The Future of IoT Software Must be Updated

FJ Acosta Padilla, E. Baccelli, T. Eichinger and K. Schleiser

Abstract – Decades of software engineering shows that updating deployed software is not a nice-to-have, but a must-have. Deployed software (e.g. an IP protocol stack) is never bug free, and these bugs must be patched to increase its robustness and security, and as communication standards and application logic evolve, deployed software typically needs to evolve in parallel. As the IoT emerges, the same will apply to deployed IoT software.

1 The Future of IoT Software

The IoT interconnects software running on low-end devices [1] via Internet protocol standards. Hence, IoT software resembles more and more software running on other machines on the Internet in that it can be roughly decomposed in:

1. a software platform e.g. a small footprint OS kernel [2] and hardware drivers,
2. a network stack e.g. based on IPv6 (and 6LoWPAN, CoAP...),
3. application(s) logic running on top of the network stack.

Concerns [3] were recently voiced, pointing out that many IoT devices deployed so far are liabilities because their firmware is flashed once-for-all, without means or plans to update it, thus left as potential prey for bugs [10] or attacks [11].

IoT software update $\neq$ Full Firmware Update. While it is paramount to provide firmware updates to IoT devices, it is important to realize that updating IoT software cannot be limited to full firmware updates. Transmitting minimum-sized binaries updates (less than full firmware) is desirable to fit extreme lack of (energy, throughput, memory) resources on IoT device networks. On one hand, compared to the rest of the Internet, the current state of the IoT is fragmented in that a plethora of heterogeneous hardware and network technologies deserve support (more or less) equally. Yet, on the other hand, the IoT is more captive in that silos dominate, whereby vendors typically sell turn-key bundles gathering proprietary IoT hardware, software, gateway and network technology (and even sometimes cloud services) in the same package. However, in the near future we can expect the market to resemble the Internet: basic-level interoperability requirements will break silos, consolidate IoT hardware and network technologies into fewer, dominant approaches. In particular, a natural consequence on IoT software will be the emergence of a market for hardware-independent IoT software, and thus lead to a state where different stake holders could originate and modify different parts of the IoT software on devices (e.g. application separately from software platform).

Updates with Multi-User Device Ownership. Another important point to realize is that, while IoT software updates might be considered a "solved" problem in proprietary contexts with homogeneous IoT devices (signed images distributed using some flavor of multicast, safe upgrading with roll-back through ping-pong), future IoT software updates will be more complex. Future IoT software updates will have to accommodate configurable, multi-user ownership, administrative groups etc. For example, a smart house or parts of an apartment complex equipped with IoT devices will frequently change owners, or host short-term leases with only restricted access rights. Standard protocols and software will be needed for grouping IoT devices and handling ownership in a flexible manner.

Standards for Community IoT Software. Once silos are broken, IoT hardware heterogeneity and open source IoT device software stacks will change the way IoT software is conceived. Future IoT software will be composed of a vendor-specific part (potentially proprietary and closed-source) de-

veloped independently from another part pulled from community-driven, open source IoT software development (e.g. RIOT [8][6]). These two parts will typically be updated independently, potentially using different means for this update. The main advantages of such software platforms [2] is to spread IoT software development/maintenance costs, and decrease time-to-market for IoT products. For example, one can envision an IoT product (e.g. the infamous Smart Coffee Machine) using RIOT combined with specific low-level drivers and application code, resulting in the ability to update each part independently of the others.

Standard mechanisms and protocols are desirable for the update of the open-source, community-driven part of IoT software. Currently, vendors update devices only when obliged by law (warranty) or through customer pressure, as keeping devices up-to-date is expensive. Many device types (e.g. washing machines) have lifetime expectancy exceeding warranties. Nevertheless, fast-evolving Internet and related threats make updates imperative even after vendor obligations run out (e.g. a functional washing machine with an expired warranty is wasted if its IoT component leaks a user's wifi credentials). With standard mechanisms and protocols as well as open-source community-driven IoT software platforms, the bulk of IoT software may thus be kept up-to-date, without significant burden nor obstacle related to vendors.

In the following, we focus mostly on updates of the open-source community-driven part of IoT software. Note however that the below considerations may also be applicable to other parts of IoT software (i.e. the proprietary part).

2 IoT Software Updates

The typical life-cycle of an IoT device is shown in Figure 1. Once the initial firmware is flashed and the device boots, a commissioning phase starts whereby the device joins the IoT network, comes online and may register with remote servers (e.g. a cloud service). After that, the life of the IoT device starts, during which software updates will be needed from time to time. Finally, it is decommissioned (this may require explicit signaling to deregister the device).

