



Tabla de Contenidos

1) Introducción

- *Motivación del proyecto.*
- *Descripción del proyecto.*
- *Esquema eléctrico del proyecto.*
- *Resumen de elementos componentes del proyecto.*

2) Primer servicio de red: Escaneado de red LAN

- *Funcionamiento general.*
- *Detalles del código del proyecto.*
- *Capturas de la ejecución del código.*

3) Segundo servicio de red: Control de temperatura y humedad

- *Funcionamiento general.*
- *Detalles del código del proyecto.*
- *Capturas de la ejecución del código.*



Introducción

Motivación del Proyecto

Existen multitud de aplicaciones posibles a desarrollar mediante **Arduino**. Sin embargo, muchas de ellas son de carácter específico y difícilmente ampliables y/o adaptables a otros usos en caso de necesidad.

Mediante el uso de servicios de red, las fronteras del sistema se expanden, las comunicaciones con el exterior se abren, la flexibilidad crece y el abanico de posibilidades se multiplica casi de manera infinita en consecuencia.

Un sistema **Arduino** preparado para acceder a la red y para ofrecer servicios basados en ésta, podrá ser fácilmente ampliado y adaptado para incorporar nuevas aplicaciones. En una era en la que todos los dispositivos han de poder estar conectados a la red de redes (Internet) y acceder a diferentes servicios a través de la misma, entendemos que un proyecto con un trasfondo de este tipo está completamente en línea con las tendencias actuales.

Descripción del proyecto

El número de aplicaciones posibles basadas en red es prácticamente infinito. En nuestro proyecto abordaremos los siguientes servicios:

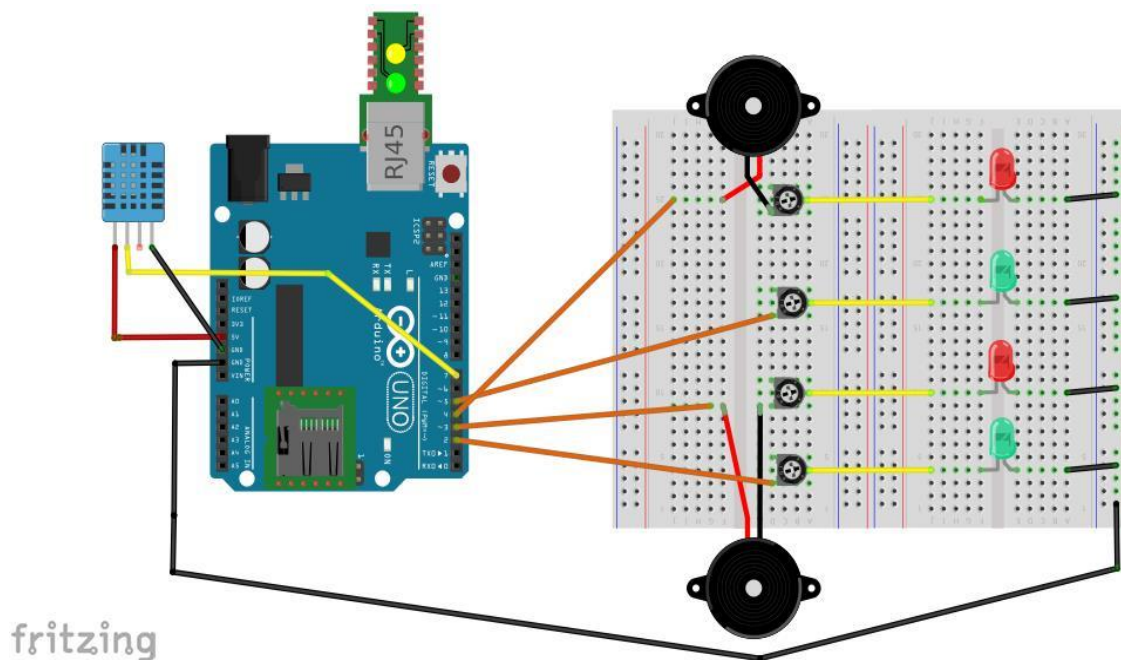
- **Monitorización de red LAN y detección de intrusos**
 - *Conexión permanente del sistema basado en **Arduino** a una LAN para control de direcciones IP presentes en la misma.
El sistema almacenará una tabla con las direcciones IP autorizadas en la red (registrada previamente en una tarjeta de memoria) e informará si detecta alguna dirección no contenida en la tabla (**detector de intrusos**). Las notificaciones con los detalles serán publicadas en “**Twitter**” y en la propia consola de **Arduino**.*
- **Uso de sensor de temperatura para control de temperatura y humedad del ambiente incorporando los siguientes servicios:**
 - *Actualización de cuenta “**Twitter**” en caso de alarma indicando los valores medidos por el mismo (con una periodicidad prefijada).*
 - ***Servidor web** con actualización periódica de página HTML con indicaciones de los valores medidos(exista o no alarma). Acceso a la misma disponible desde cualquier terminal conectado a Internet (incluidos móviles).*



- Servicio de **notificaciones visuales/sonoras** mediante el uso de diodos LED de colores y zumbadores:
 - LEDs en rojo -> Valores por encima de un cierto umbral.
Disparo de alarma sonora.
 - LEDs en verde -> Valores por debajo de un cierto umbral.
Situación de trabajo correcta.

