Panic Attack Monitoring in a Blockchain World

Andrew Stoycos (astoycos@bu.edu)
Caroline Ekchian (cekchian@bu.edu)
Eric Li (ericli21@bu.edu)

Introduction/Project Goal

In the United States today anxiety disorders affect over 40 million adults, making it the most common Mental Illness in the country. Of the the 40 million afflicted only 36.9% actually ever receive any sort of treatment or counseling. By using some readily accessible technologies, and modern software solutions, we aim to create a medical device which will tackle this problem head on.

Blockchain is a revolutionary software tool that is becoming increasingly popular in the medical community. Specifically, it is a decentralized network used for storing and sharing sensitive medical records and information across healthcare professionals, pharmacies and health insurance providers. Using a FRDM board as the sensing mechanism our goal is to create a medical device that will detect whether or not an individual is experiencing a panic attack and then store this data on the blockchain.

We use a combination of different types of collected data in order to determine if an individual is experiencing a panic attack. This data as well as the results will be stored on the blockchain, a secure and decentralized form of data storage. The custom blockchain network can then be accessed by patients and medical professionals in order to see how often a panic attack is occurring as well as to observe some basic medical data, such as heart rate.
Ultimately this information will allow the user to keep track of the frequency and severity of anxiety attacks, which will then aid a primary care physician in the the creation of a customized treatment plan.

While it is very useful for this data to be accessible by multiple parties, it is important that this information be stored securely and reliably. Blockchain solves all the security risks associated with centralized data storage by acting as a distributed database that can continually add, update, and share its information. They use hashing and proof of work as well as a peer-to-peer network to create a consensus and verify which blocks in the blockchain are valid. Also, blockchain removes the need for a trusted third party allowing the registered users to access or add data based on the individual user's security clearance.

Project Implementation

Figure 1 is a block diagram of how our project was implemented. We used the FRDM board to read in data from a temperature and humidity sensor (Si7020). This sensor communicated with

the FRDM board via the serial communication protocol I2C. The FRDM board is running Zephyr OS. Zephyr OS is being used to run a script that continuously reads in the data from the sensor, simulate vibration sensor data using a random number generator, and process these values to evaluate whether or not the user is experiencing a panic attack. The script determines whether or not the user is experiencing a panic attack by comparing the sensor values with pre-defined baseline values. The script looks at how much the values that are read in from the sensor and generated from the random number generator deviate from the baseline values. If two out of the three values (humidity, vibration, or temperature) are a certain percentage greater than the baseline values, then a panic attack is considered to have occurred. This information is printed out every minute and can be viewed via the serial terminal running on the laptop. The FRDM board is connected to the laptop using a USB cable. In addition to the data being printed out on the serial terminal, it is being saved to a log file. Every time a new set of data is being printed out to the terminal, it is also being printed out to the log file. There is a Java script that is running on the laptop produces a new data file every time a new line is being printed out to the terminal and subsequently added to the log file. GO.main starts the blockchain and everytime a new text file is created, it pushes it to IPFS, retrevies it's IPFS hash, and pushes this hash along with the filename to the blockchain.

