Toro Betancur, Veronica; Viquez Zamora, Jose; Antikainen, Markku; Di Francesco, Mario

A Scalable Software Update Service for IoT Devices in Urban Scenarios

Published in:
Proceedings of the 9th International Conference on the Internet of Things

DOI:
10.1145/3365871.3365880

Published: 01/01/2019

Document Version
Publisher's PDF, also known as Version of record

Please cite the original version:
Toro Betancur, V., Viquez Zamora, J., Antikainen, M., & Di Francesco, M. (2019). A Scalable Software Update Service for IoT Devices in Urban Scenarios. In *Proceedings of the 9th International Conference on the Internet of Things* Article 9 ACM. https://doi.org/10.1145/3365871.3365880

A Scalable Software Update Service for IoT Devices in Urban Scenarios

Verónica Toro-Betancur, José Viquez Zamora, Markku Antikainen, Mario Di Francesco
{veronica.torobetancur,jose.viquezzamora,markku.antikainen,mario.di.francesco}@aalto.fi
Aalto University, Finland

ABSTRACT

Devices in the Internet of Things (IoT) are software-driven, thus, they need not be only programmed before deployment, but also continuously updated. IoT deployments in urban scenarios are particularly relevant as enablers of smart city applications. For such a context, this work addresses the reliability and security aspects of distributing software updates to a large number of IoT devices. Specifically, it presents a design and implementation of a software update framework for IoT devices in urban scenarios. The proposed approach leverages long-range wireless broadcast to update a large number of IoT devices at the same time, which scales up to the massive networks that are typical of densely-populated and built-up metropolitan areas. Experiments on a real testbed demonstrate that the proposed approach obtains a long range (up to 350 m) and a success rate higher than 99% with a single transmission, for IoT devices deployed both outdoors and indoors. In particular, broadcast updates are always more efficient than standard updates over the Internet through enterprise WiFi for typical urban IoT deployments.

CCS CONCEPTS

• **Computer sys. organization → Embedded systems**.

KEYWORDS

Software updates, Long-range broadcast, Scalability, Security

ACM Reference Format:
Verónica Toro-Betancur, José Viquez Zamora, Markku Antikainen, Mario Di Francesco. 2019. A Scalable Software Update Service for IoT Devices in Urban Scenarios. In *9th International Conference on the Internet of Things (IoT 2019), October 22–25, 2019, Bilbao, Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3365871.3365880

IoT 2019, October 22–25, 2019, Bilbao, Spain

ACM ISBN 978-1-4503-7207-7/19/10.
https://doi.org/10.1145/3365871.3365880

1 INTRODUCTION

The Internet of Things (IoT) is around us: street lights automatically switch on and off as we drive home; a smart thermostat decides when and for how long to turn on heating while saving energy; a virtual assistant (e.g., Amazon Alexa) clears our doubts on recipes while we are cooking. Things in the IoT are *embedded* devices with communication and computing capabilities, yet constrained in terms of resources: memory, storage, energy [20]. Being software-driven, things need not be only *programmed* before deployment, but also continuously *updated* [31]. This is needed to address changing requirements of IoT applications as well as to ensure correct operations, in particular, to fix outstanding software bugs and security vulnerabilities [25].

The number of things has reached billions worldwide in diverse environments, especially in *metropolitan areas* as technology enabler of smart city applications [5]. In fact, a large amount of IoT devices are deployed in urban scenarios for air quality and traffic monitoring [16], for instance. Most of these devices are deployed outdoors and are wirelessly connected to the Internet [5]. In any case, they run unattended; clearly, updating their configuration and software cannot be done manually but needs to be automated [31]. For this purpose, over-the-air (OTA) programming has been developed to distribute software and configuration updates to embedded devices [6], particularly, set-top-boxes (Section 2).

OTA updates in urban scenarios are challenging, mainly due to the extent of deployed networks and the adverse impact of built-up areas on reliability [24]. In this context, viable long-range communication technologies include either cellular or low-power wide area networks (LPWAN) [26]. Cellular communications leverage licensed radio frequencies; they are rather reliable, but expensive and not very energy-efficient [22]. In contrast, LPWANs employ the unlicensed industrial, scientific and medical bands; they are energy-efficient and low-cost, but they have very limited bandwidth and poor reliability in large networks [30].

A different option consists in leveraging long-range broadcast transmissions, as those used for FM radio and digital television. These wireless transmissions are inherently *asymmetric*, implying that IoT devices have receive-only capabilities. Despite appearing as a limitation, this approach has several advantages. First, IoT devices are not permanently connected

to the Internet, which reduces the chance of remote security attacks [25]. Second, broadcast wireless transmissions enable reaching a large number of IoT devices at once, especially over extensive areas such as urban environments [17].

