# IoT Device Firmware Update over LoRa: The Blockchain Solution

A. Anastasiou *, P. Christodoulou*, K. Christodoulou†, V. Vassiliou ‡ and Z. Zinonos*
*Department of Computer Science, Intelligent Systems Lab, Neapolis University Pafos, Cyprus
†Institute For the Future (IFF), University of Nicosia, Cyprus
‡Department of Computer Science, University of Cyprus, Cyprus

***Abstract*—More and more Internet of Things (IoT) devices are deployed around the world, due to the convenience and extra functionality they enable. This growth, while great for the industry as a whole, has come at a price with respect to ensuring and maintaining security and privacy. Having that in mind, one of the most common solutions to the IoT security problem is to update the devices frequently. Recently, LoRa Alliance has released a new specification (FUOTA) on how to perform firmware updates using LoRa technology. In this paper, we propose a blockchain-based framework to securely update the firmware of the IoT devices using the LoRa communication protocol. As a first step, we perform an evaluation of the firmware update procedure using different network sizes and different firmware sizes. The evaluation shows that there is a need to use more gateways that will collaborate to increase the reliability and the performance of the firmware update process.**

***Index Terms*—IoT, Firmware Update, Blockchain, LoRa**

## I. Introduction

Every day thousands of new end-devices are connected to the IoT world, from smart refrigerators, smart watches, smart door locks, medical sensors, to fitness trackers and smart security systems. IoT has being integrated in many sectors such as, transportation, health-care, smart cities, agriculture, manufacturing and environmental monitoring systems.

The increasing popularity of the Internet of Things (IoT) has made IoT devices a powerful amplifying platform for cyber-attacks. Numerous vulnerabilities and attacks have been reported affecting IoT devices. The 2019 Ponemon Institute Study on the Cyber Resilient Organization [1] discusses that the number of breaches due to unsecured connected devices now accounts for 26% of all security breaches up from last years figure of 15%. The actual number may actually be greater, as most organizations are not aware of every unsecured IoT device, application, or third party platform they are using.

In the Mirai attack, criminals found a way to perform the largest DDoS attack ever known. Authors in [2] have concluded that the main reason for the attack being so successful, was the factory credentials that were left unchanged along with the inadequate firmware updates. Another example showing the importance of firmware update was observed within the Fiat Chrysler Group. Attackers gained access to the vehicles via the entertainment system and subsequently to the rest of the control software, that way obtaining control to crucial systems of the vehicle. The Fiat Chrysler Group had to recall 1.4M vehicles in the US and was forced to individually patch their systems with an updated version of the firmware.

Several architectural IoT factors should be taken into account when designing and developing secure IoT deployments. Some advocate the use of hardened border routers, or gateways; others suggest segragating IoT networks; while others propose methods for recognizing anomalies within the network [3]. We believe, like many other experts, that the most sensitive issue concerning IoT devices security is the process of updating the firmware in a way that the integrity of the firmware is validated. Without adopting any security mechanism, it is possible that the firmware update is written over the already installed one, without any revision. In cases of a malfunction during the update potential for a rollback is limited. The fact that the majority of these devices are expected to operate for years, probably performing different tasks over time, makes the update of the devices an important task for the long term viability of the device.

In this paper, we use two state-of-the art technologies to support firmware update procedures for IoT devices. More specifically, we utilize a Blockchain smart contract to effectively support IoT devices firmware updates by verifying the authenticity and integrity of the firmware. At the same time, we use the FUOTA framework and we evaluate the performance of LoRa technology during the firmware update procedure.

Blockchain (BC) is the technology used in the world's first cryptocurrency namely, bitcoin (BTC). BTC was first launched in 2008 and is considered an immutable electronic distributed ledger. By many it is considered the apogee regarding privacy and security in a decentralized system. Blockchain technology was designed for securing the transactions of the world's most recognized cryptocurrency.

On the other hand, LoRa belongs to the Low-Power Wide Area Networks (LPWAN) that are wireless technologies with specific characteristics such as large coverage areas, low bandwidth, possibly small packet and application layer data sizes and long battery life operation. LPWAN is the wireless communication technology that supports the requirements of IoT applications. In a nutshell, these requirements contain conditions related to low data rate, long-range, low energy consumption, and low cost. In this work, we will use LoRa to multicast the firmware update to all interconnected IoT

Dr. Vassiliou is also with the RISE - Research Center on Interactive Media, Smart Systems and Emerging Technologies, Nicosia, Cyprus

2325-2944/20/$31.00 ©2020 IEEE
DOI 10.1109/DCOSS49796.2020.00070

devices.

The remainder of the paper is organized as follows. Section II briefly presents the related work about firmware updates and provides background information explaining the functionality of Blockchain and Lora. The challenges in updating IoT devices are discussed in Section II. In Section III the proposed architecture with the different components is analysed. In Section IV, authors evaluate the proposed solution and present the collected results. Finally, the conclusions of the study are summarized in Section V.

## II. Background Information

Nowadays, different approaches have been proposed for updating IoT firmware of devices. Mostly, such approaches aim to maintain a balance between time security, system stability, and transmission reliability. This section focuses only on Blockchain-based proposed approaches. In addition, this section provides details on LoRa's functionality.

