Network connected temperature sensor

Mark E. Taylor

metphoto.net

Change log

Date	Version	Comment
2nd January 2010	v1.0	Initial document.
10th January 2010	v1.1	Updated and corrected.
23rd January 2010	v1.2	Arduino code re-written to make use of the “Dallas Temperature Control Library”.
29th August 2010	v1.3	Minor text corrections.

Table of contents

Network connected temperature sensor

Introduction

This document describes the steps needed to create a network connected temperature probe. The remote temperature is then displayed on a web page.

It sounds complex but each stage is reasonably straightforward. The goal of the project was to link the physical world to the virtual world. Something that is becoming increasingly called “The Internet of Things”. This is hardly a new idea; I remember Coke machines being connected to the Internet 15 years ago. A project like this also clearly associated with the world of “hacking” (in its original sense) and the more recent world of “maker”.

There is a huge amount of information on the web explaining how any particular topic works or how to implement something. However there are very few places where complete projects are described. The only example I can think of is Instructables. So I thought I would document my small project for all to see. The document does not provide every small detail on how to do a particular task, but rather it is an overall guide to the necessary steps.

Overall the project took a couple of days to complete and that included learning how to get CGI working on Apache and learning a tiny amount about Perl CGI itself. It should be noted that there are other (better, more efficient) ways to do what I have done here - but this is a learning experience rather than a full commercial project.

If you feel this guide needs more detail or if you have any questions then please let me know.

Mark E. Taylor
London, UK
January 2010

Contact: info@metphoto.net

Disclaimer and license

I cannot be held responsible for data loss or physical damage that may result from this document. **You** are responsible for your own safety and data security.

Always follow manufactures guidelines when installing electrical equipment.

Prerequisites

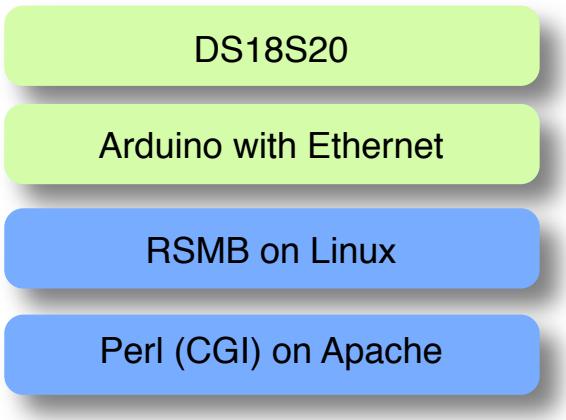
I created this project with the following components:

- Viglen MPC-L micro PC - I do not believe this wonderful little PCs are being made any more.
- Arduino with Ethernet shield
- MQTT client for Arduino
- DS18S20 1-wire digital thermometer
- A server running a web server (This article assumes Linux and Apache.)
- Really Small Message Broker (RSMB) from IBM alphaWorks
- The server needs to have Perl and CGI installed
- The Perl MQTT::Client library
- MySQL database (optional)
- RRDTOOL (optional)

The Viglen PC runs the Xunbuntu operating system and makes an excellent home server. It consumes around 10w of power continuously.

Overview

Below is a diagram showing the various layers of hardware and software used by this project. (The diagram is upside down by standard IT conventions!) There is a lot of terminology mentioned below, but don't worry it is all explained in detail further on.



In order the layers provide the following:

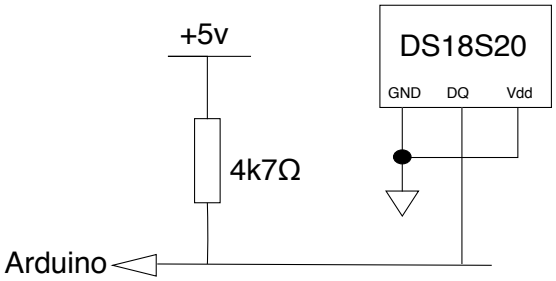
- A DS18S20 1-wire digital thermometer connected to the Arduino - this senses the ambient temperature
- An Arduino board with an Ethernet shield attached (running a MQTT client) - this processes the data from the thermometer and sends a message to the RSMB server
- A RSMB server running on Linux - receives the messages from the MQTT client on the Arduino

- A Perl CGI script to create a web page and display the temperature - the Perl script ‘subscribes’ to a ‘topic’ from the RSMB and displays a web page

Hardware setup - DS18S20

The hardware set up is very straight forward. The DS18S20 1-wire digital thermometer comes in a small three pin package. There are many tutorials on the web showing how to use them. I connected mine by following the instructions on the manufactures web site.