Using broadcast transmissions for software updates also entails several challenges. The most important is the *reliability* of communications; as IoT devices cannot send any messages in the uplink, data must be protected by adequate mechanisms to guarantee that they are correctly received despite channel impairments [2]. Moreover, the *security* of transmissions must be ensured against malicious users who could read the content of the updates and gain access to confidential information, or even try to send illegitimate updates to the IoT devices [25].

This work specifically addresses these challenges with the design (Section 3) and implementation (Section 4) of a software update framework for IoT devices in urban scenarios. The proposed approach leverages long-range wireless broadcasts to update a large number of IoT devices at the same time, which scales up to the massive networks that are typical of densely-populated and built-up metropolitan areas. The reliability of updates is guaranteed by employing erasure codes and multiple transmissions in a cyclical update schedule. Their security is ensured by cryptographically signing and encrypting the updates, so that they are only available to the intended recipients. Experiments on a real testbed demonstrate that the proposed approach obtains a long range (up to 350 m) and a success rate higher than 99% with a single transmission, for IoT devices deployed both outdoors and indoors (Section 5). In particular, broadcast updates are always more efficient than standard updates over the Internet through an enterprise WiFi infrastructure for typical urban IoT deployments.

2 RELATED WORK

Substantial research has been carried out on the Internet of Things [20]. A significant share of the literature targeted data collection, for instance, through middleware platforms [21]. Instead, the management of large networks of interconnected devices has received less attention. The work in [29] has shown the feasibility of implementing standard Internet protocols (such as the Simple Network Management Protocol) on resource-constrained devices, but has not elaborated how these can support software updates. The lifecycle of things, including the role of software updates, has been described in [10] as a starting point to discuss security issues in the IoT. Unfortunately, the work does not consider mechanisms to deliver software updates to a large number of IoT devices. A scalable directory service was devised in [13] to enable both data access and attribute management of IoT devices with a low latency. The corresponding update scheme only considers records (i.e., set of attributes), thus, it is not suitable for software updates. Indeed, remote software updates were investigated in [14] through an architecture based on mobile edge computing and long-range communications. The solution proposed therein, however, was only evaluated for the distribution of very small updates (i.e., less than one kilobyte). The use of software containers for the IoT has lastly gained attention too [1], especially as technology enabler of continuous integration and deployment in such a context. However, containers only allow to update the applications and not the operating system of IoT devices. Blockchains have also been proposed to address scalability in the IoT [7], including for device management [28]. In particular, blockchains were leveraged in [4] to ensure accountability of IoT software updates. The solution proposed therein prevents attackers to distribute malicious software updates in an IoT network, but has a significant overhead in content delivery.

Software reprogramming has long been considered for networks consisting of embedded devices [12]. However, several solutions developed for large networks rely on a high device density and propagation of updates over multiple hops through either flooding or gossiping. These are indeed not scalable, as the network can quickly become congested as the number of nodes increases [18]. More recently, OTA updates have gained attention, especially for urban scenarios. Among them, the LWMesh network protocol [19] has been leveraged to distribute firmware updates over an IoT network for smart urban applications [6]. Other methods for secure OTA update have been devised for electronic control units in smart vehicles [23]. However, these solutions assume that end devices establish bi-directional communication, which is costly and not very reliable in urban environments. Instead, this article proposes an architecture where IoT devices receive updates over a broadcast medium, without the need for bi-directional communication. A different method for sending software updates employs television broadcast. For instance, OTA updates of digital TV receivers were evaluated over local television services in [8]. Updates were successfully transmitted at bitrates up to 4.5 Mbps; however, the evaluation was performed in a lab scenario as opposed to the real urban environment considered here.

There are also a few ongoing initiatives targeting software updates of IoT devices. Among them, the suit working group at the IETF [11] and the Eclipse hawkBit framework [9] are particularly relevant. This work shares similarities with them as to the how to describe and deliver software updates for IoT devices. However, this article specifically focuses on long-range broadcast communication for urban IoT scenarios.

3 SYSTEM OVERVIEW

This section describes the proposed software update system. The reference architecture is presented first, followed by the characteristics of the supported devices. The section

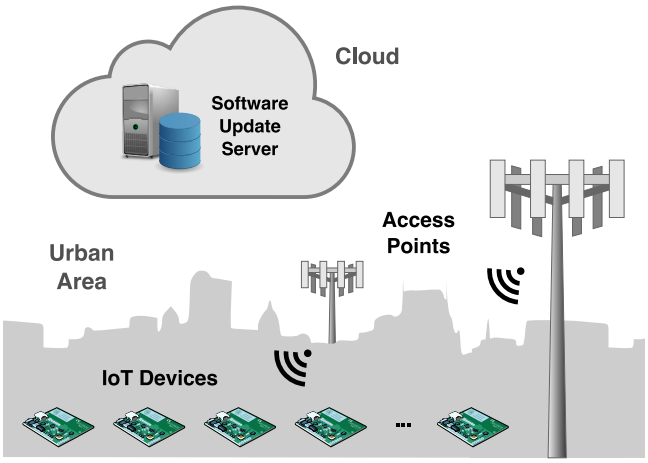

Figure 1: Reference architecture: a software update server in the cloud stores and provides updates to IoT devices through a content distribution infrastructure, consisting of one or more access points. These access points employ a long-range wireless technology and broadcast transmissions to reach IoT devices in an urban area.

concludes with a discussion of broadcast data transmission and security considerations.