### A. Blockchain

Blockchain has numerous adoptions in a number of non-monetary applications. This includes applications like a decentralized IoT parking application [4] [5], energy auction applications [6], smart city applications [7] [8] and others that make BC an attractive feature for distributing firmware updates to the IoT world.

Over the years a number of solutions regarding automatic firmware updates has been proposed to satisfy the fast growing IoT ecosystem. BC as a firmware update distribution approach was proposed by Lee et al. [9], where authors used a BC network to distribute firmware status to the devices along with peer-to-peer (P2P) delivery for the binary part. Their proposal also supports firmware integrity checking for extra safety.

In [10], authors proposed a firmware update scheme based on a decentralized smart contract designed for autonomous vehicles (AVs). The smart contract was used to ensure the authenticity and integrity of firmware updates, and more importantly to manage the reputation values of AVs that transfer the new updates to other AVs.

Another proposal describing a BC-based security update mechanism in IoT was presented in Steger et al. [11]. Authors proposed this mechanism as a solution in the automotive industry. By its fundamental design BC records a block of transactions in the chain using a cryptographic hash that is linked with the previous block that also contains time-stamps, other transactions and additional data. Even though their proposal was mainly used for the automotive industry, if we take into consideration the number of vehicles that participated using this approach, we can consider that it translates to IoT specific devices as well.

Dorri et al. [12] identifies numerous issues pointing that BC is not an off-the-shelf solution to IoT. There are several issues that need to be taken into consideration e.g., the high demands for resources, especially when using the Proof-of-Work consensus algorihtm, or scalability requirements amongst miners, and the high delays resulting from the deployment of strategies to prevent double spending. To solve these challenges, authors developed a Lightweight Scalable Blockchain (LSB). The philosophy of LSB is quite simple, the system replaces the resource intensive consensus algorithm with a timing-based value, and divides the network into manageable public blockchain clusters.

In [13] authors are focused on current research work that is based on IoT firmware updates which aims to highlight issues related with security of firmware updates. The study specifically focuses on the security challenges that face low-end IoT devices and LPWAN.

In [14] authors present a solution on how to secure the firmware updates on IoT gateway devices, that aims to assure the proof of origin, integrity and confidentiality in transit of the firmware image.

Finally, in [15] authors outline a framework that utilizes Blockchain technology to securely check the firmware version, validate the correctness of the firmware and download the latest firmware for the various embedded devices.

### B. LoRa

LoRaWAN [16] has three different classes of end-devices to address the various needs of applications:

- *Class-A*: The end-devices allow bi-directional communications whereby each end-device's uplink transmission is followed by two short downlink receive windows. The transmission slot scheduled by the end-device is based on its own communication needs with a small variation based on a random time basis.
- *Class-B*: The end-devices of Class-B have a higher number of received slots. In contrast to Class-A , Class-B devices open extra receive windows at scheduled times. In order for the end-device to open a receive window at a scheduled time it uses a time synchronized Beacon from the gateway. This allows the server to know when the end-device is listening.
- *Class-C*: The end-devices of Class-C an almost continuously open receiving windows mechanism; they are closed only when transmitting. Such devices consume more power to operate compared to Class-A and Class-B, however they offer the lowest server latency to end-device communication.

LoRa allows the configuration of four critical parameters, Bandwidth (BW), Spreading Factor (SF), Coding Rate (CR) and Transmit Power (TP).

- *Bandwidth (BW)*: The bandwidth is the width of the transmitted signal. It can only be chosen among three options: 125kHz, 250kHz or 500kHz. For a long range transmission a 125kHz BW must be configured and for a fast transmission a 500kHz BW must be configured.
- *Spreading Factor (SF)*: The Spreading Factor refers to the value which determines how spread the chirp would be. In LoRaWAN networks, SF7- SF12 are used. The selection of SF defines also the data rate. Table I presents the

chirp packet length based on the SF parameter. Modifying this parameter provides a trade-off between increasing the communication distance and decreasing the data transfer rate.

- *Coding Rate (CR)*: The Code Rate error correction code its added to a packet before transmission. To calculate the CR, the following formula is used:

$$CR = \frac{4}{4+n}, n = 1, 2, 3, 4 \quad (1)$$

- *Transmission Power (TP)*: The Transmission Power refers to the amount of power used to transmit a chirp. The higher the transmission power is the higher the power consumption is also. For example, for a transmission power of 20dBm the power consumption is 412.5mW.

TABLE I: Data Rates

| Data Rate (DR) | Spreading Factor (SF) | Bandwidth (kHz) | Max App Payload (bytes) |
|---|---|---|---|
| 0 | 12 | 125 | 51 |
| 1 | 11 | 125 | 51 |
| 2 | 10 | 125 | 51 |
| 3 | 9 | 125 | 115 |
| 4 | 8 | 125 | 222 |
| 5 | 7 | 125 | 222 |

The LoRa Alliance has published three Firmware Updates Over The Air (FUOTA) LoRaWAN application specifications. These specifications are related to time synchronization, the sending of messages to groups of end devices and the splitting of data files. The recommended FUOTA architecture is presented in [17]. In fact, without utilizing the new specifications it is impossible to successfully update the firmware because of LoRa characteristics such as the limited data rates (bits per second), the duty cycles limitations since these networks operate in the unlicensed spectrum. In addition, this type of networks suffer from packet losses due to interference. In our proposal, we utilize the recommended FUOTA architecture as shown in the section III.

