Ronald Macmaster and Parth Adhia

EID: Rpm953 and Pda375

09/30/2016

EE 445L – Lab 4: Internet of Things

1.0: Objectives:

- Implement a system that connects to the internet via an IEEE 802.11 – **Wifi** module, CC3100
- Use DNS to convert name to IP address
- Configure a smart object that can retrieve data from a weather server using TCP
- Design a smart object that can store data onto an internet server using TCP

Throughout this lab, we will implement a system that connects to the internet (via an IEEE 802.11 wifi module), utilize a DNS to lookup IP addresses by name, configure a smart object that retrieves weather data from a server using TCP, and design a smart object that can store data onto an internet server using TCP. These objectives will require us to interface with a CC3100 module for wifi capability and learn about the TCP/IP protocol stack. Also, we will have to study the client-server architecture and the relationship between TCP, UDP, and network packets. Finally, we will have to understand the engineering design tradeoffs associated with different network interfacing methods, and we will examine the measurements of the protocols we have chosen to implement.

2.0 Hardware Design:

We have decided to sample the on-board temperature sensor, so there are no external hardware connections to show. The main board (TM4C123) is connected to a CC3100 chip and interfaced through a simple link driver.

3.0: Software Design:

Much of the processing is done throughout the main driver program and the simple link driver. Most of our required and additional functionality can be summarized by five basic modules — LCD graphics, time management, UART debugging, ADC Sampling, and a simple link for WIFI communication. The simple link interface has been abstracted due to it's large amount of complexity, and most of the TCP/IP stack is handled through the CC3100. Submitted along with this report are the interface and driver files (.h and .c) for our software modules.

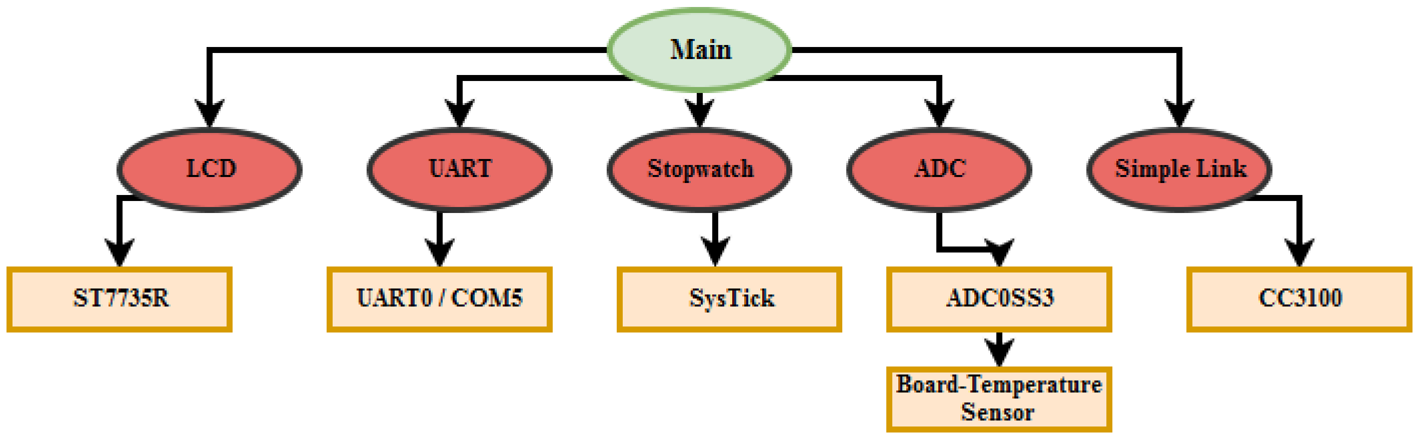

Figure 3.1: *Call graph for our smart temperature acquisition system. A main driver program manages hardware through secondary software modules. The board is connected to the CC3100 through a simple link interface.*