Reference architecture

Figure 1 illustrates the reference architecture considered in this work. A device manufacturer or service provider releases *software* or *configuration updates* for IoT devices running a certain application. Updates are uploaded on a dedicated server in the Internet which is connected to a *content distribution infrastructure* consisting of one or multiple access points. These employ long-range wireless communication to deliver updates over a metropolitan area by using, for instance, licensed / unlicensed spectrum or TV whitespaces [3]. This work targets the downlink (i.e., communication from access points to devices) as a *broadcast* communication channel. This allows to send data simultaneously to all the devices in reach of the content distribution infrastructure. Consequently, all IoT devices in the coverage area of the access points can receive updates at the same time, thereby increasing the efficiency of communications towards a large number of devices. IoT devices have suitable receiving capabilities through either special-purpose hardware (e.g., a WiFi interface) or software-defined radios. They also run a *software update service* that receives and installs the update packages.

Device software and platforms

IoT devices run an application on top of an embedded operating system, following the mobile device landscape where this case is already prevalent (e.g., for smart phones and smart watches). The maturation and the consolidation of embedded operating systems, many of which are specifically targeted for the IoT, makes this approach not only desirable, but also flexible enough to support future device classes and platforms [32]. Operating system images are general enough to be used for different platforms sharing the same hardware (for instance, all "Nest Thermostat E" boards): these are referred to as device *types*. While the sheer number of IoT devices is very large, there is significantly less variability in their types. For instance, there might be hundreds of smart locks in a corporate building, but all of them[1] belong to one or a few types (e.g., a certain model of a given manufacturer). Devices update both their operating system and their applications: the update process considers the two independently. For instance, a smart thermostat could use embedded Linux as operating system and a custom application to implement the actual temperature monitoring / control functions. The operating system could be updated to patch security vulnerabilities, while the custom application could be updated to fix software bugs or to add new functionality.

Broadcast updates

Broadcast data transmission employs multiple *channels*, each delivering a different type of update (e.g., operating system and application). Update packages are a collection of files in compressed format (e.g., zip) and also contain metadata including the name of the update package, its version, the target device class, and a cryptographic hash (for verification purposes). The proposed system also supports incremental updates [15], wherein update packages only contain the differences between a previous version and the current one. Updates are cyclically broadcasted so as to increase availability. Additional channels could also be used, for instance: to provide the schedule of upcoming software updates, based on which IoT devices can turn off their radio transceivers to save energy; to provide input data for certain applications, such as real-time public transportation timetables. Forward error correction is applied to update packages to increase the reliability of communications against channel errors [2].

Security

Two different types of adversaries are considered. The first type of adversary intends to transmit illegitimate updates to the IoT devices. The ultimate goal of such an attacker may be, for instance, to get some IoT devices under their control. The second type of adversary passively listens to transmissions so as to learn any confidential information that they might contain.

The update system is protected against the aforementioned attacks by cryptographically signing and encrypting

[1]The expected number of devices per type is at least two orders of magnitude higher than the type of devices [17].

(in this order) all transmitted data. For this to work, the IoT devices must store the public key of the server that is used for signing the data. The confidentiality and authenticity of the transmitted data could be ensured, for instance, with symmetric AES-CCM, which is done after signing. IoT devices of the same type share the same symmetric key that is used for encryption / decryption. The reason for this is to protect the confidentiality against passive adversaries and to make denial-of-service attacks that target the slow asymmetric signing more difficult – the IoT devices verify the asymmetric signature only after the symmetric decryption.

It must be noted that only attackers that operate on the wireless channel are considered here. That is, attackers that can physically access an IoT device and extract decryption keys from it are considered out of scope for this work. Such an attacker could use the extracted keys to breach the confidentiality of the future transmissions that are destined to IoT devices of the same class as that of the breached device. However, the authenticity of the follow-up transmissions would still be ensured due to the asymmetric signatures.

4 IMPLEMENTATION

The following details an implementation of the proposed software update service for IoT devices in urban scenarios. It first discusses the software update server in the backend, then the software update service running at the IoT devices. The section concludes by introducing a long-range data transmission scheme leveraging WiFi.

