IoT Device Firmware Update through Cloud Over-the-Air Interfaces

Rasmus Henriksen
henriksen1103@cs.ucla.edu
University of California, Los Angeles
Los Angeles, CA, USA

Zhi Li
mikelili@ucla.edu
University of California, Los Angeles
Los Angeles, CA, USA

Aditya Mishra
admish@ucla.edu
University of California, Los Angeles
Los Angeles, CA, USA

Turan Vural
turan@g.ucla.edu
University of California, Los Angeles
Los Angeles, CA, USA

ACM Reference Format:
Rasmus Henriksen, Zhi Li, Aditya Mishra, and Turan Vural. 2022. IoT Device Firmware Update through Cloud Over-the-Air Interfaces. In *CS 219 Spring 2023: Cloud Computing, March–June, 2023, Los Angeles, CA.* ACM, New York, NY, USA, 7 pages. https://doi.org/0.0/0.0

ABSTRACT

Firmware is an integral part of devices, which refers to the software that instructs the hardware to function and communicate with other software running on a device. Firmware updates are necessary to address security or performance issues during the lifetime of IoT (Internet of Things) devices. Over-the-Air (OTA) updates enable firmware updates to be completed remotely and securely, without the need to remove devices from their deployment.

This paper introduces a LoRa (Long Range) OTA procedure to enable cloud-based resources to initiate and complete updates to LoRa-connected IoT devices. This differs from existing OTA over LoRa solutions by allowing the LoRa gateway to simply push updates from upstream servers managing the deployment of devices. Upstream servers expose APIs to allow devices communicating over a TTS instance to initiate the OTA procedure. A PyCom FiPy was chosen as the IoT device on which to perform OTA.

Our code is included in a Github Repo [7] that others can build off of our initial LoRa OTA protocol.

1 INTRODUCTION

1.1 Over-the-Air Update (OTA)

Over-the-Air (OTA) updates is a wireless update or delivery of new software, firmware or other data to mobile devices such as smartphones, tablets, and Internet-of-Things (IoT) device. The growing popularity on mobile devices in industry and commerce has increased the need for keeping large deployments of mobile devices up-to-date with minimal overhead.

Currently, there are no high-level frameworks or APIs enabling cloud-orchestrated OTA over LoRaWAN. Such a framework would allow mission-critical security patching while maintaining the utility of existing IoT fleet management platforms. Thus, the primary goal of this project is to implement a LoRa OTA protocol that enables updates from the cloud to be pushed to a LoRa device.

1.1.1 WiFiOTA. A generalized OTA firmware update over WiFi follows these steps:

(1) Device connects with server using WiFi
(2) Device requests an update to be initiated
(3) Servers sends a manifest listing the files operations to be carried out
(4) Device receives manifest and executes file operations
   (a) In the case of an update or a new file, the device requests the file from the server
(5) Write firmware to OTA partition on device and set flag to boot from OTA
(6) Reset machine to complete the firmware update

A more detailed approach is highlighted in Figure 1 below.

