Effective Over-the-Air Reprogramming for Low-Power Devices in Cyber-Physical Systems

Ondrej Kachman(✉) and Marcel Balaz

Institute of Informatics, Slovak Academy of Sciences,
Dubravska cesta 9, 84507 Bratislava, Slovakia
{ondrej.kachman,marcel.balaz}@savba.sk

Abstract. Cyber-physical systems often include sensor devices in their structure. These devices may require firmware updates once deployed and these updates must be energy efficient for battery powered, physically inaccessible sensors. The problem of energy saving reprogramming can be split to four tasks– making old and new firmware versions more similar, generating small delta files using differencing algorithms, propagating delta files and applying updates at the end devices. This paper describes existing approaches dealing with this problem, analyzes their power consumption and introduces new optimizations for differencing algorithms. A new approach is presented that requires no external flash memory, device reboot or complex update agent at the sensor device.

Keywords: Reprogramming · Low-power devices · Over-the-air programming · Firmware similarity · Differencing algorithm · Update agent · Power consumption

1 Introduction

As the self-programming flash memories and wireless communication entered the world of embedded systems, over-the-air reprogramming became possible. It was especially examined for wireless sensor networks (WSNs), where hundreds of small sensor devices may be deployed in a large area. Firmware preloaded on devices in WSN is usually developed under test conditions and may not work properly once the devices are deployed. Over-the-air reprogramming made it easier to update firmware on the devices, but transferring the whole firmware image to many devices is energy inefficient as sending 1 bit wirelessly can consume the same energy as 1000 instructions [1]. This is critical for wireless low-power devices powered by batteries that require firmware update.

The problem of effective over-the-air reprogramming motivates the research question of this paper. How do over-the-air updates influence energy consumption of the updated devices and what can be done to improve the energy efficiency of these updates. The task is also to identify partial problems in this area and their solutions. We present our contribution to differencing algorithms and show how it can help to perform firmware updates effectively.

L. Camarinha-Matos et al. (Eds.): DoCEIS 2016, IFIP AICT 470, pp. 284–292, 2016.
DOI: 10.1007/978-3-319-31165-4_28

2 Relationship to Cyber-Physical Systems

Cyber-physical systems (CPS) are a promising direction in automated interaction between the physical and the virtual world. Based on information collected from physical world, CPS can properly respond with programmed actions. Interaction between the two worlds is provided by intelligence of CPS, which may be created by 3D displays, sensors, actuators, cameras and other devices. This links CPS with previously mentioned WSNs and low-power devices. Sensors and actuators often create an important part of CPS and face the same challenges [2], for example limited battery life, network traffic processing with limited computing capabilities or the adaptation to environment. Over-the-air reprogramming for small devices within CPS should be effective in terms of energy consumption, speed and security.

Energy can be wasted by excessive amount of update data disseminated through the network to end devices. Many protocols have been developed for WSNs [3–5]. However, WSNs usually consist of the same or similar devices and serve to collect data. CPS may consist of many devices with different resources and tasks that may be networked by different technologies, for example WLAN, Bluetooth or GSM [6]. When embedding over-the-air programming capabilities into the low-power device's firmware, developers must consider the right networking technology and protocols in order to reduce the amount of energy consumed by the firmware updates. Another way of energy wasting can be caused by the firmware code. Modern microcontrollers usually have firmware code stored in a program flash memory. Firmware updates are applied from delta files that encode the differences between older and newer versions. Some approaches proposed alteration to compilers [7, 8] and linkers [9] in order to make the versions more similar. Compiler alterations led to more instructions, thus worse execution times and some linker alterations resulted in fragmented program memory. Calls to functions in different parts of flash memory can lead to worse energy consumption as activation of different flash regions requires additional energy [10]. When considering many different devices and platforms in CPS, compiler and linker alterations would be required for every different platform. Developers can instead work with the product of compilers– object files, which usually have the standard executable and linkable (ELF) format, and work with sections and relocation entries [11].

Update speed is important for CPS too. Ineffective update dissemination can slow down network traffic and long downtime of important devices may cause CPS to malfunction. The problem of network traffic can be addressed by effective protocols, Quality of Service mechanisms (QoS) and reduced amount of update data using delta files generated by the differencing algorithms that compare the old and new firmware versions. The delta files are later processed by the end devices and their decoding followed by application of the update may greatly influence the update speed, so their structure and encoding must be taken into consideration by the firmware developers.

