Evaluating Differential Firmware Updates for Embedded IoT Device Fleets

Jayden Renee Sorensen*, Syed Aftab Rashid*, Hossam ElHussini[†], Andrei-Marian Dan*
**Hitachi Energy Research*, Segelhofstrasse 1A, Baden-Dättwil, Switzerland
[†]*Hitachi Energy Research*, Montreal, Canada

***Abstract*—Remote firmware updates are critical for maintaining the performance and security of Internet-of-Things (IoT) device fleets. Remote updates are especially critical from a fleet management perspective, requiring thousands of devices to be kept up-to-date. When considering a fleet of devices, transferring large update files is inefficient. Therefore many modern systems consider *differential* updates. In this work, we describe and evaluate a secure differential update implementation for fleets of embedded IoT devices. We demonstrate how differential updates can be performed on a real hardware platform using a popular open-source embedded update framework, SWUpdate. We also show how differential updates can integrate with redundancy concepts such as A/B partitioning and on-device security mechanisms such as secure boot. Our experiments, performed using both real hardware and a device fleet simulator, identify crucial differences between differential and full-image updates: the differential updates can decrease the file size by 49%, the install time by 78%, and require up to 77% more temporary disk memory compared to full-image updates. These key insights are essential for developers and practitioners when selecting the IoT firmware update strategy.**

I. Introduction

Internet of Things (IoT) enables thousands of devices to remotely connect over the Internet and perform a multitude of tasks. The firmware running on these devices requires frequent updates to ensure device security, maintain/improve performance, and upgrade features. For a handful of devices, firmware can be updated locally. However, for large fleets, i.e., a set of IoT enabled devices, comprising thousands of devices, remote firmware updates are the only viable solution [1].

In a remote update, applications, services, firmware, and configurations are transferred from a server to the target device(s) mainly over a network. With the proliferation of software-based solutions across different industries, remote updates are playing an increasingly pivotal role in Industrial IoT (IIoT) [1]–[3]. Remote updates enable device manufacturers to offer new features and services to customers, fix bugs, and ensure the availability of their devices, all without needing to physically access the device(s).

Although remote software and firmware updates are convenient, they also introduce several challenges, including integrity and confidentiality, version control, dependencies, update duration, and failure management [4]. One significant challenge faced when performing remote updates for a fleet of resource-constrained devices is the size of the update image [4]. Sending the entire new firmware image is easy, however, the large file sizes can lead to network bandwidth congestion, higher device memory requirements, and longer download and installation times. This is the reason why many modern systems consider *differential* updates [5], [6].

A differential update represents the *difference* between the contents of on-device firmware and the new firmware to be installed. Depending on the difference between the two, the final update image size can be significantly reduced, leading to lower network bandwidth usage and lower download/install times. Several existing works advocate the use of differential updates [4], [7], focusing on optimized differential image generation [5], [6] and update security [8], [9]. This work complements the existing work by developing and evaluating a framework for integrating differential updates with a secure boot enabled device. Given a device with an existing secure boot implementation, we designed a differential update strategy and compared it with a full image update strategy through experimental evaluation. Our objective is to evaluate the efficacy of differential updates in reducing the update file size, download and install time, and temporary disk memory usage, while preserving the existing secure boot implementation.

The main contributions of this work are:

1) The design of an efficient differential firmware update strategy for secure boot enabled embedded Linux devices, based on the popular update framework, SWUpdate. The update process is integrated with an existing secure boot mechanism and ensures general update security.
2) An end-to-end implementation of the efficient firmware update strategy and extensive evaluation using both real hardware and a device fleet simulator. Our results show that differential updates, on average, decrease the update image size by 49%, install time by 78%, and require up to 77% more temporary disk memory. These experimental insights are key for practitioners and developers when considering differential updates as a potential firmware update strategy.

The remainder of the paper is structured as follows: Section II presents background on SWUpdate, its usage, and introduces other relevant firmware update concepts. Section III provides details on how differential firmware updates can be performed using SWUpdate and integrated into the secure boot mechanism. Further, it explains the security implications of using differential updates and their comparison with a full-image update strategy. In Section IV, we discuss our experimental setup and present the results of our evaluation. Section V then

TABLE I: A/B Dual Partition Device Map

Partition	Purpose	Contents
1	Boot	Bootloader
2	Root A	A: boot files, root filesystem, application(s)
3	Root B	B: boot files, root filesystem, application(s)
4	Data	Read/Write Data

discusses further challenges and potential directions towards applying differential firmware updates to industrial systems. The state-of-the-art is presented in Section VI and Section VII concludes the paper.

II. Background

SWUpdate is a flexible, open-source library for performing firmware updates on embedded Linux devices [10]. We chose SWUpdate instead of Mender [11] or RAUC [12] because of its flexibility and support for differential updates. The differential handler provided with SWUpdate requires no on-device computation and no external chunk downloading, which is optimal for resource-constrained devices with limited connectivity. Moreover, SWUpdate can be modified to support new devices, operating systems and update handlers, making it compatible across many platforms [13].