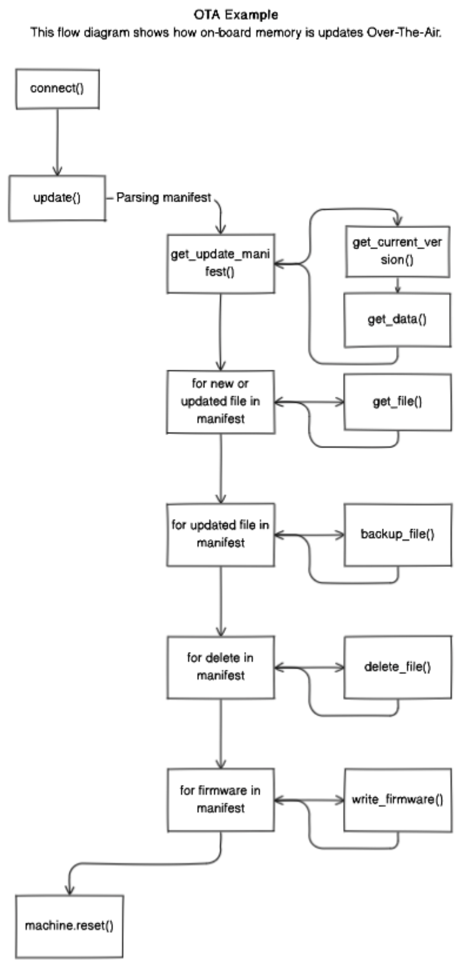


Figure 1: WiFi OTA procedure for PyCom devices

2 BACKGROUND

This section describes the technologies used to complete a cloud-supported OTA update over LoRa.

2.1 LoRa

LoRa (Long Range) is a communications protocol primarily used for Internet of Things (IoT) devices, able to communicate over ranges approximately 3 miles in urban areas, 10 miles in rural areas [1]. Downstream devices communicating over LoRa can operate for years, hibernating while not communicating and pushing a minimal amount of bytes upstream at a low power only when necessary.

LoRa uses chirp spread spectrum, using a modulation scheme that encodes data in chirps. The transmitted chirp signals cover a wide frequency range, increasing the signal-to-noise ratio (SNR), the likelihood of the signal to be received above the noise floor. This allows for LoRa signals to maintain reliability while being sent at a low power.

The high SNR for low-power transmissions contributes to LoRa's use case for IoT devices needing to operate for long periods of time without depleting on-board power sources [2]. These low power requirements and high range make it an attractive communication option for IoT deployments in agriculture and heavy industries. Furthermore, these environments are often unfriendly to weak radio signals and lack pre-existing network connectivity over other means (i.e. WiFi or cellular). Utilizing LoRa IoT devices has been shown to save farmer's money and increase yields by providing better crop monitoring. [3]

2.2 LoRaWAN

LoRaWAN is a MAC-layer communication protocol that provides standardized connectivity and security mechanisms for LoRa-based networks. It enables communication between IoT devices and LoRaWAN gateways while ensuring interoperability and scalability. LoRaWAN incorporates adaptive data rate (ADR) functionality, dynamically optimizing transmission parameters for reliable and efficient communication in varying signal conditions.

2.3 The Things Stack

The Things Stack is a LoRaWAN network server that manages applications, end devices, and LoRa gateways. End-devices are enrolled in TTS-managed applications which describes the authentication of such devices as well as the handling of uplink and downlink communication. Enrolled devices are authenticated by TTS when joining the LoRa network. Uplink messages can be forwarded further upstream from TTS to cloud applications via webhooks. Downlink communications can be forwarded from cloud applications to TTS. Communications in our experiments were unencrypted and encoded in base 64.

3 DESIGN

This section describes the transport through which an OTA update from cloud through LoRa occurs. It explains the functionality of the communication that takes place over LoRa and WiFi and its place in the architecture of our proof of concept. Figure 2 depicts the design at a high level.

Uplink communication describes communication from an IoT or *downstream* device to a cloud server. *Downlink* communication describes a communication from a cloud server to a downstream device.

3.1 LoRaWAN

Uplink communication is initiated by the downstream device and occurs over LoRa. Communication is received by a LoRa gateway, which forwards communication upstream. The FiPy end device operates as a node, the LoRaWAN protocol for wireless data transmission at a frequency of 914.9 MHz.

The communication within the LoRaWAN section is initiated when the FiPy end device undergoes a pre-scheduled firmware update event. To establish connectivity, the device initiates an Over-The-Air Activation (OTAA) process by transmitting a Join-request message to the LoRa gateway. Upon receiving the Join-request, the gateway responds with a Join-accept message, enabling the FiPy end device to create a LoRa socket for subsequent data transmission.

Inherent to the LoRaWAN protocol, the FiPy end device primarily operates in an uplink mode, where it periodically sends data to the gateway. After transmitting an uplink message, the device briefly listens for any incoming messages, including ACK (acknowledgment), NACK (negative acknowledgment), and downlink messages. This listening period ensures bidirectional communication while optimizing power resources.

