ABSTRACT
LoRa has emerged as one of the promising long-range and low-power wireless communication technologies for Internet of Things (IoT). With the massive deployment of LoRa networks, the ability to perform Firmware Update Over-The-Air (FUOTA) is becoming a necessity for unattended LoRa devices. LoRa Alliance has recently dedicated the specification for FUOTA, but the existing solution has several drawbacks, such as low energy efficiency, poor transmission reliability, and biased multicast grouping. In this paper, we propose a novel energy-efficient, reliable, and beamforming-assisted FUOTA system for LoRa networks named FLoRa, which is featured with several techniques, including delta scripting, channel coding, and beamforming. In particular, we first propose a novel joint differencing and compression algorithm to generate the delta script for processing gain, which unlocks the potential of incremental FUOTA in LoRa networks. Afterward, we design a concatenated channel coding scheme to enable reliable transmission against dynamic link quality. The proposed scheme uses a rateless code as outer code and an error detection code as inner code to achieve coding gain. Finally, we design a beamforming strategy to avoid biased multicast and compromised throughput for power gain. Experimental results on a 20-node testbed demonstrate that FLoRa improves network transmission reliability by up to 1.51× and energy efficiency by up to 2.65× compared with the existing solution in LoRaWAN.

CCS CONCEPTS
• Networks → Network protocols; • Computer systems organization → Embedded systems;

KEYWORDS
LoRa, Firmware Update Over-The-Air

1 INTRODUCTION
Recent years have witnessed the emergence of Low Power Wide Area Networks (LPWANs) as a novel paradigm of Internet of Things (IoT), which complements legacy wireless communication technologies in providing long-range and low-power communication. Among several LPWANs, such as LoRa, NB-IoT, and Sigfox, LoRa is one of the most prevalent choices in industry and research communities, currently being dedicated to 163 network operators across 177 countries globally [42].

The massive deployment of LoRa networks demands software upgrading with the latest standards, preventing security vulnerabilities, and device customization for specific purposes throughout their life cycle. Thus, the ability of Firmware Update Over-The-Air (FUOTA) is becoming a necessity for LoRa networks. FUOTA, also known as Over-The-Air Programming (OTAP), refers to remotely updating firmware images for embedded devices in an over-the-air manner without tinkering with the underlying hardware. Figure 1 illustrates a typical FUOTA process in LoRa networks. Instructed by the server, the gateway first issues a command packet to create a multicast group for FUOTA and subsequently starts transmitting firmware image fragments in the lossy channels to these LoRa nodes at the agreed time. The nodes open the receiving window accordingly, then reboot after the completion of reception and verification of the firmware. Finally, the gateway performs the update summary by collecting the status reported by these LoRa nodes.

LoRa Alliance has dedicated FUOTA specifications to standardize and refine this task [6]. However, our study shows that the current specification suffers from the following limitations. (i) Low
To improve energy efficiency, we propose a delta script algorithm to unlock the ability of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocks the potential of incremental FUOTA. Specifically, delta scripting refers to generating the delta script as the patch for processing gain, which unlocking the ability of incremental update, the channel coding scheme enhances the reliability and robustness of large-scale firmware image distribution, and the beamforming strategy further serves unicast users. We evaluate the performance of FloRa by conducting comprehensive benchmark experiments in a 20-node LoRa testbed. The results show that FloRa improves network transmission reliability efficiently.
Then the receiver applies the Fast Fourier Transform (FFT) on the signal. SF indicates the number of bits per chirp symbol can represent, ranging from 7 to 12. BW is typically 125 kHz, 250 kHz, and 500 kHz. CR, denoting the coding rate of the Forward Error Correction (FEC) mechanism, can be set to 1 to 4. Thus, the LoRa data rate DR is specified by

\[ DR = SF \times \frac{BW}{2SF} \times \frac{CR}{CR+4}. \] (3)

2.2 Firmware Update Over-The-Air
LoRaWAN is a data link layer specification built on top of LoRa, defining the typical star-topology network architecture and corresponding service specifications inclusive of FUOTA.

Network Architecture. Figure 4 illustrates a typical LoRaWAN network architecture, rendering a star topology inclusive of end devices, gateways, network servers, and application servers.

FUOTA. To enable FUOTA tasks in LoRa networks, LoRa Alliance has dedicated three FUOTA specifications [6], i.e., remote multicast setup, fragmentation data block transport, and clock synchronization. In brief, regarding uplink-oriented LoRa radios and duty cycle limitation, the remote multicast setup specification enables FUOTA downlink transmission on a group of LoRa devices. It contains two commands specifying device addresses and session keys in the multicast group, and the communication class (B or C) of LoRa devices. Concerning the large-size firmware image, the fragmentation data block transport specification defines the fragmentation session setup mechanisms, specifying parameters of firmware image size, fragment size, etc. It recommends that a Low-Density Parity-Check (LDPC) code is used for erasure correction; however, the LDPC code falls short of flexibility due to its fixed coding rate and requirement for pre-defining. Another requirement is clock synchronization, serving for agreeing on the start time to further minimize energy consumption during the process of FUOTA.

However, we reveal three fundamental problems when performing FUOTA tasks in practice. Due to the characteristics of low data rate, LoRa networks cannot afford the high energy overhead to receive the entire firmware image. Then, the deployment scenario of LoRa networks makes them suffer from transmission loss due to the dynamic and poor link quality. Additionally, multicast cannot ensure that all nodes complete FUOTA tasks. Thus, the low energy efficiency and success rate issues make the FUOTA task challenging.

3 SYSTEM OVERVIEW
Design Goals. The design of FLoRa is driven by the following considerations:

- Energy Efficiency. FUOTA is an energy-intensive task, while LoRa is a power-constrained technology. Regarding the large size
of the firmware image, we aim to unlock the potential of incremental FUOTA in LoRa networks, and design lightweight firmware decoding and reconstruction algorithms.

- **Transmission Reliability.** LoRa nodes need to construct the firmware image losslessly to reboot themselves. However, packet loss and symbol error are unavoidable to a varying degree in large-scale FUOTA downlink transmission, due to signal fading and interference. To solve this problem, channel coding is leveraged to detect or correct errors for reliable transmission.

- **Multicast Fairness.** Several multicast users with poor link quality may bring down the throughput of the multicast group. Therefore, we aim to assign these nodes in additionally created unicast groups and trade beamforming for a better link budget and strong signal strength.

- **System Compatibility.** The design of FLoRa should not be conflicted with the current LoRaWAN specification. Additionally, the hardware diversity of LoRa transceivers and the indeterminacy of the LoRa network also make it necessary for FLoRa to have the property of compatibility.