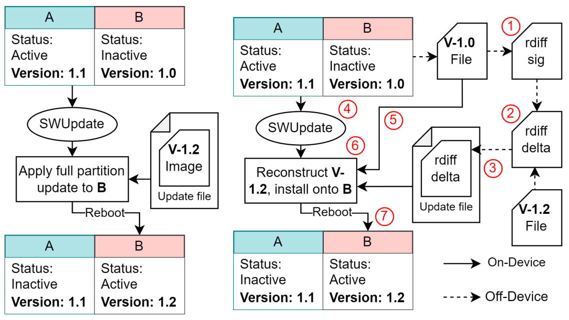
A. Using SWUpdate

As an input, SWUpdate receives a Copy In/Copy Out (CPIO) file archive, which contains: *sw-description* (the update description file), *sw-description.sig* (the security signature of sw-description), update *files* and *images*, and *pre- and post-install scripts* (shell scripts used to set the environment, trigger reboots, etc). The basic process that occurs on the target device to perform an update with SWUpdate is:

1) receive and extract the CPIO archive;
2) verify the sw-description signature with the public key stored on-device;
3) parse the sw-description file and verify the checksums of each update file;
4) run pre-install script(s);
5) perform all file and image updates;
6) run post-install scripts(s).

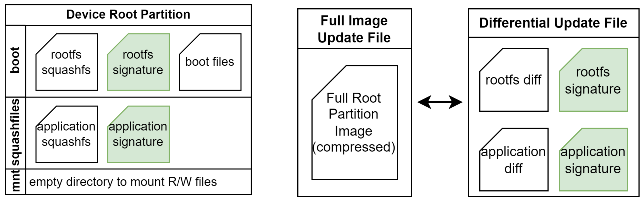
B. A/B Dual Partition

The A/B dual partition concept is used to ensure system redundancy, i.e., if the active root partition fails during an update or in normal operation, the device can restart and boot the firmware on the backup root partition. SWUpdate supports A/B partition configuration with the double-copy update strategy [10]. The dual partition strategy uses two copies of the root partition. One copy is active and running the current firmware version, while the other copy remains inactive and contains the previous version of the firmware. When an update is performed on an A/B partitioned device it is applied to the backup partition, preserving the active partition. After the update, the active and backup partitions switch to load the new firmware. Table I shows an example A/B partition map.



(a) Full Image Update (b) Differential Update

Fig. 1: Examples of an A/B Full Image and Differential Update with SWUpdate



(a) Root Partition Layout (b) Full and Differential Updates

Fig. 2: Device Root Filesystem and Update File Formats

C. Full Image Update

The basic strategy for performing firmware updates on an A/B system is to rewrite the entire backup root partition image with the new firmware image. After the update, the device reboots, switching the active partition to the new firmware. The backup partition contains a previous copy of the firmware, so that in case of a bug, security issue, or failure in the new firmware the device may easily and automatically roll back to the previous stable version. The full-image update method, shown in Fig. 1a, is straightforward and easy to implement, however, it is far from optimized. Between firmware versions, much of the device root partition stays the same. For example, the directory structure of the root partition is typically constant. Moreover, individual files within the root partition may undergo frequent updates, while other files rarely change. This means that large amounts of duplicated data are sent from the update server to the device, wasting bandwidth and increasing update download and install time.

D. RDIFF

Librsync rdiff [14] is a tool used to generate and apply differential files. SWUpdate provides a built-in differential handler, based on the rdiff tool. The rdiff algorithm generates a differential file between the base and target files. To reconstruct the target, the base and differential files are combined. When preparing an rdiff update, the differential files are generated off-device during the firmware development process. The base used to generate the differential must be the firmware file currently installed on the device. During the update, the differential file is applied by SWUpdate to construct the target file. We consider differential updates using the rdiff update handler, as there is no on-device computation required, making them ideal for resource-constrained devices.

E. Secure Boot

Our target device implements an existing hardware-based secure boot mechanism, which must be preserved for differential updates. The implementation of secure boot that we are using is based on the pattern described by Löhr, Sadeghi, and Winandy [15] and the implementation detailed by Schulz and Josserand [16]. Each stage of the boot process, from the initial bootloader to the root filesystem and application(s), is protected with a unique key and signature. All components are privately signed during development and the public keys are saved to the device. Before each file is executed, it must be verified. To ensure no tampering of the keys or firmware files on the device, the first key is immutably embedded in the device hardware. Each subsequent stage is protected by a software key. Using a hardware-rooted secure boot pattern ensures that the device only boots verified firmware [8], [17].

