Flomosys: A Flood Monitoring System

Tai Groot and Behnam Dezfouli
Internet of Things Research Lab, Department of Computer Science and Engineering, Santa Clara University, USA
agroot@alumni.scu.edu, bdezfouli@scu.edu

***Abstract*—The expansion of the Internet of Things (IoT) has led to numerous innovations in the industry, including improvements to existing systems. Disaster prevention and monitoring systems are a prime example of such systems. Every year, there are significant and preventable financial losses, not to mention the safety hazards caused by floods. To warn people ahead of time, we can deploy low-power wireless sensor nodes to send readings across any terrain to a cloud platform, which can perform pattern analysis, prediction, and alert forwarding to anyone's cellular device. In this paper, we propose *Flomosys*, a low-cost, low-power, secure, scalable, reliable, and extensible IoT system for monitoring creek and river water levels. Although there are multiple competing solutions to help mitigate this problem, Flomosys fills a niche not covered by existing solutions. Flomosys can be built inexpensively with off-the-shelf components and scales across vast territories at a low cost per sensor node. In this paper, we present the design and implementation of this system as well as real-world test results.**

***Index Terms*—Disaster Monitoring, IoT, Wireless Communication, Security.**

I. Introduction

Floods are notoriously earth's most frequent and most destructive natural hazards. While flood damage counts in the billions worldwide, climate scientists have predicted that with the rise of global warming, flood events will only intensify in number and magnitude over time. Between 1995 and 2015, flooding affected 2.3 billion people and claimed 157,000 lives across the globe [1]. Since 2000, the US has spent over $107 billion on the damages caused by floods. California's San Francisco Bay Area, in particular, is facing a grim future. In 2017, San Jose suffered flooding of the Coyote Creek, which amounted to around $100 million in total damage and displaced 14,000 residents [2]. According to a team of scientists and economists who studied the global impacts of a rising sea level coupled with a growing economy and population, the Bay Area is not alone in its dismal future. Their reports predict that flood damage worldwide will cost up to $1 trillion per year by 2050.

In this paper, we present the design and development of a *low-power*, *reliable*, *low-cost*, *scalable*, *secure*, and *extensible* flood monitoring system, referred to as *Flomosys*. This system aims to monitor flood zones and report accurate, useful data to predict impending floods, so authorities are prepared to mitigate flooding disasters and allocate proper funds for response and recovery. A Flomosys system was installed and had been running in California's Bay Area since Summer 2019.

Figure 1 presents a high-level architectural diagram of Flomosys. The system is composed of three distinct components, Sensor Nodes (referred to as *Nodes*), the *Gateway*, and the *Cloud* platform. A full installation requires at least one instance of each component, but can be scaled to support many Nodes for each Gateway, and multiple Gateways per Cloud.

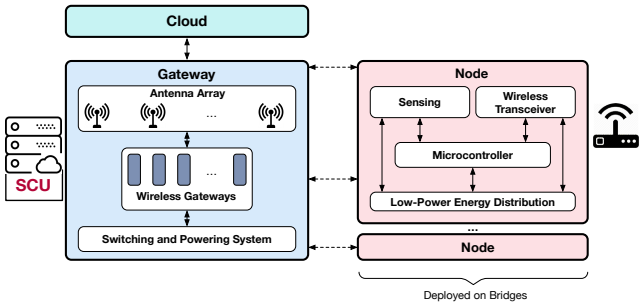


Fig. 1: Flomosys architecture. Each Node measures water level and transmits its data up to several miles away over a secure link to the Gateway. Each Gateway operates on multiple frequencies, performs duplicate packet detection, and forwards the received data to the Cloud. The Cloud provides an administration dashboard for system configuration, as well as data analysis.

Since Nodes typically rely on battery or energy harvesting as their energy source, we designed and developed a low-power circuit and software to minimize energy consumption. Nodes communicate with the Gateway using the Long Range (LoRa) wireless protocol [3]. On top of LoRa, we designed and implemented a new encryption and authentication mechanism called HMENC, which is suitable for low-bandwidth, low-power devices. This specialized encryption mechanism requires very little overhead and provides a built-in checksum feature making it more efficient than Advanced Encryption Standard (AES) or similar general-purpose ciphers, allowing Nodes to operate using very little energy. According to our power profiling results, each Node survives between 100 days and almost three years on a 2400 mAh battery, depending on water level sampling rate and wireless transmission variables (as we will explain in Section VI).

The Gateway design includes an array of receivers connected to directional antennas operating on different frequencies. This provides the Nodes with the diversity required to reach the Gateway in case of link unreliability. Also, the Gateway can be fully implemented using low-cost, single-board Linux devices such as the Raspberry Pi (RPi). If solar or limited energy resources are powering the Gateway, duty-cycling mechanisms are used to reduce energy consumption [4]. In our current deployment, the Gateway has been installed on the roof of an 11-level tall building belonging to Santa Clara University (SCU).