For the update procedure, we adopt the transmission of Class-C messages. Thus, the duty cycle of the gateway can be enhanced by using multicast Class-C messages. To achieve multicasting, a network server is responsible to create, delete or modify the multicast groups and guarantee that the multicast message will be delivered to the multicast group.

### C. Challenges in Updating IoT Devices

In this subsection, we identify some of the fundamentals steps required when updating a firmware. Note that requirements may differ depending the device type and/or its application. The major challenges are:

1) *Authentication*: The device should accept the firmware from a trusted source.
2) *Version Control*: The firmware should be accepted only if the version is suitable for the device (to prevent the installation of outdated software).
3) *Package Integrity, Complete and Error-Free Transmission*: If the firmware is somehow tampered or is incomplete it should not be accepted by the device. After the update package is transmitted, it has to be debugged for any errors.
4) *Operability Check*: The updated firmware has to be checked if it is working as it was intended to do.
5) *Reduced User Interaction*: User interaction must be limited since this is the common source of errors.

Within the world of IoT there are various challenges concerning the process of automatic firmware update. A major challenge is the lack of a standardized approach for wide-scale updating, for large scale IoT deployments. The challenge emerges from the fact that devices are from many different manufacturers, having heterogeneous ways for conducting updates. In addition, devices are often constrained in hardware resources with limited connectivity and reach-ability.

## III. Firmware Update Using Blockchain

The need for Firmware updates over the air (FUOTA) has risen due to IoT security concerns. A number of recommendations were proposed for performing embedded software management, and more specifically firmware updates.

With FUOTA's support, new functionalities, security updates, and optimization patches can be deployed with minimum human intervention to embedded devices, over their lifetime. However, supporting FUOTA over one of the most promising IoT networking technologies, LoRaWAN, is not a straightforward task. This is mainly due to LoRaWAN's limitations. LoRaWan does not provide the best channel for bulk data transfer such as a firmware image. While the LoRa Alliance proposed new specifications to support multicast, fragmentation, and clock synchronization, which are essential features to enable efficient FUOTA in LoRaWAN, we propose an added security level to the process. With this paper we propose the use of a smart contract that has been deployed over a BC as an authentication and a management mechanism for firmware updates. The proposed model is mainly based on LoRa Alliance FUOTA Process [16]. The proposed architecture is shown in Fig. 1.

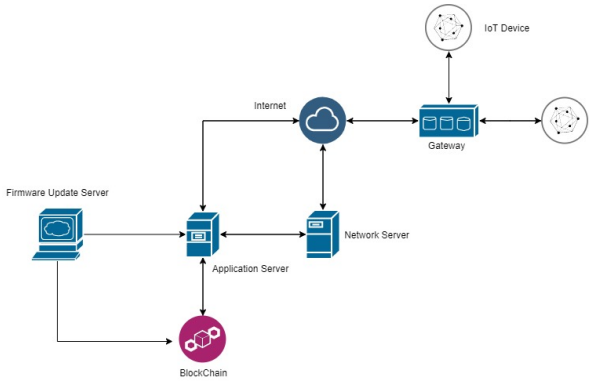



Fig. 1: Proposed Architecture

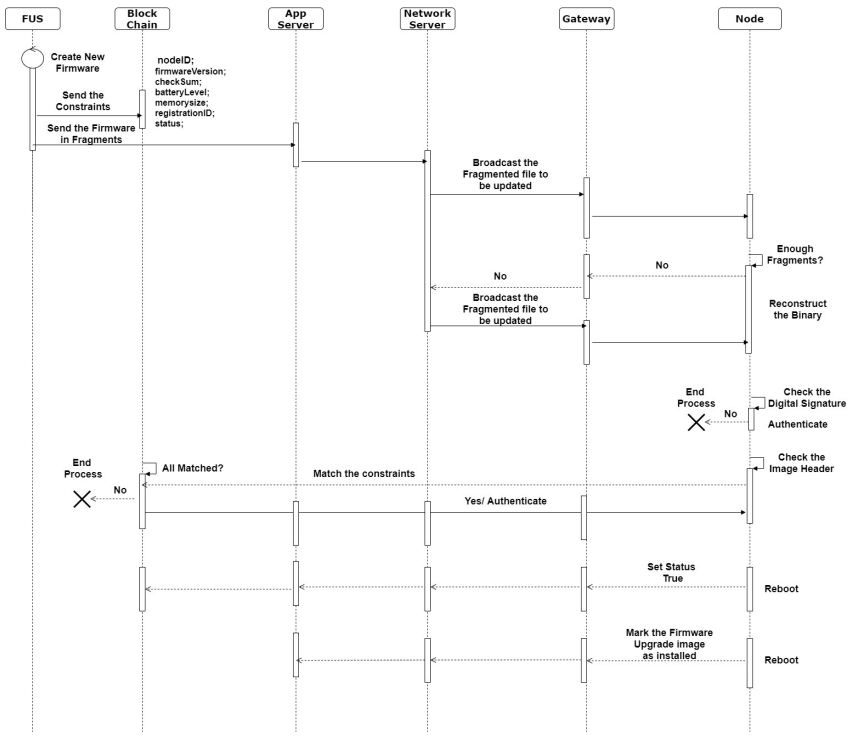



Fig. 2: Flow Diagram

A flow diagram (as shown in Fig.2) illustrates the process used by FUOTA which has been enhanced with the use of a BC to enable Blockchain Updates Over the Air (BUOTA), as follows:

- Step 1 - Firmware Update Server (FUS): During this step FUS creates the new compressed firmware image. It then selects the number of fragments, redundancy and the erasure code. The new image contains a header (at a basic level it contains the target end-devices hardware version, the current firmware version, image CRC, compression mechanism used by the image, etc.) and a digital signature which uses a private/public key scheme to authenticate the image. The image is signed using the FUS private key and creates the fragmented file. FUS also creates the constraints that need to be sent to BC, including *nodeID, firmwareVersion, checksum, battery level, memory size, registrationID*, and *status*. We explain each one of the aforementioned parameters in subsequent sections.
- Step 2 - In this step FUS sends the generated data to the Application Server (AS) and the constraints to the BC. AS also negotiates with the Network Server (NS) a Class distribution window (in our study we focus on Class-C). Class-C distribution negotiated window parameters include: the list of end-devices, the size of the fragment file to send, time criticality, coding redundancy, etc. In addition, in this paper we examine multicast distributions so the list of end-devices is already known.
- Step 3 - In step 3, the FS configures the Class-C multicast group using an applicative unicast downlink for all end-devices that require an update. In the background the end-devices have their clocks synchronized to the network's clock because absolute time is used to define the session start.
- During this step NS initiates the fragmentation session setup command. This command contains a freely assigned descriptor such that end-devices can autonomously check whether the received image is applicable to them in order to know where to store it.
- Step 4 - NS sends the fragmented file to the Gateways and the Gateway broadcasts the fragment file to end-devices to be updated. As soon as an end-device (ED) has received enough fragments (specified by the FUS), it reconstructs the binary image.
- Step 5 - In this step, the ED authenticates the sender (FUS) and the digital signature of the image by checking the sender's FUS public key stored in all end-devices. This step serves as an integrity test of the firmware upgrade image during the FUOTA process by LoRa Alliance.
- Step 6 - ED checks the image header to verify that:
  - the image is compatible with the ED hardware; and that,
  - the image is compatible with the firmware version currently running on the end-device.
- Step 7 - ED communicates via Gateway to BC to verify

the constraints. At the same time, BC matches, constraints and sends a verification for execution.

- Step 8 - During this step the ED application marks the firmware image as "ready". This means that the image will be installed by the bootloader during next reset. The application is responsible for this as if the reboot decision needs to happen immediately, postpone it to a later time, or wait an input from the firmware management command to reboot.
- Step 9 - After reboot, bootloader checks the availability of a firmware upgrade image and finds the new one. Bootloader checks again the CRC of the image and decompresses the image. The decompression method may vary, as it might happen directly in-place, and the decompressed image will overwrite the previous firmware image, or in a currently unused memory space. In the first case, the update is performed transitionally, thus the flash memory is updated per page, during this process the page's content that is written is verified, and the address of the next page is written to NVM. LoRa Alliance introduced this step to guarantee that the process will resume where it was left at the next reset even if the end-device crashes or power is interrupted during the update process)
- Step 10 - Bootloader marks the firmware upgrade image as installed and the ED sends to the BC the new status, the installed firmware version, the installation's timestamp as well as a unique ID which can be used to enable roll-back. The aforementioned data are recorded on the proposed Smart Contract (as shown in Alg. 1).
- Step 11 - Finally, ED sends a notification to FUS as an update status (success or failure).

The parameters that are included in the BC and smart contract respectively are outlined below:

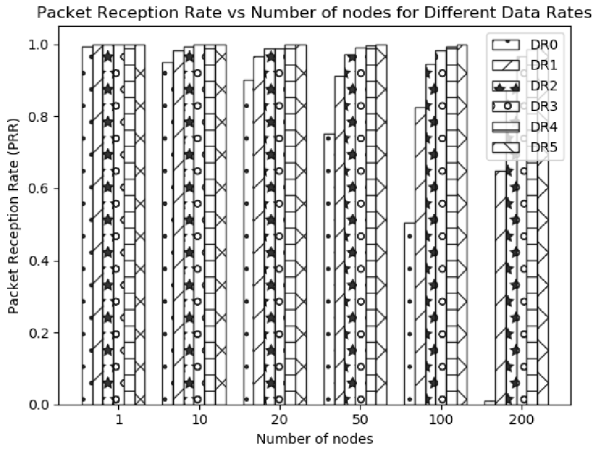
- *nodeID*, refers to the unique number of each ED.
- *FirmwareVersion*, contains the current firmware version of each ED. This parameter is used to ensure that a firmware version is applicable to a specific ED.
- *Checksum*, is used to verify that the MD5 hash originated from the NS is the same as with the one to be installed in the ED.
- *Batterylevel*, ensures that the ED has the necessary power resources to undergo the specific update. The minimum battery level is defined by the developer of the new firmware.
- *Memorysize*, refers to a constraint which can ensure that the ED has the appropriate memory size to accept the new firmware update to avoid overloading. The memory size of the new firmware is also defined by the developer.
- *registrationID*, provides a unique id to be used to enable roll-back in case of an unsuccessful firmware update.
- *Status*, is used to ensure that the firmware update was completed either successfully or unsuccessfully. BC gets a response from the ED after a reboot.