The security problem of over-the-air updates concerns CPS as well as any other systems of collaborating computational devices. A device with rogue firmware could infect the network and compromise the system. Network protocols disseminating the update data must implement some security mechanisms and the delta files should include an integrity check to prevent them from tampering.

3 State of the Art and Related Work

The area of effective over-the-air reprogramming can be split into different problems. Based on our analysis of the existing research in this area, we split it to four partial problems: (1) Firmware version similarity, (2) Delta generating differencing algorithms, (3) Update dissemination, and (4) Update application.

Overview of over-the-air update procedures is shown in Fig. 1. The following subsections analyze state of the art for each of the listed problems. One more subsection on flash memory energy consumption modeling is included as a base for our energy analysis of over-the-air updates.

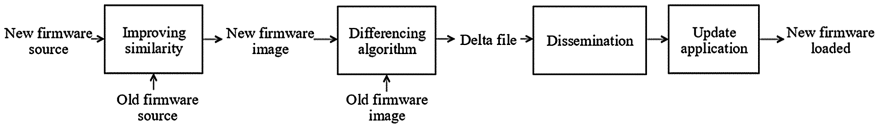

Fig. 1. Over-the-air updates overview

3.1 Firmware Version Similarity

Researchers agree, that firmware updates are more energy effective when data that must be sent through the network are minimal, so the devices save energy on the network communication. First step towards the smaller update data is through improvement in the similarity of two consecutive firmware versions. Considering multiplatform CPS, ELF files are the best way to improve firmware similarity. Authors of [12] identified 4 ELF file properties that caused larger delta file generation – branches, global variables, indirect addressing and relative jumps. Preserving addresses allocated for functions, variables and constants between the firmware versions results in a smaller difference between the linked files. Some of these problems can be addressed by changes to object file relocation entries [11, 13]. Tools altering the ELF files to improve the similarity of firmware images can help the differencing algorithms to perform significantly better.

3.2 Delta Generating Differencing Algorithms

The differencing algorithms that generate delta files (also called deltas or patches) for low-power devices must generate delta as small as possible and encode it in such a way, that the end device can decode it and apply the update easily. Basic approaches used block level comparison [14], which was fast but resulted in unnecessarily large deltas. These deltas transferred whole blocks of fixed size that have changed. Soon, byte level differencing algorithms and delta files were introduced [15–17]. These algorithms detected common sequences in the firmware images and new data. The difference information were encoded into delta files using usually two basic operations – COPY and ADD, each with different attributes. Basic format encoded into file is following:

- <COPY > <old address > <new address > <number of bytes>
- <ADD > <new address > <number of bytes > <first byte > … < last byte>

Every attribute may cost different amount of bytes. Cost for every attribute of an operation adds up to operation's total cost within a delta file, usually in bytes or words.

3.3 Update Dissemination

The delta dissemination stage greatly depends on the network protocol. One of the most notable protocols for WSNs updates is Deluge [4], included in TinyOS, running on many wireless sensor nodes. As mentioned in Sect. 2, CPS consist of different platforms and the network topology may be dynamic. Standard protocols can be used on the higher layers of the CPS. Supported by the QoS mechanisms, update data should reach end devices quickly. If these low power devices cannot support standard the communication protocols (like the most common TCP/IP protocol) due to their computational restraints, they should be provided with border routers that would communicate with them using some lightweight protocol [2].

3.4 Update Application

Updates are applied at the end devices using update agents [18]. The update agents decode the received delta file and execute decoded operations. Depending on implementation, the new firmware image can be reconstructed in an external memory and loaded into the program memory after reboot [11], or it can be applied on the fly, when the update agent edits the program memory directly and runs new firmware right after all the operations from the delta file are executed [12]. This saves the requirement of reboot and firmware loading, but may result in device malfunction if the update is not applied properly or interrupted. The Rollback option from external memory should be therefore present on these devices if possible.

3.5 Flash Memories Energy Consumption Modeling

Creating an accurate model for energy consumption of flash memories is not an easy task. There are many parameters that must be taken into account. The energy model proposed and validated in [10] is focused on embedded flash memories. The flash memory controller activates different regions of flash. These regions are created by 2^k bytes and activation of each region consumes E_k energy. Formal representation:

$$i \to j = \sum_{k=0}^{N(i,j)} E_k \tag{1}$$

i and j represent memory address. Term N(i,j) represents the largest changed region – $2^{N(i,j)}$ bytes, $N(i,j) = \log_2(i \oplus j)$. The authors of [19] created and validated an instruction-level energy model for embedded systems. They define a memory access energy consumption of a process as E_{memory}:

$$E_{memory} = E_{cntr} + E_{Flash} + E_{SRAM} \quad (2)$$

E_{cntr} is energy consumed by the memory controller. E_{flash} is energy consumed by the flash memory write, read and erase operations. E_{SRAM} is energy consumed by write and read operations in RAM memory. This basic model serves us for theoretical evaluation of our differencing algorithm optimizations.