In addition to reporting water level values to interested

agencies, Flomosys can provide endangered residents with proper flood notice. The Flomosys platform is open-source and is designed to be assembled using Commercial Off-The-Shelf (COTS) components. The source code and hardware designs are published publicly and are freely available on the SIOTLAB's GitHub account[1]. With this low-cost design, strategic flood precaution is no longer out of reach for impoverished areas and third-world countries. Ultimately, the cost of implementation no longer outweighs the impending costs of flood-related damages.

The rest of this paper is organized as follows: In Section II, we discuss related work and how Flomosys fills a new place in the landscape of flood monitoring. In Section III, we present the system requirements. In Section IV, we detail the system architecture and how the different components move data from a Node to the web application. In Section V, we discuss the wireless communication protocols used and the HMENC encryption protocol. In Section VI, we examine the power profiling results and explain the theoretical system's lifetime. We conclude the paper in Section VII.

II. Background

Current flood monitoring systems deployed worldwide have provided us with information for what methods work when developing a reliable, feasible, and sustainable solution.

The Philippines, which is among the most flood-prone regions globally, has launched a program that predominantly utilizes Light Detection and Ranging (LIDAR) 3D terrain mapping and ultrasonic sensors. The LIDAR technology is coupled with computer-assisted analyses to pinpoint landslide-prone areas, while the ultrasonic sensors are utilized to monitor water levels. More recently, there has been an increased focus on deploying flood warning systems on urban streets. All of these sensors provide data that is analyzed and interpreted, then shared with the public via online flood information websites and mobile device applications. Another organization in the Philippines developed a real-time flood monitoring and early warning system to monitor the Cagayan River's water level using ultrasonic sensors [5]. This sensor system consists of an Arduino, ultrasonic sensors, a GSM module, web-monitoring software, and SMS-notification hooks to alert stakeholders and mitigate casualties related to flooding.

In 2008, various states in the United States experienced devastating floods [6], including Iowa [7]. This led the University of Iowa to create an Iowa Flood Center (IFC), which was also supported by the Tech State of Iowa. IFC has developed many inexpensive river stage[2] sensors mounted on bridges to span rivers and streams. They have developed a self-contained and compact Bridge-Mounted River Stage Sensor (BMRSS) [8] for monitoring small rivers and streams, consisting of an ultrasonic distance sensor, a solar panel, a GPS antenna, a cellular modem antenna, and a serial port. In operation, a BMRSS wakes periodically, measures its distance from the water surface, and transmits this information via the Internet to IFC servers. BMRSS unit consists of an ultrasonic distance sensor from Senix, designed for operation up to 15.2 m, and supporting the RS-485 interfaces. BMRSS enhances the output from flood forecasting models and can operate for several years unattended, even in harsh environments.

The flood monitoring system presented in [9] uses the Blynk platform as a medium of data transmission. This system uses two WiFi-based NodeMCU development boards connected through Blynk, a platform with iOS and Android applications to control an Arduino or RPi remotely via the internet [10]. This platform provides a digital dashboard to build a graphical user interface with custom widgets. One NodeMCU is placed at the flood area while the second one acts as the control unit. The transmitter unit consists of a NodeMCU, an ultrasonic sensor, and a display to show the current reading. The ultrasonic sensor data is sent to the Blynk application over WiFi, where it is stored in a database and can be transmitted to the internet if the second NodeMCU is connected. This system provides a short communication range, especially in urban areas, and deploying nodes on many bridges would require full sets of hardware at each installation.

III. Requirements

The major requirements of the system are *reliability*, *energy efficiency*, *scalability*, and *security*.

Concerning *reliability*, in the case of system failures, flood warnings may not reach endangered communities, which is arguably worse than having no system in place at all, as residents may rely on the system. Reliability must be addressed from multiple perspectives. First, the sampled data regarding water height must be accurate. Second, the software managing the operation of Nodes, Gateway, and Cloud platform must be bug-free and always be operational. Third, the Nodes must be able to transmit their data to the Gateway reliably. Fourth, the lifetime of the system components—especially the battery—must be long and predictable.

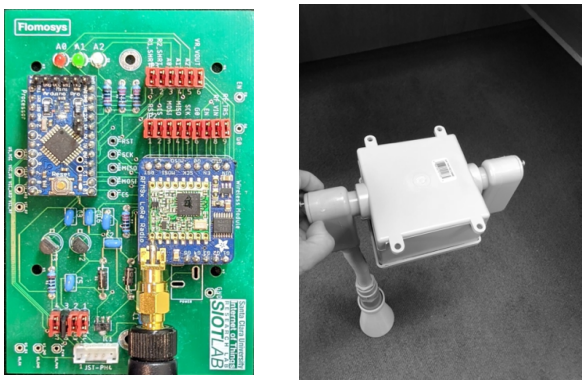
From the *energy efficiency* point of view, Nodes may rely on battery or solar harvested energy. Therefore, it is essential to minimize energy consumption to reduce the cost of an energy harvesting system [11]. Relying on renewable energy offers the additional benefit of reducing maintenance costs by eliminating the need to replace batteries frequently.