F. Root Filesystem Layout

To ensure the device firmware remains secure and resistant to tampering, the root filesystem and the main application(s) are stored in squashfs format, which is a read-only compressed filesystem. The squashfs files, which are externally signed, are verified and mounted during the boot process. A separate partition is reserved for read/write data. Files utilized by the bootloader may also be stored in the root partition. The layout of the root partition is shown in Fig. 2a.

III. Differential Updates with SWUpdate

SWUpdate's rdiff handler generates smaller update files compared to a full image update. Using the rdiff handler, SWUpdate can perform differential updates of either individual files or the entire partition. A limitation of the librsync rdiff library is that it cannot reconstruct the target file in-place. This implies that for full-partition updates, the differential file must be compared with one partition and applied to the other. Since the backup partition will be updated, the active partition must be referenced. As the state of the active partition cannot be guaranteed, it is safer to not use full-partition updates, and instead target individual files with differential updates. With SWUpdate, differential updates of individual files can be applied directly to the backup partition, without referencing the active partition. SWUpdate creates a temporary file on the backup partition, where the file is reconstructed before being installed in the target location. The rest of our work utilizes single file differential updates, and describes strategies to perform full updates by upgrading individual files.

Instead of sending the entire root partition image, the same update can be performed with targeted differential file updates. Differential files can be created for the individual files that are changed within the root partition. Fig. 2b shows the two SWUpdate files, one for a full image update and the other for the differential update. For the differential update, the partition image has been replaced with differential files that target the individual files changed between versions. The method for generating and performing the full-partition differential updates using the rdiff update strategy is shown in Fig. 1b and consists of the following steps:

1) On the development server, generate the rdiff signature of the base firmware file (e.g., V-1.0) currently installed on the backup partition, B.
2) Create the rdiff delta (differential) between the rdiff signature and the target firmware file (e.g., V-1.2).
3) Package the rdiff delta file into the SWUpdate CPIO update file archive.
4) On the device, receive and parse the CPIO archive.
5) Reference the firmware (e.g., V-1.0) currently installed on the backup partition, B.
6) Reconstruct the target firmware (e.g., V-1.2) and install to the backup partition, B.
7) Reboot with the updated partition, B, as the active partition.

The previous steps can be extended to support updating multiple firmware files on the root partition within the same update.

A. Secure Boot Implementation

A diagram of our secure boot implementation utilizing the U-Boot [18] bootloader is shown in Fig. 3, and can be described in the following stages:

1) first-stage ROM-code bootloader signature is verified using a hardware-embedded key;
2) U-Boot Secondary Program Loader (SPL) is loaded by the first-stage bootloader and verified with a software key;
3) Main U-Boot application is loaded by U-Boot SPL and verified with a software key;
4) fitImage, which contains the kernel and initramfs is loaded and verified by the main U-Boot application;
5) initramfs is verified and loaded by the fitImage;
6) external signatures of the root filesystem and application files are verified and the filesystems are mounted by the initramfs.

The first stage bootloader is device-specific and will not be updated. The U-Boot bootloader (and associated SPL) will likely be updated occasionally. The fitImage will likely be updated more frequently and may be updated with differential updates. Finally, the root filesystem and application files and their associated signatures will be updated frequently, presenting a good target for differential updates. The files contained within the dashed line in Fig. 3 are the primary targets of differential file updates.

B. Differential Updates with Secure Boot

Our proposed process to perform a differential update while maintaining the secure boot chain and updating external signature files is shown in Fig. 4. The update consists of the following steps:

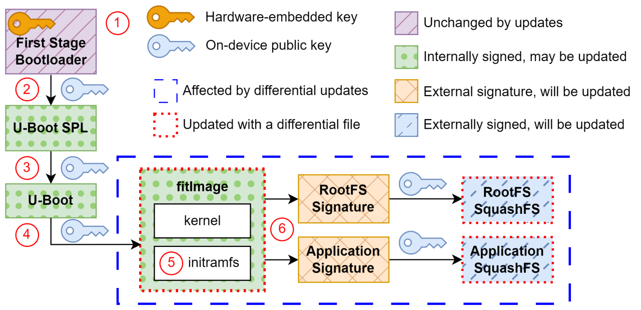


Fig. 3: Example of Secure Boot with the U-Boot Bootloader

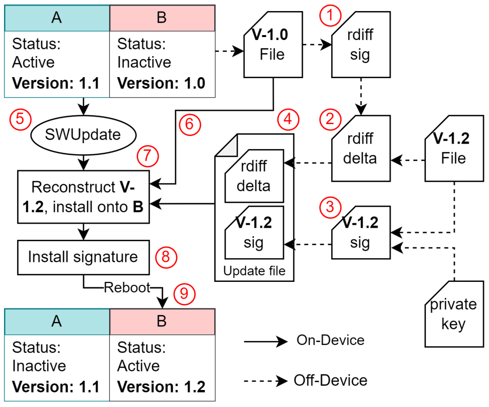


Fig. 4: Differential Update with External Signatures

1) On the development server, generate the rdiff signature of the firmware file (e.g. V-1.0) currently installed on the backup partition, B.
2) Create the delta between the base rdiff signature and the target firmware (e.g., V-1.2) file.
3) Using a securely stored private key, sign the target firmware (e.g., V-1.2) file to generate the external signature required in *Step 6* of the secure boot process shown in Fig. 3.
4) Package the delta and the external signature file into the SWUpdate CPIO archive.
5) On the device, receive and parse the CPIO archive.
6) Reference the firmware (e.g., V-1.0) currently installed on the backup partition, B.
7) Reconstruct the target firmware (e.g., V-1.2) and install it on the backup partition, B.
8) Overwrite the existing external signature (e.g., V-1.0) with the new external signature file (e.g., V-1.2).
9) Reboot with the updated partition, B, as the active partition.

In the strategy described above, the entire signature file is transmitted and the existing signature is completely overwritten. Since the signature files are small (usually only a few hundred bytes) there is no considerable benefit to performing a differential update of the signature file. On the other hand, the squashfs files and the fitImage are updated with differential files, as this can significantly reduce the amount of data needed to complete the update.

TABLE II: Security of differential and full image updates

Requirement	Differential	Full Image
Confidentiality	aes-256-cbc encryption, PKCS#11 for key management	Same. More overhead for encrypting full image.
Integrity	Digital signatures using SHA256	Same.
Availability	Pre-install scripts, backup partition	Same. Easier recovery due to size.
Authenticity	Digital signatures	Same
Non-repudiation	Digital signatures	Same

C. Update Security

To formally assess the security of the proposed firmware update method, we look at the security objectives defined in the CIA (Confidentiality, Integrity, Availability) triad:
Confidentiality: To avoid disclosing senstive information, the firmware update and the sw-description files are encrypted using AES block cipher in CBC mode and 256 bit key (aes-256-cbc). The encrypted firmware files and scripts are then sent to the device where they are decrypted. Moreover, PKCS#11 is used to securely manage and maintain the cryptographic keys used to decrypt the update files
Integrity: digital signatures ensure that the differential update files are not tampered with. The files are signed using SHA256. The SWUpdate library also offers signing the firmware update files using self-signed or Public Key Infrastrcuture (PKI) certificates.
Availability: the proposed firmware update method ensures that a device remains operational if a software update fails due to a Denial of Service (DoS) or other reasons. This is acheived through (a) a pre-install script that checks and detects any mismatches in the differential update files, and (b) a backup partition to ensure that the device does not stop working if an update fails.

Besides the CIA triad, authenticity and non-repudation are also acheived by the use of digital signatures. Particularly:
Authenticity: Digital signatures ensure that the firmware update file identity is verifiable and that it is coming from a trusted source.
Non-repudiation: Using PKI certificates to create the digital signatures of the fimrware update files, the signing entity (server) can not claim not signing/sending the files.

Table II summarizes the main security objectives acheived by the differential firmware update and how it compares to the full image update. While the security objectives are acheived in both update methods (since the underlying technologies are the same), the differential update method has the advantage of lower computational overhead due to smaller image sizes as shown in Section IV.

IV. Experimental Evaluation

We describe two types of experiments that evaluate the differential update strategy. The on-device experiments focus on the local impact of the strategy - the duration to install

TABLE III: Ubuntu Experiment Update Files

(a) Firmware Versions

Update	Base	Target
1	17.10	18.04.1
2	18.04	18.04.2
3	18.04.1	18.04.3
4	18.04.2	18.04.4
5	18.04.3	18.04.5
6	18.04.4	18.10
7	18.04.5	19.04
8	18.10	19.10
9	19.04	20.04
10	19.10	20.04.1
11	20.04	20.04.2
12	20.04.1	20.04.3
13	20.04.2	20.04.4

(b) Firmware Update Sizes

	Full [MB]	Diff [MB]	Reduc. [%]
1	77.1	72.8	5.6
2	77.8	34.6	55.6
3	78.6	39.9	49.2
4	79.0	40.0	49.3
5	79.4	54.0	32.0
6	83.1	74.8	10.0
7	85.9	83.4	2.9
8	84.7	80.8	4.6
9	312.3	311.7	0.2
10	321.1	320.3	0.3
11	324.6	211.3	34.9
12	330.6	227.9	31.1
13	351.3	250.4	28.7

the differential update and the total disk memory required. The device fleet simulator experiment measures the impact of differential updates on the time required to distribute the updates to large fleets of devices.

On-Device Experimental Setup: The on-device experiments are performed on a Raspberry Pi 4 Model B 4GB v1.1, with the following setup: a 32 GB SD card is partitioned into a 100 MiB boot partition, a 1 GiB data partition, and two equally-sized 15.45GB root partitions. One of the root partitions is active, running Ubuntu Server 22.04.3. The active partition contains SWUpdate and all required packages, while the inactive partition contains the experimental firmware.

