http://doi.org/10.35784/iapgos.3679 *received: 25.05.2023 | revised: 19.07.2023 | accepted: 24.09.2023 | available online: 30.09.2023*

REMOTE SOTA ALGORITHM FOR NB-IOT WIRELESS SENSORS – IMPLEMENTATION AND RESULTS

Piotr Szydłowski, Karol Zaręba
Efento sp. z o.o., Cracow, Poland

Abstract. *In this paper we share our experience with remote software updates for NB-IoT devices. The experience was collected over the years, when managing a fleet of tens of thousands of NB-IoT wireless sensors deployed worldwide by our customers. The paper discusses the main concerns that must be taken into account when designing the remote software over the air (SOTA) update mechanism, describes the remote update algorithm developed and used by us and presents the achieved experimental results based on remote software update of 5 000 NB-IoT sensors deployed in 10 European countries.*

Keywords: Internet of Things, wireless sensors, NB-IoT, software over the air

ALGORYTM ZDALNEJ AKTUALIZACJI OPROGRAMOWANIA W BEZPRZEWODOWYCH SENSORACH NB-IOT – IMPLEMENTACJA I REZULTATY

Streszczenie. *W tym artykule dzielimy się naszymi doświadczeniami ze zdalnymi aktualizacjami oprogramowania w urządzeniach NB-IoT. Doświadczenie zbieraliśmy przez lata, zarządzając flotą dziesiątek tysięcy czujników bezprzewodowych, które używane są na całym świecie przez naszych klientów. W artykule omówiono główne zagadnienia, które należy wziąć pod uwagę przy projektowaniu mechanizmu zdalnej aktualizacji oprogramowania (SOTA), opisano algorytm zdalnej aktualizacji opracowany i wykorzystywany przez nas oraz omówiono eksperymentalne wyniki aktualizacji oprogramowania na podstawie aktualizacji 5 000 czujników NB-IoT pracujących w 10 krajach europejskich.*

Słowa kluczowe: internet rzeczy, sensory bezprzewodowe, NB-IoT, zdalna aktualizacja oprogramowania

Introduction

One of the key challenges for IoT solution providers is to develop an efficient remote software update method for hundreds of thousands of battery-powered devices deployed around the world. The lack of secure, remote firmware updates is identified as a key security issue for IoT devices, by both organisations dedicated to security like OWASP [1] and researchers that test particular IoT solutions [3].

This may be one of the key issues that slow down the IoT adoption, as many users, especially large organisations, are not able to use IoT devices that do not meet their security requirements and standards. Moreover, the security breaches resulting from the firmware update may even lead to serious injuries of the devices' users. The most famous example of it dates back to 2015, when the researchers got almost full control over a car by injecting the malicious code into its entertainment system [1]. A recent example of large scale firmware related security breach is Mozi [10] – a botnet composed of over 1.5 million of non-computer devices used to launch DDoS attacks and steal the data.

There are many remote update mechanisms that have been used for years to update operating systems, software or hardware. However, it's impossible to implement the same mechanisms and algorithms in battery powered IoT devices due to their constrained resources (RAM, ROM, CPU) and the fact that these mechanisms require the updated device to stay long in the active state – the state, when the device communicates with the server and consumes a lot of energy.

Recently, there have been few attempts to standardise the IoT devices software updates mechanisms, which resulted in documents describing the best practices, firmware update architecture (e.g. IETF [4]) or documentation describing the updates (OMA [6]).

There are several key concerns that have to be taken into account when designing an update solution for IoT devices:

- The update must be secure – there are many security aspects that should be taken into account, but all of them are grouped in the following categories of the security threats, based on the S.T.R.I.D.E model [5]: spoofing identity, tampering with data, repudiation, information disclosure, denial of service, elevation of privilege.
- The update process must be safe – in case there are any issues with the update process, the device should be able to automatically roll back to the older version of the software.
- The update process must not impact the device's operations – e.g. in case of IoT sensors the update should not block taking the measurements for a long time (or ideally, not block it at all).
- The update package must be small – there are few very important reasons behind this:
    - The smaller the file the shorter the communication time with the server. In case of battery powered devices this has a direct impact on the battery life time.
    - In case of the cellular IoT devices the update package size has a direct impact on the cost of ownership, as the users are billed for the data sent / received over the mobile network.
    - According to different researchers, the number of cellular IoT devices will grow significantly in the coming years [8]. Sending large amount of data required to perform updates on these devices over the cellular networks that are optimised for small data packages (NB-IoT) can cause serious problems on the network side.
    - The update package must be small enough to be processed by an IoT device with constrained resources (memory and processing power).

