Firmware Updates Over The Air Mechanisms for Low Power Wide Area Networks: A Review

Mompoloki Pule[1]
[1]*Department of Electrical Engineering*
Tshwane Univeristy of Technology
Pretoria, South Africa
calvin.pule@gmail.com

Adnan M. Abu-Mahfouz[1, 2]
[2]*Council for Scientific and Industrial Research (CSIR)*
Pretoria, South Africa
a.abumahfouz@ieee.org

***Abstract*—Wireless Sensor Networks (WSN) provide an affordable infrastructure that allows numerous control and data collection processes to be conducted with high spatial and temporal resolution. The recent arrival of Low Power Wide Area Networks (LPWAN) has given rise to WSN solutions with long range capabilities and a high level of autonomy. However, this has come at a cost of having low data rates and poses quite a huge challenge on implementation of high bandwidth applications such as Firmware Updates Over the Air (FUOTA). Firmware updates are essential throughout the lifetime of a product for several reasons including post deployment bug fixes, introduction of new security features and implementation of performance optimizations. This paper proposes a generic and hypothetical approach to designing an efficient FUOTA mechanism for WSN/LPWAN. The paper also presents a review of past literature on similar works and how best existing approaches have addressed the challenges of remote sensor node reprogramming.**

Keywords—WSN, LPWAN, FUOTA, and Reprogramming

I. INTRODUCTION

Wireless Sensor Networks (WSN) provide a promising infrastructure for numerous applications ranging from asset tracking, agricultural automation, environmental monitoring, telemedicine and military surveillance. With quite a simple architecture, WSN have allowed control and monitoring processes to be conducted remotely, in real-time, with minimal human intervention and at a relatively low cost. A typical WSN system consists of two main subsystems namely nodes and gateways. Nodes are devices equipped with sensing, processing and wireless communication capabilities, and together perform a collaborative measurement process. Gateways provide a means to collect data from nodes and relay it to storage centers, analytics engines and data visualization systems [1]. The rapid deployment and reduction in cost of broadband internet has made it affordable to connect, monitor and control WSN systems through the public network, leading to a technological revolution termed Internet of Things (IoT) [2], [3]. Fig 1 is an illustration of a typical IoT system.

To support remote and real-time data collection with a high level of autonomy, most applications require nodes that are capable of long-range communication while consuming very little power such that they can operate for several years on battery power. Conventional WSN communication technologies such as Wi-Fi, Bluetooth, ZigBee and traditional cellular technologies have not been able to accommodate these requirements effectively [4], [5]. Low Power Wide Area Networks (LPWAN) have since emerged to complement the inadequacies. LPWAN support communication over long distances while consuming very little power. These benefits come at a cost of having very low data rates, which impose design restrictions when trying to implement high bandwidth applications such as Firmware Updates Over the Air (FUOTA).

This paper presents a hypothetical approach to effective implementation of FUOTA for WSN/LPWAN. The paper also reviews past literature on similar works and evaluates how proposed methodologies have addressed the challenges of FOUTA. Research gaps and potential future directions are also identified.