**System Overview.** Figure 5 illustrates the overview of FLoRa. When preparing a FUOTA task for the LoRa network, it is assumed that the gateway has been informed of the information of the corresponding LoRa nodes (e.g., identifiers and firmware versions). First, the gateway creates a multicast/unicast group by issuing a command packet specifying FUOTA parameters (e.g., firmware image size, fragment number, start time, and security key), and waits for the feedback from LoRa nodes. Afterward, the gateway constructs the delta script of the new and old firmware images by leveraging the suffix array for inter-firmware differencing and intra-firmware compression. Then, the gateway distributes the delta script to LoRa nodes after performing a concatenated coding scheme, where the fragments in the delta script are for outer rateless encoding followed by inner error detection encoding. Among them, a degree distribution function and a priority matrix are designed for reasonable and fair fragment choosing during the encoding process. The LoRa nodes receive and decode these packets, which are subsequently used to reconstruct the firmware image. After the firmware integrity check and security authentication, the nodes reboot themselves and then report the FUOTA status to the gateway for an update summary. For nodes that fail to reboot, FLoRa utilizes an explicit beam scanning method by sending probing packets every steering beamwidth to acquire azimuth information for the unicast group setup. The delta script is then distributed to the unicast groups accordingly.

4 SYSTEM DESIGN

4.1 Joint Delta Scripting Algorithm

As mentioned previously, the large-size firmware image needs to be distributed entirely to LoRa nodes during the FUOTA process. Thus, the energy-intensive and error-prone FUOTA task is challenging to LoRa technology due to its low data rate and constrained power requirement. In the field of reprogramming, incremental update has been widely used as an option to enhance energy efficiency [3], which mainly relies on generating the delta script between the new and old firmware images to be transmitted to IoT nodes. Then, nodes reconstruct the new firmware image using the stored old one. Unfortunately, although differences between the old and new firmware are generally not large, the modified code is relocated and shifted to other memory addresses, which may result in a disproportionate increase in the size of the generated delta script. To solve this problem, we first adopt an inter-firmware differencing algorithm [34] between the new and old firmware images to generate the delta script, then propose an intra-firmware compression algorithm to further minimize the size of the delta script (see Figure 6(a)).

4.1.1 Differencing Algorithm. When a FUOTA task is released, both the new and old firmware images can be acquired. The key idea of delta scripting is to use the differencing algorithm to find the common segments between these two firmware images, and inject the corresponding commands to encode the delta script. In FLoRa, we use a classic suffix array-based differencing algorithm from [34] to generate the delta script. Suffix arrays are an efficient data structure obtained by sorting all suffixes of a string. It can be used to minimize the size of the delta script, enable efficient reconstruction of the original firmware image, and further serve for the proposed compression algorithm. To make this paper more self-contained, we first succinctly describe the suffix array, then present the differencing algorithm.

**Suffix Array.** Suppose there is a string $S = c_0c_1...c_{n-1}$ of length $n$, $suffix[i]$ refers to the suffix starting from the $i^{th}$ character $c_i$ to the end character $c_{n-1}$, namely, $suffix[i] = c_1c_2...c_{n-1}$. Two arrays, suffix array $sa[n]$ and rank array $rk[n]$, are used as auxiliary tools. The suffix array $sa[i]$ is defined as the index character of the suffix ranked $i^{th}$ after sorting all suffixes of $S$ by dictionary order. The rank array $rk[i]$ denotes the rank of $suffix[i]$. Thus, it possesses the mathematical properties of $sa[rk[i]] = rk[sa[i]] = i$.

To construct the suffix array $sa[n]$ and rank array $rk[n]$ for the given string $S$, common solutions include plain, doubling [7], and
DC3 algorithms [23], possessing the time complexity of $O(n^2 \log n)$, $O(n \log n)$, and $O(n)$, respectively. Thus, we adopt the DC3 algorithm [23] for the suffix array construction. The idea of the algorithm is as follows: (1) dividing all suffixes of $S$ into two parts based on the result of their index mod 3, i.e., $A = \{\text{suffix}[i] \mid i \mod 3 \neq 0\}$ and $B = \{\text{suffix}[i] \mid i \mod 3 = 0\}$; (2) radix sorting for the first 3 characters of $A$; (3) radix sorting for $B$; (4) merging the results to construct the suffix array.

Afterward, the suffix array and rank array are used to construct the height array $\text{height}[n]$ of $S$, with which we can find the Longest Common Prefix (LCP) between the new and old firmware images to generate the delta script. We first define $\text{lcp}(i, j) (i < j)$ as the length of the LCP of $\text{suffix}[i]$ and $\text{suffix}[j]$. The height array $\text{height}[i]$ is then defined as the LCP between $\text{suffix}[sa[i]]$ and its predecessor $\text{suffix}[sa[i-1]]$, denoted as

$$\text{height}[i] = \begin{cases} \text{height}[0] = 0, & i = 0 \\ \text{height}[i] = \text{lcp}(\text{sa}[i], \text{sa}[i - 1]), & i = 1, 2, \ldots, n - 1. \end{cases}$$

In simple terms, the height array represents the LCPS of the two suffixes ranked next to each other. With the height array, $\text{lcp}(i, j)$ with arbitrary $i$ and $j$ can be calculated. It is revealed that if the values of the height array with varying indexes from $i + 1$ to $j$ are always greater than a certain number, this number is the length of the LCP, denoted as

$$\text{lcp}(i, j) = \min(\text{height}[i + 1], \ldots, \text{height}[j]) (i < j).$$

Delta Script Format. After finding LCPS by constructing the suffix arrays, we need to understand the firmware image format and then define the delta script. Generally, a firmware image is composed of many firmware records, where each firmware record is a binary/hexadecimal string with finite length, including start code, byte count, address, record type, data, and checksum [53].

To encode the delta script, we design two commands for inter-firmware differencing: KEEP and UPDATE. Specifically, the former is used to maintain the common segments between two firmware images, while the latter is used for recording different ones. Two commands would not conflict with the contents of the firmware image, defined as

$$\text{Command KEEP : } K < \text{entry}, \text{addr, offset} >,$$
$$\text{Command UPDATE : } U < \text{str} >.\text{ (6)}$$

For command KEEP, the entry indicates the index of the firmware record, addr and offset represent the starting address and length of the common segments, respectively. For command UPDATE, str indicates the different segments. For size saving and simplicity, this command can be replaced directly with str.