Backend

The backend of the system is represented by the software update server, located in the cloud. The backend includes a web application as the user-facing component of the update service and a database for storing update packages. In particular, the web application supports managing users, device types and actual updates. Moreover, software developers can create user accounts to upload software updates for specific device types. An update has an associated name, version, target device type, and an optional[2] release time. Once uploaded, the update package is encrypted and signed with the private key of the server, according to the device type (as discussed in Section 3). When the release time is reached, the server makes the update available over-the-air as part of those sent over the broadcast channel. The backend is realized by using the CakePHP web development framework[3] with the model-view-controller architectural pattern and MySQL as database.

Access point

The access point runs a client software that connects to the software update server, retrieves the available packages and builds the schedule for transmitting the actual updates over the air. Our implementation constructs a simple cyclical update schedule based on the release time of the packages: those available earlier appear at the beginning of the schedule. Afterwards, the client retrieves the actual software updates, appends the relevant metadata, and broadcasts them over WiFi. In doing so, it employs Wifibroadcast, a software[4] primarily designed for transmission of high-definition videos over WiFi, for instance, aerial video streams from drones. Wifibroadcast can actually be used with any type of source data and allows to configure different levels of forward error correction, realized through Reed Solomon codes [27]. The client software at the access point is written in python.

Software update service

IoT devices also employ Wifibroadcast and run a software update service that keeps listening for updates over the broadcast channel. In particular, the service receives the broadcast schedule / metadata to determine if a software update is available and needs to be installed. This happens when the update package is fit for the device type and has a higher version than the one currently running at the device. Eligible updates are then retrieved over the air and processed. First, error correction is applied to erase errors, then the cryptographic hash in the metadata is checked; finally, the package is decrypted and the signature is verified.

Upon success, the service proceeds onto installing the update. Our implementation leverages a seamless update process similar to Android A/B system updates[5]. Seamless updates have the advantage to reduce the downtime, as the device is still operational while the update is downloaded and installed; whereas it only becomes unavailable during restart. Specifically, seamless updates use two partitions that run different versions of an operating system: active and inactive, each considered as independent entity. The update service runs on the currently active operating system in the primary partition. It decompresses and installs the update package onto the inactive (secondary) partition, then configures the boot loader to mark the currently inactive partition as primary. As a result, the newly installed version of the operating system will be started upon restart. As the previous version of the operating system is still available, it can still be rolled back in case an error occurs with the new software image. The update service is implemented for Linux as a combination of python and bash scripts.

[2]The update is available immediately if no specific release time is supplied.
[3]https://cakephp.org/

[4]https://befinitiv.wordpress.com/wifibroadcast-analog-like-transmission-of-live-video-data/
[5]https://source.android.com/devices/tech/ota/ab/

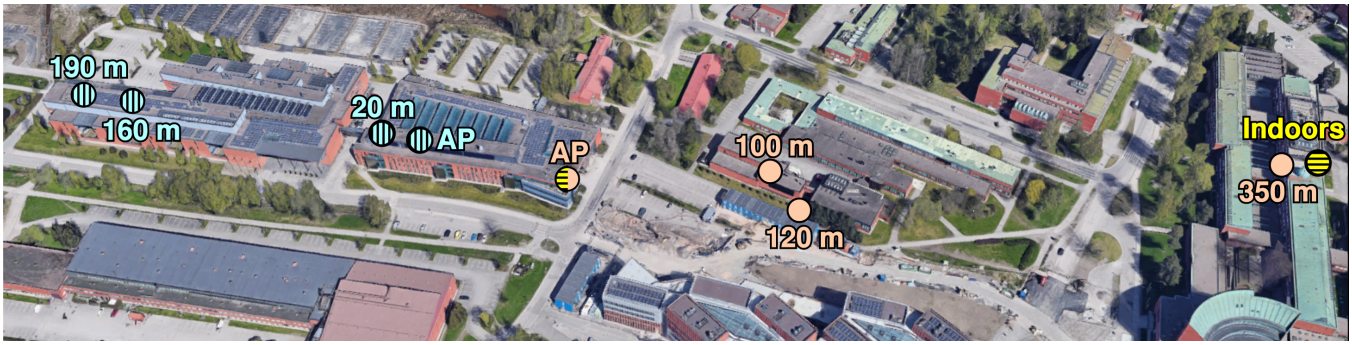

Figure 2: Testbed for the experimental evaluation, deployed at the Aalto University campus in Otaniemi. Node locations are represented by circles with different colors, according to the considered scenarios: light blue with a vertical pattern for non-line of sight (non-LOS), pink for line of sight (LOS), and yellow with a horizontal pattern for indoors. Map data from Google.

5 EXPERIMENTAL EVALUATION

The following presents an experimental evaluation of the proposed system, divided into two parts. The first focuses on the software-related aspects of the update process; whereas the second part investigates the reliability of delivering software updates in a real urban scenario. In both cases, results are obtained with the system prototype realized according to the description in the previous section. Unless otherwise specified, a full software update package (i.e., the whole operating system) with a size of 48.293 MB is considered.