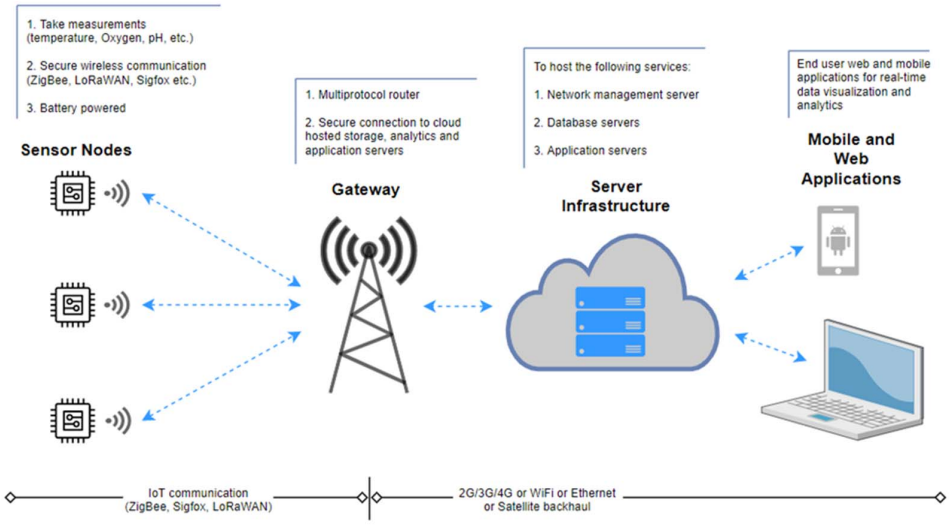

Fig 1. Typical WSN/IoT System Architecture

The remainder of this paper is organized as follows: Section 2 describes characteristics, requirements and challenges of LPWAN. Section 3 introduces FUOTA and describes the challenges and requirements for its effective implementation over WSN/LPWAN. Sections 4 and 5 provides a review and a discussion of previous work on FUOTA over WSN/LPWAN, respectively. Sections 6 and 7 discusses potential research gaps on designing effective FUOTA mechanisms for WSN/LPWAN and concludes the paper.

II. Low Power Wide Area Networks

A. Characteristics of LPWAN

LPWAN are relatively affordable and can transmit data over kilometers of range with low power consumption. Among them, Sigfox, LoRa/LoRaWAN and NB-IoT have become leaders of the LPWAN technology space and are competing for large scale IoT deployment [6]. Sigfox was developed in 2010 by a French company called Sigfox which is also its network operator. At the physical layer, Sigfox uses Binary Phase Shift Keying (BPSK) modulation in an Ultra-Narrow Band (UNB) carrier. Sigfox uses unlicensed ISM bands such as 868MHz in Europe, 915MHz in North America and 433MHz in Asia. LoRa was developed in 2009 by a French company called Cycleo and was later purchased by an American company called Semtech. LoRa uses a modulation technology called Chirp Spread Spectrum (CSS), that spreads a narrow-band signal over a wider channel bandwidth. Like Sigfox, LoRa uses similar unlicensed ISM bands. NB-IoT is a narrow-band LPWAN technology that has been standardized and specified in Release 13 of the 3GPP in June 2016. NB-IoT uses Quadrature Phase-Shift Keying (QPSK) modulation and operates on licensed frequency bands like GSM and LTE[6], [7]. TABLE 1 provides a summary of the main characteristics of the leading LPWAN technologies.

B. Requirements and challenges of LPWAN

Nonetheless, LPWAN come with some tradeoff challenges such as very low data rates and uplink-oriented communication. In addition, most of LPWAN operate in license free bands and must therefore adhere to duty cycle limitations while suffering the effects of radio interference [4]. This poses quite a huge challenge for applications with considerable bandwidth requirements such as FUOTA.

III. Firmware Updates Over-The-Air (FUOTA)

A. Characteristics of FUOTA

Conventionally, FUOTA was termed Over-The-Air Programming (OTA/OTAP), and generally referred to a process of updating device firmware over a wireless communication medium [8]. Firmware preloaded on WSN nodes may occasionally need to be updated for several reasons including post deployment bug fixes, security patches, introduction of new features and performance enhancements. In this regard, FUOTA has been considered most ideal especially when dealing with physically inaccessible, large scale WSN/LPWAN deployment scenarios where manual methods would be very costly and time consuming to realize.

B. Challenges of conducting FUOTA for WSN/LPWAN

Since WSN/LPWAN nodes are always designed to be simple and affordable so as to maintain their traditional small form factor; they are inherently resource constrained in terms of computational power, energy/battery life, communication bandwidth and memory. A study in [9] suggests that of all the constraints, energy is the most critical and its consumption can be categorized into three parts namely: 1) consumption by the sensor transducer, 2) consumption by microprocessor computation, and 3) consumption by the communication module. Authors of [5], [9] found that communication is more energy consuming than microprocessor computation as transmitting one bit wirelessly can easily consume as much energy as executing 800 to 1000 instructions. Communication overheads introduced by FUOTA could have a very huge impact on the lifespan of devices. When planning a large scale WSN/LPWAN deployment, it is very critical to have an effective strategy for conducting FUOTA.