Differencing. Given the new and old firmware images composed of $\text{F}_{\text{new FR}}$ and $\text{F}_{\text{old FR}}$ firmware records, they are denoted as $\text{F}_{\text{new FR}}$ and $\text{F}_{\text{old FR}}$, respectively. For $i$th firmware records having the same address (i.e., entry) of two firmware images $\text{F}_{\text{new FR}}[i]$ and $\text{F}_{\text{old FR}}[i]$, a connection character ‘-’ is utilized to merge the two records, i.e., $S = \text{F}_{\text{old FR}}[i] + \ldots + \text{F}_{\text{new FR}}[i]$. Then, we use the aforementioned suffix array data structure to find the LCP of $S$. For those LCPs whose lengths are greater than the pre-defined threshold, we utilize the KEEP command to encode them in the delta script; otherwise, the UPDATE command is used. The threshold is set to 11, which is the minimum length of the replaced KEEP command.

4.1.2 Compression Algorithm. Due to the disproportionate increase in the size of the generated delta script resulting from firmware modification, we need to further decrease its size for the power-constrained LoRa networks. Most previous work on compression algorithms can reach a considerable result, but at the cost of a huge computational overhead and memory usage [24]. To meet the requirements of high energy efficiency of LoRa networks, we propose a novel suffix array-based compression algorithm, with which we can achieve a trade-off between compression performance and computational overhead.

Similar to the differencing algorithm, we add one more command LABEL, for intra-firmware compression. In command LABEL, the reoccurring segments are added to the dictionary and subsequently replaced with the dictionary index, denoted as

$$\text{Command LABEL : } L < \text{index, str} > / L < \text{index} >.\text{ (7)}$$

where str appears only for the first time.

Given the generated delta script $F_{\text{FR}}$ composed of $F_{\text{FR}}$ (i.e., the larger one between $\text{F}_{\text{new FR}}$ and $\text{F}_{\text{old FR}}$) firmware records, the common segments appear within and between firmware records. Thus, we define a window containing a source

### Algorithm 1: Delta script construction algorithm.

**Input:** The new and old firmware image, $\text{F}_{\text{new FR}}$ and $\text{F}_{\text{old FR}}$.

**Output:** The delta script, $F_{\text{FR}}$.

1. Function $\text{calc.sat}(S)$:
   1. Suffix array sorting based on the DC3 algorithm;
   2. Return $\text{sa}[S.\text{length}()]$.

2. Function $\text{calc.height}(S)$:
   1. For $i \leftarrow 1$ to $S.\text{length}() - 1$ do
      1. $\text{height}[i] = \text{lcp}(\text{sa}[i], \text{sa}[i-1])$;
   2. Return $\text{height}[S.\text{length}()]$.

3. Function $\text{calc.lcp}(S)$:
   1. $\text{calc.lcp}(S \leftarrow \text{F}_{\text{old FR}}[\text{entry}] + \ldots + \text{F}_{\text{new FR}}[\text{entry}])$;
   2. If offset $\geq$ threshold then
      1. $F_{\text{FR}} \leftarrow K < \text{entry}, \text{addr, offset} >$;
   3. Else
      1. $F_{\text{FR}} \leftarrow U < S.\text{substr}() >$;
   4. Return $\text{addr, offset}$.

4. Function $\text{calc Satellite}(S)$:
   1. $\text{calc.sat}(S)$;
   2. $\text{calc.height}(S)$;
   3. $< \text{addr, offset} > \leftarrow \text{max}(\text{height}[S.\text{length}()])$;
   4. Return $\text{addr, offset}$.

5. **Differencing Algorithm:**
   1. For entry $\leftarrow 0$ to max($\text{F}_{\text{new FR}}, \text{F}_{\text{old FR}}) - 1$ do
      1. $\text{calc.lcp}(S \leftarrow \text{F}_{\text{old FR}}[\text{entry}] + \ldots + \text{F}_{\text{new FR}}[\text{entry}])$;
      2. If offset $\geq$ threshold then
         1. $F_{\text{FR}} \leftarrow K < \text{entry}, \text{addr, offset} >$;
      3. Else
         1. $F_{\text{FR}} \leftarrow U < S.\text{substr}() >$;
   4. Return $\text{addr, offset}$.

6. **Compression Algorithm:**
   1. For entry $\leftarrow 0$ to $F_{\text{FR}} - 1$ do
      1. $c.\text{buffer} \leftarrow F_{\text{FR}}[\text{entry}]$;
      2. If entry $\neq entry2$ then
         1. $s.\text{buffer} \leftarrow F_{\text{FR}}[\text{entry}]$;
      3. Else
         1. $s.\text{buffer} \leftarrow F_{\text{FR}}[\text{entry}2]$;
   4. $c.\text{buffer} \leftarrow F_{\text{FR}}[\text{entry}]$;
   5. If entry $\neq entry2$ then
      1. $s.\text{buffer} \leftarrow F_{\text{FR}}[\text{entry}]$;
   6. Else
      1. $c.\text{buffer} \leftarrow F_{\text{FR}}[\text{entry2}]$;
   7. If offset $\geq$ threshold then
      1. $F_{\text{FR}} \leftarrow L < \text{index, str} > / L < \text{index} >$;
   8. Else
      1. $F_{\text{FR}} \leftarrow U < S.\text{substr}() >$;
   9. Return $\text{addr, offset}$.

10. Return the delta script, $F_{\text{FR}}$.  

buffer $s_{buffer}$ and a compression buffer $c_{buffer}$, denoted as $W = s_{buffer} + \cdots + c_{buffer}$. For $i$th firmware record $F_A[i]$, we first feed it into the compression buffer to find LCP in intra-record level. Then, $F_A[i]$ slides into the source buffer, and the rest of firmware records $F_A[j \neq i]$ are fed into the compression buffer to find LCP in inter-record level. We repeat the above operations until all records are traversed. Similar to the differencing algorithm, for those LCPs whose lengths are greater than the pre-defined threshold, we utilize the $LABEL$ command to encode them in the delta script; otherwise, the $UPDATE$ command is used. We set the threshold to 5, which is the minimum length of the replaced $LABEL$ command. The whole delta scripting process is summarized in Algorithm 1.

### 4.2 Concatenated Channel Coding Scheme

After preparing the delta script, FloRa distributes it to LoRa nodes in fragments. However, as discussed above, although LoRa signals can provide high sensitivity and be resilient to in-band/out-of-band interference, the long-range deployment scenarios make packet loss and symbol error inevitable. Additionally, Cyclic Redundancy Check (CRC) is not enabled in the downlink transmission according to the LoRaWAN specification. Thus, the unreliable link quality problem needs to be resolved before firmware reconstruction during the error-prone FUOTA task. To this end, we propose a concatenated channel coding scheme (see Figure 6(b)), where a rateless code [32] serves as outer code to combat packet loss and an error detection code as inner code to avoid symbol error.

#### 4.2.1 Concatenated Encoding

We introduce the concatenated encoding process for the firmware image, including outer rateless encoding followed by inner error detection encoding.