**Algorithm 1** Pseudocode for the *FirmwareUpdate* smart contract

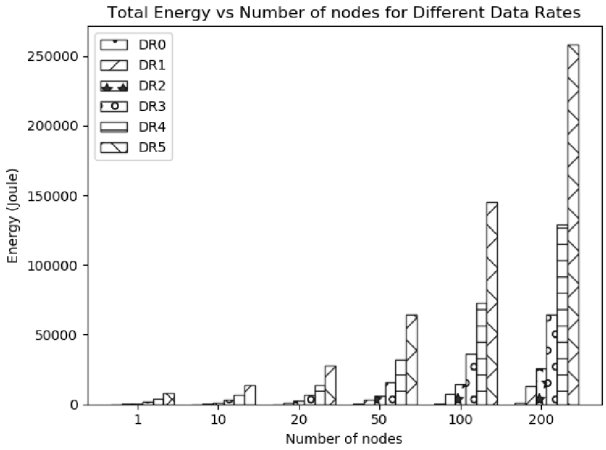
```
1:  contract FirmwareUpdate
2:
3:    struct rF {nID, FM, CS, BL, MS, RID, status}
4:    struct rF {nID, FM, time, status, nUUID}
5:
6:    rFR ← rF[]
7:    nSR ← nS[]
8:    aNR ← uint[]
9:    aURID ← uint[]
10:   aUP ← uint[]
11:
12:   function registerNewFirmware(uint _nID, string _FM,
13:                                 string _CS, uint _BL, uint _MS)
14:     if (msg.sender == contract.owner) then
15:       s ← rFR.length++
16:       rFR[s-1].nID ← _nID
17:       rFR[s-1].FM ← _FM
18:       rFR[s-1].CS ← _CS
19:       rFR[s-1].BL ← _BL
20:       rFR[s-1].MS ← MS
21:       rFR[s-1].status ← false
22:       rFR[s-1].RID ← s+10000
23:       aNR.push(rFR[s-1].nID)
24:       aURID.push(rFR[s-1].RID)
25:     end if
26:   end function
27:
28:   function updateNodeStatus(uint _checkRID, bool _status)
29:     if (msg.sender == contract.owner) then
30:       index ← 0
31:       s2 ← nSR.length++
32:       for (i ← 0 to rFR.length)
33:         if (rFR.RID == _checkRID) then
34:           index ← i
35:           return index
36:         end if
37:       end for
38:       nSR[s2-1].nID ← rFR[index].nID
39:       nSR[s2-1].FM ← rFR[index].FM
40:       nSR[s2-1].time ← now()
41:       nSR[s2-1].nUUID ← rFR[index].RID
42:       nSR[s2-1].status ← _status
43:       rFR[index].status ← _status
44:       aUP.push(nSR.nUUID)
45:     end if
46:   end function
47:
48:   function readFirmwareVersion(uint nID) returns (...)
49:     index ← i
50:     for (i ← 0 to rFR.length)
51:       if (rFR.RID == nID) then
52:         index ← i
53:         return index
54:       end if
55:     end for
56:     return (rFR[index].nID, rFR[index].FM, ...)
57:   end function
58: contract end
```

## IV. Evaluation

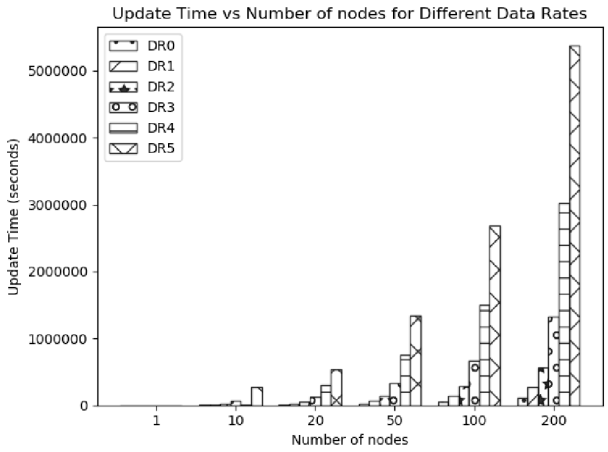
In this section, we evaluate the firmware update process using the FUOTASim simulator [18]. A detailed description of the simulator can be found in [19]. The metrics that were used for the evaluation are the *Packet Reception Rate (PRR)*, the *Total Network Energy* required, the *Update Time* and, finally, the *Update Efficiency* (nodes that were updated). The evaluation was performed with aim to analyse the performance of the firmware update process during the multicast transmission phase using different number of end devices and different size of firmware. Each simulation was run 1000 times with different random seeds. In all the simulations, we used Class C messages and one gateway node.