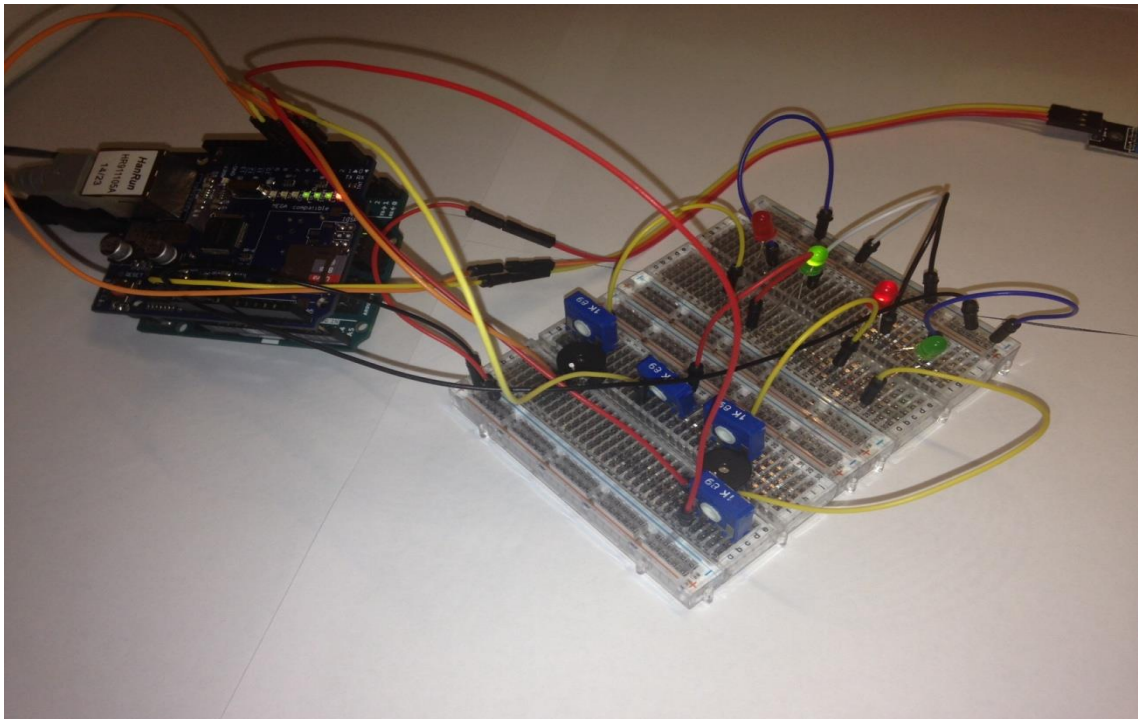
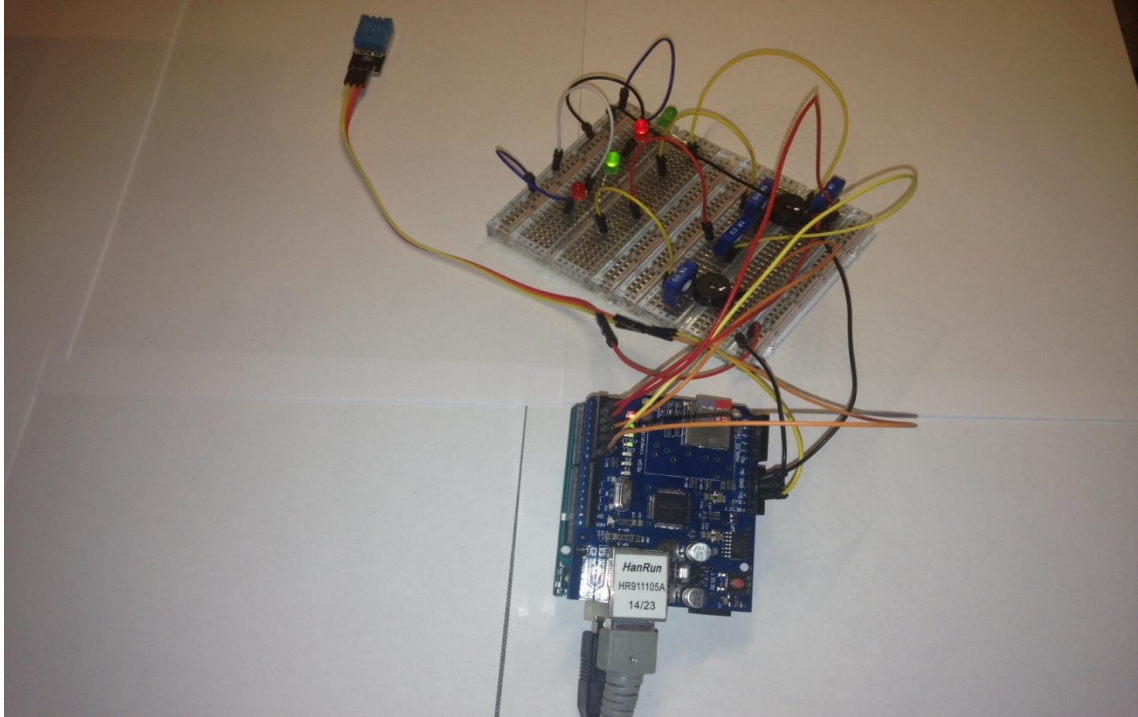
Esquema eléctrico del proyecto

A continuación mostramos el esquema del proyecto generado mediante la aplicación “Fritzing”. Pueden observarse los pines digitales usados para alimentar el circuito:





En las siguientes imágenes mostramos el aspecto del montaje real:





Resumen de elementos componentes del proyecto

- 1 x Arduino Uno.
- 1 x Ethernet Shield.
- 1 x Tarjeta de memoria SD.
- 1 x DHT11 Sensor digital de temperatura y humedad.
- 4 x Potenciómetros de 1K para conexionado de diodos LED a **Arduino**.
- 2 x Diodos LED verdes.
- 2 x Diodos LED rojos.
- 2 x Zumbadores.
- 2 x Placas adicionales para interconexión de componentes.
- Cable rígido para conexionado de **Arduino** con placas.
- Cable de red RJ45 para conexión con router de salida a Internet.



Primer servicio : Escaneado de red LAN

Funcionamiento general

En los siguientes vídeos mostramos cómo funciona el servicio de escaneado de red LAN:

- <https://www.youtube.com/watch?v=rNFGsw1rUGs>
- <https://www.youtube.com/watch?v=Vhed-nPdHpo>

Previamente a detallar el funcionamiento, destacar que:

- El circuito implementado en **Arduino** es conectado a la red local LAN mediante un cable RJ45 y en la propia configuración software fijamos de manera estática una dirección IP para la tarjeta “**Ethernet Shield**” acoplada a **Arduino**.
- La tarjeta SD conectada a dicha tarjeta almacenará las direcciones IPs autorizadas de la LAN. Se trata de un fichero de texto que habremos creado previamente y en el que habremos indicado las direcciones permitidas en la red local.