**Outer Encoding.** Regarding the packet loss issues in multicast scenarios, rateless code has been proven as an effective method [41], which demonstrates a strong capability of resilience to dynamic channel quality. The key idea of conventional rateless codes is to continuously select an appropriate number according to the degree distribution function and generate the encoded blocks containing this number of source symbols. Unfortunately, this may lead to a biased choice of the encoding source symbols. To this end, FloRa designs a priority matrix with a dynamic updating strategy to adjust the weights of all firmware records for fair choosing whilst maintaining historical transmission information during multicast.

Below we introduce the outer encoding process, along with the degree distribution function and priority matrix.

Given the aforementioned delta script with multiple firmware records $F_A[F_A, FR]$, we suppose the number of firmware records is $n = F_A, FR$ and the largest length of the firmware record is $a$. Thus, $i$th record is denoted as $F_A[i] = c_1 c_2 \ldots c_{i-1}$, with $a$ symbols. For the sake of completeness of the expression, those records whose lengths are less than $a$ are filled by the character ‘*’ for the rest of the digits. The encoder selects a set of $d$ original firmware records to generate an encoded block $EB[j]$ at the $j$th time. The degree $d$ conforms to a special degree distribution, while $d$ records are chosen based on the priority matrix. The encoder performs a bit-wise XOR operation for those $d$ original firmware records to dynamically generate the encoded blocks. The outer encoding process is represented by

$$EB[j] = (c_{d_1} \oplus c_{d_2} \oplus \cdots \oplus c_{d_d}) (c_{d_1} \oplus c_{d_2} \oplus \cdots \oplus c_{d_d}) \ldots (c_{d_{m_d-1}} \oplus c_{d_{m_d}} \oplus \cdots \oplus c_{d_{m_d}}).$$

where $d_1, d_2, \ldots, d_d$ represent the selected $d$ entries. After constructing the blocks, two kinds of information are appended as the header: degree $d$ and the entries of these selected firmware records. The encoded block is then denoted as $EB[j] = c_{d_1} c_{d_2} \ldots c_{d_{m_d-1}}$, where $b$ is the updated length. The encoder can continuously select a degree $d$ and generate corresponding encoded blocks.

**Degree Distribution Function.** To enable reasonable degree choosing, we apply the Robust Soliton Distribution (RSD) $\mu(d)$ as the degree distribution function. RSD $\mu(d)$ is originally designed for Luby Transformer (LT) code [32], one of the representative rateless codes:

$$\mu(d) = \frac{\rho(d) + \tau(d))}{\rho(d) + \tau(d)}, d = 1, 2, \ldots, n. \quad (9)$$

It is derived based on the Ideal Soliton Distribution (ISD) $\rho(d)$ by adding an additional component $\tau(d)$, denoted as

$$\rho(d) = \left\{ \begin{array}{ll} \frac{1}{n} d = 1 \\ \frac{1}{d^2} d = 2, 3, \ldots, n \end{array} \right. \quad (10)$$

$$\tau(d) = \left\{ \begin{array}{ll} n/(nd), d = 1, 2, \ldots, [n/P] - 1 \\ \left( P/(n/\delta) + \sum_{d=n/P}^{n} \right)/(n/d) - 1 \\ 0, d = [n/P] + 1, \ldots, n \end{array} \right. \quad (11)$$

Among these, $\rho(d)$ is the probability distribution over the integers varying from 1 to $n$, while $\tau(d)$ relies on a set of parameters $P = q \ln(n/\delta)\sqrt{n}$, where $q > 0$ is a constant and $\delta \in (0, 1]$ represents the upper bound for the decoding failure probability. As shown in Figure 7, $\rho(d)$ has a spike at 2, but the extra component $\tau(d)$ in RSD adds an extra spike at $n/P$. Thus, when receiving a certain number of blocks, the lower bound for the success probability of decoding is $1 - \delta$. In our experiments, we utilize $\delta = 0.1$ and $q = 0.4$ to enable high decoding success probability and moderate size of encoded blocks.

**Priority Matrix.** To avoid biased firmware record choosing, we design a priority matrix $\Gamma$ of size $m \times n$ ($m > n$) for outer encoding. Among them, $n$ is the number of firmware records, while

![Figure 7: Degree distribution function with $n = 100$ (only the first 20 indices shown).](image)

![Figure 8: Priority matrix $\Gamma$ of size $m \times n$ (an example of $m = 20, n = 10$).](image)
\( m \) represents the number of encoded blocks, i.e., the number of transmission iterations. Each element in \( \Gamma \), denoted as \( \gamma_{j,k} \in [0,1] \), represents its priority of the firmware record \( F_A[j] \) during the transmission of encoded blocks \( EB[j] \). As shown in Figure 8, we split \( \Gamma \) into two parts: the fixed source submatrix \( \Gamma_S \) containing the first \( n \) rows of \( \Gamma \) and the varying redundancy submatrix \( \Gamma_R \) containing the rest of \( m - n \) rows of \( \Gamma \). For the source submatrix \( \Gamma_S \), the values of the diagonal positions are set to 1 while the others are 0. This makes sure that only one source firmware record is encoded into the block for transmission in turn, regardless of the degree selected. For redundancy submatrix \( \Gamma_R \), the initial elements are all set to 1, which are updated whenever an encoded block is sent.

We adopt the exponential smoothing method \([13]\) for updating \( \Delta \) values to start a new round of redundancy encoding. The above learning rate updating strategy is used to adjust the weights of all firmware records for adapting to fair source firmware record choosing whilst maintaining historical transmission information during multicast. We set \( a \) to 0.2 to ensure that higher importance is placed on the previously un-selected firmware records.

**Inner Encoding.** After outer encoding, the encoded block is fed for inner encoding to avoid symbol error. FLoRa designs an error detection code rather than an error correction code, since error correction typically requires longer redundancy bytes, which may degrade transmission efficiency. On the contrary, with the feature of infinite transmission of rateless code, the packet can be simply discarded when a symbol error is found.

Inspired by the idea of linear block code \([46]\), we design a novel error detection code. In particular, we define an error detection code \( EB[j,k] \) for the encoded block \( EB[j] \) by dividing it into \([b/k] \) segments. For each segment, we use the XOR operation for these \( k \) characters to generate the check byte, which is appended at the end of each segment. The inner encoding process is represented by

\[
EB[j,k] = \begin{pmatrix}
\gamma_{j,k} & c_{j,k} \\
\gamma_{j,k} & c_{j,k-1} \\
& & & \vdots & \vdots & \vdots \\
& & & & c_{j,k} & c_{j,k-1} \\
& & & & & & \vdots & \vdots & \vdots \\
& & & & & & & c_{j,k} & c_{j,k-1} \\
& & & & & & & & \cdots & \cdots & \cdots
\end{pmatrix}
\]

In our experiment, we set \( k \) to 8 for a trade-off between accuracy and energy overhead. After inner encoding, the encode block is denoted as \( \hat{EB}[j,k] = c_k \hat{c}_{k-1} \cdots \hat{c}_1 \), where \( c \) is the updated length.