From the *scalability* point of view, the system must be easy to extend without deploying a large number of Gateways. Compared to the other flood-monitoring projects mentioned in Section II, Flomosys will achieve scalability by keeping Node costs low and shifting more expensive hardware and internet requirements to the Gateway. Because dedicated GSM modules and expensive recurring cellular plans are not required, deploying tens or hundreds of Nodes will cost much less than full systems, therefore dropping the average cost per monitored site for every Node installed and keeping total recurring costs very low. Also, the long communication range of the Nodes will reduce the number of Gateways required to cover an area.

Last but not least, *secure* communication between Nodes and Gateways is essential. In this regard, authenticity is necessary

[1] https://github.com/SIOTLAB/Flomosys

[2] Water level above a locally defined reference elevation



(a) (b)

Fig. 2: (a) Node's circuitry, and (b) packaging used for installation on bridges. Low-power circuit design and software improvements were employed to ensure minimum power consumption. The current draw of the circuitry during sleep mode is 30 μA.

to ensure a valid Node has generated the data received by a Gateway. Also, integrity is required to ensure the received data is tamper-free. Confidentiality is not essential because the data sampled by Nodes are not confidential.

IV. Architecture and System Overview

At a high level, the system architecture is composed of three components, as Figure 1 shows: one or multiple *Nodes*, one or multiple *Gateways*, and a cloud-based coordination and management system, which is simply referred to as the *Cloud*.

A. Nodes

Figure 2 shows a Node's circuitry and packaging. The two major responsibilities of a Node are collecting measurements and sending data packets to the Gateway. Each Node's sampling rate is adjusted based on several factors, including the last-measured water level and custom settings configured via the Cloud. In the current deployment, during normal operation (i.e., no hazard detected), the sampling interval is within the range of 15 to 30 minutes, depending on a per-Node configuration set in the firmware. As the water level increases, the Node's software reduces the sampling interval to ensure timely detection and response. These parameters are configurable via the Cloud platform.

The Node's circuitry includes a microcontroller, wireless transceiver, power distribution circuit, and interface to communicate with an ultrasonic sensor. To reduce the power consumption of the Nodes, software and hardware optimizations were made. These optimizations allow the Node to achieve a 30 μA current consumption during sleep mode. The microcontroller used is ATMEGA328P, which is available on Arduino Pro Mini boards [12]. The software improvements include enabling a low-power mode named *power-down* and disabling all unnecessary functionalities. Power-down is a power-saving configuration supported by ATMEGA328P, which disables everything except the watchdog timer, Two-Wire Interface (TWI) Address Match, and interrupts. The watchdog timer allows the Node to automatically wake from power-down mode without requiring an external clock or interrupt. The watchdog periodically generates an interrupt, which transitions the microcontroller from power-down mode to normal operation. If the wake-up is early, the MCU will revert to power-down mode until the next timer interrupt; if it is the time to wake up, the MCU remains in normal mode to sample the sensor and transmit data.

The energy efficiency of the circuitry is improved as follows. First, the regulator of the Arduino Pro Mini has been removed. Instead, we use the MCP1703 regulator [13], which has a low quiescent current of 2 μA. MCP1703 also allows input voltage within the range 2.7 to 16 V; thereby simplifying interfacing various energy sources such as solar energy harvesting systems. During sleep mode, the microcontroller cuts the power to the wireless transceiver and the ultrasonic sensor to avoid any current leakage. This is achieved by using two onboard transistors. The debugging LEDs (A0-A2) can also be fully disabled by jumpers. We also have removed the onboard LED of the Arduino Pro Mini.

1) Sensor: In addition to energy efficiency, the sensor must provide accurate water level measurements by reporting the Node's distance to the surface of the water. The three main types of distance sensors are ultrasound, LIDAR, and radar. Many LIDAR sensors' signals do not properly reflect off the water and instead penetrate the surface. Ultrasound modules are cheaper than their radar counterparts. Therefore, because ultrasound is affordable, reflects off the water properly, and is easy to source from a manufacturing standpoint compared to other technologies, it makes for the ideal distance sensor for Flomosys.

Passive ultrasound sensors send a set of very short pulses and detect the reflection of these signals. A transducer emits these sound pulses, which reflect off objects back to the sensor, and the transducer detects the echoes. The device can then measure the time it takes between the sending and receiving of these pulses. Using this time interval, we can determine the distance between the sensor and the object that has been detected.

2) Control Program: Algorithm 1 presents the pseudocode of the program controlling the Node's operation. When a Node is awake during its duty-cycle, it first samples using the ultrasonic sensor. Next, it uses HMENC to encrypt the packet, and enters the transmission loop. In the transmission loop, an encrypted packet is transmitted on channel 914, and the Node waits for an acknowledgment (ACK) packet from the Gateway. If no ACK is received before a timeout, the Node transmits again on channel 915. The Node will transmit on channel 916 if the transmission on channel 915 failed as well. This channel hopping ensures the Node does not violate Federal Communications Commission (FCC) regulations concerning LoRa communication. Once a packet is received, it is validated. If the packet is too long or short, it is immediately discarded. The retry counter is incremented, and the system attempts to transmit and receive it again, provided that it has not already retried too many times. If the packet is the right size, it is decrypted, and the checksum is calculated. Invalid checksums