Una vez ejecutado el código en **Arduino**, se mostrarán primeramente todas las IPs autorizadas contenidas en la tarjeta SD. A continuación, se iniciará un bucle en el que se recorrerán todas las direcciones contenidas en la subred (para nuestra demostración hemos utilizado la subred dada por “**192.168.1.x** “, recorriendo por tanto 255 direcciones y yendo desde la IP acabada en 1 hasta la acabada en 255. En cualquier caso, la subred es configurable en el código).

El número de paquetes que se envían en cada ejecución del comando “ping” es igualmente configurable en el código. A efectos de demostración en los vídeos, hemos fijado un número de paquetes igual a uno. Idealmente, debería usarse un número mayor (como por ejemplo, los cuatro paquetes que se utilizan en el caso de sistemas Windows) para evitar posibles errores si el dispositivo presente en la red no puede responder (por la razón que fuera) al único paquete lanzado.

Según se van recorriendo las IPs, si Arduino obtiene respuesta al “ping” enviado a una determinada dirección, comprobará si esa IP está contenida dentro de la lista de IPs autorizadas. En caso afirmativo, continuará con la ejecución. En caso contrario, enviará un “**tweet**” a una cuenta creada en la red social “**Twitter**” (**@esca_neador**) específicamente para mostrar las alarmas de IPs no autorizadas presentes en la LAN.

Una vez que ha terminado de recorrer toda la red, el programa volverá al principio (primera dirección IP de la lista) y comenzará todo el proceso de nuevo.



Detalles del código del proyecto

El código hace uso de una serie de librerías, en concreto:

- **SD.h** → Usada para gestiones de la tarjeta de memoria instalada en la “**Ethernet Shield**”.
- **SPI.h** → Comunicación con periféricos SPI.
- **Ethernet.h** → Necesaria para la gestión de la tarjeta “**Ethernet Shield**”.
- **Twitter.h** → Usada para envío de “**tweets**” a red social “**Twitter**”.
- **ICMPPing.h** → Requerida para el envío de paquetes a direcciones IP mediante “ping”.

Las siguientes capturas muestran las distintas funciones implementadas con algunos detalles de las mismas:

Comienzo del programa

Primeramente definimos las librerías que usará el programa y creamos una serie de variables y constantes.

```

/* Control de Ip's en red local:
*Programa que controla si las ip's presentes en un LAN se encuentran registradas en una tabla de control.
*De no estarlo se genera un mensaje de alerta que se envia a la red social Twitter, con expresión de la IP
*no registrada
*/

#include <SD.h>
#include <SPI.h>
#include <Ethernet.h>
#include <Twitter.h>
#include <ICMPPing.h>

#define FBYTE 192
#define SEBYTE 168
#define TBYTE 1
#define FFBYTE 144

#define NUMPAQUETES 4

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ipConecton[] = {FBYTE,SEBYTE,TBYTE,FFBYTE};
IPAddress myDns(8,8,8,8);
Twitter twitter("3095700296-qfImjpwAY2Q2p3pxIUEU49QmJTy6ExXUfXetc27");
SOCKET pingSocket = 0;
char buffer [64];
ICMPPing ping(pingSocket, (uint16_t)random(0, 255));
EthernetClient client;
int chipSelect=4;
int ipsLeidas[5][4];
int numIps=0;

```

Librerías necesarias para la ejecución del código

Definición de la IP estática de la tarjeta de red

Definición de las variables necesarias

//mac para la placa Arduino
//IP para la placa Arduino
//Servidor DNS, establecemos la dirección de DNS's de google.com
//Token del que nos provee Twitter para identificarnos como usuarios
//socket para establecer comunicación en la LAN
//un array de caracteres para envío de mensajes a la consola Serie
//objeto de la clase ICMPPing para hacer pings en la red local
//este puerto será utilizado para la conexión con la tarjeta SD
//un array de enteros de 5x4 que almacenará un máximo de cinco direcciones ip autorizadas en la lan
//numero de ips leidas del fichero de ips autorizadas



Función “Setup”

```
void setup(){
  Ethernet.begin(mac,ipConection,myDns);
  delay(100);
  Serial.begin(9600);
  delay(1000);
  Serial.print(F("Conectado con IP: "));
  for (byte i = 0; i < 4; i++){
    Serial.print(ipConection[i], DEC);
    //Serial.print(ipConection[i]);
    Serial.print(F(", "));
  }
  Serial.println();
  // Si ha habido error al leer la tarjeta informamos por el puerto serie.
  if (!SD.begin(chipSelect)){
    Serial.println(F("Error al leer la tarjeta."));
  }
  // Abrimos el archivo.
  File dataFile = SD.open("ipsAuth3.txt");
  if (dataFile) {
    //abro if
    //Mientras el fichero tenga contenido
    while (dataFile.available()) { //abro while
      for(byte i=0;i<4;i++){ //abro for
        char crct;
        byte index=0;
        char valor[4];
        while((crct=dataFile.read())!='\0'){ //abro while
          valor[index]=crct;
          valor[index+1]='\0';
          index++;
        } //cierro while
        ipsLeidas[numIps][i]=atoi(valor);
      } //cierro for
      Serial.print(F("ip leida del fichero: "));
      Serial.print(ipsLeidas[numIps][0]);
      Serial.print(F(", "));
    }
  }
}
```

La función "setup" se encarga de leer las IPs autorizadas contenidas en la tarjeta de memoria así como de inicializar la tarjeta de red con los parámetros adecuados (IP, MAC y DNS)

Función “Loop”

```
void loop()
{
  for(int i=1;i<256;i++){ //for
    IPAddress pingAddr(FBYTE,SBYTE,TBYTE,i);
    ICMPEchoReply echoReply = ping(pingAddr, NUMPAQUETES);
    if (echoReply.status == SUCCESS){ //if ping resultado
      sprintf(buffer,"Reply[%d] from:%d.%d.%d.%d:bytes=%d time=%ldms TTL=%d",
        echoReply.data.seq,
        echoReply.addr[0],
        echoReply.addr[1],
        echoReply.addr[2],
        echoReply.addr[3],
        REQ_DATASIZE,
        millis() - echoReply.data.time,
        echoReply.ttl);

      if(!(comprobarIP(echoReply.addr))){ // si la ipE está en el fichero de ips
        Serial.println(F("Ip NO autorizada"));
        sprintf(buffer,"Ip no autorizada en:%d.%d.%d.%d, cd.-%d", FBYTE,SBYTE,TBYTE,i,millis());
        tweet(buffer);
      }else{
        Serial.println(F("IP AUTORIZADA"));
      } //cierro if si la ipE está en el fichero de ips

    }else{
      sprintf(buffer, "Echo request failed:%d.%d.%d.%d", FBYTE,SBYTE,TBYTE,i);
    } // cierre if ping resuelto
    //imprimo el buffer
    Serial.println(buffer);
  } //cierro for
}
```