It is noted that we set a threshold-enabled stop condition that the number of redundancy packets does not exceed 25% of the firmware records, if not enough update reports are received from LoRa nodes.

**4.2.2 Concatenated Decoding.** After the firmware records of the delta script are encoded and then distributed, LoRa nodes receive these packets to decode them. Specifically, the concatenated decoding process includes the inner error detection decoding followed by the outer rateless decoding to reconstruct the delta script.

**Inner Decoding.** For a received packet \( EB[j,k] \), the LoRa nodes utilize the XOR operation to compute the check codes for each segment, and then compare these computed ones with the appended ones. The received packet can then be fed to the outer decoder for further decoding if the check codes of all segments are correct; otherwise, LoRa nodes discard this packet.

**Outer Decoding.** For outer decoding, LoRa nodes first find those encoding blocks of degree 1. They perform XOR operations between these blocks of degree 1 and the rest of the blocks. In such a process, more encoding blocks of degree 1 are generated. LoRa nodes repeat the above process until all firmware records of \( F_A[\{F_A.FR\}] \) are reconstructed. The decoding process terminates with failure if there is no such block of degree 1. Since the whole decoding process involves the XOR operations only, the decoding scheme is lightweight to power-constrained LoRa networks.

**4.2.3 Firmware Image Reconstruction.** After receiving all firmware records of the delta script correctly and completely, the firmware image reconstruction can be performed. Recall Section 4.1, by simply traversing each firmware record of delta script and performing corresponding command recovery operations with an order of LABEL, KEEP, and UPDATE, LoRa nodes can losslessly recover the firmware image. With the above design, the firmware image reconstruction algorithm is energy efficient for LoRa nodes due to its time complexity of \( O(n) \). At this stage, the majority of LoRa nodes have successfully acquired the firmware image to reboot themselves.

**4.3 Beamforming Strategy**

Several multicast users with poor link quality may fail the FUOTA task. To solve this problem, we additionally group these LoRa nodes into several unicast deliveries and utilize beamforming to achieve a better link budget and stronger signal strength \([25]\). However, to acquire the key information to beamforming, such as azimuth angles of candidate LoRa nodes, the existing LoRa AoA methods \([5,31]\) are not desirable due to their limited resolution and range. To overcome this limitation, we design an explicit beam scanning-based azimuth estimation method, along with an optimal unicast group setup policy (see Figure 6(c)).

**4.3.1 Beam Scanning-Based Azimuth Estimation Method.** Before presenting the beam scanning-based azimuth estimation method, it is essential to discuss the impact of beamforming. By definition, beamforming is a spatial filtering technique to radiate or capture energy in a specific direction based on an array of radiators over
its aperture [25]. Thus, there are two primary factors to beamforming, types of antenna arrays and beam-steering azimuth angles. For instance, suppose a LoRa transmitter has an antenna array of Uniform Linear Array (ULA), a commonly-used type in beamforming, in which antennas are arranged in a straight line with a distance of half the wavelength between every two adjacent ones. Figure 9 illustrates several beamforming patterns with different numbers of antennas $u$ and beam-steering azimuth angles $\theta$. We can observe that $u$ and $\theta$ render the angle and the main lobe of the beam, respectively. However, the main lobe of the beam is too abstract a performance metric to quantify, thus we utilize Half Power Beam Width (HPBW) [18] to indicate the beamwidth $\psi$. Specifically, HPBW refers to the angular separation width, in which the magnitude of the radiation pattern decreases by $-3$ dB (typically 50%) from the peak of the main lobe of the beam.

To efficiently and dependably acquire the azimuth angles of the candidate LoRa nodes, FLoRa implements an explicit beam scanning method. The idea of this method involves sending probing packets in the directions with the highest gain of beamforming and relying on the returned packets from nodes to collaboratively determine the azimuth angle. With beamforming, the LoRa transmitter sends $p$ probing packets every beamwidth $\psi$ to all candidate LoRa nodes that wait for the unicast delivery, thus resulting in $p \times \lceil 2\pi/\psi \rceil$ probing packets in total. Each probing packet contains the beam-steering azimuth angle information $\theta$ (i.e., the direction with the highest gain of beamforming) during the current transmission. Afterward, LoRa nodes may receive some of the probing packets, and then respond to the LoRa transmitter the ones with top-$l$ largest Received Signal Strength Indicator (RSSI) values. In our experiments, the values of $p$ and $l$ are both empirically set as 3 to ensure stable and accurate estimation of the azimuth angle whilst maintaining high energy efficiency. It is noted that the time duration and energy consumption during beam scanning are trivial compared to FUOTA firmware distribution.

4.3.2 Optimal Unicast Group Setup Policy. Given the candidate LoRa nodes that wait for the unicast delivery and the information of beamforming with respect to steering azimuth angle and beamwidth, we need to set up the unicast groups elaborately to maximize the performance of FUOTA tasks. To this end, we design an optimal unicast group setup policy based on the hierarchical clustering method [35] to acquire the number of unicast groups. In brief, FLoRa computes the azimuth angle differences between every two of these nodes, where Manhattan distance is utilized as the similarity metric, denoted as

$$\|\theta_i - \theta_j\| = \sum_{l} |\theta_{i_l} - \theta_{j_l}|,$$

(13)

where $\theta_i$ and $\theta_j$ represent the azimuth angle of LoRa node $i$ and $j$. With the computed similarity matrix, the beamwidth $\psi$ is utilized as the constraint to assign different unicast groups for these candidate LoRa nodes. Afterward, the firmware image can be distributed accordingly until all unicast deliveries are complete to finish the FUOTA task.