The LoRa gateway, hosted on a dedicated server station, remains vigilant, continuously monitoring LoRa messages. It forwards all received LoRa communications to the TTS infrastructure using the Universal Serial Bus (USB) protocol. The gateway awaits responses from the TTS, including downlink messages, ACK, or NACK, which it then broadcasts to the registered devices.

3.2 IEEE 802.11

The standard IEEE 802.11 section of our design capitalizes on the pervasive internet protocol, facilitating seamless connectivity and interoperability. This section harnesses the TTS infrastructure and a Flask-based web application backend, enabling efficient data processing, management, and function handling.

Upon receiving a join request from a device, the TTS process the request, originating from the LoRa gateway, to verify the device's registration. A successful accept event triggers the TTS to invoke the Join-accept webhook function, employing an HTTP POST method to send the join-accept event to our Flask-based web application backend. The backend application, upon receiving the join-accept event, responds with a 200 code to confirm the successful registration.

The TTS triggers different webhook paths upon receiving various communications from devices. For instance, an uplink message from a LoRa device invokes the "uplink" webhook path, with the message subsequently forwarded to the backend via a POST request. The Flask-based web application backend, upon receiving the POST request, responds accordingly based on the path of the request and the content of the transmitted data.

In summary, the design of our LoRa OTA system encompasses the LoRaWAN and IEEE 802.11 sections, each carefully engineered to ensure seamless communication, robust connectivity, and effective data management. The LoRaWAN section facilitates wireless communication between the FiPy end device and the LoRa gateway, while the IEEE 802.11 section leverages the TTS infrastructure and Flask-based web application backend for efficient data processing and visualization. This synergistic integration empowers our LoRa OTA system with the ability to seamlessly transmit, manage, and analyze data, enabling a wide range of IoT applications.

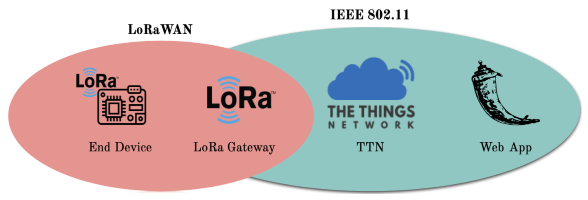


Figure 2: LoRa OTA Design

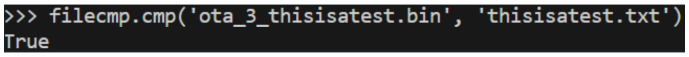
4 IMPLEMENTATION

We can support OTA on the cloud and have provided an API for IoT devices. We created the current LoRa firmware to support OTA over LoRa and we also wrote a demo application to initiate OTA.

So far, we have finished the creation of a LoRa OTA protocol that can send files from a WiFi Server to a LoRa microcontroller.

We have defined a LoRa OTA protocol that dictates what the device, gateway and server need to do to ensure successful firmware transfer. Moreover, since no preexisting, formalized OTA Protocol existed - our high-level ideas (i.e OTA Trigger Standardization, Firmware Indexing, etc.) that serve as the foundation of our work can be extended to OTA implementation with other Wireless Communication methods as well. It was necessary to experiment with the implementation of the LoRa OTA Protocol as a 1 to 1 mapping of the WiFi OTA Protocol was not possible (as discussed in 5.1).

We have defined the format for triggers, ACKs and NACKs as they did not exist before. The Wifi OTA code that served as a basis for our project used an arbitrary but known set of bytes as a trigger. We extend this idea to create ACKs and NACKs to enable reliable transmission of data in the LoRa-Gateway-Server Network.



```
>>> filecmp.cmp('ota_3_thisisatest.bin', 'thisisatest.txt')
True
```

Figure 3: A comparison of our transmitted file (left) with our original file (right) using Python's filecmp module

Because LoRa does not have retransmission, we defined a protocol to initiate retransmission when the TTS to device connection fails (which is unfortunately frequent). The retransmission is triggered in the event of a socket timeout (configured by user) in which case the LoRa device will transmit a NACK to the server. We have also explored the effect of different socket timeout values on total transmission times. Once the server receives the NACK it will retransmit the previously sent packet.