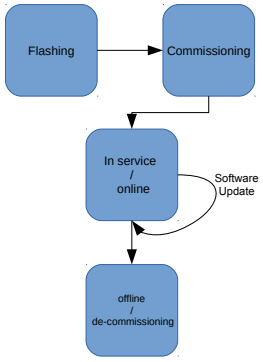

Figure 1: IoT device life-cycle.

We can distinguish 3 levels of IoT software updates:

- **Full firmware update**. An entirely new ROM image is fetched and installed.
- **Platform update**. ROM is partially updated, e.g. a peripheral driver module, or part(s) of the network stack.
- **Application update**. ROM is partially updated, to replace e.g. higher level, application logic, which can originate either from native code [9], or from interpreters [4][7].

Some parts of the ROM may be updated much more often (e.g. fixing critical network stack bugs, or slight modifications of application logic), compared to other parts of the ROM, hence the advantage of a modular approach (e.g. using a micro-kernel architecture such as RIOT's [8]) and not systematically resorting to full-firmware updates.

Basic software updates can be coarsely decomposed in the following phases.

1. IoT software repositories push meta-data information about software updates to the IoT device, when newer updates become available
2. The IoT device selects what updates it should/can accommodate

3. The IoT device pulls selected updates from a repository
4. The IoT device proceeds to verify and install received update(s)
5. If the installation fails, roll-back to previous version

At first sight, establishing standards would be useful for (i) meta-data information, (ii) communication protocols between IoT devices and repositories, (iii) authentication of updates, and (iv) potentially, also for in-network caching or proxying of repository role and contents. Some further considerations below.

2.1 When to update?

Some logic is needed on the IoT device to evaluate if conditions for the software update are satisfied locally (checking parameters such as: battery level, radio signal strength, free storage memory to store the update, enough memory to run it, sanity/signature checks...). However, updates always imply (even if minimal) downtime. Some level of human control may thus be desirable to ensure downtime does not happen at a critical time (e.g. rebooting smart lights go off during a speech of the Queen of England). The criticality of the update depends both on (i) how central the module is, e.g. if it affects online status for the device, and (ii) how central the IoT device is, e.g. updating and rebooting intermediate, upstream devices may induce failed connectivity for down-stream devices. For large IoT deployments, standard IoT software update strategies may need to capture the system as a whole, instead of per-device.

2.2 Who updates?

Humans involvement should be limited to pushing (properly signed) new versions of IoT software to a well-known IoT software repository. The rest should be automatic. However, as mentioned in Section 2.1, authorized humans should also be able to signal *"please don't update within this time window"* to their IoT devices. Furthermore, as mentioned in Section 1, different authorities may update different parts of IoT software on the same device, independently.

2.3 What to update?

Standardization of IoT software name-space and meta-data is needed to identify what to update. These standards will be necessary both for IoT software repository management, and for IoT device system aspects e.g. representing on the device current modules, kernel, applications and detecting new components and versions on the fly. Prior work in this area includes for instance adapting to resource constraints of low-end IoT devices the model@runtime paradigm [13] which provides a systematic representation of repository locations, software module specifications, communication bindings between IoT devices. In the future, IoT software will be in large part hardware independent, and may also be software platform independent (e.g similar to POSIX compliant software, outside of IoT), which highlights further the need for standardization in this domain.

2.4 How to distribute updates?

There are two aspects to distribution of software updates, (i) where the software updates are stored and (ii) how the software updates are accessed.

Where are software updates stored? Updates may be in a single repository, or may be replicated in several places in the network. Replication "somewhere on the Internet" is a well-known field and out of scope of this paper. On the contrary, replication (or cache/proxy) of software update contents within the LoWPAN is in scope a priori. The wireless sensor network research community worked on this topic, and new approaches such as Information Centric Networking (ICN) in IoT contexts may also be considered to manage in-network caching of software update contents.

How are software updates accessed? The part of the communication outside the LoWPAN is a well-known field, out of scope of this paper. Inside the LoWPAN, one must distinguish two cases: single-hop scenarios and multi-hop scenarios.

The *single-hop scenario* typically consists in a star-shaped topology, centered on a gateway to the Inter-

net. In this case distribution of updates is straightforward through the gateway's point-to-point link with each IoT device, typically over standard protocols including UDP, IPv6, 6LoWPAN and a low-power MAC (optimizations using broadcast from the gateway are of course possible, e.g. to announce the availability of new updates).

The *multi-hop scenario* is more diverse, and requires routing inside the LoWPAN, and may benefit of in-network caching of software components. Approaches were proposed such as Deluge[15] and FiGaRo[16] for software update distribution algorithms on multi-hop networks, to all devices (Deluge), or to a subset fo devices (FiGaRo).