System performance

The performance of the proposed system is characterized in terms of *availability*, as the time needed to install and activate the newly downloaded software (the lower the better). Experiments are carried out on a Raspberry Pi 3B as a representative IoT device[6] class. Ten iterations of each experiment are run according to the independent replication method.

Table 1 shows the time taken to install and activate a software update. Note that the installation phase in the table includes the time to process the metadata, decrypt the update package and verify its signature, extract files onto the inactive partition, and configure the boot loader; it does not consider the time needed to receive the update package over the air (evaluated in the next section). The results clearly show how the installation time is much longer than the reboot time, i.e., from 30% to 75% longer. This result demonstrates the advantage of seamless updates, as they significantly reduce the downtime, namely, the time during which the IoT device is not operational. It is also worth noting that the total time for the update, including both installation and reboot, is below 1.5 minutes on the average despite consisting of a full (operating system) update.

[6] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Table 1: Update time by component.

Phase	Time (s)		
	Minimum	Maximum	Average
Installation	42.58	65.75	52.49
Reboot	32.67	37.43	35.25
Both	75.25	103.18	87.74

Communication performance

Setup and methodology. Two Raspberry Pi 3B, one acting as the IoT device and another as the access point, are employed to assess the performance in delivering software updates over the air. Both devices use an Alfa Network long-range USB adapter based on the Atheros AR9271 chipset; the device operating as access point uses a 12 dBi-gain antenna instead of the default one shipped with the adapter. Wifibroadcast uses messages with a 1024-byte payload and operates on a *block* basis; each block includes 10 source data packets and a variable number of redundant packets to achieve a target error correction level.

Devices are deployed at the Aalto University campus as a representative urban environment inclusive of different types of buildings (e.g., university buildings and a shopping mall), a public transportation hub (with several bus stops and a metro station), roads, parking spaces, lawns and trees. Three different scenarios are considered: *outdoors line-of-sight* (LOS), where the access point and the IoT device are approximately located at the same height; *outdoors non-LOS*, where the IoT devices are located at different vertical distances with respect to the access point; *indoors*, where the IoT devices are located inside a building, while the access point is located on another building. These scenarios are representative of use cases wherein IoT devices are deployed either outdoors or indoors, with long-range access points

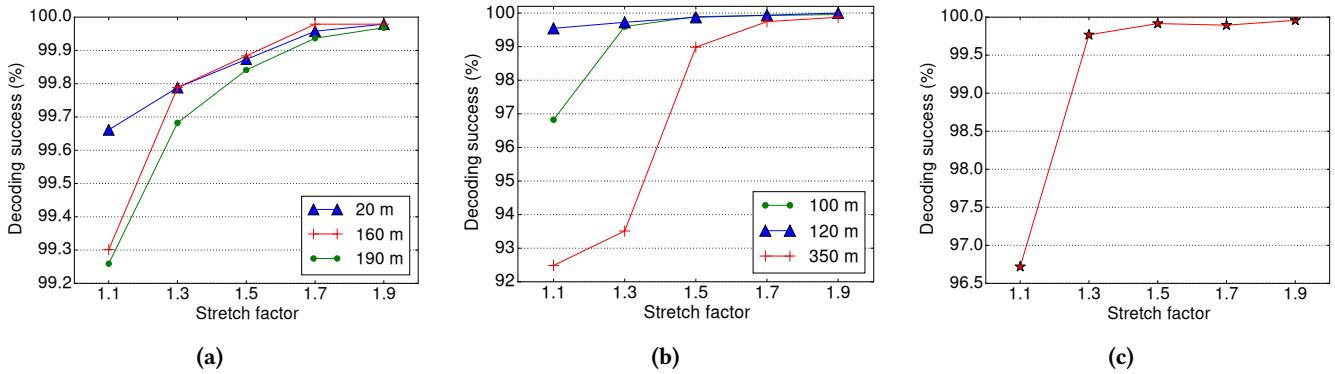

Figure 3: Decoding success as a function of the stretch factor for different scenarios: outdoors (a) LOS, (b) non-LOS, and (c) indoors.

primarily installed outside buildings (i.e., similar to cellular base stations). Experiments are carried out for different distances between the IoT device and the access point for the outdoor scenarios as detailed in Figure 2. The vertical offsets[7] between the access point and the IoT device for the non-LOS scenario are 9 m and -6 m for the distances of 120 m and 350 m, respectively.

The communication performance is characterized in terms of *reliability*, namely, the success probability in decoding the blocks in a software update (i.e., after applying error correction) and *download delay*, as the time it takes to fully receive an update. Both metrics are derived for different values of the *stretch factor*, that is, the ratio between the size of the update package after applying error correction and the original size of the data. Each experiment is repeated five times and the median is reported in the figures.