La función "Loop" recorrerá las IPs de la subred e irá haciendo "ping" a cada una de ellas con el número de paquetes configurado previamente. Cuando obtenga respuesta en el comando "ping" para una determinada dirección, comprobará con la llamada a la función "ComprobarIP" si dicha dirección está dentro de la lista de IPs autorizadas



Función “comprobarIP” y “tweet”

```

boolean comprobarIP(IPAddress a){

  Serial.print(F("\n\nPing atendido en ip: "));
  //Serial.print(echoReply[0]);
  Serial.print(a[0]);
  Serial.print(F("."));
  //Serial.print(echoReply[1]);
  Serial.print(a[1]);
  Serial.print(F("."));
  //Serial.print(echoReply[2]);
  Serial.print(a[2]);
  Serial.print(F("."));
  //Serial.print(echoReply[3]);
  Serial.print(a[3]);
  Serial.println();
  for(byte i=0;i<numIps;i++){
    boolean chivato=true;
    for(byte j=0;j<4;j++){
      if(!(a[j]==ipsLeidas[i][j])){
        chivato=false;
      }
    }
    if(chivato){
      return chivato;
    }
  }
  return false;
}

```

La función "comprobarIP" se encargará de comparar la IP leída con las IPs autorizadas

```

//Función que recibe un Array de caracteres y lo envía a Twitter usando la librería Twitter.h
void tweet(char msg[]){
  char textoTweet[144];
  sprintf(textoTweet,"%s",msg);
  if (twitter.post(textoTweet)){
    int status = twitter.wait(&Serial);
    if (status == 200){
      Serial.println(F("Tweet enviado"));
    }else{
      Serial.print(F("Fallo subida tweet:cdg:"));
      Serial.println(status);
    }
  }else{
    Serial.println(F("Fallo conexión."));
  }
}

```

Esta función enviará un "tweet" cuando sea llamada para reportar que una IP no autorizada ha sido detectada en la LAN



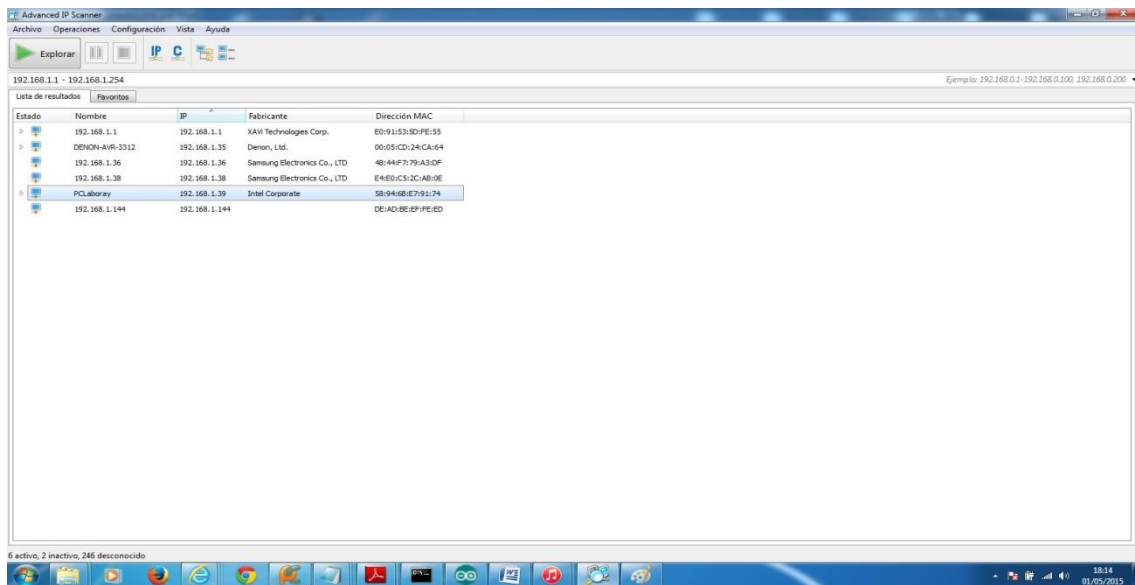
Detalles de la ejecución

El detalle de la ejecución del código puede encontrarse en el siguiente vídeo:

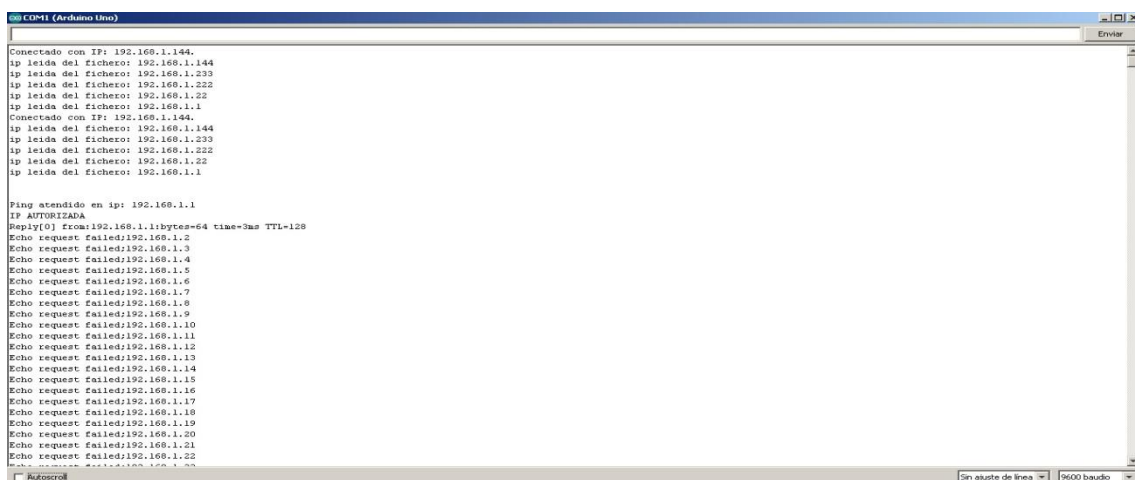
- <https://www.youtube.com/watch?v=Vhed-nPdHpo>

A continuación incluimos algunas capturas de dicha ejecución para indicar algunos comentarios:

1) **Dispositivos dentro de la LAN** → Previamente a la ejecución del código, analizamos la red LAN para conocer qué dispositivos existían en la misma. De esta forma, podíamos valorar la corrección de los resultados del código.



