**EE345L – Lab 4: Internet of Things**

Ty Winkler and Jeremiah Bartlett

02/24/16

1. **OBJECTIVE**

The objective of this lab was to introduce us to wifi communications using the TM4C123 microcontroller. In this lab we used a CC3100Boost to connect to openweathermap.org and retrieve a TCP payload containing the weather information of Austin, Texas. We then parsed this payload for Austin’s temperature which was then printed to the LCD board. We then sent a TCP packet to embsysmooc.appspot.com containing the voltage acquired through the on board ADC.

1. **ANALYSIS AND DISCUSSION**
   1. **Communication Sequence**

First the client creates a socket and opens it so that it can send a TCP packet to the server. The client then sends a TCP packet to the server and waits for a reply of the requested information. Upon getting a reply the client then closes the socket.

* 1. **DNS**

Domain Name Servers allow clients to call upon internet domain names and re-routs them to the respective IP Address. For example, a user could enter google.com and using dns would automatically get re-routed to the IP Address 216.58.192.46.

* 1. **UDP vs TCP**

Universal Datgram Protocol and Transmission Control Protocol are two types of IP traffic. UDP is the simpler version of TCP where a client will continuously send packets of data to the server regardless of the information being sent back and regardless of whether that information is arriving to the server. TCP requires the server to respond back to the client confirming that a connection has been established between the client and server. TCP is most often used in web browsing and communication between multiple computers. UDP is faster as it does not require a confirmation and is most commonly used in games and servers that collect data from a large number of users.

1. **Software & Hardware Design Solutions**
   1. **Software Design**
2. #define REQUEST "GET /data/2.5/weather?q=Austin%20Texas&units=metric&APPID=955db084d47dd332d35dafe1a3e2881e HTTP/1.1\r\nUser-Agent: Keil\r\nHost:api.openweathermap.org\r\nAccept: \*/\*\r\n\r\n"
3. **int** main(**void**){int32\_t retVal;  SlSecParams\_t secParams;
4. **char** \*pConfig = NULL; **INT32** ASize = 0; SlSockAddrIn\_t  Addr;
5. ADC0\_InitSWTriggerSeq3\_Ch9();         // allow time to finish activating
6. initClk();        // PLL 50 MHz
7. Output\_On();
8. UART\_Init();      // Send data to PC, 115200 bps
9. Timer1\_Init();
10. LED\_Init();       // initialize LaunchPad I/O
11. UARTprintf("Weather App\n");
12. retVal = configureSimpleLinkToDefaultState(pConfig); // set policies
13. **if**(retVal < 0)Crash(4000000);
14. retVal = sl\_Start(0, pConfig, 0);
15. **if**((retVal < 0) || (ROLE\_STA != retVal) ) Crash(8000000);
16. secParams.Key = PASSKEY;
17. secParams.KeyLen = strlen(PASSKEY);
18. secParams.Type = SEC\_TYPE; // OPEN, WPA, or WEP
19. sl\_WlanConnect(SSID\_NAME, strlen(SSID\_NAME), 0, &secParams, 0);
20. **while**((0 == (g\_Status&CONNECTED)) || (0 == (g\_Status&IP\_AQUIRED))){
21. \_SlNonOsMainLoopTask();
22. }
23. UARTprintf("Connected\n");
24. **while**(1){
25. **int** i = 0;
26. **while**(i < 10){
27. **int** sendc = 0;
28. strcpy(HostName,"openweathermap.org");
29. retVal = sl\_NetAppDnsGetHostByName(HostName,
30. strlen(HostName),&DestinationIP, SL\_AF\_INET);
31. **if**(retVal == 0){
32. Addr.sin\_family = SL\_AF\_INET;
33. Addr.sin\_port = sl\_Htons(80);
34. Addr.sin\_addr.s\_addr = sl\_Htonl(DestinationIP);// IP to big endian
35. ASize = **sizeof**(SlSockAddrIn\_t);
36. SockID = sl\_Socket(SL\_AF\_INET,SL\_SOCK\_STREAM, 0);
37. **if**( SockID >= 0 ){
38. retVal = sl\_Connect(SockID, ( SlSockAddr\_t \*)&Addr, ASize);
39. }
40. **if**((SockID >= 0)&&(retVal >= 0)){
41. strcpy(SendBuff,REQUEST);
42. sl\_Send(SockID, SendBuff, strlen(SendBuff), 0);// Send the HTTP
43. sl\_Recv(SockID, Recvbuff, MAX\_RECV\_BUFF\_SIZE, 0);// Receive resp
44. sl\_Close(SockID);
45. LED\_GreenOn();
46. UARTprintf("\r\n\r\n");
47. UARTprintf(Recvbuff);  UARTprintf("\r\n");
48. }
49. }
50. ST7735\_OutUDec(sendc);
51. ST7735\_OutString("\n");
52. i++;
53. }
54. //while(Board\_Input()==0){}; // wait for touch
55. LED\_GreenOff();
56. //Temp Part e
57. getTemp(Recvbuff);
58. ST7735\_OutChar('\n');
59. ST7735\_OutChar('e');
60. ST7735\_OutChar('m');
61. ST7735\_OutChar('p');
62. ST7735\_OutChar(' ');
63. ST7735\_OutChar('=');
64. ST7735\_OutChar(' ');
65. **for**(**int** i = 0; i < 5; i++){
66. ST7735\_OutChar(myArray[i]);
67. }
68. ST7735\_OutChar('\n');
70. //ADC Part f
71. ADC0\_SAC\_R = ADC\_SAC\_AVG\_64X;    //enable 64 times average before obtaining
72. **int** voltage = ADC0\_InSeq3();
73. ST7735\_OutString("Voltage~");
74. ST7735\_sDecOut3(voltage);
76. **char**\* voltageString;
77. **char** voltageStringNum[5];
78. sprintf(voltageStringNum, "%.1d.%.3d", voltage/1000, voltage%1000);
79. //ST7735\_OutString(voltageStringNum);
81. **char**\* sendString;
82. **char** str1[173] = "GET /query?city=Austin%20Texas&id=Ty%20Winkler%20Jeremiah%20Bartlett&greet=Voltage%3D";
83. strcat(str1, voltageStringNum);
84. strcat(str1, "V&edxcode=8086 HTTP/1.1\r\nUser-Agent: Keil\r\nHost: embsysmooc.appspot.com\r\n\r\n");