5 EVALUATION

5.1 Experimental Setup

We now move to evaluate FLoRa. We first describe a LoRa testbed we build, then present the benchmarks of FLoRa and the existing solution in LoRaWAN, followed by a set of performance metrics. Testbed. We implement FLoRa in a LoRa testbed with Commercial Off-The-Shelf (COTS) LoRa devices, and deploy the testbed on our campus to evaluate the performance of FLoRa for a period of 1 month. Figure 10 and 11 show our experimental setup. The testbed is composed of 1 LoRa transmitter (gateway) and 20 LoRa receivers (end devices), deployed in both outdoor and indoor scenarios. Each LoRa transceiver is assembled by integrating an Arduino ATmega328P microcontroller unit (MCU), an SD card shield with 8 GB capacity, a Dragino LoRa SX1276 Shield operating at 868 MHz, and an omnidirectional antenna with 2 dB gain. Then, each transceiver is connected to a Raspberry Pi 4b single-board computer supplied by the battery via a USB port. All the Raspberry Pis are connected to campus Wi-Fi for experiment control and result visualization. It is noted that the size of the testbed is comparable to several state-of-the-art works, such as CurvingLoRa [29] (12 nodes, 1.45 km $\times$ 0.35 km outdoors) and FTrack [49] (20 nodes, 47 m $\times$ 20 m indoors).

Benchmarks. We compare FLoRa with the existing solution in LoRaWAN, i.e., Baseline. Baseline adopts LDPC as the error erasure code with its coding rate set to 5–15%, which means Baseline can transmit 5–15% redundancy packets for error recovery. Additionally, we utilize FLoRa-Multicast and FLoRa-Unicast to represent the FUOTA process in the multicast and unicast manner, respectively.
Performance Metrics. We use the following performance metrics: 
(1) **Success Rate (SR)** is defined as the ratio of successfully rebooted LoRa nodes to the total number of ones, which measures the transmission reliability during FUOTA tasks. 
(2) **Energy Overhead (EO)** is defined as the average energy consumption of all LoRa nodes in the testbed for packet receiving and decoding, which measures the energy efficiency during FUOTA tasks. EO is measured by the Monsoon Power Monitor [40]. For each metric, we report the average results and the standard deviations by conducting sufficient times.

5.2 Overall Performance

We evaluate the performance of FLoRa with different firmware image sizes and LoRa configuration parameters.

5.2.1 Evaluation of Firmware Image Size.

**Setup.** We compare the performances of FLoRa with Baseline in ten firmware image sizes, increasing from 10.40 kB to 71.83 kB. By default, our experiments are conducted with LoRa parameter settings of $SF = 9$, $BW = 125$ kHz, and $TP = 17$ dBm, where SF represents the spreading factor, BW is the bandwidth, and TP is the transmission power. Each firmware record in the firmware image is modulated as one LoRa packet to be sent at intervals. The interval is set to 0.5 s to meet the requirements for greater than the LoRa packet air-time.

**Result.** Figure 12(a) and 12(b) show the evaluation results of different firmware image sizes on SR. We observe that the SR-Size curves of Baseline, FLoRa-Multicast, and FLoRa-Unicast show a similar decreasing trend as size increases. The performance of Baseline shows a significant decrease at a large size, but FLoRa-Multicast and FLoRa-Unicast demonstrate robustness. The SR of Baseline is 0.76 at a size of 10.40 kB and 0.59 at a size increased to 71.83 kB in an outdoor environment, while the SR of FLoRa-Multicast is ranged from 0.91 to 0.81. The mean SR of FLoRa-Unicast reaches 0.95 due to the use of beamforming for the unicast group. The performance boosted by FLoRa-Unicast based on FLoRa-Multicast indoors is slightly lower than that outdoors. This may be due to multipath effects in an indoor environment. Thus, both FLoRa-Multicast and FLoRa-Unicast have obtained consistently higher SR than Baseline for all firmware sizes. Overall, the results verify the reliability of FLoRa, which improves 1.30–1.51× outdoors and 1.36–1.47× indoors than Baseline.

Similarly, Figure 12(c) and 12(d) show the evaluation results of different firmware image sizes on EO. The EO-Size curves of Baseline, FLoRa-Multicast, and FLoRa-Unicast show a similar increasing trend as size increases. The EO of Baseline is 21.67 J at a size of 10.40 kB and 167.02 J at a size increased to 71.83 kB, while that of FLoRa-Multicast is ranged between 9.06 J and 70.45 J outdoors. Both FLoRa-Multicast and FLoRa-Unicast have obtained consistently lower EO than Baseline, especially at a large size. For example, the EO of Baseline at a size of 25.12 kB is close to that of FLoRa-Multicast at a size of 42.24 kB outdoors. The EO of FLoRa-Unicast is slightly lower than that of FLoRa-Multicast, since beamforming offers better link quality and a smaller number of redundancy packets are sent. The results verify the energy efficiency of FLoRa, which improves 1.91–2.65× outdoors and 1.90–2.60× indoors than Baseline.

In addition, we provide an intuitive EO comparison using battery life as follows. We use a 5 V battery with a capacity of 500 mAh (9,000 J) to power the LoRa transceivers. Suppose the firmware image is 28.82 kB, the EO of Baseline and FLoRa-Unicast is 61.87 J and 25.49 J, respectively. Thus, the energy costs of Baseline and FLoRa-Unicast amount to 0.69% and 0.28% of the total energy supply of the battery. For a fully charged battery, a LoRa node can complete...
5.2.2 Evaluation of LoRa Configuration Parameters. Setup. We evaluate the performance of Baseline and FloRa with various LoRa configuration parameters, including 3 SFs (i.e., 7, 8, and 9) and 3 BWs (i.e., 125 kHz, 250 kHz, and 500 kHz). By default, our experiments are conducted using a firmware image with a size of 28.82 kB outdoors. Result. Figure 13(a) and 13(b) show the error rates of different LoRa configuration parameters on SR. For SF, we observe that the SR of Baseline, FloRa-Multicast, and FloRa-Unicast show a slight increase as SF increases. The SR of Baseline is 0.67 at SF=7 and 0.73 at SF=9, while that of FloRa-Multicast is ranged between 0.88 and 0.89. This is because a larger SF incurs a lower data rate but provides better resilience for interference and fading. Additionally, the characteristic of robustness to packet loss allows FloRa to show resilience at low SF configuration. For BW, we observe that Baseline, FloRa-Multicast, and FloRa-Unicast show a slight decrease as BW increases. The SR of Baseline is 0.73 at BW=125 kHz and 0.66 at BW=500 kHz, while that of FloRa-Multicast is ranged from 0.89 to 0.87. This is because a higher BW provides a higher data rate, but lower sensitivity due to the additional noise integrated. Thus, both FloRa-Multicast and FloRa-Unicast have obtained consistently higher SR than Baseline under different configuration parameters.

Figure 13(c) and 13(d) show the error rates of beamforming strategy for ULA antennas. (a) and (b) illustrate the error rate of the beamforming strategy. (b) and (c) illustrate the SR and EO, respectively.

145 times of FUOTA tasks under Baseline and 353 times under FloRa. These results demonstrate that FloRa incurs a low EO and is more efficient than the existing solution in LoRaWAN.