TABLE 1. Characteristics of LPWAN technologies [6], [7]

	Sigfox	**LoRa/LoRaWAN**	**NB-IoT**
Modulation	BPSK	CSS	QPSK
Frequency	Sub-GHz ISM: EU868, US918, AS433	Sub-GHz ISM: EU868, US918, AS433	Licensed LTE frequency band
Bandwidth	UL: 100/600 Hz DL: 1.5 kHz	125 kHz and 250 kHz	200 kHz
Data rate	UL: 100/600 bps DL: 600 bps	0.3-5 kbps	UL: 158.5 kbps
Range	10 km (urban) 40 km (rural)	5 km (urban) 20 km (rural)	1 km (urban) 10 km (rural)
Maximum payload size (bytes)	UL: 12 DL: 8	250	13
Error correction	UL: CRC-16 DL: CRC-8	CRC-8/16	CRC
Bidirectional	Limited/Half-duplex	Yes/Half-duplex	Yes/Half-duplex
Topology	Star	Star	Star
Localization	Yes (RSSI)	Yes (TDOA)	No (under specification)
Security	No (must be implemented in application layer)	Yes (AES 128b)	Yes (LTE encryption)
Standardization	Collaborating with ETSI to standardize the network	LoRa-Alliance	3GPP

C. Requirements of effective FOUTA for WSN/LPWAN

To analyze FUOTA for WSN/LPWAN, we have broken down the system into the following subset of IoT devices and services.

Nodes: These are devices to be updated through FUOTA.

Gateways: These are devices which provide a means for nodes to communicate with servers and vice versa.

Network server: Provides services that regulate how devices can access and share network resources.

Application server: Provides services to route device traffic to appropriate applications and vice-versa.

Update server: Provides the core services for implementing FUOTA over WSN/LPWAN.

Services can be viewed as independent entities or as combined where necessary.

The problem of developing an efficient FUOTA mechanism can be broken down into several processes and protocols, some of which can be addressed individually or in combination where necessary.

Firmware differencing: This is a process for computing the difference between new and old firmware images to reduce the amount of data that needs to be transmitted to perform an update.

Data encoding: This is a process for encoding the difference into a script that can easily be reconstructed by the intended recipient.

Data fragmentation: This is a process for breaking down the update script into sizeable packets/fragments that can effectively be transmitted through the communication channel.

Update session scheduling: This is a protocol that is meant to prepare appropriate devices to get ready for the update session.

Packet dissemination: This is a protocol to effectively distribute update packets to the intended recipients.

Error correction and recovery: This protocol is for recovering from packet losses and errors.

Firmware reconstruction: This is a process for decoding the update script to generate the new firmware image.

Update session termination: This is a protocol to effectively conclude the update session.

Client reprogramming: This is a process to rewrite the old firmware image with the newly reconstructed one.

Security: These are protocols for establishing trust and authenticating communicating entities of the FUOTA mechanism. The protocols must ensure integrity of the update script and the reconstructed firmware image, before reprogramming.

Fig 2 shows how the devices, services, processes and protocols would typically interact during an active FUOTA process.

To address the challenges of FOUTA, it is imperative to come up with a comprehensive approach to assess the effects of each process and protocol on the overall performance of the update mechanism. Below are 6 generic and hypothetical design approaches that we propose for effective implementation of efficient FUOTA for WSN/LPWAN:

- All FUOTA protocols and processes must work within the resource constraints of WSN/LPWAN.
- All FUOTA processes and protocols must be executed effectively and efficiently to not hinder sensor nodes from executing their core application.
- All FOUTA protocols must be secure enough to not compromise sensor nodes and the network.