All the above requirements are considered and satisfied in our remote differential SOTA update algorithm presented in section 2. Section 3 presents experimental results that show all aspects and advantages provided by our approach.

1. Typical method of Software over the air (SOTA) updates

Many IoT solution providers allow remote updates on their devices. The typical software over the air update procedure is simple: a development team compiles the code after the modifications and generates an entirely new software image, which is then distributed to the devices over the air [12]. However, the typical Over the Air (OTA) update is very troublesome, with millions of battery-powered sensors based on cellular networks (NB-IoT and LTE-M). The amount of data transmitted can not only cause destabilisation of the network but also seriously burden those devices that are optimised for short transmission sessions.

2. Proposed solution

2.1. Key requirements

We develop and produce wireless NB-IoT sensors used by customers around the globe. To provide the end users with the best experience and address the potential issues resulting from software bugs we had to implement an effective, remote software over the air update mechanism. On top of the general software over the air updates requirement described in section 1, the update process must be handled by the wireless sensors with the constrained resourced. The sensors are based on 64 MHz Cortex-M4, 512 KB Flash, 64 KB RAM and the only power source used by them are batteries.

2.2. Proposed SOTA update algorithm

We have developed an update process that seems to fit the IoT devices very well. The solution incorporates a differential update mechanism – a software update system based on sending only the changes in the software to the already-deployed devices. This update method has many advantages over the classic OTA updates, including:

- the amount of data transmitted to the device is several times reduced compared to the classic OTA update;
- the network load is minimised;
- longer device battery life is ensured, as due to the small size of the update package, the device spends less time in the active (connected) mode,
- the measuring system is not destabilised by the lack of measurements during the long update process.

The update process was designed to be used with UDP protocol, as it is supported by NB-IoT networks and minimises the data sent between the sensors and the servers. On top of the UDP we use the CoAP as it is one of the most popular and widely used IoT protocols and thanks to its small overhead it can be used by almost any IoT device.

The security threats are taken into the account during the update process. The developed algorithm guarantees that each of the devices will check the authenticity of the software, make sure that the update file was not corrupted during the transmission over the NB-IoT network and, if it was, request the server to resend the missing packages. The full encryption of the software transmission between the server and IoT devices makes it impossible for any third party to interfere the update process. Thanks to the server authentication and the mechanisms that verify the correctness of the received differential software file, the software update process is completely secure.

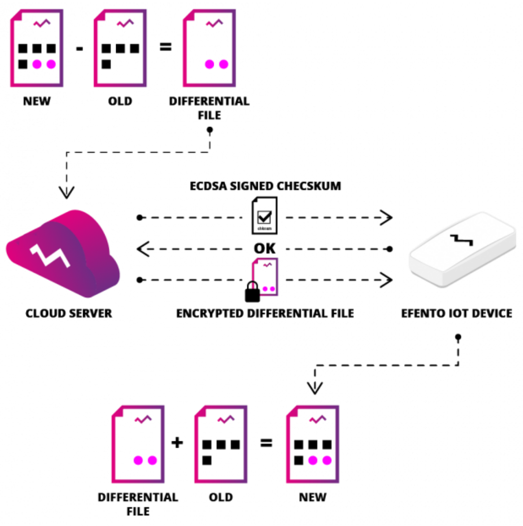

Fig. 1. Remote software over the air update developed by us

The security mechanisms incorporated in the SOTA update mechanism are based on the Elliptic-curve Diffie–Hellman (ECDH) [9] key agreement protocol. Each of the sensors has its own pair of public and private keys, securely generated and flashed on each device during the production. The server's pair of public and private keys are generated for each update session. The differential software file is signed during the compilation using the ECDSA SEC256k1 algorithm [11] with a private key securely stored in our CI infrastructure. The overview of the SOTA update mechanism is presented on Fig. 1.

The new SOTA update algorithm consists of the following steps:

1. After completing the work on the new version of the software, we prepare a differential file that is the difference between the new and the current versions of the software. The software is signed with ECDSA signature.
2. The differential file is sent to the update server and IoT devices are notified of the available update.
3. Sensor sends a CoAP message to the update server with its current software version number and its own public key.
4. The update server calculates a shared secret from its own private key and sensor's public key. The first 16 bytes of the secret will be used as the AES128 encryption key for the communication with the sensor, for a single update session.
5. The update server responds to the sensor with an encrypted CoAP message that contains the new software version number, hash (SHA256) from the new software, ECDSA signature of the new software hash and CRC16 from the whole frame. Along with the encrypted payload the update server sends its public key (not encrypted).
6. The sensor receives the update server's public key and calculates a shared secret from the server's public key and sensor's private key. The first 16 bytes of it will be used as an AES128 encryption key, valid for a single update session.
7. The sensor decrypts the response from the update server and decides, based on the new software version number, whether the software update is needed. If the update is not needed, the sensor terminates the update procedure.
8. If the update is needed the sensor verifies the authenticity of the update file based on the software hash and ECDSA signature. If the authentication fails, the sensor terminates the update procedure
9. If the authentication is successful, the sensor requests the differential software file from the update server.
10. The update server provides the sensor with the differential software file by sending encrypted data packages over raw UDP sockets.
11. Once the download of all the packages is completed, the sensor checks, if none of the packages is missing. If any of the packages is missing, the sensor requests the missing package(s) using a CoAP message. Downloading of the missing packages may be repeated twice. If the sensor was not able to download and build the differential file, the update is terminated and the sensor will try to perform the update again at the next communication.
12. The sensor enters the bootloader. The bootloader calculates the hash (SHA256) from the new software (old software merged with the differential software file received from the update server). If the calculated hash matches the hash sent by the update server, the sensor performs the software update. Otherwise, the update process is terminated and sensor reboots with the old software version.
13. Once the software update is successfully finished, the sensor notifies the update server about that by sending a message with its new software version number.

3. Algorithm testing and results

The results are based on remote software updates performed by us on 5 000 NB-IoT sensors located in 10 European countries, operating in NB-IoT networks of different mobile operators. The network coverage in the places where the sensors operate varies. For the simplicity of presenting the results, we decided to group the devices by their Coverage Enhancement Level (ECL) [2] – a parameter, dynamically set by an NB-IoT device based on the signal quality indicators (RSSI, RSRP, RSRQ). Each ECL determines the number of times downlink and uplink messages can be repeated to reach devices in poor coverage and the number of repetitions in each ECL is predefined by the network. ECL can have one of three values: 0 – used when coverage is good, 1 – used with moderate coverage, 2 – used with poor coverage.

Out of the 5 000 devices, 2 964 were in good coverage (ECL0), 1 424 were in moderate coverage (ECL1) and 612 were in poor coverage (ECL2).

The maximum number of the update attempts (point 11 of the Proposed SOTA update algorithm description in section 2) was set to 3. If a device had not been able to successfully perform the software update three times, the update server will not initiate the process anymore.

We defined the following metrics for the evaluation of the remote SOTA algorithm performance:

- Success rate - how many devices out of the whole test batch (5 000) successfully performed the software update. We also analysed the impact of the signal related parameters on the SOTA update
- Data usage – data required to send the update file to the device over the NB-IoT network.
- Energy consumption – energy consumed by a sensor for the update process.
- Downtime – total time during which the sensor is not able to perform its regular operations (taking the measurements and sending it to the server) due to the update process.

3.1. Success rate

Out of 5 000 devices, 4 988 (99.76%) managed to successfully perform the software update. Some of the devices required more than one attempt to successfully update their software. The 12 devices that failed to update the software were located in poor coverage (ECL2).

Table 1. Results of the remote software update of 5 000 devices

Coverage	Number of devices	Successful update at the 1st attempt	Successful update after repetition(s)	Success rate
ECL0	2 964	2 842	122	100%
ECL1	1 424	1 066	358	100%
ECL2	612	172	428	98%
Total	5 000	4 080	908	99.76%

3.2. Data usage

The size of the differential software file depends on the changes implemented in the software – the larger the changes, the larger the differential file. The total size of the current version of sensors' software is 250 kB. Based on the history of the remote updates performed by us, the smallest differential file was 7 B, the largest 33 kB and the average size of the differential software file was 21 kB. Using the remote update mechanism based on the differential files decreases the data consumption for the remote software update by 229 kB (91.6%) on average. This is a large difference as during its regular operations, with the measurement interval set to 5 minutes and the transmission interval set to 60 minutes, a single sensor consumes 183 kB of data per month.

3.3. Energy consumption

During the tests performed in our office, in good network coverage (ECL0), the total energy consumption during the update was 0.07 mAh for the differential update file of 7 B and 0.5 mAh for the differential file of 33 kB. As the majority of the energy consumed during the update is used to download the update file from the server, the conclusion is clear: the larger the update file, the more energy is required to perform the update. The total capacity of the batteries used in our wireless sensors is 6 300 mAh, so the remote update process based on the differential update files does not drain the battery too much.

The energy consumption during the remote software update process may vary, depending on the network coverage. As in the poor network coverage (ECL2), some packets of the differential file may not be delivered to the sensor at the first attempt and may require repetition(s), the energy consumption will be higher. Due to the specification of the test setup (5 000 devices are deployed in 10 countries around Europe), we were not able to measure to energy consumption of every single device during the update process and asses the network coverage impact on the energy consumption.