A. On-Device Update File Size

We generated update files for several different firmware versions in both the full image and differential update format. We used past versions of Ubuntu [19] (Table IIIa) and OpenWRT [20] (Table IVa). To simplify the experiments, the application squashfs file is omitted and only a root filesystem squashfs file is used. For each version, we downloaded the ARM64 firmware image, extracted the root filesystem, and compressed it into squashfs format. These squashfs files were then used to generate the experimental firmware updates. Each update is applied to a base firmware that is two versions behind the target, to simulate dual-copy updating, where the backup partition contains the preceding version of the firmware. The full image updates contain only a gzip-compressed copy of the root EXT4 partition containing the new root filesystem squashfs (format shown in Fig. 2a, excluding the application squashfs). The differential update files contain only the rdiff delta required to achieve the same target firmware version (the differential file for the root filesystem squashfs). No extra files or scripts are included in the updates, and the signatures were omitted for the experiments. The contents of the two update files are shown in Fig. 2b. The file sizes for each full image and differential update are shown in Table IIIb for Ubuntu and Table IVb for OpenWRT.

The reduction in update file size for differential updates depends on the changes between the base and target versions, e.g., updates between minor versions of the firmware

TABLE IV: OpenWRT Experiment Update Files

(a) Firmware Versions

Update	Base	Target
1	22.03.0	22.03.2
2	22.03.1	22.03.3
3	22.03.2	22.03.4
4	22.03.3	22.03.5
5	22.03.4	22.03.6
6	22.03.5	23.05.0
7	22.03.6	23.05.1
8	23.05.0	23.05.2

(b) Firmware Update Sizes

	Full [MB]	Diff [MB]	Reduc. [%]
1	6.3	2.1	66.1
2	6.3	2.7	57.1
3	6.3	3.4	45.7
4	6.3	2.1	66.1
5	6.4	2.2	65.6
6	6.2	5.7	8.3
7	6.2	5.6	10.2
8	6.2	1.8	70.9

are significantly reduced with differential updates. For larger updates with many changes, e.g., between major firmware versions, using differential updates may fail to provide provide a meaningful reduction. The average OpenWRT update size is reduced by 49%, while the Ubuntu updates, which include more major versions, are reduced by 23%, on average.

B. On-Device Update Duration

We measure the update duration for both the differential and full image updates. Each full image and differential update is repeatedly applied to its respective base firmware. The duration of each update is recorded with the Linux time utility and each update type is run 10 times. The data in Table Va for Ubuntu and Table VIa represents the average update time (both real time and CPU time) for each version.

The real time reflects how long the update actually takes to complete, from start to finish. The CPU time represents the time spent performing actual calculations. It is not influenced by the time spent waiting for file I/O and other processes, which makes it more consistent between runs. For the OpenWRT updates, using differential updates reduced the real time by 78%, on average, and the CPU time by 64%. The Ubuntu updates saw 37% decrease, on average in real time and a 25% decrease, on average, in CPU time when using differential updates. The difference in update duration is not proportional to the file size reduction between the full image and differential updates. Instead, the differential update duration is dependent on the size of the target file. This is because the rdiff library reconstructs the target file block-by-block, copying blocks either from the base file or from the differential file. The size of the target file controls how many blocks must be copied, which therefore determines the duration of the update. However, even in cases where there is nearly no reduction in file size (rows 9 and 10 of Table Va) the differential updates are faster than the full image updates. One contributing factor is that decompressing the gzip-compressed image file takes longer than reconstructing the differential files for the same update. We measured the real time for the file decompression and the rdiff reconstruction for each Ubuntu update 10 times. On average, the rdiff reconstruction is 23% faster (real time) than the gzip decompression for the same update. However, this difference does not explain the full discrepancy between the differential and full image updates, as it amounts to a 12 second decrease in elapsed time, on

TABLE V: Ubuntu Experiment Results

(a) Average Update Duration

	Full [s]		Diff [s]		Reduc. [%]	
	Real	CPU	Real	CPU	Real	CPU
1	63.7	10.1	37.0	8.4	42.0	17.2
2	75.3	10.0	32.8	5.5	56.3	45.0
3	70.0	10.3	32.9	6.3	52.9	38.4
4	75.1	10.6	36.0	5.8	52.1	45.9
5	67.1	10.5	36.5	7.3	45.6	30.4
6	82.2	10.6	43.8	9.0	46.8	15.0
7	63.2	11.2	50.9	9.6	19.4	14.6
8	75.4	11.2	36.3	9.3	51.9	16.6
9	297.6	39.0	227.9	34.8	23.4	10.8
10	295.3	37.1	233.7	35.7	20.9	3.7
11	306.1	39.5	232.0	26.7	24.2	32.4
12	297.8	39.2	253.7	29.1	14.8	25.9
13	364.6	43.0	263.3	30.9	27.8	28.2