We have detailed how the server should segment and index files (the indexing process is currently evolving) to ensure that all OTA data is sent to LoRa Device. We have also experimented with the effect of varying segment sizes on total transmission time of a file.

We have been able to write firmware data to the OTA Firmware Partition on the FiPy microcontroller. This process requires writing the combined chunks to protected device memory.

We have finished the transfer of a 1.6 Kb, 16Kb and a 100 Kb firmware file from a server to our LoRa device. We were able to compare the received file on the device with the transmitted file on the server and see that they are identical. Figure 3 showcases one such comparison.

4.1 Feasibility of WiFi OTA Approach for LoRa OTA

We initially attempted to implement a 1 to 1 mapping of the WiFi OTA Process to our LoRa OTA implementation. However, this was not possible.

Due to LoRa's Unreliable Transmission and slow data rate, the sending of the manifest, application code and firmware was infeasible.

The LoRa Gateway had to be physically shared with other groups and additionally, the updates take hours without considering retransmissions. This made running our experiments (as discussed in the evaluation section) difficult.

Finally, there is high latency between device and the gateway, the gateway and TTS and with TTS and the Server. This made us have some departures from the WiFi OTA Approach as a whole.

4.2 Implementation Architecture

The diagram detailing our implementation architecture is shown in Figure 4. The left-half (anything in green) showcases parts of the solution that use LoRaWAN, the right-half (anything in red) showcases parts of the solution that use WiFi. In the middle (anything in purple) is TTS and the LoRa Gateway which permit LoRaWAN and WiFi to communicate.

Arrows pointing to the right indicate upstream communication, arrows pointing to the left indicate downstream communication. At a high level, communication is done between:

(1) The webserver and the LoRa Gateway/TTS
(2) The LoRa Gateway/TTS and the webhook/filesystem

The LoRa gateway/TTS[1] serves as the bridge between the webserver and webhooks + filesystem. In our experiment, we enrolled our end device in an application on TTS. The device sends a join request and authenticates using Over-the-Air-Authentication (OTAA).

The webserver transmits the firmware version, ACKs and NACKs, the webhooks transmit firmware length and segmented data packets. TTS then forwards the data from the webserver to the webhook and vice versa.

Uplink communication is initiated from the end device and sent to the TTS-managed LoRa gateway. We chose to use sockets to send data upstream. Lower-level on-board implementations are available if future projects wish to exercise more control over the board's transmissions. The LoRa gateway forwards the received packet over WiFi to TTS, where it can be processed by the TTS application. In our application's configuration, we have webhooks configured to be sent to a webserver on certain events that are triggered when the TTS receives a communication from the end device. We enabled two webhooks to be called: one on a join-accept event, and one on the receipt of any uplink communication from the device.

On the join-accept webhook, the webserver prepares the file to be transported to the device. This includes clearing of any caches or data structures used in previous transmissions and parsing the file into 63-byte chunks to be sent. A downlink transmission is sent to TTS to clear the scheduled transmission buffer.

Downlink transmissions are initiated from the webserver or TTS and are pushed over LoRa via TTS and the gateway to the end device. TTS allows for calls via a REST API to schedule downlink calls from applications further upstream. Downlink messages from upstream entities are pushed to a queue in TTS and are sent when possible.

[1]Although the gateway and TTS are two separate components, they gateway operation is abstracted by TTS's role as a network manager. When we refer to TTS in this section, we include the function of the LoRa gateway.