2) **Comienzo de la ejecución** → Como vemos, primeramente se muestra la lista de IPs autorizadas contenidas en la tarjeta y a continuación comienza el bucle en el que lanza “ping” a todas las direcciones IP de la subred. La primera de ellas (192.168.1.1) responde y, al estar dentro de la lista de IPs autorizadas, no produce ningún error.

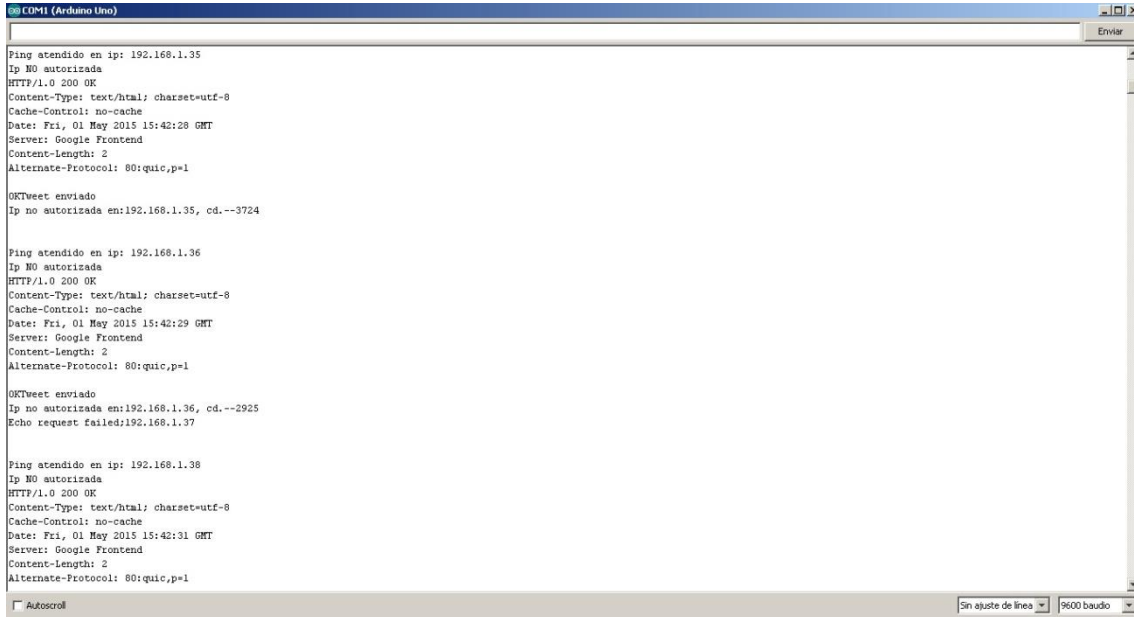




3) **Detección de IPs no autorizadas** → De acuerdo con la lista de IPs presentes en la LAN y teniendo en cuenta las direcciones autorizadas que hemos definido, las direcciones IP dadas por la siguiente lista deberían generar error:

192.168.1.35 // 192.168.1.36 // 192.168.1.38 // 192.168.1.39

En la propia consola de **Arduino** se muestra que, efectivamente, estas direcciones no están autorizadas (y que se procederá al envío de “**tweets**”):



```

COM1 (Arduino Uno)
Enviar

Ping atendido en ip: 192.168.1.35
Ip NO autorizada
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: no-cache
Date: Fri, 01 May 2015 15:42:28 GMT
Server: Google Frontend
Content-Length: 2
Alternate-Protocol: 80:quic,p=1

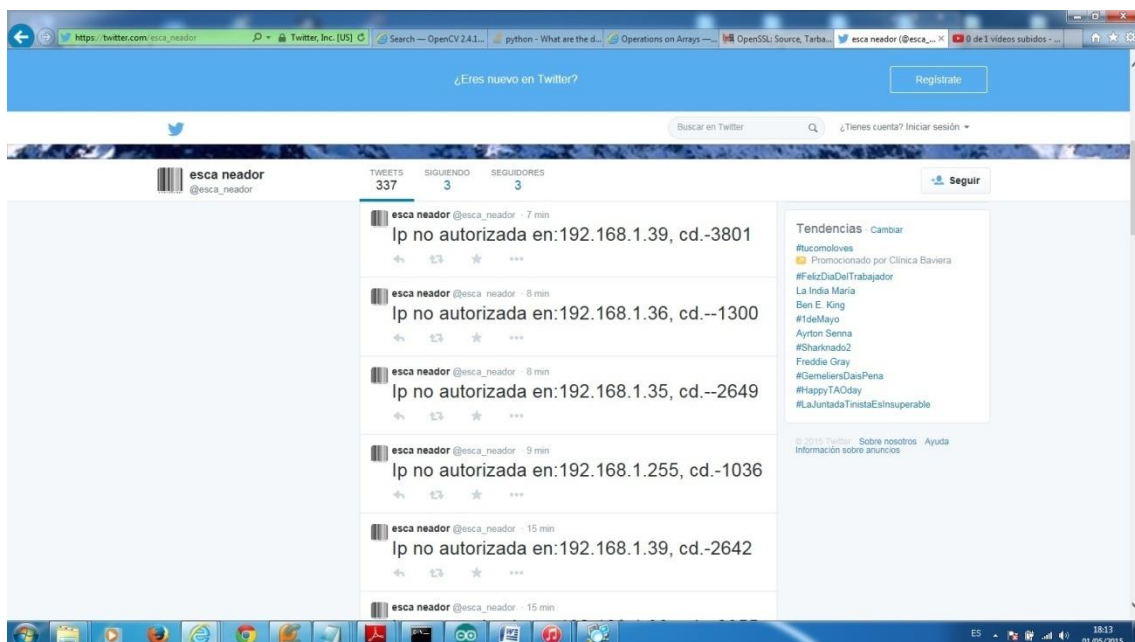
OKTweet enviado
Ip no autorizada en:192.168.1.35, cd.--3724

Ping atendido en ip: 192.168.1.36
Ip NO autorizada
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: no-cache
Date: Fri, 01 May 2015 15:42:29 GMT
Server: Google Frontend
Content-Length: 2
Alternate-Protocol: 80:quic,p=1

OKTweet enviado
Ip no autorizada en:192.168.1.36, cd.--2925
Echo request failed:192.168.1.37

Ping atendido en ip: 192.168.1.38
Ip NO autorizada
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: no-cache
Date: Fri, 01 May 2015 15:42:31 GMT
Server: Google Frontend
Content-Length: 2
Alternate-Protocol: 80:quic,p=1
Autoscroll
Sin ajuste de línea 9600 baudio
  