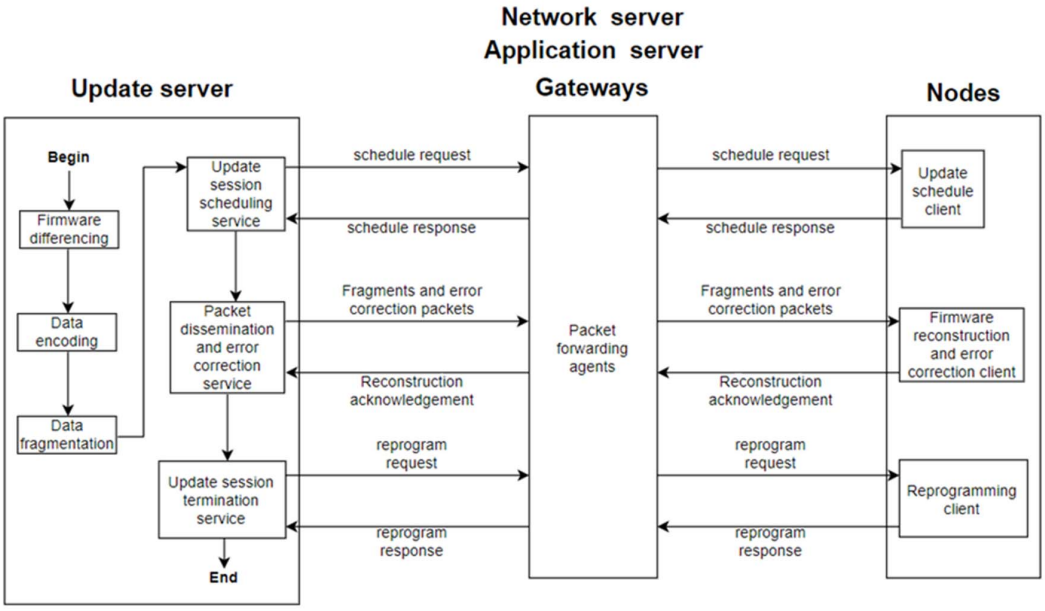

Fig 2. Typical FUOTA Flow Diagram

- All FOUTA protocols and processes must consume as little energy as possible to preserve the lifetime and autonomy of sensor nodes.
- All FUOTA protocols must have the ability to effectively recover from packet losses or errors.
- Lastly, integration of FUOTA should strive to be seamless and not cause major changes to the existing code base of the underlying system.

IV. Review of FUOTA Mechanisms for WSN/LPWAN

Past literature on FUOTA mechanisms for WSN/LPWAN is reviewed and evaluated based on how best authors have addressed the challenges of FOUTA with respect to the processes and protocols previously suggested.

Researchers at Berkeley, University of California [8], [10] developed a very basic OTA protocol for WSN called Crossbow Network Programming (XNP). XNP was developed as a single-hop protocol and was implemented and tested on TinyOS release version 1.1. However, the protocol was considered not to be scalable since it could only disseminate the program code to nodes within the vicinity of the host machine or gateway. XNP was also considered inefficient for distributing the complete program code instead of just the difference. In addition, the protocol did not address issues of security or effectiveness of the client reprogramming agent, all of which have an impact on the performance of the update process.

Reijers and Langendoen [11] presented an efficient code distribution mechanism for WSN. Authors developed a UNIX diff-like algorithm to find the shortest edit script to convert an old firmware image to a new one. The algorithm was optimized with REPAIR and PATCH command operations to produce a shorter edit script. The script was then fragmented into 64-byte packets to be transmitted effectively through the communication channel. Authors developed an initialization phase to inform nodes of the beginning of the reprogramming cycle, so they make necessary preparations. During transmission, nodes would build the new code image in external EEPROM by sequentially processing the packets bottom up and creating necessary gaps for any missing information. To ensure proper reconstruction of the firmware image, authors developed a verification protocol that allows nodes to request for the missing binaries from their neighbors. Upon successful code reconstruction, nodes would be given the instruction to start running the update process by loading a small piece of code into RAM that gives the instruction to copy the image from EEPROM to flash and reboot on completion. On startup, the nodes would then run the new code. Authors implemented and tested their scheme on EYES nodes, featuring Texas Instruments MSP430F149 microcontrollers with 2KB RAM and 60KB flash memory. The nodes were also equipped with RFM TR1001 868.35MHz hybrid wireless transceivers with 115kbps maximum data rate, and 256KB external EEPROM. Experimental results showed a significant improvement as compared to simply transferring the complete binary code, and further showed that the application code only had to be disrupted a few times when transferring the new code. However, the proposed data recovery protocol imposed considerable communication overheads on sensor nodes by having them request for missing packets and sending acknowledgements. This approach is also applicable only in multi-hop network topologies. Furthermore, the protocol did not implement any security mechanism to negotiate trust to authenticate update files, or to even check the integrity of the reconstructed code file. This introduces a vulnerability for attackers to introduce rogue firmware that could break or cause the nodes to malfunction.

Jeong, *et al.* [10] extended the network programming implementation of TinyOS release 1.1 by developing a mechanism that transmits incremental changes from an old firmware image to a new one. The difference was generated using an optimized block level comparison algorithm called Rsync. The difference was encoded into a script with DOWNLOAD and COPY commands, to show which sections would be added to the new image and which would be copied from the old image, respectively. The script was then fragmented, transmitted across the communication medium and stored in external flash by the nodes. The host program would then query the nodes to confirm receipt of all fragments, so they could request for retransmission of those that are missing. The host would then send a decode command when all nodes have received all fragments, after which they would start rebuilding the new firmware code in external flash. The algorithm was implemented and tested on TinyOS and was compared to fixed block comparison. Results showed a speed-up of 9.1 for changing a constant and that of 2.1 to 2.5 for changing a few lines in the source code. Nonetheless, the proposed solution had no initialization procedure to make the nodes aware of and help them prepare for the update session. In addition, the proposed error correction mechanism imposed considerable communication overheads on the nodes with retransmission requests and acknowledgements. Moreover, the solution did not address security concerns for ensuring the authenticity and integrity of the update script or the reconstructed firmware.