Algorithm 1: Pseudocode of a Node's Controlling Program

```
while true do
    retries = 0;
    read_ultrasonic();
    while retries <MAX_PACKET_LISTENS OR time <TIMEOUT do
        encrypt_packet();
        transmit_packet();
        wait_for_ack_packet();
        if packet length not correct then
            drop packet;
            ++retries;
            continue;
        else
            if packet fails checksum then
                drop packet;
                ++retries;
                continue;
            else
                if msg_type == seed upgrade then
                    if seed packet has correct seq_no then
                        current_seed = message_seed;
                        encrypt_packet();
                        retries = 0;
                        continue;
                    else
                        ++retries;
                        continue;
                    end
                else
                    // data transmission is successful
                    break;
                end
            end
        end
    end
    increment_seed();
    sleep();
end
```

result in another transmission attempt, retries permitting. Valid checksums result in one of two situations: either the packet is a valid ACK, or it is a seed upgrade. If the packet is an ACK, the loop is finished, and the Node goes into a power-down sleep mode until the duty cycle is finished. Seed upgrades tell the Node to update their sequence number and attempt to transmit again, regardless of retries. As this case is infrequent, and the FCC regulations require an *average dwell time* on each channel to be lower than 400 ms, an immediate retry every few hundred packets in the rare event of Node reboot is not an issue.

B. Gateway

A Gateway is responsible for collecting sensor data readings from Nodes, acknowledging the reception of measurements, filtering duplicate receptions, and forwarding data to the Cloud. A Gateway implementation includes multiple Linux-based boards, such as the RPi, where each is connected to a wireless transceiver and antenna. To interface each RPi with a wireless transceiver, we have designed daughterboards that are compatible with RPi header pins. In our installation, the Gateway was installed on top of an 11-level building belonging to SCU with a steady power supply and reliable internet access.

C. Cloud

The Cloud Application runs on a managed server hosted on Linode, a hosting service providing a 99.9% uptime. The Cloud's application itself consists of a front-end and back-end. The back-end, implemented in Golang, accepts authenticated incoming connections from Gateways streaming measurements and serves requests for the data to the front-end. The front-end, written in VueJS, displays several graphs to the user, displaying readings for the previous week, month, and all-time data. An administrator console allows vendors such as water districts to register webhooks for events such as water level passing a certain threshold. All connections to and from the server are served using Transport Layer Security (TLS) with a certificate auto-renewed by LetsEncrypt, a free Secure Sockets Layer (SSL) Certificate Authority (CA). The domain registration is handled through Google Domains, which costs $12 USD per year. The total cost to run the cloud services for a year comes to less than $75 USD without requiring expertise with serverless functions.

V. Wireless Communication

This section discusses the importance and challenges of achieving long-range, secure wireless communication. We also present the details of the wireless communication and security protocol implemented in Flomosys.

A. Reliable, Long-Range Wireless Links

The choice of wireless technology has a significant impact in terms of reliability, energy efficiency, range, and security. We did not choose cellular communication for several reasons. First, cellular service may not be available in remote areas, and it is unreasonable to expect users to install new towers in the event of no coverage. Second, cellular communication requires a recurring, often expensive subscription. Although WiFi offers a high data rate and standardized security practices, its range is short and requires access point installation near ($\sim$100 m) each Node. The WiFi communication protocol also requires establishing and maintaining association to allow Nodes to communicate with WiFi access points. Both 802.15.4 and Bluetooth Low Energy (BLE) offer energy-efficient communication. However, similar to WiFi, the communication range of these technologies is short ($\sim$100 m) [3].

Given the above discussion, the application at hand requires a Low-Power Wide-Area Network (LPWAN) technology stack. Currently, the three leading LPWAN technologies are SigFox, Ingenu, and LoRa. We chose LoRa because it satisfies our system requirements in terms of message rate, range, and energy efficiency [14], [15]. LoRa, developed by SemTech, operates in the 915 MHz (North America), 829 MHz (Europe), and 433 MHz (Asia) bands. It has no duty cycle restrictions in North America [16], but a maximum dwell time of 400 ms per channel per 20-second interval is enforced. It can achieve a data rate of up to 300 Kbps, but typical data rates are between 300 bps and 27 Kbps, depending on the Coding Rate (CR) and Spreading Factor (SF) used, which must be adjusted based on distance. The LoRa module that we use is Semtech 1276 [17],

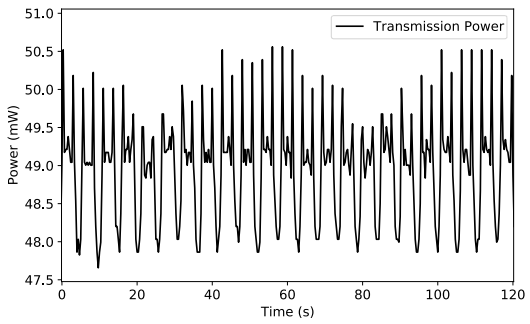


Fig. 3: The power required to transmit LoRa packets over time. Each spike corresponds to a packet transmission.