```

3) **Envío de “tweets” reportando la información** → Por último, el programa enviará “**tweets**” con las direcciones IP no autorizadas encontradas en la red. Destacar que para evitar errores en la subida de dichos “**tweets**”, el código utilizará la función “**millis()**” para poder crear diferencias entre mensajes y que “**Twitter**” no los rechace.





Segundo servicio : Control de temperatura y humedad

Funcionamiento general

En los siguientes vídeos mostramos cómo funciona el servicio de control de temperatura y humedad:

- <https://www.youtube.com/watch?v=7FnVYPk7hr4>

- <https://www.youtube.com/watch?v=Ytx7QGvnbQE>

El sistema está pensado para que obtenga la temperatura y la humedad mediante un sensor y que compare dichas medidas con unos umbrales establecidos (y configurables en el código). Si las mediciones están por debajo de los umbrales, entonces las condiciones ambientales serán correctas desde el punto de vista de nuestras necesidades. En caso contrario, será necesario disparar una alarma.

El sistema incorpora facilidades para proceder con la monitorización a dos niveles:

- Local → El sistema utiliza diodos LED de color verde para indicar que los valores de temperatura y humedad medidos por el sensor están por debajo de los umbrales establecidos. Si se supera dicho umbral, entonces se apagará el diodo LED verde correspondiente y se encenderá un LED rojo, activando a su vez una alarma sonora para indicar el problema (mediante zumbador).
- Por red → El sistema incorpora un servidor web al que los usuarios interesados se pueden conectar vía Internet para comprobar en tiempo real cuales son las medidas de temperatura y humedad que está recogiendo el sensor. Dichas medidas se actualizarán cada cierto tiempo (siendo éste un parámetro configurable dentro del código). Por otro lado, en caso de que la temperatura y/o humedad superen los umbrales pre-configurados, el sistema enviará una notificación a la red social “**Twitter**” reportando tanto el tipo de alarma como el valor medido en la misma. Mientras que se estén produciendo medidas por encima del umbral, el sistema continuará enviando “**tweets**” a la red social. La periodicidad de dichos envíos es igualmente configurable.

Detalles del código del proyecto

Las librerías usadas por el código son las siguientes:

- **Twitter.h** → Usada para envío de “**tweets**” a red social “**Twitter**”.
- **SPI.h** → Comunicación con periféricos SPI.
- **Ethernet.h** → Necesaria para la gestión de la tarjeta “**Ethernet Shield**”.
- **DHT.h** → Gestión de las operaciones con el sensor de temperatura y humedad.



Comienzo del programa

```
#include <Twitter.h>
#include <SPI.h>
#include <Ethernet.h>
#include <DHT.h>

#define MAXTEMP 28
#define MAXHUME 20
#define HUMVAL 2
#define HUMAL 3
#define TEMPVAL 5
#define TEMPAL 4
#define PINDHT 7

byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
IPAddress ip(192,168,1,144); // Ip del servidor 192.168.1.144
IPAddress myDns(8,8,8,8);
EthernetServer server(80); // Arrancamos el servidor en el puerto estandar 80
DHT dht( PINDHT, DHT11);
char buffer[140];
Twitter twitter("3095700296-qHmjpwEAY2Q2p3pxIUEU49QmJTy6ExXuFXetc27"); //Token del que nos provee Twitter para identificarnos como usuarios
```

Inicio del programa en el que definimos una serie de constantes que luego usaremos a lo largo del programa. En concreto, definimos los umbrales de temperatura y humedad por encima de los cuales lanzaremos las alarmas. Por otro lado, creamos constantes para indicar los pines digitales en los que conectaremos las partes del circuito dedicadas a la gestión de la temperatura y humedad así como los dedicados para el conexionado del Arduino con el sensor DHT11

De la misma manera, creamos las variables necesarias para poder inicializar la tarjeta de red conectada a Arduino que luego nos permitirá acceder al servidor web y crear tweets en la red social Twitter.

Función “Setup”

```
void setup()
{
    pinMode(HUMVAL,OUTPUT);
    pinMode(HUMAL,OUTPUT);
    pinMode(TEMPVAL,OUTPUT);
    pinMode(TEMPAL,OUTPUT);
    dht.begin();
    Serial.begin(9600);
    while (!Serial) ;

    Ethernet.begin(mac, ip,myDns);
    delay(3000);
    server.begin(); // Inicia el servidor web
    Serial.print("Servidor Web en la direccion: ");
    Serial.println(Ethernet.localIP());
}
```

La función setup() establece el modo en el que funcionarán los pines digitales conectados a las secciones de control de temperatura y humedad así como inicializará la tarjeta de red con los valores definidos.