5.3 Microbenchmarking
We now evaluate FloRa with respect to delta scripting (§5.3.1), channel coding (§5.3.2), beamforming (§5.3.3), ablation study (§5.3.4), and computational overhead (§5.3.5). Unless otherwise specified, the experiments will use the default settings in §5.2.

5.3.1 Evaluation on Delta Scripting. Setup. We evaluate the performance of delta scripting on ten trials of firmware image sets with a size varying from 10.40 kB to 71.83 kB. By default, the new firmware image is a random modification of the old firmware, thus possessing a similar size to the old one.

Result. Figure 14 shows the evaluation results of delta scripting with ten trials of firmware image sets. We observe that FloRa consistently achieves a much smaller size of differenced and compressed delta script compared with the new firmware image. For example, in trial 5 with the new firmware at a size of 28.82 kB and the old one at a size of 28.78 kB, the size of the differenced and the compressed delta script is 14.76 kB and 11.67 kB, respectively. For all firmware image sets, the size of the differenced delta script decreases by 36.8–59.4% on the basis of the new firmware. While that of the compressed one further decreases by 10.4–20.9% based on the differenced one. This verifies the effectiveness of our delta scripting algorithm.

5.3.2 Evaluation on Channel Coding. Setup. We evaluate the performance of channel coding under eight different deployed TX-RX distances that vary from 0.25 km to 2 km.

Figure 15: Performance of channel coding under different TX-RX distances, including link quality, SR, and EO, respectively.

Figure 16: Performance of beamforming under different numbers of ULA antennas. (a) illustrates the error rate of the beamforming strategy. (b) and (c) illustrate the SR and EO, respectively.
Result. Figure 15(a) shows the results of link quality under eight TX-RX distances, represented by the number of redundancy encoded blocks sent of FloRa-Multicast. We observe that the box plot shows an increasing trend as distance increases, which means link quality becomes worse. The average number of redundancy encoded blocks sent is 17.2 at a distance of 0.25 km, while becomes 87.8 at a distance of 2 km. The reason is that the occurrence of packet loss and symbol error tends to increase the number of redundancy blocks as TX-RX distance increases. This also suggests the necessity of reliable channel coding during FUOTA tasks.

Figure 15(b) shows the evaluation results of eight TX-RX distances on SR. We observe that the SR-Distance curves of Baseline, FloRa-Multicast, and FloRa-Unicast all show a decreasing trend as distance increases. For TX-RX distances ranging between 0.25 km and 2 km, the SR of Baseline is ranged from 0.87 to 0.55, which encounters a significant decrease with a higher distance. While that of FloRa-Multicast and FloRa-Unicast are ranged from 0.99 to 0.85 and 0.99 to 0.92, which are consistently higher than Baseline due to the proposed reliable channel coding scheme. For example, the SR of Baseline at a distance of 0.25 km is close to that of FloRa-Multicast at a distance of 1.25 km. The SR of FloRa-Unicast approaches an average value of 95.1% due to the use of beamforming.

Figure 15(c) shows the evaluation results of eight TX-RX distances on EO. The EO-Distance curves of Baseline, FloRa-Multicast, and FloRa-Unicast also show a similar increasing trend as distance increases. The EO of Baseline is 57.56 at a distance of 0.25 km, while becomes 66.94 at a distance of 2 km. Both FloRa-Multicast and FloRa-Unicast have obtained consistently lower EO than Baseline, especially at a longer TX-RX distance. The EO of FloRa-Multicast and FloRa-Unicast are ranged from 25.02 to 30.88 and 23.56 to 27.32, respectively. This verifies the effectiveness of our concatenated channel coding scheme.

5.3.3 Evaluation on Beamforming. Setup. We evaluate the performance of beamforming under different numbers ofULA antennas. Specifically, we deploy three ULA as 2, 4, and 8 antennas, denoted as ULA_2, ULA_4, and ULA_8, respectively. Each antenna is calibrated to be placed half a wavelength (i.e., 17.28 cm) away from its adjacent one. LoRa transmitting signal is fed to the ULA through an equal power splitter.

Result. Figure 16(a) shows the evaluation results of the beamforming strategy of ULA_8 under eight TX-RX distances. Specifically, we utilize the error rate, the ratio of the absolute error to the maximum measurement range (i.e., 360°), as the metric to indicate the accuracy of the proposed beamforming strategy. We observe that the error rates show a decreasing trend as distance increases. The beamforming strategy has an average error rate of 24.80% when the TX-RX distance is 0.25 km. As TX-RX distance increases, the average error rate reaches 6.20% at a distance of 2 km. This means that the accuracy of the azimuth estimation method is susceptible to interference by other probing packets at a close distance. The low error rate level verifies the effectiveness of our beamforming strategy.

Figure 16(b) shows the evaluation results of eight TX-RX distances on SR. We observe that the SR-Distance curves of ULA_2, ULA_4, and ULA_8 all show a decreasing trend as distance increases. For ULA_2, the SR is 0.99 at a distance of 0.25 km, but it decreases to 0.86 at a distance of 2 km. While that of ULA_4 and ULA_8 are ranged from 0.99 to 0.89 and 0.99 to 0.92, respectively, which are consistently higher than ULA_2 due to better link quality offered. For example, the SR of ULA_2 at a distance of 0.75 km is close to that of ULA_8 at a distance of 1.5 km.

Figure 16(c) shows the evaluation results of eight TX-RX distances on EO. The EO-Distance curves of ULA_2, ULA_4, and ULA_8 show a similar increasing trend as distance increases. The EO of ULA_2 is 24.49 at a distance of 0.25 km, while becomes 30.03 at a distance of 2 km. Both ULA_4 and ULA_8 have obtained consistently lower EO than Baseline, especially at a longer TX-RX distance. The EO of ULA_4 and ULA_8 are ranged from 28.92 to 23.56, respectively. This is because a larger number of ULA antennas can result in a higher power gain.

5.3.4 Ablation Study. Setup. We add an ablative study to show the impact of independent modules of FloRa on the effectiveness of FUOTA tasks.

Result. Table 1 shows the results of the ablation study. We observe that these three key components of FloRa contribute to the performance gain of FUOTA tasks. Specifically, the delta scripting algorithm can significantly reduce energy consumption, from 61.87 J to 26.31 J of EO. The channel coding scheme is mainly beneficial to increase transmission reliability, from 0.73 to 0.86 of SR. While beamforming results in enhanced performances with respect to both SR and EO due to better link quality offered.

5.3.5 Computational Overhead. Setup. We now evaluate the computational overhead by running the firmware reconstruction algorithm on the Raspberry Pi single-board computer. The computational overhead is indicated by the running time and energy consumption, both of which are averaged by measuring 20 times.