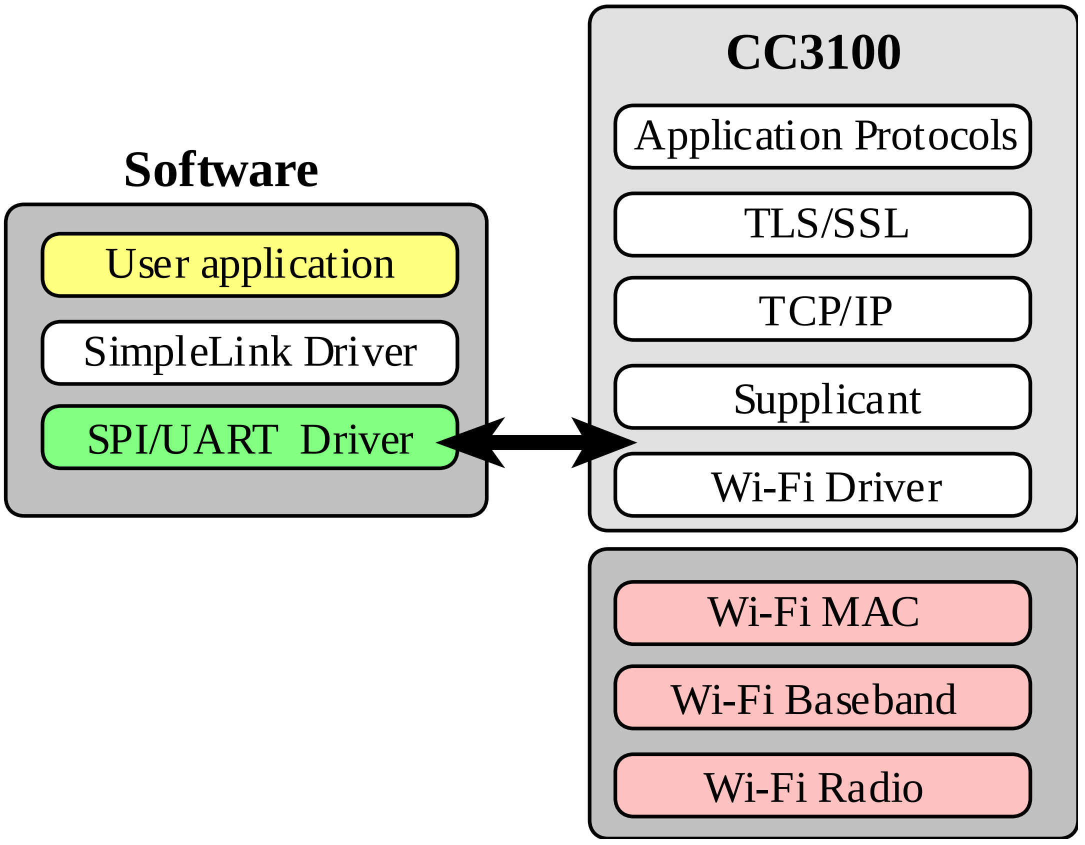

Figure 3.2: *Interaction between our main software stack and the CC3100. Most of the TCP/IP networking stack is abstracted through the Simple Link driver. The primary TM4C123 board doesn't have to devote resources to handling the protocols.*

4.0: Measurement Data:

Instrument the system so you can measure the time it takes to collect weather data from the internet and how long it takes to store data on the EE445L server. Run it 10 times and calculate the minimum, maximum, and average times and record the number of lost packets (my experience is that it either works and there are no lost packets, or it is broken and every packet is lost).

4.1: Percentage of lost packets.

Packet loss for this real-time system is **~0%** because the the application runs to completion. We are using TCP via a socket on the CC3100 simple link, so packets are usually dropped only if transmission outright fails. The sl_Send() and sl_Recv() methods we call operate on a TCP socket.

4.2: Transmissions to Openweathermap.org

Minimum: **18.3865 ms** Average: **26.285218 ms** Maximum: **41.86884 ms**

Table 4.2: GET Request measurements to api.openweathermap.org

Trial	Stopwatch Cycles	Transmission Time (ms)
0	1655857	33.11714
1	919325	18.3865
2	1166095	23.3219
3	1255021	25.10042
4	1281270	25.6254
5	1159975	23.1995
6	1280650	25.613
7	2093442	41.86884
8	1089817	21.79634
9	1241157	24.82314
min	919325	18.3865
avg	1314260.9	26.285218
max	2093442	41.86884

4.3: Transmissions to EE445L Server

Minimum: **15.40152 ms** Average: **25.658822 ms** Maximum: **38.9075 ms**

Table 4.3: GET Request for posts to embedded-systems-server.appspot.com

Trial	Stopwatch Cycles	Transmission Time (ms)
0	1281885	25.6377
1	774639	15.49278
2	1303004	26.06008
3	1654625	33.0925
4	1945375	38.9075
5	1317424	26.34848
6	1205452	24.10904
7	1482936	29.65872
8	770076	15.40152
9	1093995	21.8799
min	770076	15.40152
avg	1282941.1	25.658822
max	1945375	38.9075

5.0: Analysis and Discussion:

1) In the client server paradigm, explain the sequence of internet communications sent from client to server and from server to client as the client saves data on the server. Assume the client already is connected to the wifi AP and the client knows the IP address of the server.

Initially, the server has already opened a connection socket that is listening on a specific port (port 80 for our web server). First, the client opens a client socket and sends a request to connect to the server. The server socket receives the request and spins off a new thread to service the client request. The server does not service the request on the connection socket, but on a new server socket with access to the client socket. The client and server then exchange messages via socket (one in our case) to transfer data to and from the server.

2) What is the purpose of the DNS?

DNS is used to translate a local IP address to a physical IP address. Example: openweathermap.org is translated to 144.76.83.20 by a domain name server. Without the existence of DNS, humans would always have to remember the physical address of machines.

3) What is the difference between UDP and TCP communication? More specifically when should we use UDP and when should we use TCP?

While TCP communication provides reliable and ordered delivery of data through an established connection, UDP delivers packets in chunks through a connectionless protocol. We should use UDP when we do not require a reliable data stream and want to emphasize reduced latency over reliability. We should use TCP for applications that require a high amount of reliability. As an example, use TCP for file transfer applications and use UDP for media streaming.