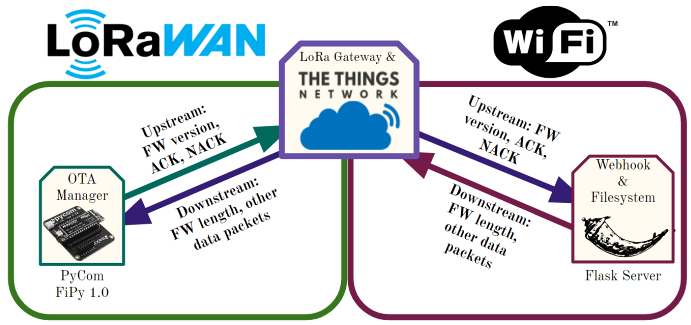

LoRaWAN
Wi Fi
LoRa Gateway &
THE THINGS NETWORK
Upstream: FW version, ACK, NACK
Downstream: FW length, other data packets
OTA Manager
PyCom FiPy 1.0
Upstream: FW version, ACK, NACK
Downstream: FW length, other data packets
Webhook & Filesystem
Flask Server

Figure 4: Solution architecture for LoRa OTA showcasing the interaction between the webserver, LoRa gateway and the webhook + filesystem

4.3 Implementation Details

The diagram showcasing the procedure for our LoRa OTA protocol is showcased in Figure 5.

(1) First Row: Connect FiPy to Gateway
  (a) The procedure for LoRa OTA starts at the top of the diagram where the the FiPy device connects to the gateway using LoRa Over the Air Authentication (OTAA). LoRa OTAA initiates the join procedure with a LoRaWAN network.
  (b) TTS sends a join-accept to the Server - at this point the device will be connected to the server. Additionally, the join-accept request triggers the server into segmenting the most recent firmware into chunks of fixed length.

(2) Second Row: FiPy Initiates OTA, Server indexes most recent Firmware File
  (a) The device sends the OTA Trigger.
  (b) From there, TTS will forward an uplink message containing the OTA Trigger to the Server.
  (c) The Server fetches the most recent Firmware (FW) File and segments it to known chunk sizes, the Server also gets the entire length of the FW.

(3) Third Row: Server Transmits FW Length
  (a) The server initiates a downlink push containing the FW Length.
  (b) The FiPy will receive the FW Length.

(4) Fourth Row: FiPy Acknowledges Length
  (a) The FiPy will either ACK or NACK the length message. A NACK occurs in the event that the socket times out.
    (i) **Note:** The FW Length is used by the FiPy to continuously receive and acknowledge Firmware chunks sent by the Server "FW Length" times.
  (b) The Server receives this ACK or NACK. If the FW Length message was NACKed, retransmission of FW Length occurs.
    (i) **Note:** Retransmission is initiated whenever a NACK is received by the Server.

(5) Fifth Row: Server Data Transfer
  (a) The server initiates a downlink push containing a segment of FW Data.
  (b) The FiPy receives the segment of FW Data.

(6) Sixth Row: FiPy Acknowledges the FW Data
  (a) FiPy wil either ACK or NACK the FW Data message. A NACK occurs in the event that the socket times out.
  (b) The Server receives this ACK or NACK. If the FW Data message was NACKed, retransmission of the FW Length occurs.

The process in the last two rows is repeated "FW Message Length" times the ensure the FiPy device receives the entire FW. Once FW Transmission is complete, all the received data is combined and written to the OTA partition on the microcontroller - completing the LoRa OTA process.

5 SETUP CONFIGURATIONS

5.1 LoRa Socket

(1) LoRaWAN Mode - US-compliant bandwidth, channels and frequencies. Zone: US915
(2) LoRa OTAA - Over the Air Authentication to join LoRaWAN Network
(3) Blocking Set - device waits for packets to be received
(4) Socket Timeout - **Varied for our experiments**. Set number of seconds for client to wait before initiating retransmission procedure.

5.2 Data Segmentation

(1) Done entirely on the server.
(2) Chunk Size - **Varied for our experiments**. The length of the FW Data in Bytes the server sends at a time.

6 RESULTS

We were able to transfer files from a webserver to a LoRa device. We experimented with tuning certain LoRa/Server Parameters. The summary of our results can be seen in Figure 6.