Fig. 4: Four directional antennas connected to the Gateway. The Gateway has been installed on top of a 11-level building in Santa Clara, California.

Fig. 5: The Gateway and Nodes use directional antennas to improve connectivity. This figure shows two sample locations where the Nodes were placed and their connectivity with the Gateway were tested.

which can operate in the 137 MHz to 1020 MHz frequency band. After performing several power profiling tests, such as the one represented in Figure 3, it was determined the total energy required to transmit a LoRa packet at +20 dBm with this module ranged from 0.702 Joules to 0.960 Joules, depending on the CR and SF.

Nodes, as well as the Gateway, use the same transceiver (Semtech 1276). The radio control software utilizes Radiohead, an open-source RF library licensed under GPLv2 [18]. Each Node is equipped with a directional antenna (mounted on a tall pole near its bridge) pointing toward the Gateway. The current Gateway deployment includes six RPi boards. These RPi boards operate on three different frequencies with two boards operating on each frequency, providing redundancy in case of failure. The three frequencies used are 914, 915, and 916 MHz. The transceivers use a SF of 1024 Chips-Per-Second (CPS), a 4/7 CR, and have a transmission power of +20 dBm. Each RPi couple is connected to a directional antenna, mounted on a mast installed on the roof of Swig Hall, the tallest building on the campus of SCU, at over 11 stories. Figure 4 shows antennas connected to the Gateway. These four antennas span the entire area we need to cover to measure the bridges along Coyote Creek and Guadalupe Creek. We observed reliable coverage extending just over 9 miles away from the Gateway. Figure 5 presents two sample locations communicating with the Flomosys Gateway.

B. Data Security and Integrity

Because the LoRa protocol does not provide any inbuilt encryption method for secure channel transmission, we have implemented a low-cost data authentication and integrity scheme.

LoRa facilitates the transmission of small packets over a long distance (i.e., several miles), and the reliability of packet transmission degrades as packet size increases. The data that we need to send is the water level, which is represented by a 4-byte variable. Important metadata includes a client ID number to determine which Node is transmitting the reading, and a sequence number, so the Gateway can detect duplicate readings. Duplicate transmissions occur when a Node sends a packet, the Gateway receives it and sends back an ACK, but the ACK is lost; therefore, the Node retransmits the packet. This is demonstrated in Figure 6.

We also added a 2-byte checksum for data validation. After including all the metadata, the message size is 16 bytes. Figure 7(a) shows the packet format. Such a small packet size is not effectively encrypted with modern encryption algorithms, as they require extra bytes for padding. Typically, using AES-128 would require us to transmit a ciphertext that is a multiple of 16 bytes, which means we are unnecessarily increasing packet size and reducing the chance of successful transmission. Additionally, there is a hard limit to the proper transmission length. In the US, the FCC mandated the maximum dwell time per channel must not exceed 400 ms over any 20 second period. However, to establish stable long-distance (around 9 miles) links, a high SF and CR must be used. These variables significantly increase the time spent transmitting per byte of

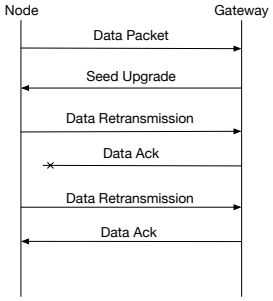


Fig. 6: This diagram illustrates three possible cases for message transmission. The Gateway rejects the first transmission for using an invalid `rand_seqno`, or seed. The Gateway responds with a seed upgrade. The Gateway receives the second transmission, but the ACK message is lost, therefore the Node retransmits the message. Finally, the third transmission is successful, and the Node receives the ACK sent by the Gateway.

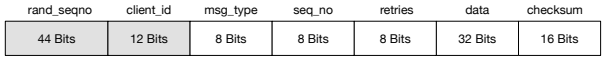
payload data. Sending 25 bytes or more with a CR of 4/7 and a SF of 10 results in an illegal transmission rate. Importantly, if the explicit header and Cyclic Redundancy Check (CRC) are used, there are only 19 available bytes for the payload section (see Figure 7a) of the packet. Similarly, Europe has duty-cycle limitations per channel. As a result, the longer the message is sent, the more infrequent messages must be sent to prevent breaking regulations.

Since extra bytes in the total packet size must be avoided to improve transmission reliability, we implemented our encryption and authentication solution for small data payloads, backed by SHA-256.

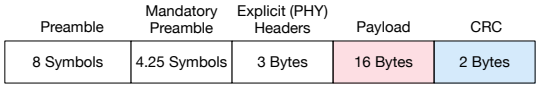
C. HMENC: Low-Power Encryption

HMENC, the encryption solution implemented in Flomosys, uses Hash-based Message Authentication Code (HMAC) and is inspired by the HMAC-based One-time Password algorithm and Time-based One-time Password algorithm (HOTP/TOTP) authentication schemes widely in use today for 2-factor authentication. The distinction between authentication and confidentiality is essential, as generally, HTOP/TOTP is only defined for authentication and does not meet the criteria for safe confidentiality. Additionally, HMENC has not been certified, and although it offers sufficient data confidentiality for Flomosys use case, the HMENC is better suited to ensure integrity and authenticity. Notably, the name HMENC is not an acronym, but rather a hybrid between HMAC and Encryption. It is important to emphasize that the data payload, in this case, is tiny, and only due to this small size does our particular implementation work. As such, HMENC is only defined for small data payloads, such as those under 64 Bytes. Figure 7 presents the HMENC message format.