Result. Table 2 illustrates the results of the computational overhead of FloRa under different firmware sizes. We observe that the running time increases as firmware size goes up. The running time is 0.125 s at a size of 3.78 kB, while 0.195 s at a size of 29.53 kB. Similarly, energy consumption increases as running time rises. The energy consumption is 1.376 J at a size of 3.78 kB, while 2.237 J at a size of 29.53 kB. The results show that firmware reconstruction can be finished in 0.2 s at a low energy consumption level. The low level of computational overhead verifies the efficiency of FloRa.
6 DISCUSSION

Beamforming. Antenna arrays and beamforming techniques can offer better link quality, thus improving the success rate and energy efficiency of FUOTA tasks in LoRa networks. In our experiments, we use ULAs with manually steering the azimuth angle to strengthen the signal in specific directions from a low-cost perspective. Thus, the experimental results are sensitive to the power loss of intermediate hardware elements and the non-uniformity of each antenna channel. To solve this problem, we plan to utilize the phased array antennas [25] in our further work to achieve precise beamforming by flexibly modifying the feed-in phase and amplitude of signals in each antenna. In the meantime, extra hardware updates and maintenance costs should also be key considerations for commercial use.

Experimental Setting. The experimental setting is flexible and can be tailored to different scenarios. For example, FLoRa-Multicast and FLoRa-Unicast can serve as trade-off options. FLoRa-Multicast is suitable for large-scale LoRa networks in a short FUOTA time demand. On the other hand, FLoRa-Unicast can be employed under link quality detection in advance, serving for small-scale nodes with subpar link quality, albeit over a longer time duration.

Security Vulnerability. The FUOTA task is a security-critical task, which may be vulnerable to various attacks with respect to the generation, distribution, and even storage of the firmware image. Additionally, it is of utmost importance to maintain the security of the command packet that issues FUOTA tasks. Therefore, firmware image integrity and user confidentiality can be considered by involving some state-of-the-art security schemes [11, 14] in future work.

7 RELATED WORK

LoRa. LoRa technology has been attracting significant efforts due to its promising prospects, with respect to PHY demodulation [28, 47], Media Access Control (MAC) protocols [10, 15], security [20, 39], and applications [16, 55]. For PHY demodulation, NELoRa [28] adopts deep learning networks for the extracted fine-grained LoRa chip to achieve a high upper limit of the SNR gain. LoRa parallel demodulation capability also receives much attention [43, 48]. For MAC protocol, LMAC [10] and LoRaCP [15] empowered LoRa with Carrier Sense Multiple Access (CSMA) and Time Division Multiple Access (TDMA) capabilities, respectively. Against various attacks, a large number of countermeasures, such as key generation [52, 54], authentication [14, 45], and packet recovery [20], were proposed. Additionally, recent attempts have built various LoRa-enabled applications, such as backscatter [17, 36], sensing [50, 55], and localization [5, 8]. These works are in parallel to FLoRa.

OTA Programming. The existing OTAP protocols mainly target for communication technologies and network topology. Mature OTAP specifications and platforms are developed for legacy wireless communication technologies, including Wi-Fi, Zigbee, and Bluetooth [3]. Additionally, plenty of firmware dissemination protocols have been proposed for Wireless Sensor Networks (WSNs), such as Trickle [27], Deluge [21], and Seluge [22]. However, these methods are designed for multi-hop WSN networks, which are hence not suitable for single-hop LoRaWAN networks. Compared with these works, FLoRa fills the gap of OTAP in LoRa technology.

Incremental Update. As an energy-efficient option, incremental update mainly relies on the differencing algorithm to generate the delta script, including block-level [44] and byte-level [9, 34] ones based on the matching granularity [3]. Rsync [44] relies on a sliding window to find LCP in the fixed blocks, while R3diff [9] computes the hash values for every three continuous bytes. Unlike these methods, FLoRa proposes a joint differencing and compression algorithm to minimize the size of the delta script.

Channel Coding. Channel coding is widely adopted for robust transmission. For example, DaRe [33] integrates convolution and fountain code for data recovery in LoRaWAN. OPR [4] exploits gateway spatial diversity with packet RSSI value for error detection, while leverages the Message Integrity Check (MIC) field for error correction in LoRa networks. Differently, FLoRa designs a concatenated channel code for FUOTA downlink transmission, including outer rateless code and inner error detection code.

Beamforming. Beamforming can enhance the signal power in interested directions and enable Space-Division Multiple Access (SDMA). Beamforming potential has been unlocked in LoRa sensing with respect to long-range and multi-target respiration monitoring [51, 55]. Differently, FLoRa is the first work adopting beamforming to boost the performance of FUOTA tasks in LoRa networks.

8 CONCLUSION

In this paper, we present the design, implementation, and evaluation of FLoRa to accomplish and boost FUOTA in LoRa networks. FLoRa consists of three essential techniques. First, we propose a joint delta scripting algorithm to unlock the potential of incremental FUOTA in LoRa networks. Then, we propose a concatenated channel coding scheme to resolve the link quality indeterminacy. Last, we design a beamforming strategy to prevent biased multicast and compromised throughput. We conduct extensive experiments on a 20-node LoRa testbed to evaluate the performance of FLoRa. The results illustrate that FLoRa improves network transmission reliability by up to 1.51× and energy efficiency by up to 2.65× compared with the existing solution in LoRaWAN.

ACKNOWLEDGMENTS

We thank the anonymous shepherd and reviewers for their constructive comments. The work described in this paper was substantially sponsored by the project 62101471 supported by NSFC and was partially supported by the Shenzhen Research Institute, City University of Hong Kong. The work was also partially supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 21201420 and CityU 11201422). Shenzhen Science and Technology Funding Fundamental Research Program (Project No. 2021SZ01210), Chow Sang Sang Group Research Fund sponsored by Chow Sang Sang Holdings International Limited (Project No. 9229062). The work was also partially supported by CityU MPFR grant 9680333, CityU SIRG grant 7020057, CityU APRC grant 9610485, CityU ARG grant 9667225 and CityU SIRG-Fd grant 7005666.

REFERENCES

solution. In IEEE DCOSS.


[26] Yidong Ren, Li Liu, Chenmin Li, Zhihao Cao, and Shiguang Chen. 2012. Is lorawan really wide? fine-grained lora link-level measurement in an urban environment. In IEEE ICNP.


[29] Yin Sun, C Emre Koksal, Kye-Han Kim, and Nes B Shroff. 2014. Scheduling of multicast and unicast services under limited feedback by using rateless codes. In IEEE INFOCOM.


[38] Binbin Xie and Jie Xiong. 2020. Combating interference for long range LoRa sensing. In ACM SenSys.