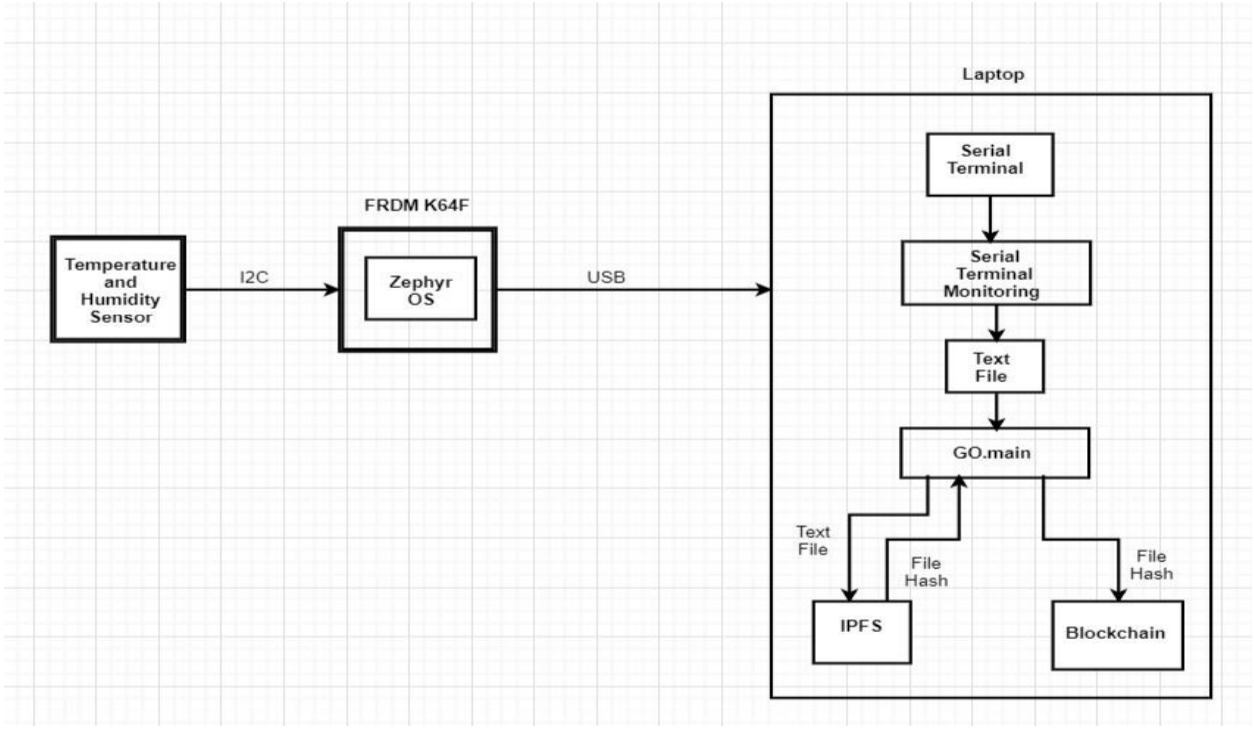

Fig. 1 Block Diagram of Project Implementation

This implementation deviates from the implementation that we had originally planned on. For example, in our project proposal we stated that we were planning to read data from 3 different sensors (two I2C sensors, one ADC sensor) using the FRDM board and that we were then

going to use software from AnyLedger to enable communication between the FRDM board and the blockchain. While the main goal and idea of our project remained the same, as we progressed, we ran into a number of issues that required us to readjust our original design implementation. These adjustments are described in the following paragraphs.

As previously stated, we had originally planned on using three different sensors to determine if someone was having a panic attack. The three sensors that we were planning on using were a heart rate monitor, a vibration sensor, and a temperature/humidity sensor. Both the heart rate monitor and temperature/humidity sensors had a I2C communication interface, while the vibration sensor was intended to be connected to the onboard ADC on the FRDM board. However, in our final implementation we ended up using only the temperature/humidity sensor. We did attempt to get the vibration sensor and the heart rate monitor to run on the FRDM board, however we ran into some problems when trying to configure Zephyr OS to enable the functionality needed to communicate with them.

In order to configure the peripherals on the Zephyr OS properly, there are a number of steps that need to be taken. Some of the configuration files that the Zephyr OS repository uses to build the OS for the target platform (in our case, the FRDM-K64f) needed to be changed in order to enable the different peripherals. The files that had to be altered were frdm_k64f_defconfig and Kconfig.defconfig. These two files were both located in the zephyr\boards\arm\frdm_k64f\ folder. This folder contains the files that are used to build a binary that is specifically suited for the target board. These files contain information that enables different peripherals through configuring the ports and pin muxing. By altering these files, we were successfully able to read and write to the temperature and humidity sensor using the I2C_0 interface. Additionally, we were able to enable the ADC_1 interface. However, attempting to perform continuous reads from the ADC interface was causing the board to auto-reboot after a few reads. This was indicative of a possible memory related issue as a result of incorrect use of the ADC interface. After thorough review of the Zephyr OS and FRDM board documentation, this issue was unable to be resolved. Therefore, instead of reading from the vibration sensor, we are simulating vibration sensor readings using the random number generator that is available through Zephyr OS. The random number generator does require additional setup through the configuration files, similar to the setup for the I2C peripheral. Trying to build I2C_1 produced a build error about an undeclared variable. This error is shown in Figure 2 below. There is a similar variable that is defined for I2C_0, however this variable seems to be missing for the I2C_1 interface. Based on reviewing the various configuration files that contain similar variables for other interfaces, it seems that it is possible that the I2C_1 interface for the FRDM board is not set up to be configured and supported by Zephyr OS.

Fig. 2 I2C_1 Build Error

In order to send medical data to a blockchain through the FRDM board, we looked into using a set of blockchain projects called AnyLedger. The AnyLedger team worked on a blockchain wallet implementation similar to what we were planning to make, because it utilized Zephyr OS to connect sensors to an nRF board and sent the data sampled by the sensors to an ethereum node directly from the board. The tools in the repository were important for us to be able to send panic sensor data directly to a blockchain. The AnyLedger wallet implementation, along with Zephyr OS, is built with certain dependencies (compilers and CMake). Due to a variety of issues, we could not build the AnyLedger wallet for the Zephyr OS on the FRDM board. One problem we ran into was that our compilers and cross-compilers, and other dependencies (Python libraries, etc.) were not compatible for the building process. According to the issues listed on the github repository, other people ran into similar problems when building. Because we ran into several errors during the building process without a proper solution, we were forced to abandon using AnyLedger for the project.

