Contents

1 Coding Evaluation

1.1 Instructions

This is a coding challenge and a few questions to see how you approach and work through problems. The end result should be a git repo you can share with us through github (or other git "forges"), email, or some other means. Email frank@radiatorlabs.com when you're done and we'll figure out how to share your work, sometimes gmail makes it difficult. Your code should accomplish the task, but beyond that there are no right answers. The goal is to get an idea of your thought process. If you're unsure of a definite answer to a given question, list the pros and cons of the possible answers you're considering.

1.2 Coding Challenge

One important task we often find ourselves doing is uploading new firmware over-the-air to devices that are deployed in the field. Many (all) of these devices are very low-powered microcontrollers, so standard methods of uploading firmware don't work. Instead, a given firmware image needs to be parceled out, piece-by-piece, in chunks of 20 bytes or less. Once the firmware image has been uploaded, we need to check the integrity of it by calculating its checksum. This is accomplished by taking modulo 256 of the sum of all of the bytes in the firmware image. If the checksum of the new image on the device matches that of the image we sent, we know we can safely reboot the embedded device into its new firmware.

Imagine such a device is accessible only through a REST endpoint. Data is sent to the device in the payload of a POST request, and the request's response contains the device's response. To simplify things, each device can receive only two types of messages: payload chunks, and requests for the checksum of the firmware loaded so far.

The protocol is:

- POST, payload: `CHUNK: <base64-encoded-firmware-chunk>`

Responds with `OK` if the chunk was written successfully, `ERROR PROCESSING CONTENTS` if it wasn't

- POST, payload: `CHECKSUM`

  Responds with `Checksum: 0x<checksum-in-hex>`

- Any other requests result in a response of `Bad request`

We've included a minimal node.js server program that implements this protocol and mocks this over-the-air firmware update process. You can run it by typing `node app.js` (with nodejs installed).

Write a CLI program that sends a `.hex` (Intel Hex format) file to this node.js application using the protocol described above. `.hex` is a text format, so it will need to be parsed to extract the data to send. An example `.hex` file has been included.

1.3 Questions

- Describe how you would accomplish updating 100+ devices with the same firmware.
- Imagine the devices respond to messages over a different channel than an HTTP response. For example, imagine the server responds to every valid message to a device with HTTP 200, and the device's actual response arrives asynchronously over a websocket. What architecture would you use?
- In addition to updating the firmware for 100+ devices, imagine each device takes ~30 seconds to respond to each message. Would this change anything?
- Imagine that in addition to performing firmware updates to devices over a REST endpoint, you also need to communicate with devices over other protocols like MQTT, CoAP, or a custom protocol over TCP. How could your design accomodate this?