The encryption algorithm is presented in Algorithm 2. Both the Gateway and the Node are preloaded with a shared 32-byte encryption key. Additionally, the Gateway maintains a registry of client IDs that are unique and flashed into the Node firmware before deployment. The first time a Node is activated, the random number generator is seeded using a static input read from an unused analog pin. Note that it is not critical



(a) The packet generated by HMENC for float data types. Unencrypted data is filled in gray, and encrypted data is in white. The values in each field represents the number of bits used.



(b) The physical layer frame used in Flomosys consists of a 2-part preamble, explicit physical headers, the actual data payload, and finally a 2-byte CRC.

Fig. 7: (a) Message format. (b) Physical layer frame format.

Algorithm 2: Encryption and Integrity Algorithm

```
1 Function encrypt_packet(char message[7], char packet[16]):
2     char checksum[2], pad[9], buffer[9];
3     checksum ⇐ calc_checksum(checksum);
4     pad ⇐ gen_HMAC(rand_seqno);
5     buffer ⇐ message + checksum
6     ciphertext ⇐ buffer ⊕ pad;
7     packet ⇐ rand_seqno + ciphertext;
```

that this pseudorandom generator be cryptographically random. The random generator fills the first five and a half bytes in a seven-byte array with a pseudorandom sequence number, and the client ID is inserted into the last byte and a half. This seven-byte sequence is used as m, the message variable defined by HMAC, and the pre-shared key is used as the HMAC key.

Given

$$K' = \begin{cases} H(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

Given K, m, and H, HMAC is defined as follows:

$$HMAC(K, m) = H((K' \oplus opad) \| H((K' \oplus ipad) \| m)) \quad (1)$$

where H is a cryptographic hash function (such as SHA-256), K is the key, opad and ipad are magic numbers multiplied by the block size as defined in the HMAC specification, and m is a message, in this case, our *random_seqno* [19].

The HMAC algorithm is run, and the output is truncated to 9 bytes. This truncated hash is the pad. The message payload is four bytes, long enough to contain a float. The metadata included along with the payload consists of a byte for the message type (to allow for future extensibility), one byte for a sequence number, and one byte to hold the number of retries. At the end of the plaintext, the two bytes store a two-dimensional checksum value calculated from the rest of the plaintext to verify message authenticity after it is decrypted.

After the metadata, data and checksum are loaded into a buffer, the plaintext is XORed with the pad, and the *rand_seqno* is prepended to the packet. Lastly, the packet is handed off to the RadioHead LoRa functions for transmission.

Upon receiving a message, the Gateway first unpacks the client ID from the 6th and 7th bytes to verify the packet sender is a valid client. Next, the first 7 bytes of the packet are loaded

into another buffer and run through the HMAC function along with the pre-shared token to reproduce the pad. The remaining 9 bytes of the packet (the ciphertext) are XORed with the pad to reveal the plaintext. Then, the first 7 bytes of the plaintext are passed through the checksum function, and if the 2-byte checksum matches the checksum provided by the remaining 2 bytes of the plaintext, the packet is determined to be valid. The sequence number is compared to the last seen sequence number. If it is a repeated number, then the Gateway will ignore the data but still sends an ACK.

In order to prevent replay attacks, the Gateway keeps a list of the last-used `rand_bytes` sequences per Node. As `rand_bytes` sequences increase by a constant every message sent, only the last seen sequence per device needs to be remembered.

If a Node sends a new message with an old random seed (distinguishable from a retransmission thanks to the sequence number), the Gateway will respond with a seed upgrade packet. This packet uses the 6 bytes otherwise occupied by the float, sequence number, and retries counter to store the first 6 bytes of the rand_bytes the device should use next. The seventh byte is already known, as it is derived from the client ID. This leaves the second half of the 6th byte unused, as the client ID also occupies it. Instead, it is set to the value of the invoking packet's sequence number divided by 2 to help mitigate seed upgrade replay attacks. When a device receives this seed upgrade message, it compares the sequence number sent to the sequence number included in the packet. If it is determined to be valid, the Node updates the seed stored in memory to match what the Gateway sent, and repacks the original message with the new seed. Therefore, one retransmission is necessary before the data is accepted and processed by the Gateway.

The HMENC algorithm is very fast and requires very little energy overhead. Our system profiling shows that the HMENC encryption operation requires approximately 3.28 ms of computing time on a Node. To run these tests, the HMENC codebase was profiled. In Figure 8, results from these tests show the cyclical power consumption curve is smoother and shifted higher on the Y-axis, signifying higher energy consumption. Specifically, the baseline tests required 0.071 mJ per operation, and each needed HMENC encryption 0.299 mJ per operation. These results were collected using the EMPIOT board [20].