Because AnyLedger could not be built properly, we decided to send data to the blockchain through a computer terminal. Instead of sending data to the blockchain directly from the board, we created print statements from the FRDM board containing the sensor data. Through a serial port, we can see this data on a terminal such as Teraterm or ZOC. In order to automate the process of handling new lines of data, we put the data that is outputted on the terminal into a log plaintext file. Originally we had planned to store this data with ethereum, since that’s what was used by AnyLedger, however it seemed too heavy of a blockchain so we decided to make our own. Before polling the data from the serial port we needed start the truly decentralized peer 2 peer powered blockchain. The custom blockchain which was implemented had the following structure.

```
> ([]main.Block) (len=25 cap=34) {
 (main.Block) {
  Index: (int) 0,
  Timestamp: (string) (len=51) "2019-04-30 23:43:22.280195 -0400 EDT m=+0.002775035",
  fileName: (string) "",
  fileHash: (string) "",
  Hash: (string) (len=64) "5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9",
  PrevHash: (string) ""
 },
```

The hash was generated with a custom SHA-256 function which hashed all of the fields within the block. The main.go script handled blockchain initialization and also gave instructions on how to connect to the chain from other terminals. Go is a great language for blockchain implementations since it supports goroutines making concurrent execution simpler then through the use of threads. A Java program is then used to poll the serial data log text file for any new data outputs (every 1-2 seconds). When the Java program sees a new data packet in the log it writes it to a text file and saves it to an internal directory, medicalData. The go script in main.go monitors this directory so that whenever a new file is created by the java script it completes two tasks. Firstly it pushes the file to the IPFS, an advanced distributed file system, and then pushes the file's hash (returned by the IPFS) and its name to the blockchain. Now that this data is secured in the blockchain anyone connected to the network can download any of the data files with an "ipfs get filehash" command. IPFS is crucial in this design since in large blockchains are extremely inadequate when it comes to storing large file or amounts of data. IPFS plays so nicely with blockchain since it needs only the hash to be pushed to the blockchain in order for other users to access it.

Project Relevance to Course Material

This course has covered many topics, and our project relates a lot to the cryptography and real time operating system modules. The blockchain component of our project utilizes SHA-256 hashing in order to keep the sampled medical data secure enough so that it cannot be tampered with. It is also important to understand concepts such as decentralized systems in order to decide how to set up the blockchain, and who can access the specific medical data. The project also related a lot to real time operating systems, as we used the FRDM board in order to sample data from sensors to a computer. We had to understand serial ports and to connect the board to the sensors through I2C. The project also involved learning build and flash operations, as well as knowing the importance of sending medical data without a time delay failure. Though our specific case constantly samples data over a long time, we wanted to send data as fast as possible in real time to ensure that there can be quick alerts and subsequent responses when a panic attack is detected.

Resources

As mentioned above, the real time operating system that we used was the Zephyr OS. We were able to download the Zephyr OS repository and build it on our computers, as well as flash it onto

the FRDM board. Building the Zephyr OS required some dependencies to be installed, such as CMake, West, Ninja (for flashing), as well as the GNU ARM Toolchain.

The board is connected through a serial port, the lines of data (printk statements) could be accessed through a serial terminal, such as Teraterm or ZOC (depending on the OS).

Below are sets of links that were particularly useful to us.

The datasheet for the temperature and humidity sensor:
https://www.silabs.com/documents/public/data-sheets/Si7020-A20.pdf

Zephyr OS, entire website, but particularly:
https://docs.zephyrproject.org/latest/getting_started/index.html
https://docs.zephyrproject.org/latest/reference/kconfig/CONFIG_I2C_0.html
https://docs.zephyrproject.org/latest/guides/porting/board_porting.html#board-porting-guide
https://docs.zephyrproject.org/1.11.0/boards/arm/frdm_k64f/doc/frdm_k64f.html
https://docs.zephyrproject.org/latest/reference/peripherals/i2c.html
https://docs.zephyrproject.org/latest/reference/drivers/index.html#_CPPv36device
https://github.com/zephyrproject-rtos/zephyr

Ninja:
https://ninja-build.org/

FRDM-K64f:
https://os.mbed.com/platforms/FRDM-K64F/#board-pinout

AnyLedger repository:
https://github.com/AnyLedger/anyledger-wallet

GNU ARM tool chain:
https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm

InterPlanetary File System (IFPS):
https://ipfs.io/

LibP2P:
https://github.com/libp2p/go-libp2p

Blockchain
https://medium.com/@mycoralhealth/code-your-own-blockchain-in-less-than-200-lines-of-go-e296282bcffc
https://medium.com/@mycoralhealth/code-a-simple-p2p-blockchain-in-go-46662601f417

Conclusion

Prior to the project, none of us had any experience with real time operating systems or blockchain. Through this project we were able to gain knowledge and hands-on experience working in these areas. More specifically, it taught us about networking, decentralized systems, communication interfaces, hardware/software interaction, and the difficulties of building software reliably across different platforms. Working with Zephyr OS proved to be extremely challenging due to dense documentation and a wide variance of usable boards, however it forced us to be extremely methodical and focused in order to get it up and running. In addition, the failure with Zephyr OS allowed us to encounter a real world failure situation, and forced us to come up with a new solution to the problem. Also, we were able to take a deeper dive into current blockchain technology such as eutherum, P2P, IPFS along with many others. Not only did this provide ample implementation practices with new languages such as GO, but it also gave us a better perspective on the advantages and disadvantages to using blockchain technology in many real world situations. Overall this project provided us with numerous skills, both mental and practical, which will be extremely useful upon entering the workforce.