3.4. Downtime

As the majority of the remote software update tasks are performed by the main application and only the verification of the software hash and the update itself are performed in the bootloader, the device downtime is minimised. The total downtime (time required to perform the update and restart the device once the update is finished) is 24.5 seconds, no matter what is the update file size.

As the new software file is built by the device once the download of the differential file is completed, the network coverage and the number of repetitions required to deliver the differential file have no impact on the device's downtime.

4. Conclusions

The software over the air algorithm developed and implemented by us meets the security and performance requirements for the remote software update. The conducted tests proved, that the software update mechanism is reliable and the impact on both sensors' batteries and the network is minimised. The proposed software over the air algorithm can be successfully used to perform the updates not only of NB-IoT wireless sensors but to update any type of IoT devices with constrained resources.

The update process based on sending only the differential file to the wireless sensors decreases the data usage, what is beneficial for their owners (lower amount of data sent / received by the devices equals lower fees paid to the mobile operator), but also decreases the impact of the update process on the stability of the NB-IoT network.

We did not notice any impact of the particular operator's NB-IoT network configuration on the update process. The devices that failed to update their software were located in different countries and the common point, that made the update impossible was the poor network coverage.

We think, that it would be possible to achieve 100% success rate (remotely update all the devices, including the 12 devices located in poor coverage that were not able to perform the update), if the maximum number of the allowed update attempts was increased. This however would also increase the battery consumption and decrease the device life time.

Due to the tests specification (devices are deployed in 10 countries) we were not able to fully assess the impact of the network coverage on the energy consumption during the update. In order to get the full picture of that, further tests are required.

Acknowledgments

The research described in this paper was part of project „Opracowanie systemu monitoringu danych fizycznych opartego na technologii NB-IoT / LTE Cat M1 i platformie w chmurze, z wykorzystaniem nowej generacji bezprzewodowych sensorów", co-financed by European Union, program "Regionalny Program Operacyjny Województwa Małopolskiego na lata 2014-2020"

References

[1] Greenberg A.: Hackers Remotely Kill a Jeep on the Highway, With Me in It. [http://www.wired.com] (21.07.2015).
[2] Khan S. M. Z.: Narrowband Internet of Things (NB-IoT): from Radio Network Coverage to Device Energy Consumption Modeling and Energy-Efficient Application. Tallinn University of Technology [http://doi.org/10.23658/TALTECH.7/2022].
[3] Klinedinst D., King C.: On Board Diagnostics: Risks and Vulnerabilities of the Connected Vehicle. Software Engineering Institute, Carnegie Mellon University, 2016.
[4] Moran B. et al.: A Firmware Update Architecture for Internet of Things. RFC 9019. Internet Engineering Task Force, April 2021.
[5] Moran B. et al.: A Manifest Information Model for Firmware Updates in IoT Devices. RFC 9124. Internet Engineering Task Force, 2021.
[6] OMA SpecWorks, OMA LightweightM2M (LwM2M) Object and Resource Registry, 2023.
[7] OWASP IoT Security Team, OWASP Internet of Things Top10, 2018.
[8] Sinha S.: State of IoT 2023: Number of connected IoT devices growing 16% to 16.7 billion globally, IoT Analytics, May 24, 2023.
[9] Standards for Efficient Cryptography, Certicom Research, SEC 1: Elliptic Curve Cryptography, 2009.
[10] Tu T. F. et al.: A comprehensive study of Mozi botnet. International Journal of Intelligent Systems 37, 2022, 6877–6908 [http://doi.org/10.1002/int.22866].
[11] Vanstone S. A.: Responses to NISTs Proposal. Communications of the ACM 35(7), 1992, 50–52.
[12] Zandberg K. et al.: Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check. IEEE Access 7, 2019, 71907–71920 [http://doi.org/10.1109/ACCESS.2019.2919760].

M.Sc. Eng. Piotr Szydłowski
e-mail: piotr.szydlowski@efento.pl

Cofounder and CEO of Efento Ltd., a company that designs, develops and produces wireless sensors and platform for device management. Prior funding Efento, he had experience in consulting (management, technology).

He graduated from AGH University of Krakow in 2011 and holds magister degree in computational physics.

http://orcid.org/0009-0007-1384-2011

M.Sc. Eng. Karol Zaręba
e-mail: karol.zareba@efento.pl

Cofounder and CTO of Efento Ltd., a company that designs, develops and produces wireless sensors and platform for device management. Prior funding Efento, he had experience at various software development positions.
He graduated from AGH University of Krakow in 2011 and holds magister degree in automation and robotics.

http://orcid.org/0009-0009-7886-5163