Shivam Desai
Joseph Torres
Professor Wilken
06/06/19

# EEC 172: Lab 5 Report

## Introduction

The purpose of this lab is to investigate the use of the RESTful (representational state transfer) API to a "thing" to the internet (in the IoT sense) and preserve its *shadow*; the RESTful API is implemented using HTTP GET and POST methods. For our purposes, the thing is our CC3200 Launchpad, and the internet service is Amazon Web Services (AWS) in parts I and II, and If This Then That (IFTTT) in part III. We also examine the use of AWS rules and IFTTT's triggers and actions to receive Launchpad information and send texts (SMS) and emails, respectively. Finally, this lab also incorporates the use of keys and certificates for establishing an encrypted connection.

## Description and Implementation

The first part of the project, we practiced storing the "state" of our Launchpad on AWS, where the state was simply the hard coded string "It's a sunny day!". We used the North Virginia AWS server for this lab. As part of preparing our Launchpad for connection with AWS, we generated our certificates and keys, changed them using Open SSL to the .der format, used UniFlash to flash them to the board, attached them to our "thing" on AWS, and granted it permission to AWS IoT functions. Then we were ready to begin to update and retrieve our device shadow via the REST API endpoint. This also necessitated the creation of a policy to define what actions were possible. The states are stored on AWS using JSON. Once on the server having used POST, we could modify the state online, and then call use GET to find that the string had changed to "It's a rainy day."

In part II, we borrow the DTMF (Goertzel) decoding implementation used in Lab 4. Again, we decode DTMF tones by defusing the superposition of each tone to find which composite frequencies are most prominent. From the decodings we form a message, a string that we can then post and preserve to the AWS server as a shadow, and later retrieve. Indeed, this time, the retrieving process calls for AWS to send a text to a specific phone number with the message parsed out of the JSON using SQL. We were able to simply use the same "thing" as in part I, to avoid having to create new certificates.

In part III, we again used the DTMF decodings to form a message on the CC3200 Launchpad. Instead of creating a thing and shadow on AWS, we used the free web based service If This Then That to create an applet whose trigger was a post to a URL (webhook) and whose trigger was to send an email with the message (again parsed from JSON) that was originally transcribed on the CC3200 Launchpad.

Our code builds off the SSL Demo code provided in SSL_REST_API_AWS.zip on Canvas. This meant that we had to make several changes to the code for our purposes, including to common.h, pinmux.c, the board through Uniflash for certificate/key downloading, and to main.c. In main.c, we had to use our own AWS endpoint and add both get and post requests. In common.h, we had to alter the SSID_NAME, SECURITY_TYPE, SECURITY KEY, and so on as specified. On AWS, we had to create a thing and register a policy. Of course, to use DMTF in part II, we had to import our code from lab 4, mostly unchanged. Other aspects of part II were completed online using our AWS account to create an SNS topic and rule to send a push notification after receiving an post to update the shadow. The part III section was similar, except that the server references had to be changed to reflect the use of IFTTT instead of AWS, and our IFTTT account was set up for a webhook to send an email with the message composed on the Launchpad with DTMF.

## Discussion of Issues and Solutions

We struggled initially with the conversion of the generated certificates to the .der format, as we were unable to locate OpenSSL on the lab machines. Instead, we emailed ourselves the certificates and completed the conversion on a laptop (having downloaded OpenSSL. It took us some time to learn how to successfully manipulate the AWS and IFTTT interfaces to set them up to preserve, update, and transmit shadows. Getting the demo code to compile was difficult as we needed to explicitly include or re-included several libraries. Having copied and pasting the server name to our code, we ran into the known issue of incorrect copying of certain characters, which was resolved after we tried typing the name out character by character. Initially, while trying to connect to WiFi outside the lab, our program would hang while trying to connect. Flashing the service pack onto the board helped us solve this and we were thus able to connect to any access point outside the lab. We ran into an issue where we kept getting a “Bad request” error every time we attempted to post our HTTP request. To solve this, we used trial and error to make sure we were using the right format and ended up using the one that worked.

## Conclusion

In this lab, we explored implementing the RESTful API through the use of GET and POST methods to update, preserve, and retrieve the shadow of a thing connected to the internet. We found that no matter how the information of a message was created on the Launchpad, we could post the information to both AWS and IFTTT to save its state. We could then use the wealth of features on those web services to transmit the state back to the board, or to a phone over the SMS, or over email.