Experimental results. The reliability in receiving software updates is illustrated in Figure 3. In particular, Figure 3a shows the decoding success as a function of the stretch factor for different distances in the LOS scenario. The results clearly show that software updates can be obtained with a success probability higher than 99%; stretch factors over 1.5 even increase the decoding success above 99.9%. Clearly, longer distances between the access point and the IoT device incur in lower success rates. The difference is substantial for a low stretch factor; however, results get very similar for stretch factors of 1.3 and above, consistently over 99.6% decoding success. There were no more than 21 unrecoverable blocks in these cases; receiving one additional update (i.e., two transmissions in total) sufficed for successful decoding.

Figure 3b also shows the decoding success as a function of the stretch factor, again for different distances but in the non-LOS scenario. The decoding success is above 96% for distances up to 120 m; the decoding success at the highest distance of 350 m is close to 92% but it significantly increases with stretch factors greater than 1.3, resulting in at most 25 unrecoverable blocks on the average. The different heights of the IoT devices in this scenario results in a higher variability of the decoding success, which not always decreases with the distance from the access point. This can be explained by actual shadowing and multipath fading that depend on the specific physical elements in the environments. As before, the update package could be successfully decoded after two transmissions for every single iteration of the experiments, irrespective of the stretch factor.

Finally, Figure 3c shows the decoding success as a function of the stretch factor for the indoors scenario. The results clearly show that high decoding success percentages – consistently above 99.5% – can also be obtained by IoT devices located indoors for an access point deployed outdoors and rather far away (i.e., approximately 353 m). Specifically, there were no more than 63 unrecoverable blocks for stretch factors of at least 1.3; at most two transmissions were needed to successfully decode all blocks in every single iteration of the related experiments also in this case.

Table 2 shows the time taken to receive a full software update package as a function of the stretch factor. The table shows the average value obtained over all experiments, along with the corresponding minimum and maximum values. The results clearly show how the download delay roughly doubles as the stretch factor increases from 1.1 to 1.9, ranging from about 7 to 12.5 minutes for an update package with a size of 48.293 MB. Moreover, there is no significant variability in the measured download delays. The results are consistent with the WiFi bitrate used for long-range communications, approximately corresponding to 1 Mbps.

Discussion. The performance evaluation above has demonstrated the feasibility of using long-range broadcast communications to deliver software updates in urban scenarios.

[7] Offsets are calculated by subtracting the height of the IoT device from that of the access point; thus, a negative value indicates that the IoT device is higher than the access point.

Table 2: Time to receive the update.

Stretch factor	Time (s)		
	Minimum	Maximum	Average
1.1	403	411	407
1.3	490	517	501
1.5	576	584	580
1.7	662	667	664
1.9	745	752	748

It is important to note that unlicensed bands in the considered deployment are quite congested as there are several co-located WiFi networks used for both office, education and research purposes. Moreover, experiments were carried out at different times due to logistic constraints (including weather conditions) during workdays. These factors affected the consistency of the results; in particular, those for the indoors scenario experienced a considerable improvement in reliability when performed after working hours – the time when the experiments were carried out.

The joint use of stretch factors and multiple transmissions of update packages (as part of a cyclical schedule) allows for enough flexibility to cope with diverse environmental conditions which are deployment-dependent. According to the obtained results, a stretch factor of 1.3 is adequate to significantly boost the reliability of receiving software updates in most cases; if needed, the IoT device could listen for more than one transmission. Clearly, listening for re-transmissions of the same update package incurs in a delay that may be significant, depending on the length of the update schedule. Nevertheless, this is not generally a problem, as IoT devices could go to sleep and save energy in the meantime, while still being fully operational, although not up-to-date.

It is also worth recalling that the case of a full (i.e., operating system) update was considered in the experiments. This is a worst-case scenario as, in practice, incremental updates would have a much smaller size (a few megabytes or less), especially for updating applications as opposed to the operating system of the devices.

Comparison with enterprise WiFi

A "standard" software update process could be used in certain application scenarios, wherein an enterprise WiFi network is deployed with enough coverage to reach all relevant IoT devices. In this case, devices would independently fetch their updates over an Internet connection established through a conventional bi-directional WiFi link, with acknowledgments and retransmissions employed for reliable communication. The following compares such an option with the broadcast update solution proposed in this work.

In doing so, the enterprise WiFi network of Aalto University is employed. In particular, experiments are carried out at different times of the day to characterize the download time achievable in practice, given the other (background) traffic in the network. The related effective download rate[8] is close to 14 Mbps, thereby requiring about 28 seconds to successfully download the update package. The following assumes that all IoT devices download their update packages at different times (e.g., as a result of a scheduling policy), so that the achievable bandwidth is not reduced due to concurrent access to the wireless medium. Note that this is an optimistic scenario, as in practice download of updates can be uncoordinated, thus, possibly also overlapping in time.