Hu, *et al.* [12] developed an algorithm called RMTD, for reprogramming WSN with minimal transfer data. RMTD used byte level comparison to find common segments between old and new code images, and employed a dynamic programming to find the optimal combination of COPY and DOWNLOAD commands to reconstruct the new code image. The algorithm's performance was evaluated by comparing its packet transmission overheads with those of the fine-tuned Rsync differencing algorithm [10] under the same experimental conditions. Results showed considerable reduction in data transfer for RMTD over Rsync of 93.25% and 59.82% when there are small changes in the source file and on overage, respectively. However, RMTD incurs additional communication and computational overheads for copying all common code segments including those that have maintained the same addresses in both firmware images.

Dong, *et al.* [13] developed a holistic over the air reprogramming mechanism for WSN called Elon, based on the TinyOS operating system. Elon minimized transmission overheads for both the TinyOS kernel services and the reprogramming protocol by introducing the concept of replaceable components. This was achieved through modifying of the nesC compiler to create a boundary between replaceable components and the OS kernel components. This isolation allowed nodes to be rebooted through software means to avoid the high energy consumption and data loss that is usually associated with a

hardware reboot. Elon placed replaceable components on RAM to avoid flash writes thereby prolonging the reprogramming lifetime of the nodes. Authors implemented and tested Elon on a testbed of 10 TelosB nodes running TinyOS 2.1, and experimental results showed considerable reduction in code size of about 120-389 and 18-42 times compared to Deluge and Stream, respectively. Elon also significantly reduced the loading cost of rebooting core OS components such as CTP, Drip and FTSP by 53.8%, 50.4% and 56.83%, respectively. Furthermore, Elon extended the reprogramming lifetime of the TelosB nodes by a factor of 2.3 compared to other approaches. Nonetheless, the mechanism was designed for and limited to Von-Neumann architecture based TinyOS sensor nodes and cannot be easily ported to other microcontroller platforms with different compiler and core architectures. Additionally, Elon is likely to incur communication overheads for not optimizing the payload size with firmware differencing between successive code images of replaceable components.

Munawar, *et al.* [14] developed Dynamic TinyOS to support efficient remote reprogramming of TinyOS based sensor nodes. The solution introduced extensions to support dynamic adaptation of the OS and application features by allowing users to define components to be kept modular in the binary executable for them to be easily replaced during runtime. This was achieved through modifying the TinyOS compiler NesC. Authors also developed a runtime system for sensor nodes called Tiny Manager, to handle storage and integration of new components with the system binary image. In addition, authors also suggested optimizations for ELF objects to reduce communication and processing overheads. The solution was implemented on the TelosB platform running TinyOS release 2.1, and its performance was compared to Deluge and Zephyr for different software change scenarios. Zephyr showed better performance in update size for small changes to the application code, but was outperformed by Dynamic TinyOS for component level changes. In terms of memory footprint, runtime performance overhead and energy consumption, the proposed solution showed significant improvement over Deluge. However, adaptations introduced by Dynamic TinyOs to compile parts of an application in isolation limits the compiler's optimization attempts to ensure coverage of the smallest possible memory footprint. Additionally, the solution does not offer a provision to update the runtime system, and will not be easily ported to other sensor networks with different microcontroller and compiler architectures since was specifically developed for TinyOS and the NesC compiler.

Wen, *et al.* [15] developed an OTA protocol that employs Parallel Diffusion Mechanism (PDM) to distribute code segments in a multi-hop, self-organized WSN environment. In PDM, forwarding nodes would be selected based on the distance between adjacent candidate nodes, and their transmission powers would be adjusted dynamically upon convergence of the algorithm. The authors used TOSSIM simulation environment, which runs TinyOS, to implement, test and evaluate PDM in comparison with other protocols such as ripple and sender selection protocol, under similar conditions. With increasing distance between nodes, PDM was observed to have the fastest execution process completion time. In addition, PMD also had the best overall network energy performance. Nonetheless, the protocol performed poorly under much dense WSN deployments, as it incured considerable communication overheads to converge and select appropriate forwarding nodes and transmit powers. The solution also did not address other critical protocol related features such security and error detection and recovery, which have a great impact on the overall energy performance on an OTA mechanism.

