anteriormente. O servidor (EthernetServer) também necessita na criação, como parâmetro, da porta utilizada para receber as requisições HTTP (por padrão a 80).

```
EthernetServer <nome_objeto>( <porta_http> );
```

O cliente (EthernetCliente) também recebe uma instância detectada do servidor após a autenticação do mesmo.

```
EthernetClient <nome_objeto_EthernetClient> =
  <nome_objeto_EthernetServer>.available();
```

Uma das diferenças em relação à biblioteca WiFiEsp são as inicializações dos objetos. O servidor é iniciado com um simples método begin, sem parâmetros.

```
<nome_objeto_EthernetServer>begin();
```

Já o objeto que representa a conexão Ethernet, precisa de mais informações do que o objeto WiFi que recebia apenas o IP. Um objeto Ethernet em sua criação nos obriga a informar 4 parâmetros para configuração da rede: um MacAddress para o módulo (através de um array com 6 números hexadecimais), um endereço IP fixo, um endereço de gateway da rede e um endereço que represente a sub-máscara de rede (todos esses 3 últimos, através de um objeto IPAddress, o mesmo utilizado no exemplo anterior para criar e informa o IP do WiFi). Assim, temos:

```
byte <nome_variavel_mac> = {<n1>,<n2>,<n3>,<n4>,<n5>,<n6>}
IPAddress <nome_variavel_ip>(<n1>,<n2>,<n3>,<n4>);
IPAddress <nome_variavel_mascara>(<n1>,<n2>,<n3>,<n4>);
IPAddress <nome_variavel_gateway>(<n1>,<n2>,<n3>,<n4>);

Ethernet.begin(<nome_variavel_mac>, <nome_variavel_ip>,
<nome_variavel_gateway>, <nome_variavel_mascara>);
```

Todos os demais métodos do EthernetClient são análogos aos do WiFiEspClient, como pode ser conferido a seguir.

Testar o objeto EthernetClient para identificar novo cliente está tentando conectar:

```
if( <nome_objeto_EthernetClient> ) ...
```

Verificar se a conexão do cliente já aconteceu e está ativa:

```
<nome_objeto_EthernetClient>.connected()
```

Verificar se estamos recebendo alguma coisa (requisição HTTP) do nosso cliente:

```
<nome_objeto_EthernetClient>.available()
```

Para o recebimento dessas informações através da serial da conexão estabelecida, também podemos criar um objeto do tipo RingBuffer, exatamente como mostrado anteriormente (e por isso mesmo não iremos repetir os métodos). Também utilizamos o método push para depositar no buffer byte a byte lido do objeto EthernetClient:

```
<nome_objeto_RingBuffer>.push(<nome_objeto_EthernetClient>.read())
```

Já quando aos status da conexão, podemos utilizar dois métodos distintos do objeto Ethernet. O primeiro serve para indicar a não detecção de hardware ethernet compatível (retornando a constante EthernetNoHardware).

```
Ethernet.hardwareStatus()
```

Já o segundo serve para indicar se a conexão foi estabelecida com sucesso e existe comunicação ativa (retornando as constantes LinkON e LinkOFF).

```
Ethernet.linkStatus()
```

Conhecendo estes comandos, podemos ver agora as diferenças no código ao adaptar o exemplo anterior para o uso de conexão ethernet em lugar de conexão WiFi para a comunicação.

```
#include <Ultrasonic.h>
#include <SPI.h>                                                    //mudou
#include <Ethernet.h>                                               //mudou

//Variáveis auxiliares
char macRede[] = {0xAB, 0xCD, 0xEF, 0x01, 0x23, 0x45};              //mudou
IPAddress ipRede(192,168,0,100);                                    //mudou
IPAddress mascaraRede(255,255,255,0);                               //mudou
IPAddress gatewayRede(192,168,0,1);                                 //mudou

int statusLed = LOW;
int distancia = 0;

//Objetos das bibliotecas
Ultrasonic ultrassom(7,6);
EthernetServer servidor(80);                                        //mudou
RingBuffer bufferAux(8);

void setup(){
  pinMode(5, OUTPUT);
  digitalWrite(5, LOW);
  Serial.begin(9600);
}

void loop() {

  if(Ethernet.harwareStatus() == EthernetNoHardware){               //mudou
    return;
  }
  if(Ethernet.linkStatus() != LinkON){                              //mudou
    Ethernet.begin(macRede, ipRede, gatewayRede, mascaraRede);      //mudou
    return;
  }       else       {
    servidor.begin();
  }

  statusLed = ultrassom.Ranging(CM);

  EthernetClient cliente = servidor.available();                    //mudou

  if(cliente) {
    bufferAux.init();
    while(cliente.connected()){
      if(cliente.available()){
        bufferAux.push(cliente.read());

        if(bufferAux.endsWith("\r\n\r\n")) {
          respostaRequisicao(cliente);
          break;
        }
        if(bufferAux.endsWith("GET /L")){
          digitalWrite(5, HIGH);
          statusLed = 1;
        }
        else{
          if (bufferAux.endsWith("GET /D")) {
            digitalWrite(5, LOW);
            statusLed = 0;
          }
        }
      }
    }
    cliente.stop();
  }
} //fim do loop
```

Perceba que ao longo do código, existem alguns comentários "//mudou" que identificam que aquela linha específica sofreu alguma alteração do exemplo do código do projeto com WiFiEsp em relação ao código do projeto adaptado para Ethernet. Esta alteração pode ser o nome do objeto, o valor dos parâmetros, o total de parâmetros, etc. Porém, queremos reforçar com estas indicações visuais que a lógica não se altera e todo o restante do código permanece exatamente igual. Logo, a mesma explicação passo a passo dada para o código do exemplo anterior (da página xxx até a página xxx), vale para o código deste exemplo. Inclusive o método respostaRequisicao que monta a página retornada mostrando o valor do ultrassônico e criando um botão para ligar e desligar o led, não sofre qualquer tipo de alteração (e por isso nem foi trazido novamente neste exemplo).

Assim conseguimos perceber que colocar o Arduino na internet, seja com uso de WiFi ou de Ethernet, é um processo muito similar e o domínio de uma das técnicas implica no fácil entendimento e adaptação da outra. Deste modo, ampliamos significativamente as possibilidades em nossos projetos IOT, não limitando os dispositivos à comunicação local, mas permitindo que nossas soluções estejam, agora, na nuvem. Assim, quase de forma literal, o céu é nosso limite.