(b) Tmp. Memory Usage

	Full [MB]	Diff [MB]	Reduc. [%]
1	77.1	150.9	-95.6
2	77.8	113.3	-45.6
3	78.6	119.4	-51.9
4	79.0	120.0	-51.8
5	79.4	134.3	-69.2
6	83.1	159.0	-91.2
7	85.9	170.3	-98.3
8	84.7	166.5	-96.6
9	312.3	624.0	-99.8
10	321.1	641.3	-99.7
11	324.6	535.9	-65.1
12	330.6	558.5	-68.9
13	351.3	601.6	-71.3

TABLE VI: OpenWRT Experiment Results

(a) Average Update Duration

	Full [s]		Diff [s]		Reduc. [%]	
	Real	CPU	Real	CPU	Real	CPU
1	11.4	1.8	2.3	0.5	80.3	70.3
2	15.0	1.8	2.6	0.6	82.9	67.2
3	8.7	1.7	2.4	0.6	72.3	63.0
4	20.5	1.7	2.7	0.5	87.0	68.6
5	13.8	1.7	3.0	0.5	78.3	70.4
6	9.0	1.7	2.5	0.9	72.3	49.7
7	9.2	1.7	2.5	0.8	73.4	53.0
8	9.8	1.7	2.4	0.5	75.4	73.2

(b) Tmp. Memory Usage

	Full [MB]	Diff [MB]	Reduc. [%]
1	6.3	8.5	-33.8
2	6.3	9.0	-42.8
3	6.3	9.8	-54.2
4	6.3	8.5	-33.7
5	6.4	8.5	-34.3
6	6.2	11.9	-91.8
7	6.2	11.8	-90.0
8	6.2	8.0	-29.2

average, while the Ubuntu differential updates are 47 seconds faster than the full image updates, on average. The rest of the discrepancy could be attributed to file I/O waiting and the SWUpdate handler implementations, as directions for further investigation.

C. On-Device Disk Memory

Performing any type of update requires sufficient disk memory to be available for temporary copies of the update images, files, and scripts. Streaming is not supported for SWUpdate differential updates, and for full image updates, it eliminates the possibility of verifying the integrity of the entire update before starting the installation. For this reason, only non-streamed updates are considered in this experiment. The update file is parsed and extracted into a temporary directory (e.g, /tmp). For differential updates, the differential files are reconstructed in the temporary directory before being copied to the target location. If there is insufficient space to extract the update files or reconstruct the differential files, the update will fail. To determine which files SWUpdate created, when they were created and deleted, and the size of the files, a watch was set on the temporary directory. The amount of disk memory required to complete a full image update is the same as the size of the update file after the decompression of the update archive. Differential updates require disk memory to extract the update file and reconstruct the differential files. The maximum total disk memory required is the size of the update file *plus* the size of the largest reconstructed differential file, as SWUpdate reconstructs the differential files sequentially. In this experiment, the only file is the root filesystem squashfs archive. Therefore, the total amount of memory required to complete the differential update is the size of the differential update file (the file sizes are shown in Tables IIIb and IVb) *plus* the size of the root filesystem squashfs (approximately the size of the full update).

Table Vb shows the total disk memory required to complete each Ubuntu update, while the OpenWRT data is shown in Table VIb. On average, the OpenWRT differential updates require 51% *more* disk memory than the equivalent full image updates. The Ubuntu differential updates require, on average, 77% more disk memory than the full image updates. In the experiments where only one squashfs file is updated, the differential updates require more disk memory than the full image updates. This is a key insight when implementing differential updates on constrained devices. In the case where multiple files are updated (for example, the rootfs squashfs, application squashfs, and fitImage) the differential updates become more efficient (as only one target file is reconstructed at a time, instead of the full partition image being downloaded), and may even consume less memory than the full image updates, depending on the sizes of the target and differential files.

D. Device Fleet Simulator

This experiment measures how decreasing the update file size impacts the update distribution time across fleets of devices. We have used an open-source network simulator, i.e., Fleet sImulator for Scalability Tests (FIST) [21] to perform this experiment. FIST is an extendable network simulator that simulates a variety of realistic IoT scenarios, including large fleets of devices and realistic network conditions, with several configuration options. The simulator is implemented with three distinct components: the update manager, the network simulator, and the device simulator. We use Eclipse hawkBit Update Server v0.4.1 [22] as the update manager. Updates are sent from the hawkBit server to the simulated devices via a network simulator. The network simulator is implemented with Docker Traffic Control [23], which simulates a realistic network with the following parameters: *delay* - 50 ms (time that each packet is delayed), *loss* - 2% (probability that a packet will be lost), *corrupt* - 1% (probability of packet corruption), and *duplicate* - 2% (probability of packet duplication). The simulated devices are implemented in the Go programming language [24]. Each device runs as a goroutine, which is a lightweight execution thread. The simulated devices implement the API required to connect to the hawkBit update manager, as well as the infrastructure required to receive the update file.