Función “Loop”

```
void loop(){
  EthernetClient client = server.available(); // Cuando se produce una petición al puerto 80 de la ip local 192.168.1.144
  if (client)
  { Serial.println("CONECTADO");
    boolean currentLineIsBlank = true; // Las peticiones HTTP finalizan con linea en blanco
    while (client.connected())
    { if (client.available())
      { char c = client.read();
        Serial.write(c); // mostramos gestión http por la consola
        // A partir de aquí mandamos nuestra respuesta
        if (c == '\n' && currentLineIsBlank)
        { // Enviar una respuesta típica
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");
          client.println("Refresh: 20"); // Actualizar cada minuto
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");

          float h = dht.readHumidity(); // Leer el sensor
          float t = dht.readTemperature();
          Serial.println(t);
          Serial.println(h);
          // Creamos la página web
          client.print("<head><title>Observacion ambiental</title></head>");
          client.print("<body><IMG SRC='https://encrypted-tbn3.gstatic.com/images?q=tbn:ANd9GcTUM_2J_jYp4s96GZEAd16LOWk-wfY7bWB1Yuz0tHC29Eqv2YaU'>");
          client.print("<h1> Humedad y temperatura en el local</h1><p>Temperatura: ");
          client.print(t); // Aquí va la temperatura
          client.print(" grados Celsius</p>");
          client.print("<p>Humedad: ");
          client.print(h); // Aquí va la humedad
          client.print(" por ciento</p>");
          client.print("<p><em> La pagina se actualiza cada 20 segundos.</em></p></body></html>");
          GestionarLeds(t,h);

          break;
        }
      }
    }
    if (c == '\n')
      currentLineIsBlank = true; // nueva linea
    else if (c != '\r')
      currentLineIsBlank = false;
  }
  }else{ (***)
    float h = dht.readHumidity(); // Leer el sensor
    float t = dht.readTemperature();
    GestionarLeds(t,h);
  }
  delay(10); // Para asegurarnos de que los datos se envia
  client.stop(); // Cerramos la conexion
  delay(20000);
}
```

La función loop() posee dos modos de funcionamiento principalmente. Si no existe ningún cliente web haciendo peticiones al servidor web, entonces el primer if fallará ("if(client)") y se ejecutará el "else" correspondiente (***)

En caso de existencia de clientes web, el programa evaluará "if(client)" y accederá a la zona delimitada por él.

En cualquiera de las dos situaciones se efectúa llamada a la función para gestionar las notificaciones visuales y sonoras, llamada "GestionarLeds".

Los detalles de la misma los veremos en la captura siguiente.



Función “GestionarLeds” y “Tweet”

```
//Procedimiento que recibe un Array de cotes y lo envia a Twitter usando la libreria Twitter.h
void tweet(char msg[]){
    char textoTweet[144];
    sprintf(textoTweet,"%s",msg);
    if (twitter.post(textoTweet)){
        int status = twitter.wait(&Serial);
        if (status == 200){
            Serial.println(F("Tweet enviado"));
        }else{
            Serial.print(F("Fallo subida tweet:cdg:"));
            Serial.println(status);
        }
    }else{
        Serial.println(F("Fallo conexión."));
    }
}

void GestionarLeds(float t,float h){
    if(t>MAXTEMP){
        digitalWrite(TEMPVAL,LOW);
        digitalWrite(TEMPAL,HIGH);
        char temperatura[6];
        dtostrf(t,5, 2, temperatura);
        temperatura[5]='\0';
        sprintf(buffer,"Alarma ambiental. Temperatura %s grados.- cod - %d",temperatura,millis());
        Serial.println(buffer);
        delay (1000);
        tweet(buffer);
    }else{
        digitalWrite(TEMPVAL,HIGH); //PONEMOS HUMEDAD EN VALIDO
        digitalWrite(TEMPAL,LOW);
    }

    if(h>MAXHUME){
        digitalWrite(HUMVAL,LOW);
        digitalWrite(HUMAL,HIGH);
        char humedad[6];
        dtostrf(h,5, 2, humedad);
        humedad[5]='\0';
        sprintf(buffer,"Alarma ambiental. Humedad %s por ciento.- cod - %d",humedad,millis());
        Serial.println(buffer);
        delay (1000);
        tweet(buffer);
    }else{
        digitalWrite(HUMVAL,HIGH); //PONEMOS HUMEDAD EN VALIDO
        digitalWrite(HUMAL,LOW);
    }
}
}
```

Esta función se encarga de enviar un tweet a Twitter en caso de que sea necesario reportar alarmas de temperatura y/o humedad

La función "GestionarLeds" se encarga de comparar los valores de temperatura y humedad medidos con los umbrales que nosotros hemos predefinido para nuestro sistema. En caso de que alguna de ellas esté por encima de dichos umbrales, entonces se dispara la alarma, tanto visual activando un diodo LED rojo como sonora, dejando pasar corriente por un zumbador.

Si no se producen alarmas, los pines digitales que alimentan a los diodo LED de color verde permanecerán a HIGH, notando por tanto, que el sistema está detectando unos valores de temperatura y humedad dentro de los márgenes correctos de funcionamiento.

En caso de envío de tweet para notificar una alarma, se hace uso de función millis() para que los mensajes tengan un código asociado y puedan diferenciarse.



Detalles de la ejecución

El detalle de la ejecución del código puede encontrarse en el siguiente vídeo:

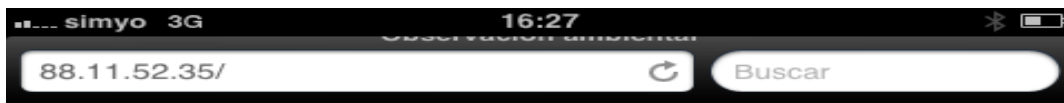
- <https://www.youtube.com/watch?v=Ytx7QGvhbQE>

A continuación incluimos algunas capturas de dicha ejecución para indicar algunos comentarios:

1) **Inicio de la ejecución** → Una vez cargado y ejecutado el código en **Arduino**, el sistema dará de alta el servidor web y quedará a la espera de que los clientes web accedan al mismo (para la monitorización por red). Paralelamente, el sistema medirá los valores de temperatura y humedad presentes con una periodicidad prefijada y mostrará mediante en los LED el estatus.

2) **Acceso al servidor web** → Dado que el sistema se encontrará en una red privada, será necesario habilitar la funcionalidad de “**port forwarding**” en el router de salida a Internet de tal forma que nuestro servidor web sea accesible a través de dicha red. Por otro lado, también es necesario conocer la IP pública del router para poder llamar al mismo.