6.1 Experiments

We ran two experiments:

(1) The effect of changing the client-side socket-timeout value on the total elapsed transmission time. (Rows 1-3 in Figure 6)
(2) The effect of changing the server-side chunk-size (Segment Size) on the total elapsed transmission time. (Rows 4-5 in Figure 6)

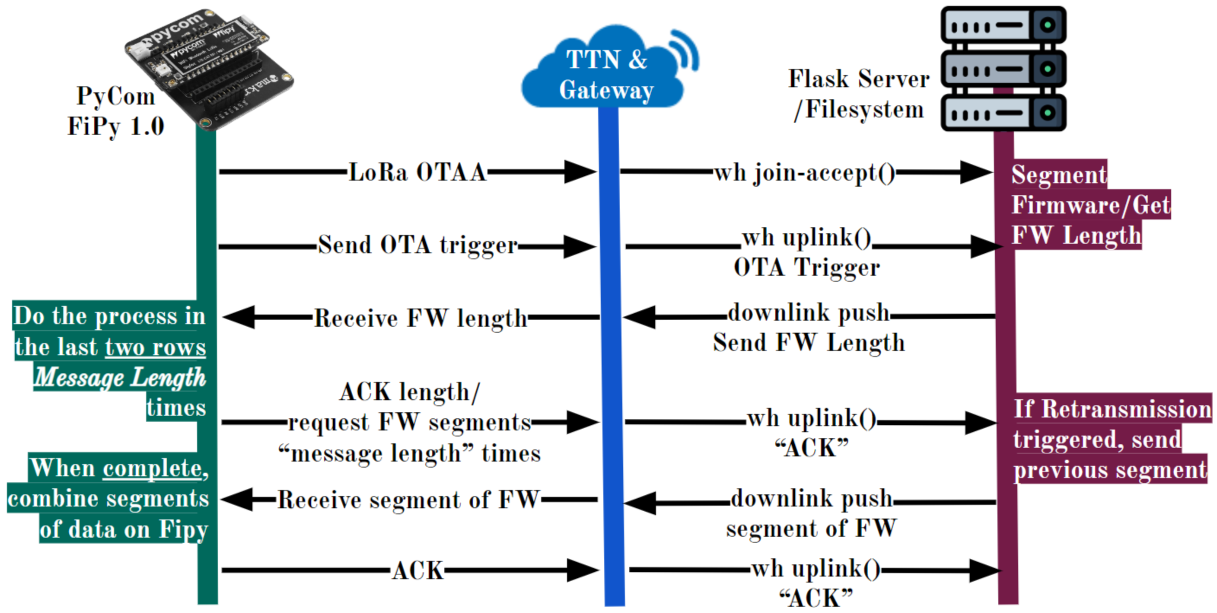
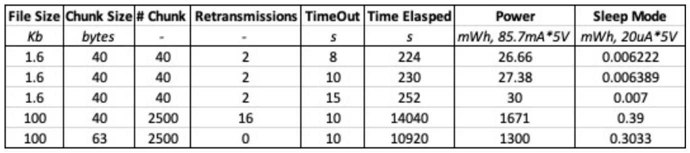

PyCom FiPy 1.0
TTN & Gateway
Flask Server /Filesystem
LoRa OTAA
wh join-accept()
Segment Firmware/Get FW Length
Send OTA trigger
wh uplink() OTA Trigger
Do the process in the last two rows Message Length times
Receive FW length
downlink push Send FW Length
ACK length/ request FW segments "message length" times
wh uplink() "ACK"
If Retransmission triggered, send previous segment
When complete, combine segments of data on Fipy
Receive segment of FW
downlink push segment of FW
ACK
wh uplink() "ACK"

Figure 5: Diagram showcasing the sequential procedure enabling a cloud push for OTA updates to a LoRa device