The simulation was run with fleet sizes of 1000, 2000, 3000, 4000, and 5000 devices.The update transmission process was simulated with various firmware sizes. A maximum firmware size of 70 MiB was selected and scaled down to 80%, 60%, 40%, and 20% of the maximum. Fig. 5 shows the results for all fleet and update file sizes, with the total duration measured in minutes. Reducing the update file size leads to a directly proportional decrease in the total download time. For example, reducing the update file size by 80% leads to an approximately

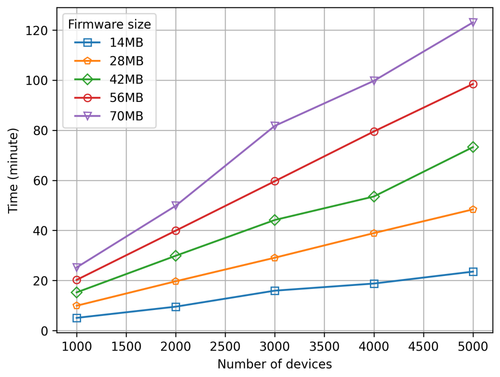


Fig. 5: Device Fleet Simulator Experiment Results

80% decrease in download time for all tested fleet sizes. This linear relationship holds for all update and fleet sizes tested in the experiment. In cases where minimizing the update download/transmission time is important, differential updates should be considered to reduce the update file size.

V. Towards Differential Updates for Industrial Automation Systems

In addition to the performance aspects considered in the previous sections, implementing differential updates in industrial automation systems introduces several challenges related to ensuring update security, managing update lifecycles, and properly applying updates. In this section, we discuss some of the challenges that might be encountered when using differential updates in an industrial setup, future work directions, and potential solutions.

A. Security and Robustness

Differential updates must be performed without compromising the update or device security. Towards a secure and efficient firmware update solution, section III considers the security aspects of differential updates, with a focus on preserving security while performing differential updates. This work also provides a comparison between a differential and a full image update, as using the SWUpdate differential handler must not open new security vulnerabilities in the firmware update process. The SWUpdate rdiff differential update handler uses the same foundation as the raw image update handler, and therefore shares the same security features. The update files are signed with a private key on the server and must be verified by a public key on the device before being installed. As well, the individual files within the update are verified with SHA256 checksums before the update can proceed. To ensure the secure boot process is not compromised, the external signatures of the squashfs files must be securely updated when their associated squashfs files are modified. In the proposed differential update strategy, the external signature files are sent alongside the differential files as raw files. Since the signature files are small, there is no benefit to updating them with a differential file. These external signatures can be securely generated during the development process so the private keys are kept secure.

B. Applying the Correct Update

The current SWUpdate rdiff handler does not check the version of the base file before applying the differential update. If the wrong base file is referenced, the update will either fail or produce an invalid file. A potential solution can use a pre-install script, which runs before the update starts and ensures that the firmware on the backup partition of the device matches the firmware used to create the differential file. If the firmware on the device does not match, the update is aborted before any changes are made.

C. Understanding Device History

The firmware versions installed on both the active and backup partitions must be known to transmit and apply the correct differential update. The solution relies on a fleet management system that is in place and allows the operator to access information about each device, including the active and backup firmware versions.

D. Update Lifecycle

Since differential updates must be applied to a specific base, generating updates for a new release means creating differentials between potentially many existing versions, leading to increased development and testing effort. Future work might investigate a strategy to limit the effort and to prepare differential updates between the new version and the previous N versions. For example, assuming $N = 3$, if a new version, 1.16, is released, differential updates are prepared for versions 1.15, 1.14, or 1.13. If the target is version 1.12 or older, the full update is used.

VI. Related Work

The secure boot implementation presented in this paper follows the pattern set out in [15] and the implementation in [16]. Related work on remote updates focuses on individual aspects of the process. [4] describes current challenges and considerations for completing remote updates for IoT devices and presents two possible frameworks for performing updates. In regards to differential updates, [5] demonstrates a method for generating minimal differential files, which is tested on several low-powered microcontrollers. [6] presents a method of applying minimal firmware patches to perform efficient updates on resource-constrained devices. From a security perspective, [8] reviews secure boot patterns for IoT devices. [9] discusses how to use multiple layers of security both in the update process and on the device to perform secure OTA updates for IIoT devices. In [9], strategies are presented for securely generating, transmitting, and applying updates, as well as for implementing secure boot with signed firmware on the device. In [7] a custom update manager is developed to perform firmware updates on high-powered IoT devices. Full firmware updates are completed on a Linux operating system running on an IoT-connected Raspberry Pi 4. The experimental device setup described in [7] utilizes the dual-copy A/B root partition setup. [13] provides a framework for implementing OTA updates with SWUpdate on RISC-V devices. The authors integrated SWUpdate with U-Boot to test several types of updates, including differential updates. s Our work compliments the existing work on differential updates but primarily focuses on the practical aspects of implementing differential updates in a secure manner and evaluating their performance on real devices with publicly available firmware.