Pote, *et al.* [16] presented a performance evaluation of six symmetric key ciphers (Blowfish, RC6, RC2, AES (Rijndael), 3DES and DES) recommended for secure OTA programming of WSN. To assess the performance the algorithms, authors adopted an evaluation criterion that considered several factors such as: 1) the effect of changing file data types to text, audio and video; 2) the effect of varying packet sizes; 3) the effect of using different data encoding bases such as hexadecimal and base64; and 4) the effect of varying cryptographic key lengths. Experiments were conducted using a Laptop IV with a 2.4 GHz CPU. Results showed that there was no significant change for encoding data with hexadecimal or base64, or for using different file data types. In the case of varying the packet size, Blowfish was observed to perform better than other algorithms, followed by RC6 and AES. RC2 had the worst performance in execution time. Increasing the key length introduced a correspondingly noticeable increase in the power consumption and execution time. However, the authors work was limited only to performance evaluation of symmetric key ciphers than to investigate how they could be employed in OTA reprogramming. In addition, all experiments were conducted on a general-purpose computer with abundant compute resources than is possible in a typical WSN setup and may not necessarily yield the same results under constrained environments. Symmetric crypto systems are also not enough to address all the security requirements for OTA reprogramming of WSN.

Kachman and Balaz [5], [17] suggest that effective OTA reprogramming requires considering several factors such as firmware similarity improvements, differencing algorithms, delta file dissemination and update agent complexity. The authors developed an algorithm called delta generator (DG), which used XOR operation to compute the difference between two firmware images and encoded it into a delta file with COPY and ADD operations. The algorithm was designed for OTA reprogramming of devices without external flash, and authors suggested that the proposed optimizations reduced space complexity and lead to better execution times. In addition, the authors developed a basic energy consumption model for reprogrammed memories, and an encoding scheme that splits a delta file into three parts namely, header, operation data, and integrity check data. The header consisted of two numbers specifying the number of COPY and ADD operations, respectively. The operation data encoded addresses and bytes for each operation, and the integrity check data encoded a CRC-16 code that would be checked to verify the integrity of the delta file before application. The algorithm was implemented on an ATmega32U4 microcontroller with 32KB of self-programming flash memory and was tested with 7 firmware change cases and compared with the R3diff differencing algorithm, which is one of the best algorithms for generating deltas for firmware updates in external memory. Results showed an upgrade over R3diff with reduced delta file sizes and correspondingly reduced amount of page writes to flash memory. Nevertheless, the proposed solution and energy consumption model did not account for other processes and protocols such as delta fragmentation, update session

scheduling and termination, packet dissemination and error recovery, authentication and security; all of which have a significant contribution to the overall effectiveness and energy consumption of the OTA process. In addition, on the fly reprogramming limits the ability to check the integrity of the full firmware image before writing to program memory. The process may also result in device malfunction if it gets interrupted before completion and should therefore have a rollback option if possible.

Jongboom and Stokking [4], [18] presented a solution for enabling FUOTA over LPWAN. To update multiple devices at the same time, authors proposed scheduling a multicast session to get all devices to listen at the same time, frequency, data rate and security session. To minimize network traffic and save energy, authors suggested employing a linear patch format, where the network sends incremental changes instead of the full firmware image. To deal with packet loss, authors recommend low-density parity check coding (LDPC) error-correction mechanism. Upon receiving a complete update file, devices would calculate a checksum of the data and send it to the network through their own private secure sessions. The server would then compare this checksum with the data that it sent and would indicate the correctness of the checksum to each device individually through private secure sessions. As part of the response, the server would send a message integrity code (MIC) to guarantee data integrity to the devices. Authors suggested additional network and application layer security measures for each end device is to be programmed with a public key of the owner who is authorized to update its firmware, a manufacturer's universally unique identifier and a device type identifier. This was to allow creation of a secure firmware update manifest that contained a cryptographic hash of the update, a manufacturer and device type the update applies to, all signed with the manufacturer's private key. Single curve ECDSA/SHA256 were suggested for computing the cryptographic hash, since they are sufficiently secure and can efficiently be implemented on constrained devices. Authors also suggest limiting the multicast session with a lifetime based on a fixed number of messages to avoid draining devices with endless error correction packets. When the limit is reached, devices would revert to their power efficient mode and discard all data. A reference implementation of the proposed solution was demonstrated by ARM and The Things Industries, on top of LoRaWAN, at the LoRa-Alliance Annual Members Meeting (AMM) in 2017. They used custom boards created with Multi-Tech xDot radio modules and NXP FRDM-K22F MCUs and managed to update them with a 52KB full firmware image under 6 minutes. Nevertheless, the proposed solution would only be suitable for updating stationary devices, whose communication requirements can be determined with relative ease. The solution was only tested on a setup with dual MCU integration, which has more compute and memory resources than typical IoT nodes, and therefore may not be easily ported to most IoT device implementations. The work was more focused on the update protocol and security mechanisms and made very little effort to address the effectiveness of the differencing algorithm and the update agent.