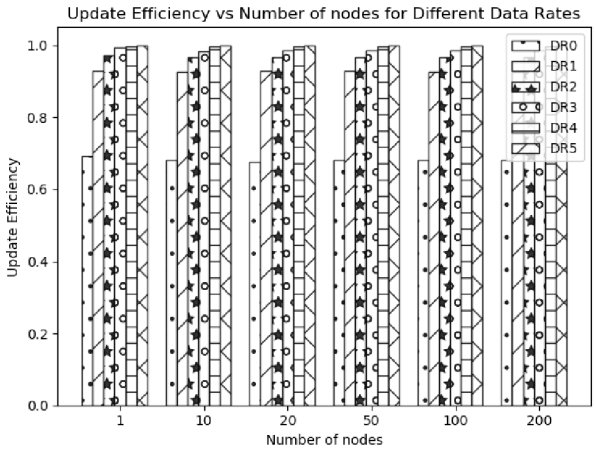

(a) Packet Reception Rate vs Number of Nodes for Different Data Rate




(b) Total Energy vs Number of Nodes for Different Data Rate




(c) Update Time vs Number of Nodes for Different Data Rate




(d) Update Efficiency vs Number of Nodes for Different Data Rate

Fig. 3: Evaluation for different number of nodes and Data Rate and Firmware=50k

Furthermore, the fragment size is set to the maximum payload supported by each data rate. Clock synchronization, multicast session setup and fragmentation session is out of the scope of this paper.

### *A. Number of Nodes Evaluation*

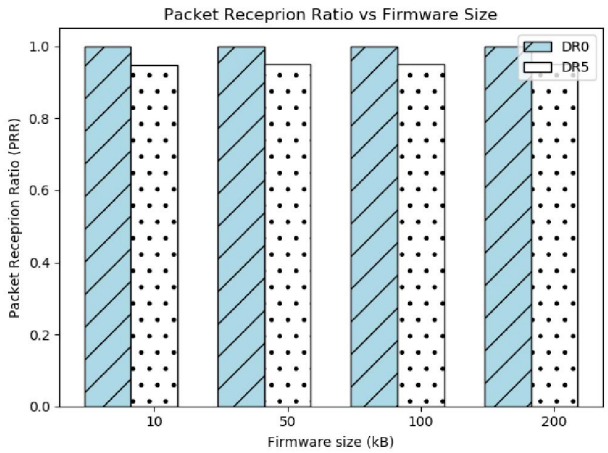
In this subsection, we investigate how the number of nodes in the network affects the performance of the firmware update. For the simulations, we used a fixed size of firmware equal to 50KB and we were changing the number of nodes and the data rate.

Fig. 3 shows the results. We observe that as the number of nodes is increasing the Packet Reception Rate in decreasing. The impact of the number of nodes increment is higher for DR5 and decreases as we move to DR0. This can be explained due to the spreading factor used in each data rate. SF12 provides higher Packet Reception Rate comparing to SF7. Furthermore, we observe a proportional increment of the total network energy consumed with the increment of the number of nodes.

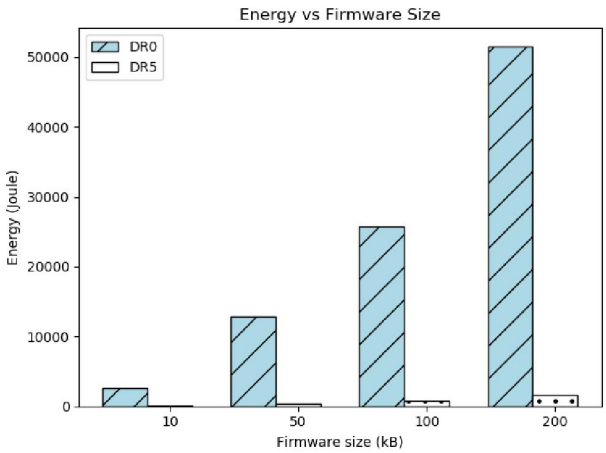
Fig. 3(c) shows the total update time required for all nodes to receive the firmware update. We observe that the update time is proportional to the data rate and the number of nodes. In case of DR0, due to the high airtime of spreading factor SF=12 there is a huge increment of the update time. Based on the evaluation, in order to update the firmware of a network consisted of 200 nodes and using one gateway, 62 days are required. If we use DR5, update time is reduced to 1.5 days. In case of 100 nodes, the update time is reduced to 31 days and 15 hours accordingly. Obviously, these update delays are not acceptable for the majority of applications since during the update phase the normal operation of the device is paused.

Finally, Fig. 3(d) shows the update efficiency for different data rates and different network size. We observe that using DR0 all nodes are updated where the number of nodes updated when using DR5 is 68%. We conclude that the update efficiency is not related to the number of nodes but is related to the data rate used.