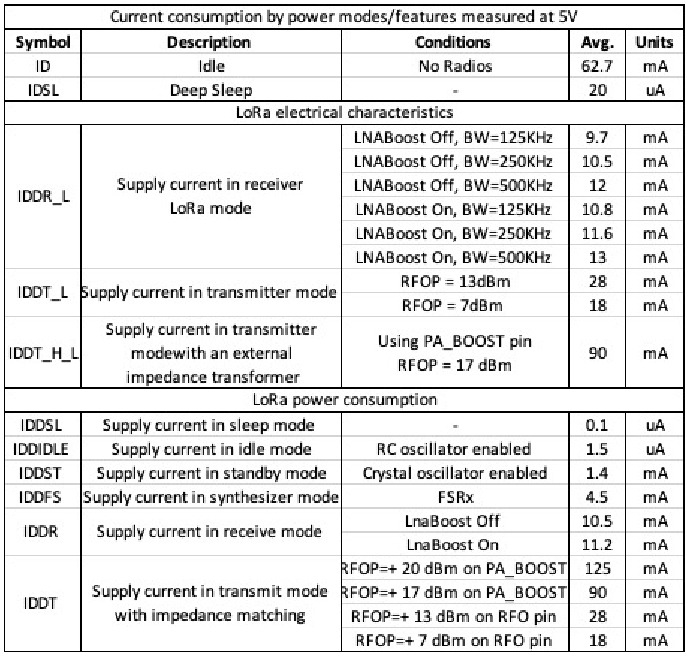


File Size	Chunk Size	# Chunk	Retransmissions	TimeOut	Time Elasped	Power	Sleep Mode
Kb	*bytes*	-	-	*s*	*s*	*mWh, 85.7mA*5V*	*mWh, 20uA*5V*
1.6	40	40	2	8	224	26.66	0.006222
1.6	40	40	2	10	230	27.38	0.006389
1.6	40	40	2	15	252	30	0.007
100	40	2500	16	10	14040	1671	0.39
100	63	2500	0	10	10920	1300	0.3033

Figure 6: Evaluation

We observed that increasing the client-side socket-timeout value did ultimately increase the total transmission time. We ultimately decided on a socket-timeout value of 10 seconds because the marginal increase in transmission time outweighed the potential retransmission that could result from a shorter socket-timeout value.

We observed that increasing the server-side chunk-size value significantly decreased the total transmission time. This is expected as there are fewer total FW packets to transmit from the Server to the Client. What is unexpected is the elimination of retransmissions by increasing the chunk-size. We expected an increase in chunk-size to increase retransmission - we are unsure of why this occurs.



Current consumption by power modes/features measured at 5V				
Symbol	**Description**	**Conditions**	**Avg.**	**Units**
ID	Idle	No Radios	62.7	mA
IDSL	Deep Sleep	-	20	uA
LoRa electrical characteristics				
IDDR_L	Supply current in receiver LoRa mode	LNABoost Off, BW=125KHz	9.7	mA
		LNABoost Off, BW=250KHz	10.5	mA
		LNABoost Off, BW=500KHz	12	mA
		LNABoost On, BW=125KHz	10.8	mA
		LNABoost On, BW=250KHz	11.6	mA
		LNABoost On, BW=500KHz	13	mA
IDDT_L	Supply current in transmitter mode	RFOP = 13dBm	28	mA
		RFOP = 7dBm	18	mA
IDDT_H_L	Supply current in transmitter modewith an external impedance transformer	Using PA_BOOST pin RFOP = 17 dBm	90	mA
LoRa power consumption				
IDDSL	Supply current in sleep mode	-	0.1	uA
IDDIDLE	Supply current in idle mode	RC oscillator enabled	1.5	uA
IDDST	Supply current in standby mode	Crystal oscillator enabled	1.4	mA
IDDFS	Supply current in synthesizer mode	FSRx	4.5	mA
IDDR	Supply current in receive mode	LnaBoost Off	10.5	mA
		LnaBoost On	11.2	mA
IDDT	Supply current in transmit mode with impedance matching	RFOP=+ 20 dBm on PA_BOOST	125	mA
		RFOP=+ 17 dBm on PA_BOOST	90	mA
		RFOP=+ 13 dBm on RFO pin	28	mA
		RFOP=+ 7 dBm on RFO pin	18	mA