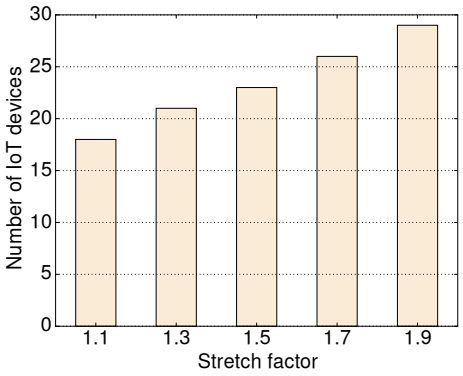

Figure 4: Minimum number of IoT devices for which broadcast updates take less time than individual downloads.

Accordingly, Figure 4 shows the tradeoff between using the proposed broadcast update solution as compared to a WiFi enterprise network. In particular, the figure shows the minimum number of IoT devices for which broadcast updates take less time. The derivation accounts for the time taken to receive corrupted blocks in the updates multiple times until they are successfully decoded, as measured in the experiments. The results clearly show that the solution in this work is always beneficial when the number of devices is higher than 30, which occurs in practice for most deployments [17]. This happens due to the nature of the broadcast transmission of the software updates, which are received by all devices in range at the same time. Broadcast updates are slower, as they employ a lower data rate to obtain a longer range; however, their duration depends only on the update package and not on the number of devices. As a consequence, they are scalable, especially since the number of devices under the coverage of the long-range access point is higher than that in enterprise WiFi.

[8]The WiFi card is using the IEEE 802.11g standard with a nominal maximum bitrate of 54 Mbps.

6 CONCLUSION

This work has introduced a novel software update service based on long-range broadcast transmissions, specifically targeted to urban IoT scenarios. The proposed system has also been implemented and evaluated on a real urban testbed by leveraging WiFi as communication technology. The obtained results showed that the proposed approach can provide updates to devices in a large area securely and reliably, resulting in a much higher scalability than using standard Internet-based approaches.

It would be especially interesting to evaluate other communication technologies instead of WiFi. A promising option would be to leverage either TV whitespaces or licensed bands (such as VHF or UHF) by means of custom modulation and access schemes, realized through software-defined radios.

ACKNOWLEDGMENTS

This work was partially supported by the Academy of Finland under grants number 299222 and 319710. The authors would like to thank to Alexandre Bosser and the Aalto Space Technology Group for their help with the experiments.

REFERENCES

[1] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen. 2018. Orchestration of Microservices for IoT Using Docker and Edge Computing. *IEEE Comm. Magazine* 56, 9 (September 2018), 118–123.

[2] G. Anastasi, E. Borgia, M. Conti, and M. Di Francesco. 2011. Reliable Data Delivery in sparse WSNs with Multiple Mobile Sinks: an Experimental Analysis. In *The 16th IEEE Symposium on Computers and Communications (ISCC 2011)*. 698–705.

[3] S. Bayhan, G. Premsankar, M. Di Francesco, and J. Kangasharju. 2016. Mobile Content Offloading in Database-Assisted White Space Networks. In *The 11th EAI International Conference on Cognitive Radio Oriented Wireless Networks (CROWNCOM 2016)*. 129–141.

[4] A. Boudguiga, N. Bouzerna, L. Granboulan, A. Olivereau, F. Quesnel, A. Roger, and R. Sirdey. 2017. Towards Better Availability and Accountability for IoT Updates by Means of a Blockchain. In *2017 IEEE European Symposium on Security and Privacy Workshops*. 50–58.

[5] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi. 2016. Long-range communications in unlicensed bands: The rising stars in the IoT and smart city scenarios. *IEEE Wireless Comm.* 23, 5 (2016), 60–67.

[6] H. Chandra, E. Anggadjaja, P. S. Wijaya, and E. Gunawan. 2016. Internet of Things: Over-the-Air (OTA) firmware update in Lightweight mesh network protocol for smart urban development. In *2016 22nd Asia-Pacific Conference on Communications (APCC)*. 115–118.

[7] K. Christidis and M. Devetsikiotis. 2016. Blockchains and Smart Contracts for the Internet of Things. *IEEE Access* 4 (2016), 2292–2303.

[8] L. C. P. Costa, R. A. Herrero, M. G. De Biase, R. P. Nunes, and M. K. Zuffo. 2010. Over the air download for digital television receivers upgrade. *IEEE Transactions on Consumer Electronics* 56, 1 (February 2010), 261–268.

[9] Eclipse. 2019. hawkBit. https://www.eclipse.org/hawkbit/.