V. DISCUSSION

TABLE 2 provides a summarized comparison of the reviewed literature. NULL implies there was limited, or no information provided for the process/protocol under consideration.

TABLE 2. Summary of reviewed literature

Ref.	Firmware differencing; encoding; and fragmentation	Update session scheduling; and termination	Packet distribution protocol	Error correction and recovery	Firmware reconstruction; & client programming agent	Security; and network topology
[8], [10]	NULL; NULL; NULL	NULL	NULL	NULL	NULL	NULL; Single-hop
[11]	UNIX diff-like algorithm; COPY, INSERT, REPAIR and PATCH commands; 64-byte fragments	Yes; Yes	NULL	Re-Tx requests and Acks	In external EEPROM; Piece of code loaded into RAM	NULL; Multi-hop
[10]	Rsync algorithm; COPY and DOWNLOAD commands; NULL	NULL; Yes	NULL	Queries, Re-Tx[a] requests and Acks[b]	In external flash; NULL	NULL; Single-hop
[12]	RMTD; COPY and DOWNLOAD commands; NULL	NULL; NULL	NULL	NULL	NULL; NULL	NULL; NULL
[13]	NULL; NULL; NULL (Compiler modification)	NULL; NULL	NULL	NULL	Software reboot mechanism	NULL; Multi-hop
[14]	NULL; NULL; NULL (Compiler modification)	NULL; NULL	NULL	NULL	Tiny Manager runtime system	NULL; Multi-hop
[15]	NULL; NULL; NULL	NULL; NULL	Parallel Diffusion Mechanism	NULL	NULL; NULL	NULL; Multi-hop
[16]	NULL; NULL; NULL	NULL; NULL	NULL	NULL	NULL; NULL	Symmetric key ciphers; NULL
[5]	Delta Generator (DG); COPY and ADD commands; NULL	NULL; NULL	NULL	NULL	On the fly in program memory	CRC-16 integrity check; NULL
[4], [18]	NULL; NULL; NULL	Yes; Yes	Multicast	LDPC, checksum and MIC	NULL; NULL	Symmetric key cipher, public key cypher and hash algorithm; Single-hop

[a] Retransmission, [b] Acknowledgements

Since communication is the most energy consuming attribute of WSN/LPWAN, authors made efforts to design FUOTA mechanisms with effectively minimized communication overheads. The first step was to introduce delta generation and transmit only incremental changes instead of the full firmware image. This was achieved through differencing algorithms and data encoding formats. Not much work was done in developing protocols for scheduling and terminating the update session. Most of the proposed packet distribution protocols assumed a multi-hop network topology and suggested use of intelligent routing techniques to send updates and/or missing packets through the network. Authors also suggested employing retransmission requests and acknowledgements for error correction and recovery, but this has a potential to degrade the performance of the network as it scales, since more collisions are likely to occur. Effective firmware reconstruction depends on the complexity of the adopted encoding scheme and authors have made several suggestions and recommendations for its implementation on both internal and external memory.

On the fly reprogramming is quite risky and could lead to node malfunction if not implemented properly. As a result, it is advisable to always couple it with a rollback option from external memory if possible. Most of the suggested protocols made very little effort to address security and how to properly implement it within WSN/LPWAN constraints. All proposed FUOTA mechanisms assumed a fixed IoT network deployment, and therefore cannot be easily extended to applications with mobile nodes such as transport and logistics tracking.

VI. Future Work

Delta generation, encoding and fragmentation processes can be further optimized to produce the shortest possible edit script to convert an old firmware image to a new one. All FUOTA related protocols such as update session scheduling/termination, packet distribution, error correction and recovery, can be further optimized to reduce data transmission requirements of sensor nodes, also bearing in mind the adopted network topology. It is essential for all the protocols to implement efficient security mechanisms to properly authenticate communicating entities and ensure data integrity and confidentiality. There is also a need to develop an effective FUOTA mechanism for mobile IoT nodes.