Figure 7: Power Consumption

6.2 Power Consumption

Given that LoRa is meant to enable connectivity while maintaining low-power operation, it is worth examining the overhead of OTA updates. The limited maximum transmission unit (MTU) makes updates longer than WiFi or cellular updates with higher bandwidth. Thus, we examine the additional power consumption needed for the transmission of our test files. The results of our calculations can be found in Figure 6. Power consumption is taken from [4], the FiPy specsheet. To calculate power usage, we follow $TimeToUpdate * 5 * Current$, where TimeToUpdate is the time elapsed during the update, 5 volts is the operating voltage, and current is the current the usage of the device while updating or in sleep. To calculate current while operating, we took the idle current usage *ID* and added 23 mA, 23mA being an approximation of LoRa usage in IDDT. To calculate the sleep current usage, we use IDSL.

6.3 Persistent data

In certain scenarios, we witnessed the FiPy receive out of order messages, duplicated messages, and messages on startup that had been sent in a previous session. For example, we have witnessed the FiPy "receive" messages even before the webserver had been started. This erroneous behavior was a challenge to reliable delivery of uncorrupted data. We were not able to diagnose the issue, but given the behaviour of messages from previous sessions being incorrectly received by the board with no webserver running and no corresponding logs in TTS to indicate messages in the downlink queue being sent, we hypothesize that this is an issue with an onboard buffer not being cleared on startup.

To minimize this issue, we clear buffers where we can on startup and wait until the FiPy has a chance to clear any lingering messages before bringing the webserver online and performing the update. This approach was effective in ensuring the correct transmission and reception of data.

A more comprehensive understanding of the buffer's behavior and potential methods for clearing it would be beneficial. Further investigation and experimentation are required to verify or disprove our hypothesis and solve more reliable transmission.

7 NEXT STEPS

(1) Experiment with LoRa's confirmed mode. This would help us attain better reliable delivery.
(2) Include sequence numbers in the FW Segments that are transmitted from the Server. This would mitigate the FiPy Buffer Problem.
(3) Further testing with additional tuning of the settings we currently use. We attempted to do more testing but were unable to due to time constraints.
(4) Experiment with compression of the firmware. Possibly only transmit the diffs between FW files to minimize transmission time.
(5) Write formal documentation for a general OTA procedure. This does not exist and additional standardization would make it easier for others to implement OTA for other wireless protocols.

8 ACKNOWLEDGMENTS

We would like to express our thanks to our professor, Dr. Songwu Lu, for teaching us the fundamental concepts that allowed us to build the experimental solution that we did. Thank you to our advisor, Dr. Zhaowei Tan, who provided us with feedback and guidance as we learned the technology stack necessary to make our solution work.

We express our gratitude to Dr. Lu and Dr. Tan for the superb guidance on not just this project, but on this course as a whole. We thoroughly enjoyed this course. We learned valuable content that made us more interested in new connectivity technologies. Our interest would not have been possible without your engaging instruction.

9 REFERENCES

[1] "What Are Lora® and Lorawan®?" LoRa Developer Portal, lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/.

[2] Delgado, C. et al. 2020 "Battery-Less LoRaWAN Communications using Energy Harvesting: Modeling and Characterization" in IEEE Internet of Things Journal

[3] R. Sokullu, "LoRa Based Smart Agriculture Network," 2022 8th International Conference on Energy Efficiency and Agricultural Engineering (EE&AE), Ruse, Bulgaria, 2022, pp. 1-4, doi: 10.1109/EEAE53789.2022.9831210.

[4] Pycom, "Pycom Fipy Specsheets" Pycom Documentation, 2017, https://docs.pycom.io/gitbook/assets/specsheets/Pycom_002_Specsheets_FiPy_v2.pdf.

[5]Pycom, "Pycom Fipy PinOut", Pycom Documentation, 2017, https://docs.pycom.io/gitbook/assets/fipy-pinout.pdf.

[6]RS Components. "Pycom Fipy Spec." RS Online Documentation, 2023, https://docs.rs-online.com/77c6/0900766b815d0a8d.pdf.

[7] https://github.com/turanzv/CS219LoRaWANOTA