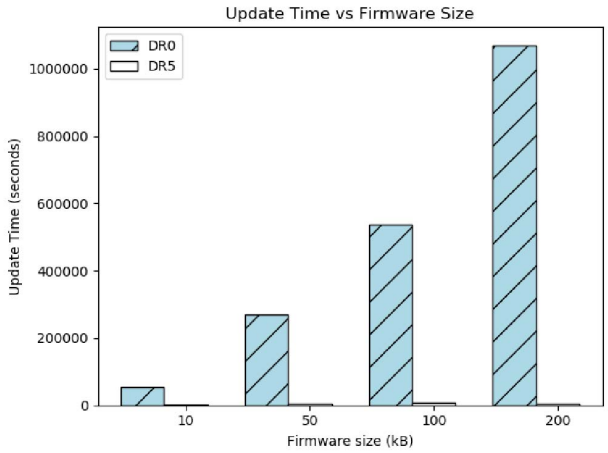
The evaluation in this subsection clearly indicates that there is a huge trade-off between the update time, the update



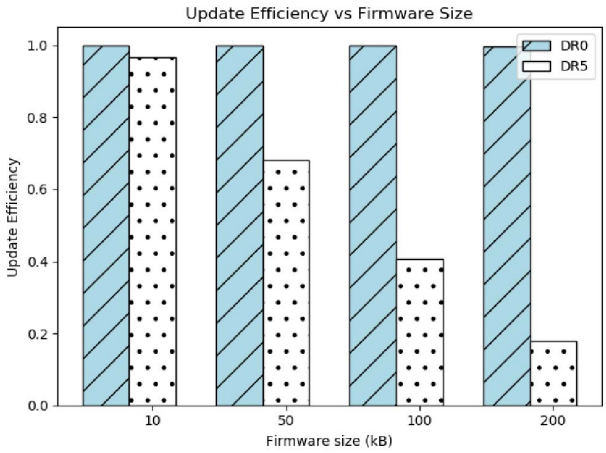

(a) Packet Reception Rate vs Firmware Size for DR0 and DR5 with 10 nodes.




(b) Total Energy vs Firmware Size for DR0 and DR5 with 10 nodes.




(c) Update Time vs Firmware Size for DR0 and DR5 with 10 nodes.




(d) Update Efficiency vs Firmware Size for DR0 and DR5 with 10 nodes.

Fig. 4: Evaluation using different firmware sizes.

efficiency, the energy consumption and the data rates used. In order to reduce this trade-off and achieve better results multiple gateways could be used.

### *B. Firmware Size Evaluation*

In this subsection, we investigate how the firmware size affects the firmware update performance. The results of the evaluation are shown in Fig. 4.

Fig. 4(a) shows the Packet Reception Rate for DRO and DR5 for firmware sizes of 10KB, 50KB, 100KB and 200KB. Based on the results, we conclude that PRR is depended on the data rate used and not on the firmware size.

In Fig. 3(b) we observe the total network energy for different firmware sizes. When using DR0 the energy consumption is much more higher than DR5 (32 times more). Fig. 3(c) shows the update time for the different firmware sizes. We observe that the update time is proportional to the firmware size. Furthermore, comparing the update time of DR0 is again 32 times more than the update time of DR5. Both figures indicates the same behaviour. This is explained due to the higher airtime of SF12 and the higher number of fragments sent in case of DRO (max payload 55 bytes vs 222 bytes of DR0).

Finally, Fig. 4(d) shows the update efficiency for different firmware sizes. We observe that DR0 offers 100% of update efficiency where in case of DR5 we observe reduction of the update efficiency as the size of the firmware increases. More specifically, the update efficiency using DR5 is reduced from 96% for firmware size of 10k to 18% for firmware size of 200k.

Again, there is a trade-off between the metrics and the firmware size which indicates that the use of more gateways will be beneficial for the firmware update.

### *C. Blockchain Evaluation*

In regards to the blockchain operation, the proposed blockchain approach was deployed and evaluated on the Ethereum Ropsten TestNet. The list of the executed transactions can be found on the following smart contract address 0x216ae7bae3bb4150077fbd4e53904b95fdacbae5.

Table II outlines the min values of the Gas Limit and Gas Price that are required for the deployment of the smart contract as well as for the execution of the system's core functions. Costs are calculated based on the current price of Ethereum[1].

TABLE II: Gas limits and prices for contract deployment and function calls.

| | **Gas limit** | **Gas price (Gwei)** | **Cost (USD)** |
|---|---|---|---|
| Smart contract deployment deployment | 1,821,902 | 1 | 0.383 |
| registerNewFirmware(...) | 273,330 | 1 | 0.057 |
| updateNodeStatus(...) | 232,424 | 1 | 0.048 |

As it can be observed from Table II for the deployment of the smart contract on the Ethereum blockchain 0.383 USD are required and for executing core functionalities a max amount of 0.057 USD is needed.

Table III outlines the execution times of the core functions. Based on our findings it takes less than 10s to register a new firmware update on the Blockchain and less than 8s to verify the update of the IoT device. High execution times can be significantly reduced when increasing functions' Gas prices.