VII. Conclusion

In this paper, we developed a differential firmware update solution and created a prototype using an open-source embedded update framework, SWUpdate. We showed how our differential update solution can be applied to secure boot enabled devices and discuss its security implications. We then used our experimental setup and a device fleet simulator to compare the performance of differential and full-image updates. Experimental results reveal that differential updates reduce the file size, especially for routine, minor updates. This decrease in update file size directly correlates to a shorter update distribution time and a faster installation process. In certain cases, the differential updates require more disk memory to complete than a full image update. Overall, these key insights enable practitioners to select the IoT firmware update strategy most suitable to each use case.

References

[1] S. Halder, A. Ghosal, and M. Conti, "Secure over-the-air software updates in connected vehicles: A survey," *Computer Networks*, vol. 178, p. 107343, 2020.

[2] F. Kauer, F. Meyer, and V. Turau, "A holistic solution for reliable over-the-air software updates in large industrial plants," in *WISES*, 2017, pp. 29–34.

[3] J. Bauwens, P. Ruckebusch, S. Giannoulis, I. Moerman, and E. De Poorter, "Over-the-air software updates in the internet of things: An overview of key principles," *IEEE Communications Magazine*, vol. 58, no. 2, pp. 35–41, 2020.

[4] J. Hernández-Ramos, G. Baldini, S. Matheu, and A. Skarmeta, "Updating iot devices: challenges and approaches," in *GIoTS*, 2020, pp. 1–5.

[5] O. Kachman, M. Balaz, and P. Malik, "Universal framework for remote updates of low-power devices," *Computer Communications*, vol. 139, 2019.

[6] O. Kachman and M. Balaz, "Efficient patch module for single-bank or dual-bank firmware updates for embedded devices," in *DDECS*, 2020.

[7] N. Nikolov, O. Nakov, and G. Popov, "Research firmware update over the air for high performance embedded systems," in *COMSCI*, 2023, pp. 1–4.

[8] R. R.V. and K. A., "Secure boot of embedded applications - a review," in *ICECA*, 2018, pp. 291–298.

[9] K. G. Crowther, R. Upadrashta, and G. Ramachandra, "Securing fota for iiot devices," in *2022 IEEE HST*, 2022, pp. 1–8.

[10] S. Babic, "Swupdate: software update for embedded systems," accessed on: 02.02.2024. [Online]. Available: https://sbabic.github.io/swupdate/swupdate.html

[11] Northern.tech, "Mender - how it works," accessed on: 02.02.2024. [Online]. Available: https://mender.io/how-it-works

[12] J. Luebbe and E. Joerns, "Rauc documentation," accessed on: 02.02.2024. [Online]. Available: https://rauc.readthedocs.io/en/latest/

[13] A. K. Srivastava, K. CS, D. Lilaramani, R. R, and K. Sree, "An open-source swupdate and hawkbit framework for ota updates of risc-v based resource constrained devices," in *C2I4*, 2021, pp. 1–6.

[14] M. Pool, "Librsync: Rdiff command," accessed on: 02.02.2024. [Online]. Available: https://github.com/librsync/librsync/blob/master/doc/rdiff.md

[15] H. Löhr, A.-R. Sadeghi, and M. Winandy, "Patterns for secure boot and secure storage in computer systems," in *ARES*, 2010, pp. 569–573.

[16] Q. Schulz and M. Josserand, "Secure boot from a to z," 2018, accessed on: 21.03.2024. [Online]. Available: elinux.org/images/e/e0/Josserand-schulz-secure-boot.pdf

[17] R. Wang and Y. Yan, "A survey of secure boot schemes for embedded devices," in *ICACT*, 2022, pp. 224–227.

[18] The U-Boot development community, "The u-boot documentation," accessed on: 02.02.2024. [Online]. Available: https://docs.u-boot.org/en/latest/

[19] The Ubuntu development team, "Ubuntu old releases," accessed on: 08.02.2024. [Online]. Available: https://old-releases.ubuntu.com/releases/

[20] OpenWRT Project, "Welcome to the openwrt project," accessed on: 25.03.2024. [Online]. Available: https://openwrt.org/start

[21] F. Muratori, Y. Yang, Z. Rejiba, and A. M. Dan, "A framework for configurable scalability evaluations of iot platforms," *The 39th International Conference on Advanced Information Networking and Applications (AINA-2025)*, 2025.

[22] "hawkbit update server," accessed on: 12.02.2024. [Online]. Available: github.com/eclipse/hawkbit/blob/master/hawkbit-runtime/hawkbit-update-server/

[23] L. Lacht, "Docker traffic control," accessed on: 12.02.2024. [Online]. Available: https://github.com/lukaszlach/docker-tc

[24] "The go project," accessed on: 12.02.2024. [Online]. Available: https://go.dev/project