Realizadas estas configuraciones, podemos acceder a dicho servidor web mediante terminales móviles conectados a redes **3G/4G**, tal y como se muestra en las siguientes imágenes:



Humedad y temperatura en el local

Temperatura: 20.00 grados Celsius

Humedad: 38.00 por ciento

La pagina se actualiza cada 20 segundos.



Humedad y temperatura en el local

Temperatura: 20.00 grados Celsius

Humedad: 42.00 por ciento

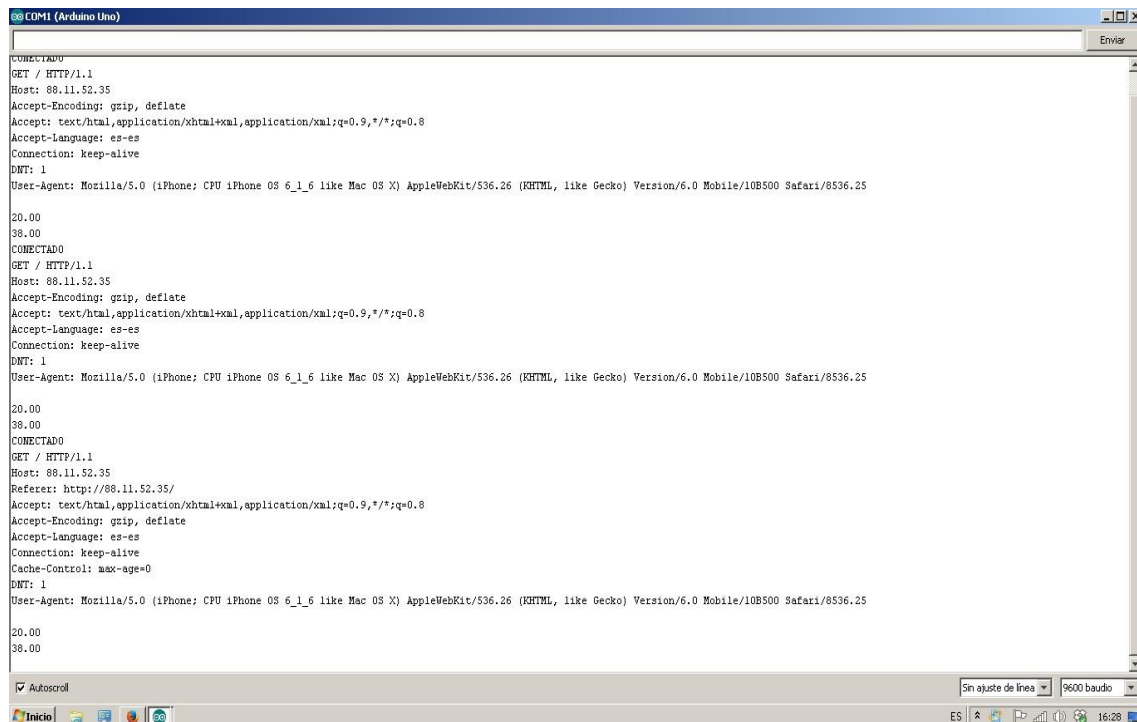
La pagina se actualiza una vez por minuto.





En ellas observamos cómo los terminales acceden a la IP pública del router (figuran 88.11.52.35 y 88.18.181.120 dado que, tras cada apagado y encendido del router, el proveedor asigna una IP pública distinta y dichas capturas fueron obtenidas en distintos momentos) y cómo éste responde accediendo al servidor web implementado en **Arduino** para devolver la página **HTML** con los datos medidos de temperatura y humedad. Igualmente se informa a los usuarios de que la página se actualizará cada cierto tiempo (parámetro configurable en el código).

Por otro lado, cada vez que se realiza una petición web de algún cliente, **Arduino** mostrará por pantalla el origen de la petición así como otros detalles propios del mensaje de tipo “GET”, tal y como se aprecia en la siguiente captura:



```

COM1 (Arduino Uno)
CONECTADO
GET / HTTP/1.1
Host: 88.11.52.35
Accept-Encoding: gzip, deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-es
Connection: keep-alive
DNT: 1
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 6_1_6 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10B500 Safari/8536.25
20.00
38.00
CONECTADO
GET / HTTP/1.1
Host: 88.11.52.35
Accept-Encoding: gzip, deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-es
Connection: keep-alive
DNT: 1
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 6_1_6 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10B500 Safari/8536.25
20.00
38.00
CONECTADO
GET / HTTP/1.1
Host: 88.11.52.35
Referer: http://88.11.52.35/
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: es-es
Connection: keep-alive
Cache-Control: max-age=0
DNT: 1
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 6_1_6 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10B500 Safari/8536.25
20.00
38.00
Autoscroll
Sin ajuste de línea 9600 baudio

```

3) **Tratamiento de alarmas**→ Como hemos comentado, cada vez que el valor de temperatura y/o humedad medidos están por encima de un umbral prefijado, el sistema debe disparar una alarma. El sistema monitoriza continuamente dichos valores y en caso de alarma actuará de la siguiente forma:

- A nivel local activará el/los LED rojos correspondientes y su alarma sonora.
- A nivel de red enviará un “**tweet**” a una cuenta pre-configurada para avisar del valor de medido.

Toda esta funcionalidad es independiente del servidor web, dado que éste es usado para conocer cuáles son los valores de temperatura y humedad medidos por el sensor en cada momento concreto.

El vídeo muestra las cuatro posibles combinaciones de funcionamiento (alarma/ no alarma de ambas mediciones) y cómo actúa el sistema cuando se producen.



Las siguientes capturas muestran algunas de las publicaciones en la red social “**Twitter**” con las mediciones de alarma:



Incluimos también los mensajes que el sistema envía por consola cuando se envían los “**tweets**” con las alarmas:

```
Alarma ambiental. Humedad 38.00 por ciento.- cod - 26092
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: no-cache
Date: Fri, 01 May 2015 14:30:34 GMT
Server: Google Frontend
Content-Length: 2
Alternate-Protocol: 80:quic,p=1

OKTweet enviado
Alarma ambiental. Humedad 38.00 por ciento.- cod - -17077
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: no-cache
Date: Fri, 01 May 2015 14:30:56 GMT
Server: Google Frontend
Content-Length: 2
Alternate-Protocol: 80:quic,p=1

OKTweet enviado
```