TABLE III: Functions' execution time

| | **Times (sec)** |
|---|---|
| Smart contract deployment | <10 |
| registerNewFirmware(...) | <8 |
| updateNodeStatus(...) | <1 |

## V. Conclusion

Firmware updates fix security vulnerabilities and are considered to be an important building block in securing IoT devices. This challenge remains in the interest of researchers from the community that have not yet standardized the process. In this paper, we present a firmware update procedure based on FUOTA that utilizes blockchain to increase security during the updating process. Since we assume the usage of LoRa technology, we performed a firmware update evaluation in order to extract useful conclusions on the reliability and effectiveness of the procedure using LoRa. Based on the evaluation, we conclude that there is a trade-off between network scalability, the firmware update size, and other metrics like: data rate, update efficiency, update time and energy consumption. Therefore, there is a need for using more that one gateways that they will collaborate to increase the reliability and the performance of the firmware update process. For future work, we intend to investigate the performance of the proposed firmware update process using more gateways. Finally, we plan to evaluate the proposed blockchain solution under specific network attacks.

## VI. Acknowledgements

Dr. Vassiliou's acknowledges the support of the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 739578 and the Government of the Republic of Cyprus through the Directorate General for European Programmes, Coordination and Development.

[1]at the time of execution ETH price was 210 USD

## References

[1] Ponemon Institute, *The 2019 Study on the Cyber Resilient Organization.*, 2019.

[2] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, pp. 80–84, 01 2017.

[3] C. Ioannou and V. Vassiliou, "Security Agent Location in the Internet of Things," *IEEE Access*, vol. 7, pp. 95 844–95 856, 2019.

[4] Z. Zinonos, P. Christodoulou, A. S. Andreou, and S. A. Chatzichristofis, "Parkchain: An iot parking service based on blockchain," in *15th Int. Conference on Distributed Computing in Sensor Systems, DCOSS 2019, Santorini, Greece, May 29-31, 2019*. IEEE, 2019, pp. 687–693.

[5] W. A. Amiri, M. Baza, K. Banawan, M. Mahmoud, W. Alasmary, and K. Akkaya, "Towards secure smart parking system using blockchain technology," in *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*, 2020, pp. 1–2.

[6] A. Hahn, R. Singh, C. Liu, and S. Chen, "Smart contract-based campus demonstration of decentralized transactive energy auctions," in *2017 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 2017, pp. 1–5.

[7] J. Qiu, X. Liang, S. Shetty, and D. Bowden, "Towards secure and smart healthcare in smart cities using blockchain," in *2018 IEEE International Smart Cities Conference (ISC2)*, 2018, pp. 1–4.

[8] G. S. Aujla, M. Singh, A. Bose, N. Kumar, G. Han, and R. Buyya, "Blocksdn: Blockchain-as-a-service for software defined networking in smart city applications," *IEEE Network*, vol. 34, no. 2, pp. 83–91, 2020.

[9] B. Lee and J.-H. Lee, "Blockchain-based secure firmware update for embedded devices in an internet of things environment," *The Journal of Supercomputing*, vol. 73, no. 3, pp. 1152–1167, 2017. [Online]. Available: https://doi.org/10.1007/s11227-016-1870-0

[10] M. Baza, M. Nabil, N. Lasla, K. Fidan, M. Mahmoud, and M. Abdallah, "Blockchain-based firmware update scheme tailored for autonomous vehicles," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, 2019, pp. 1–7.

[11] M. Steger, A. Dorri, S. S. Kanhere, K. Römer, R. Jurdak, and M. Karner, "Secure wireless automotive software updates using blockchains: A proof of concept," in *Advanced Microsystems for Automotive Applications 2017*, C. Zachäus, B. Müller, and G. Meyer, Eds. Cham: Springer International Publishing, 2018, pp. 137–149.

[12] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an optimized blockchain for iot," in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, ser. IoTDI '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 173–178. [Online]. Available: https://doi.org/10.1145/3054977.3055003

[13] N. S. Mtetwa, P. Tarwireyi, A. M. Abu-Mahfouz, and M. O. Adigun, "Secure firmware updates in the internet of things: A survey," in *2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, 2019, pp. 1–7.

[14] K. Zandberg, K. Schleiser, F. J. Acosta Padilla, H. Tschofenig, and E. Baccelli, "Secure firmware updates for constrained iot devices using open standards: A reality check," *IEEE Access*, vol. PP, pp. 1–1, 05 2019.

[15] A. Yohan and N.-W. Lo, "An over-the-blockchain firmware update framework for iot devices," in *2018 IEEE Conference on Dependable and Secure Computing (DSC)*, 12 2018, pp. 1–8.

[16] LoRa Alliance Technical Committee, "Lorawan 1.0.3 specification," Tech. Rep., 2018. [Online]. Available: https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdf

[17] FUOTA Working Group of the LoRa Alliance Technical Committee, "Fuota process summary," Tech. Rep., 2018. [Online]. Available: https://lora-alliance.org/sites/default/files//remotemulticastsetupv0.0.pdf

[18] K. Abdelfadeel. Fuotasim simulator. [Online]. Available: https://github.com/kqorany/FUOTASim

[19] K. Abdelfadeel, T. Farrell, D. McDonald, and D. Pesch, "How to make firmware updates over lorawan possible," 2020. [Online]. Available: https://arxiv.org/abs/2002.08735