VII. Conclusion

Designing an efficient FUOTA mechanism for WSN/LPWAN requires a strategy to ensure the limited resources are used wisely. We proposed breaking down the process into several subprocesses and protocols that can easily be analyzed and optimized. These include delta generation, update scheduling and termination, packet distribution, error correction and recovery, security, data reconstruction and client reprogramming. We reviewed some of the existing work and identified potential gaps and future works.

References

[1] M. Pule, A. Yahya, and J. Chuma, "Wireless sensor networks: A survey on monitoring water quality," *J. Appl. Res. Technol.*, vol. 15, no. 6, pp. 562–570, Dec. 2017.

[2] D. Evans, "The Internet of Things - How the Next Evolution of the Internet Is Changing Everything," Cisco Internet Business Solutions Group (IBSG), 2011.

[3] Lopez Research LLC, "An Introduction to the Internet of Things (IoT)," 2013.

[4] J. Jongboom and J. Stokking, "Enabling firmware updates over LPWANs," in *Embedded World Conference*, 2018.

[5] O. Kachman and M. Balaz, "Effective Over-the-Air Reprogramming for Low-Power Devices in Cyber-Physical Systems," in *7th Doctoral Conference on Computing, Electrical and Industrial Systems (DoCEIS)*, 2016, pp. 284–292.

[6] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A Comparative Study of LPWAN Technologies for Large-Scale IoT Deployment," *ICT Express*, vol. 5, no. 1, pp. 1–7, 2018.

[7] Q. M. Qadir, T. A. Rashid, N. K. Al-Salihi, B. Ismael, A. A. Kist, and Z. Zhang, "Low Power Wide Area Networks: A Survey of Enabling Technologies, Applications and Interoperability Needs," *IEEE Access*, vol. 6, pp. 77454–77473, 2018.

[8] A. S. A. Quadri and B. O. Sidek, "An Introduction to Over-the-Air Programming in Wireless Sensor Networks," *Int. J. Comput. Sci. Netw. Solut.*, vol. 2, pp. 33–49, 2014.

[9] J. Sen, "A Survey on Wireless Sensor Network Security," *Int. J. Commun. Networks Inf. Secur.*, vol. 1, no. 2, pp. 55–78, 2009.

[10] J. Jeong and D. Culler, "Incremental Network Programming for Wireless Sensors," *Int. J. Commun. Netw. Syst. Sci.*, vol. 5, pp. 433–452, 2009.

[11] N. Reijers and K. Langendoen, "Efficient Code Distribution in Wireless Sensor Networks," in *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications*, 2003, pp. 60–67.

[12] J. Hu, C. J. Xue, Y. He, and E. H. M. Sha, "Reprogramming with Minimal Transferred Data on Wireless Sensor Network," in *IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*, 2009, pp. 160–167.

[13] W. Dong, Y. Liu, X. Wu, L. Gu, C. Chen, and † Zhejiang, "Elon: Enabling Efficient and Long-Term Reprogramming for Wireless Sensor Networks," in *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2010, pp. 49–60.

[14] W. Munawar, M. H. Alizai, O. Landsiedel, and K. Wehrle, "Dynamic TinyOS: Modular and Transparent Incremental Code-Updates for Sensor Networks," in *IEEE International Conference on Communications*, 2010, pp. 1–6.

[15] T. Wen, Z. Li, and Q. Li, "An Efficient Code Distribution Protocol for OTAP in WSNs," in *5th International Conference on Wireless Communications, Networking and Mobile Computing*, 2009, pp. 1–4.

[16] C. R. Pote, P. U. Tembhare, and M. G. Lade, "Secure Wireless Sensor Network Updates Using OTAP And Performance of Symmetric Encryption Algorithms on Power Consumption," *Int. J. Eng. Res. Appl.*, pp. 75–81, 2014.

[17] O. Kachman and M. Balaz, "Optimized Differencing Algorithm for Firmware Updates of Low-Power Devices," in *19th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2016, pp. 1–4.

[18] J. Jongboom, "Firmware updates over Low-Powered Wide Area Networks | Mbed," 2018. [Online]. Available: https://os.mbed.com/blog/entry/firmware-updates-over-lpwan-lora/?_ga=2.105836355.1504264362.1562850494-164173221.1562850494. [Accessed: 13-Jul-2019].