[10] T. Heer, O. Garcia-Morchon, R. Hummen, S.-L. Keoh, S. S. Kumar, and K. Wehrle. 2011. Security Challenges in the IP-based Internet of Things. *Wireless Personal Comm.* 61, 3 (01 Dec 2011), 527–542.

[11] IETF Working group. 2019. Software Updates for Internet of Things (suit). https://datatracker.ietf.org/wg/suit/.

[12] F. Javed, M. K. Afzal, M. Sharif, and B. Kim. 2018. Internet of Things (IoT) Operating Systems Support, Networking Technologies, Applications, and Challenges: A Comparative Review. *IEEE Communications Surveys Tutorials* 20, 3 (thirdquarter 2018), 2062–2100.

[13] V. P. Kafle, Y. Fukushima, P. Martinez-Julia, and H. Harai. 2017. Scalable Directory Service for IoT Applications. *IEEE Communications Standards Magazine* 1, 3 (September 2017), 58–65.

[14] D. Y. Kim, S. Kim, and J. H. Park. 2017. Remote Software Update in Trusted Connection of Long Range IoT Networking Integrated with Mobile Edge Cloud. *IEEE Access* 6 (2017), 66831–66840.

[15] K. Kolomvatsos. 2019. An efficient scheme for applying software updates in pervasive computing applications. *J. Parallel and Distrib. Comput.* 128 (2019), 1 – 14.

[16] P. Kortoçi, L. Zheng, C. Joe-Wong, M. Di Francesco, and M. Chiang. 2019. Fog-based Data Offloading in Urban IoT Scenarios. In *The 38th IEEE Conf. on Computer Communications (INFOCOM 2019)*. 784–792.

[17] Z. Li, T. Nguyen, Q. Lampin, I. Sivignon, and S. Zozor. 2017. Ensuring k-coverage in Low-Power Wide Area Networks for Internet of Things. In *ICNC 2017*. 26–30.

[18] T. M. M. Meyfroyt, S. C. Borst, O. J. Boxma, and D. Denteneer. 2014. Data Dissemination Performance in Large-scale Sensor Networks. *SIGMETRICS Perform. Eval. Rev.* 42, 1 (jun 2014), 395–406.

[19] Microchip. 2016. Atmel AVR2131: Lightweight Mesh Application Note. http://www.microchip.com//wwwAppNotes/AppNotes.aspx?appnote=en591089.

[20] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. 2012. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks* 10, 7 (2012), 1497–1516.

[21] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng. 2017. IoT Middleware: A Survey on Issues and Enabling Technologies. *IEEE Internet of Things Journal* 4, 1 (Feb 2017), 1–20.

[22] J. J. Nielsen, G. C. Madueño, N. K. Pratas, R. B. Sørensen, C. Stefanovic, and P. Popovski. 2015. What can wireless cellular technologies do about the upcoming smart metering traffic? *IEEE Communications Magazine* 53, 9 (September 2015), 41–47.

[23] D. K. Nilsson, L. Sun, and T. Nakajima. 2008. A Framework for Self-Verification of Firmware Updates over the Air in Vehicle ECUs. In *2008 IEEE Globecom Workshops*. 1–5.

[24] G. Premsankar, B. Ghaddar, M. Di Francesco, and R. Verago. 2018. Efficient Placement of Edge Computing Devices for Vehicular Applications in Smart Cities. In *The 16th IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*.

[25] S. Ray, A. Basak, and S. Bhunia. 2017. Patching the Internet of Things. *IEEE Spectrum* 54, 11 (November 2017), 30–35.

[26] U. Raza, P. Kulkarni, and M. Sooriyabandara. 2017. Low Power Wide Area Networks: An Overview. *IEEE Communications Surveys Tutorials* 19, 2 (Second quarter 2017), 855–873.

[27] I. S. Reed and G. Solomon. 1960. Polynomial Codes Over Certain Finite Fields. *Journ. Soc. for Ind. and Applied Math.* 8, 2 (1960), 300–304.

[28] M. Samaniego and R. Deters. 2016. Blockchain as a Service for IoT. In *The 2016 IEEE Conference on Internet of Things (iThings)*. 433–436.

[29] A. Sehgal, V. Perelman, S. Kuryla, and J. Schönwälder. 2012. Management of resource constrained devices in the Internet of Things. *IEEE Communications Magazine* 50, 12 (December 2012), 144–149.

[30] M. Słabicki, G. Premsankar, and M. Di Francesco. 2018. Adaptive Configuration of LoRa Networks for Dense IoT Deployments. In *The 16th IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*.

[31] A. Taivalsaari and T. Mikkonen. 2017. A Roadmap to the Programmable World: Software Challenges in the IoT Era. *IEEE Software* 34, 1 (Jan 2017), 72–80.

[32] A. Taivalsaari and T. Mikkonen. 2018. A Taxonomy of IoT Client Architectures. *IEEE Software* 35, 3 (May 2018), 83–88.