Pins one and three are joined and linked to ground. Use the ground pin from the Arduino. The digital out pin also provides the power to the device. Use a 4.7k ohm resistor between the +5v supply, again from the Arduino, and the centre pin on the device. The centre pin is also taken to a digital input pin on the Arduino, in my case that is pin three.



Arduino hardware and software

The Arduino hardware is very straight forward. A standard Arduino micro controller and a Ethernet shield. Attach the Arduino and the Ethernet shield and apply power; I connect mine to a spare USB port on my Linux server. Then connect the DS18S20 to the Arduino, as described above. Finally connect the Ethernet shield to your network.

For more information on the Arduino see:

- Arduino main site
- Arduino Ethernet shield

My sensor is very close to the Arduino, however I believe it is possible to position it some distance from the Arduino, using suitable lengths of cable. I have yet to experiment with this aspect.

You will need to use the Arduino software suite to program the Arduino.

The application running on the Arduino has two tasks to perform. These are:

- Capture data from the DS18S20 and decode it
- Send a message containing this data to the RSMB server

One question may occur to you; why process the temperature data on the Arduino? Why not just take the raw data and process it on the server? There are two simple answers. Firstly I already had the Arduino routine to decode the data to ASCII (an ideal format to send via MQTT). So why waste time reinventing something? Furthermore, why not get

the Arduino to do some of the hard work? It is a tiny computer after all. Also to my mind the idea of sending a fully formatted ‘message’ to the RSMB seemed like the most elegant way of completing the task.

Thanks go to SK Pang electronics who wrote the function to convert a float to a string. See here: http://www.skpang.co.uk/content/view/31/42/

Below is my sketch for the Arduino.

```
#include <Ethernet.h>
#include <PubSubClient.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// Completely re-written to make use of the Dallas Temperature Control Library.
// http://milesburton.com/wiki/index.php?title=Dallas_Temperature_Control_Library
// Mark E Taylor - 23rd January 2010.

// The DS18S20 is connected on pin digital 3.
#define ONE_WIRE_BUS 3
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

int ledPin=7; // LED connected to digital pin 7.
int def_delay=0xbb8; //The default delay, 3000 milliseconds.
int send_delay=def_delay;
char* ardTempTopic = "arduinotemperature"; //The temperature is published on this
topic.
char* ardLEDSetTopic = "arduinotemperatureset"; //The LED is set on this topic.
char* ardLEDStatusTopic = "arduinotemperaturestatus"; //The LED status is published on
this topic.
char buff[6];

byte mac[] = {  0xAE, 0xDD, 0xCE, 0xEF, 0xFE, 0xED };
byte ip[] = { 10, 101, 1, 14 }; //Address of Arduino.
byte server[] = { 10, 101, 1, 15 }; //Address of RSMB server.

void callback(char* topic, byte* payload,int length) {
// Check there is a payload at all.
if (length > 0) {
  if (strcmp(topic,ardLEDSetTopic)==0) { //Which topic is this message for?
    byte ardValue = payload[0];
    switch (ardValue) {
    case 0x31: //ASCII "1" in hex.
    LED_ON();
```

```
    break;
    case 0x30: //ASCII "0" in hex.
    LED_OFF();
    break;
    default:
    Serial.print("Now doing the default action.\n");
  }//End of switch section.
} //End of check topic name section.
} //End of check length section.
} //End of callback function.

PubSubClient client(server, 1883, callback);

void setup(void)
{
  Ethernet.begin(mac, ip);
  client.connect("Arduino","",0,1,"ArduinoWILL");
  client.subscribe(ardLEDSetTopic); //Subscribe to topic to receive messages.
  pinMode(ledPin, OUTPUT);
  LED_OFF(); //Set the LED off and send a message.
  sensors.begin();
  Serial.begin(28800);
}

void LED_ON()
{
  digitalWrite(ledPin, HIGH); //Set the LED on.
  client.publish(ardLEDStatusTopic,"The LED on the Arduino is now on."); //Publish
message on topic arduinotemperaturestatus.
}

void LED_OFF()
{
  digitalWrite(ledPin,LOW); //Set the LED off.
  client.publish(ardLEDStatusTopic,"The LED on the Arduino is now off."); //Publish
message on topic arduinotemperaturestatus.
}

void loop()
{
client.loop(); //Checks for incoming MQTT messages.
//Call sensors.requestTemperatures() to issue a global temperature request to all
sensors on the bus.
sensors.requestTemperatures(); // Send the command to get temperatures.
```

```
float reading = sensors.getTempCByIndex(0);//Index 0 means the first sensor on the one
wire bus.
tempToAscii(reading,buff);//Now convert to a string to be published.
client.publish(ardTempTopic,buff); //Name of topic and data to publish.
delay(send_delay); //Wait before sending the message again.
}

void tempToAscii(double temp, char *buff) {
// This code from a project by SK Pang. See http://www.skpang.co.uk/content/view/31/42/
int frac;
frac=(unsigned int)(temp*1000)%1000;
itoa((int)temp,buff,10);
strcat(buff,".");
itoa(frac,&buff[strlen(buff)],10);
}
```