In the end, HMENC requires at least 16 bytes of ciphertext, like AES-128, but it offers several benefits over AES. HMENC offers an in-built checksum solution to prevent data corruption, and it allows us to reuse one key across all the devices as no two devices will ever use the same `rand_seqno` or client ID. HMENC allows us to reject packets received by one Node but intended for another without decrypting them first, providing speed and, therefore, power savings. Additionally, if we choose to add features to the device in the future and we must increase our packet size by one byte, we can send a 17-byte packet, whereas using AES-128 would require us to use at least 32 bytes, possibly more depending on how the Initialization Vector (IV) is generated.

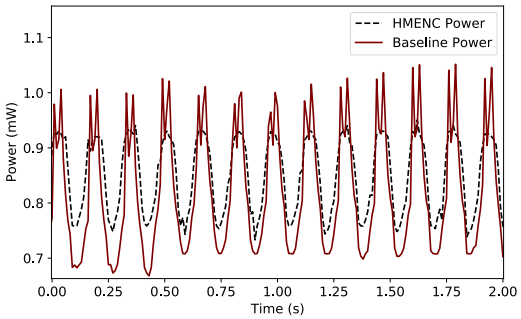


Fig. 8: The power consumption of a full HMENC function performing both memory allocation (on the stack) and encryption. We compare this full function with a baseline function only performing memory allocation.

VI. Lifetime Analysis

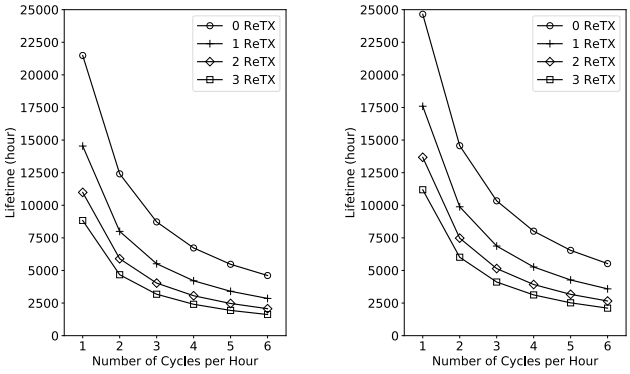
As mentioned in Section III, energy efficiency is a major concern for Flomosys. Additionally, energy efficiency is directly related to the reliability concerns for the system. As Flomosys is designed for remote locations, often without electricity, it must be feasible to power the system with renewable energy. For example, it should not drain a battery faster than the recharging rate of an inexpensive COTS solar panel. However, given the many variables involved in calculating expected energy harvesting rate from renewable sources, we have chosen to determine how long the system can survive powered by a 2400 mAh battery.

The energy cost of each operation, including data sample collection, packet encryption, and wireless transmission, have been recorded and analyzed using the EMPIOT board [20]. These measurements, as well as sleep power consumption, were also measured and verified using a high-precision digital multi-meter (Tektronix 7510). To determine the theoretical lifetime of Flomosys with different average retries and wireless configurations, we have used the following equation:

$$\begin{aligned} \text{lifetime} &= \frac{E_{bat}}{(E_u + (E_e + E_t) \times (R+1) + E_s) \times N} \\ &= \frac{3600 \times 2400 \times 10^{-3} \times 5}{(E_u + (E_e + E_t) \times (R+1) + E_s) \times N} \end{aligned} \tag{2}$$

where E_u, E_e, E_t, and E_s are the energy consumption for sampling with the ultrasonic sensor, encrypting a packet, transmitting a packet, and sleeping. E_{bat} is the available energy of the battery, N is the number of cycles (sampling and transmission) per hour, and R is the average number of retransmissions (ReTX) per cycle.

As mentioned in Section V-B, the most reliable transmission results were observed with a CR of 4/7 and a SF of 1024. However, as the distance between the Node and the Gateway reduces, a lower SF and CR may be acceptable. For this reason, two graphs have been generated: Figure 9(a), which displays the theoretical system lifetime for various retransmission rates with the CR and SF used at the longest distance (9 miles), and Figure 9(b) which displays the theoretical system lifetime for



(a) CR: 4/7; SF: 1024 CPS (b) CR: 4/5; SF: 128 CPS

Fig. 9: Theoretical lifetime assuming a 2400 mAh battery with no self-discharge for different average retransmissions, SF, and CR.

various retransmission rates with lower SF and CR values, to show maximum lifetime.

Finally, it is important to note these figures do not take into account battery self-discharge, which will significantly lower the system's lifetime. The particular battery we use has a lifespan of 5 years [21]. Consequently, the system remains operational for a substantial period before any key component replacement is necessary. Ultimately, the longevity of this component reduces the need for frequent maintenance.