In all cases, simpler logic on IoT devices may be possible if one assumes the gateway is a trusted proxy and takes some tasks from the IoT devices (The update process could also benefit from its potential knowledge of the LoWPAN e.g. duty cycles of devices, parameters on MAC layer level, routing information etc.).

2.5 How to update securely?

Aside of network-level security (securing communication channels to transmit the update binary etc.) security is typically provided by checks based on signing [14] the binary (or a hash, the signature check of which is lighter work for small processors). A trend is developing whereby IoT hardware embarks security chips [17][18] to speed up cryptographic computations, and that provide means to store keys securely (such hardware also provide means to store keys securely and aim to make it impossible to install updates when the signature does not match a stored key). Alternatively, the gateway (or home router) might take over some security overhead by checking signatures on behalf of IoT devices etc.

Limits of security through updates? On one hand, the unavailability of updates can be destructive: early adopters of smart home hub Wink were frustrated [5] when their devices could not be remotely updated to fix a critical bug. On the other hand, forced updates can also be destructive. For example, Nest reportedly decided to remotely "update" IoT software on home automation hubs, in effect bricking [12] them intentionally, apparently without consent of users...

References

[1] C. Bormann, M. Ersue, A Keranen. "Terminology for constrained-node networks." Internet Engineering Task Force (IETF), RFC 7228, May 2014.

[2] O. Hahm, E. Baccelli, H. Petersen, N. Tsiftes, "Operating Systems for Low-End Devices in the Internet of Things : a Survey," in IEEE Internet of Things Journal, Dec. 2015.

[3] B. Schneier. "The Internet of things is wildly insecure – and often unpatchable." Schneier on Security https://www.schneier.com/essays/archives/2014/01/the_internet_of_thin.html January 2014.

[4] Brouwers, N., Langendoen, K., & Corke, P. "Darjeeling, a feature-rich VM for the resource poor." In Proceedings of ACM SenSys. Nov. 2009.

[5] B. Barrett, "Wink?s Outage Shows Us How Frustrating Smart Homes Could Be." WIRED, April 2015. http://www.wired.com/2015/04/smart-home-headaches/

[6] E. Baccelli, O. Hahm, H. Petersen, and K. Schleiser. "RIOT and the Evolution of IoT Operating Systems and Applications." ERCIM News no. 101, April 2015.

[7] Solorzano, J. "leJOS: Java based OS for Lego RCX." Online at: http://lejos.sourceforge.net

[8] E. Baccelli, O. Hahm, M. Wählisch, M. Günes, T. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in Proceedings of IEEE INFOCOM, April 2013.

[9] Oliver, R., Wilde, A., & Zaluska, E. (2014). "Reprogramming embedded systems at run-time." In Proceedings of IEEE ICST, Sept. 2014

[10] H. Tanriverdi, "Als eine Glühbirne das Smart Home lahmlegte." Süddeutsche Zeitung, March 2015. http://www.sueddeutsche.de/digital/dos-attacke-als-eine-gluehbirne-das-smart-home-lahmlegte-1.2380844

[11] Trend Micro Report, "Researchers Discover a Not-So-Smart Flaw In Smart Toy Bear", February 2016 http://www.trendmicro.com/vinfo/us/security/news/internet-of-things/researchers-discover-flaw-in-smart-toy-bear

[12] A. Gilbert. "The time that Tony Fadell sold me a container of hummus". https://arlogilbert.com/the-time-that-tony-fadell-sold-me-a-container-of-hummus-cb0941c762c1

[13] FJ. Acosta Padilla, F. Weis, J. Bourcier. "Towards a model@ runtime middleware for cyber physical systems." Proceedings of ACM / USENIX DeDiSys, 2014.

[14] Why You Want Firmware Updates, ThingSquare Blog, http://www.thingsquare.com/blog/articles/firmware-updates/

[15] Hui, Jonathan W., and David Culler. "The dynamic behavior of a data dissemination protocol for network programming at scale." Proceedings of the 2nd international conference on Embedded networked sensor systems. ACM, 2004.

[16] Mottola, Luca, Gian Pietro Picco, and Adil Amjad Sheikh. "FiGaRo: fine-grained software reconfiguration for wireless sensor networks." Wireless Sensor Networks. Springer Berlin Heidelberg, 2008. 286-304.

[17] NXP, "A710x family: Secure authentication microcontroller", May 2016 http://www.nxp.com/products/identification-and-security/authentication/secure-authentication-microcontroller:A710X_FAMILY

[18] Infineon Technologies AG, "OPTIGA TRUST P SLJ 52ACA", May 2016 http://www.infineon.com/cms/en/product/productType.html?productType=5546d4624f205c9a014f6eec8c007b9a