Really Small Message Broker

The Really Small Message Broker (RSMB) is a derivative of a large enterprise product made by IBM called MQ. The RSMB makes use of a protocol called MQ Telemetry Transport (MQTT). Basically this application passes messages (telemetry) from one from application to another. In this case the RSMB is listening for messages from the Arduino. These messages are then sent back out to any application that cares to listen ('subscribe') to them.

Rather than describe how the application works (I only have a basic understanding anyway) I thought I would describe how I use the RSMB.

There are several tutorials on the web that show how a small web server could be embedded into a Arduino board. It would therefore be possible to create a complete web page on the Arduino that displayed the temperature from the DS18S20. However for me this was very limiting, as the pages that could be created were small and complex to maintain. It seemed to me to be more sensible to abstract the data from the web layer. [This is my IT background kicking in here. My day job is to look after some major systems.] I spent a few days looking at ways of getting data from an Arduino to a server. Eventually I found the RSMB; this provides a generalized way of sending messages, of any content, between the Arduino and server.

The RSMB can be obtained from the IBM alphaWorks site, here. You will need to register an account before being able to download the application. After that just follow the instructions to install. It is very easy. There is nothing that needs to configured on the RSMB, it just sits there and sends and receives messages.

Perl/CGI web page

The web is created by a Perl CGI script, shown below. As with many things, and with Perl in particular, there are many ways to do this. In this case my objective was to create a single web page that did all the work. In other words the processing is being done within the web page as it was being displayed.

You will need to download and install the WebSphere::MQTT::Client Perl library from CPAN.