VII. Conclusion

Multiple competing solutions offer comparable features and use similar methods to monitor flood zones. Unlike the other solutions, Flomosys can be built inexpensively with off-the-shelf components and scales across vast territories at a low cost per Node. In the worst case, Flomosys can continuously report data over several miles for 100 days using a 2400 mAh battery, or for 2.8 years under perfect conditions, marking it as very low-power and energy efficient. HMENC, the low-power encryption algorithm, secures all data sent between Nodes and Gateways, and the protocol supports adding new message types in case additional modules need to be added to the board. Together, these properties ensure Flomosys is a low-cost, low-power, secure, scalable, reliable, and extensible flood monitoring solution.

Acknowledgment

This project has been funded by the Santa Clara Valley Water District, the City of San Jose, Santa Clara University's Frugal Innovation Hub, and Cisco Systems. We would like to thank all the students that have contributed to the success of this project: Chelsey Li, JB Anderson, Michael Cannife, Navid Shaghaghi, Jaykumar Sheth, Peter Ferguson, Salma Abdel Magid, Francesco Petrini, and Zach Cameron. We would like to thank Dr. Winncy Du and her team at San Jose State University for working on the solar panel component. We also thank Semtech and MachineQ for providing us with the LoRa modules and development boards.

References

[1] M. Williams, "2.3 Billion People Affected by Flooding Disasters in 20 Years," https://www.channel4.com/news/factcheck/2-3-billion-people-affected-by-flooding-disasters-in-20-years, 2017.

[2] E. Deruy, "Coyote Creek Victims Sue a Year After Disastrous Flood," https://www.mercurynews.com/2018/02/17/a-year-after-devastating-coyote-creek-floods-some-victims-still-struggling/, 2018.

[3] A. Nikoukar, S. Raza, A. Poole, M. Güneş, and B. Dezfouli, "Low-power wireless for the internet of things: Standards and applications," *IEEE Access*, vol. 6, pp. 67 893–67 926, 2018.

[4] I. Amirtharaj, T. Groot, and B. Dezfouli, "Profiling and improving the duty-cycling performance of Linux-based IoT devices," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–29, 2019.

[5] J. Natividad and J. Mendez, "Flood monitoring and early warning system using ultrasonic sensor," in *IOP Conference Series: Materials Science and Engineering*, vol. 325, no. 1, 2018.

[6] J. Ambrose, "A watershed year: Anatomy of the iowa floods of 2008 and the 1,000-year flood: Destruction, loss, rescue, and redemption along the mississippi river," *The Annals of Iowa*, vol. 70, no. 2, pp. 191–194, 2011.

[7] Senix, "Water Level Sensors Provide Real-time Flood Warnings in Iowa," https://senix.com/2015/04/29/sensors-provide-iowa-flood-warnings/, 2015.

[8] A. Kruger, W. F. Krajewski, J. J. Niemeier, D. L. Ceynar, and R. Goska, "Bridge Mounted River Stage Sensors (BMRSS)," *Materials Science and Engineering*, vol. 4, 2016.

[9] N. A. Z. M. Noar and M. M. Kamal, "The development of smart flood monitoring system using ultrasonic sensor with blynk applications," in *IEEE ICSIMA*, 2017, pp. 1–6.

[10] "Democratizing the Internet of Things Blynk," http://www.blynk.io.

[11] B. Dezfouli, M. Radi, S. A. Razak, T. Hwee-Pink, and K. A. Bakar, "Modeling low-power wireless communications," *Journal of Network and Computer Applications*, vol. 51, pp. 102–126, 2015.

[12] ATMEL. (2020) ATmega328P Datasheet, 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash. [Online]. Available: https://cdn.sparkfun.com/assets/c/a/8/e/4/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

[13] MCP1703. [Online]. Available: https://www.microchip.com/wwwproducts/en/en530838

[14] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, "A study of LoRa: Long range & low power networks for the internet of things," *Sensors*, vol. 16, no. 9, p. 1466, 2016.

[15] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne, "Understanding the limits of LoRaWAN," *IEEE Communications magazine*, vol. 55, no. 9, pp. 34–40, 2017.

[16] M. Loy, R. Karingattil, and L. Williams, "ISM-band and short range device regulatory compliance overview," *Texas Instruments*, 2005.

[17] Semtech, "Semtech SX1276, 137 MHz to 1020 MHz Long Range Low Power Transceiver," 2017. [Online]. Available: https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1276#download-resources

[18] RH_RF95 Class Reference. [Online]. Available: https://www.airspayce.com/mikem/arduino/RadioHead/classRH__RF95.html

[19] NIST, "FIPS 198-1, The Keyed-Hash Message Authentication Code (HMAC)," Tech. Rep., Jul 2008. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf

[20] B. Dezfouli, I. Amirtharaj, and C.-C. Li, "EMPIOT: An energy measurement platform for wireless IoT devices," *Journal of Network and Computer Applications*, vol. 121, pp. 135 – 148, 2018.

[21] Tycon Solar. (2018) Outdoor Remote Power Systems, Tycon Solar RPDC Series Data Sheet. http://tyconsystems.com/documentation/Spec%20Sheets/RPDC%20RemotePro_Spec_Sheet.pdf.