87. strcpy(HostName,"embsysmooc.appspot.com");
88. retVal = sl\_NetAppDnsGetHostByName(HostName,
89. strlen(HostName),&DestinationIP, SL\_AF\_INET);
90. **if**(retVal == 0){
91. Addr.sin\_family = SL\_AF\_INET;
92. Addr.sin\_port = sl\_Htons(80);
93. Addr.sin\_addr.s\_addr = sl\_Htonl(DestinationIP);// IP to big endian
94. ASize = **sizeof**(SlSockAddrIn\_t);
95. SockID = sl\_Socket(SL\_AF\_INET,SL\_SOCK\_STREAM, 0);
96. **if**( SockID >= 0 ){
97. retVal = sl\_Connect(SockID, ( SlSockAddr\_t \*)&Addr, ASize);
98. }
99. **if**((SockID >= 0)&&(retVal >= 0)){
100. strcpy(SendBuff, str1);
101. count = 0;
102. sl\_Send(SockID, SendBuff, strlen(SendBuff), 0);// Send the
103. sl\_Recv(SockID, Recvbuff, MAX\_RECV\_BUFF\_SIZE, 0);// Receive
104. sl\_Close(SockID);
105. LED\_GreenOn();
106. UARTprintf("\r\n\r\n");
107. //ST7735\_OutString(Recvbuff);
108. UARTprintf("\r\n");
109. }
110. }
111. **while**(1);
112. }
113. }
     1. **Measurement Data**

Percent of lost packets using TCP

0%

Minimum, maximum, and average times from 10 transmissions to openweathermap.org

|  |  |  |  |
| --- | --- | --- | --- |
| **Seconds** |  | **Minimum Time** | |
| 9.27 |  | 6.74 | s |
| 9.37 |  |  |  |
| 13.54 |  | **Maximum Time** | |
| 6.83 |  | 16.73 | s |
| 6.74 |  |  |  |
| 10.33 |  | **Average Time** | |
| 11.47 |  | 10.52 | s |
| 9.67 |  |  |  |
| 16.73 |  |  |  |
| 11.28 |  |  |  |

Minimum, maximum, and average times from 10 transmissions to EE445L server

|  |  |  |  |
| --- | --- | --- | --- |
| **Seconds** |  | **Minimum Time** | |
| 3.86 |  | 3.57 | s |
| 3.57 |  |  |  |
| 3.99 |  | **Maximum Time** | |
| 6.69 |  | 6.69 | s |
| 4.00 |  |  |  |
| 3.82 |  | **Average Time** | |
|  |  | 4.32 | s |