My starting point was the example code supplied with the WebSphere::MQTT::Client. This showed me how to connect to the RSMB server, subscribe to a 'topic' and receive messages from that topic. The rest of the code is pure Perl CGI.

```
#!/usr/bin/perl -wT
# A small application to create a web page and display the temperature from remote
sensor.
# The DS18S20 is connected to an Arduino micro controller with an Ethernet shield.
# The Arduino runs a MQTT client. This sends messages to a RSBM broker on metphoto.
# This application subscribes to the topic published by the RSMB and displays the
results.
# Version 0.2 10/01/10. Mark E Taylor.

use CGI ":standard";
use WebSphere::MQTT::Client;
use Date::Format;
use Data::Dumper;
use CGI::Carp qw(warningsToBrowser fatalsToBrowser);#Warn of errors from perspective of
caller.
$CGI::POST_MAX=1024 * 10;#Max 10K posts.
$CGI::DISABLE_UPLOADS = 1;#No uploads.

# Start of HTML, using the CGI module.
print header;
print start_html(-title=>'Arduino temperature sensor',
             -style=>{'src'=>'../css/ardtemp.css'},
             -author=>'info@metphoto.net',
             -meta=>{'keywords'=>'Arduino ethernet temperature
DS18S20','copyright'=>'&copy Mark E Taylor '.time2str("%Y",time)});

print h1("This page displays the temperature of network connected DS18S20 sensor"),hr;
my $display_temp = &arduino_temperature;
# my $LED_status = &LED_status;
my $date_format = "%H:%M:%S %A %o %B %Y."; # Saturday 2nd January 2010.
print p({-class=>'at_temp_text'},"The current temperature is: ${display_temp}c.");
print p({-class=>'at_date_text'},"Last updated at: ".time2str($date_format, time));
print p({-class=>'at_date_text'},"Subscribed to Really Small Message broker topic:
'$ardTempTopic'.");
```

```
print a({href=>'http://www.metphoto.net/PDF/Network%20connected%20temperature
%20sensor.pdf'},"Click here for a PDF document describing how this page was
created."),br,br;
print a({href=>'http://metphoto.homeunix.net/met_current/index.php'},"Click here for
CurrentCost power consumption for iPhone."),br,br;
print a({href=>'http://metphoto.homeunix.net'},"Back to the home page.");
print end_html; # End of HTML.

sub arduino_temperature{
$ardTempTopic = "arduinotemperature";

my $mqtt = new WebSphere::MQTT::Client(
       Hostname => 'server_name',
       Port => 1883,
       Debug => 0,
       );

# Connect to Really Small Message Broker on server.
my $conn_result = $mqtt->connect();
die "Failed to connect: $conn_result\r\n" if ($conn_result);

# Subscribe to topic.
my $temp_result = $mqtt->subscribe($ardTempTopic);

# Get messages from topic. Errors can be caught by eval { }
my @message = $mqtt->receivePub();
$mqtt->unsubscribe($ardTempTopic);
$mqtt->terminate();
return $message[1]; # The actual data is in element 1 - 0 is the topic and 2 is the
QOS.
}
```

The result

The result of all of this technology is a very expensive thermometer. See here: http://metphoto.homeunix.net/met_current/cgi-bin/ardtemp.pl

Receiving messages on the Arduino

As an after thought I added a section to the Arduino code that received messages from the RSMB. The code looks for a message containing an ASCII “1” or “0”. Depending on the message it turns an LED on or off. In addition the Arduino then publishes the status of the LED on a different MQTT topic.

Further thoughts

A few further thoughts to take this project further.

You could...

- Record the current temperature to mySQL or RRDT
- Do something when the temperature reaches a certain threshold:
  - Turn something on/off depending on the temperature - home automation
  - Send a message - email, SMS
  - Send a message back to the Arduino MQTT client. So the Arduino does something
- Get the Arduino to process incoming messages from the RSMB
- Stop/start sending the temperature by sending a specific message from the RSMB
- Change the scale from Celsius to Fahrenheit on the web page

Conclusion

The question I get often asked is “Why do this?”. Well, it is clearly great fun to undertake a small project and it seeing it successfully completed.

In recent weeks I learnt a lot about Linux, Apache, Perl, CGI and the Arduino. Also a little about MQTT. I am a geek and I am having a great time.

Thanks

Thanks goes to Andy Stanford-Clark and Nick O’Leary for all things MQTT.

Google for being a great teacher.

Internet links

Information	Link
Viglen MPC-L Linux PC - I do not believe these wonderful little PCs are being made any more.	http://www.viglen.co.uk/
Andy Stanford-Clark (creator of MQTT)	http://stanford-clark.com/
Nick O’Leary (MQTT client for Arduino)	http://knolleary.net/arduino-client-for-mqtt/
Perl module WebSphere::MQTT::Client	http://search.cpan.org/~njh/WebSphere-MQTT-Client-0.03/lib/WebSphere/MQTT/Client.pm
IBM alphaWorks (RSMB)	http://www.alphaworks.ibm.com/
SK Pang Electronics	http://www.skpang.co.uk/
Perl CGI information	http://perldoc.perl.org/CGI.html
DS18S20 information	http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2815
Arduino micro controller	http://arduino.cc/
MQ Telemetry Transport (MQTT) information	http://mqtt.org/

END OF DOCUMENT