4 Research Contribution and Innovation

In our research of update energy consumption, apart from delta file size, we also take into account encoding of a delta file and update application. We propose optimizations of the differencing algorithms and delta file encoding for updates without external flash. We also create an energy consumption model that can evaluate our solution.

4.1 Differencing Algorithms for Updates Without Use of an External Memory

Low-power devices can be updated without use of an external memory. Self-programmability allows the device to rewrite its program memory on the fly. Some low-power devices used in CPS for control may not have an external memory, so they have to be updated without it if needed. On the fly update does not require copying of identical firmware parts like reconstruction in the external memory. Update application in device's program memory therefore requires different delta file encoding.

Delta file encoding. Updates executed on the fly require different delta file encoding. The traditional way that the reconstructs firmware mixes ADD and COPY operations.

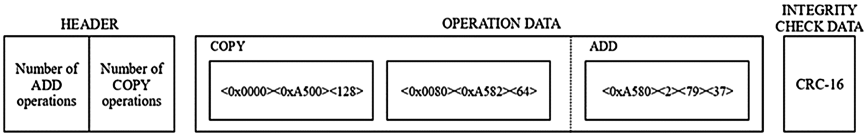

Fig. 2. Delta file format for updates without external memory usage

Now, COPY operations must be carried out first before new data are added. We propose the delta file to be split into three parts – header, operation data and integrity check data (Fig. 2). The header consists of two numbers, the number of COPY and the number of ADD operations. The operation data encode addresses and bytes for each operation. Integrity check data encode CRC-16 code that is checked prior to the firmware update in order to prevent corrupted delta files from updating firmware.

Optimizing differencing algorithms. All analyzed differencing algorithms work with full firmware images. For two images of length n and m, the space complexity in the worse cases is $O(n \times m)$ [15], in the better cases $O(n + m)$ [13, 17]. If we apply xor operation on these images and only compare non-matching sections of a new image to an old image, it is possible to improve algorithm's space complexity, which also results

in better execution time. Let k be the number of non-matching bytes between n and m bytes of compared images. Space complexity is reduced to $O(n + k)$ as k is less than m.

Non-matching segments are built using COPY and ADD operations. The generated operations may be sometimes so small that they can be merged into preceding or following operation. For ADD operation with cost x bytes (without data bytes) and COPY operation with cost y bytes, we propose following optimizations:

- If COPY copies at most y bytes, it can be merged into preceding ADD
- If two consecutive COPY operations copy less than $2 \times$ bytes, they can be merged into ADD operation
- If result of COPY operations merging are two consecutive ADD operations, they can be merged into one ADD operation

4.2 Modeling Energy Consumption of the Update

We define energy consumed by flash write operation as $\lambda_{\text{flash(write)}}$. This is energy consumed by the erase and write operations for one page. Let s be the flash page size, *addr* the address we start writing data to and n the number of bytes we write. NAND memories can only be written one page at time, so the number P of pages we must write in order to store n bytes is:

$$P = \lfloor (addr + n) \div s \rfloor - \lfloor addr \div s \rfloor + 1 \tag{3}$$

Using this formula, we can also determine the number of pages we read from, but flash memory can read bytes directly. If energy needed to perform each read operation is $\lambda_{\text{flash(read)}}$, then reading n bytes and writing them during COPY and ADD operations will result in flash memory energy consumption:

$$E_{flash} = n \cdot \lambda_{flash(read)} + P \cdot \lambda_{flash(write)} \tag{4}$$

E_{cntr} can be calculated based on Eq. (1). We omit small region activations, for example reading a file byte by byte, and take into account only jumps from location we read to page we write and back ($2P$ jumps total). This usually causes activation of more regions. If jump between two addresses i and j requires activation of R regions, jumping between these addresses during writing of P pages will consume energy:

$$E_{cntr} = 2P \sum_{k=0}^{R} E_k \tag{5}$$

4.3 Experimental Results and Discussion

We implemented the differencing algorithm called Delta Generator (DG) with proposed optimizations that generates delta files in the proposed format. The evaluation was done for 7 firmware change cases on ATmega32U4 microcontroller with 32 KB of self-programming flash memory. The page size of this memory is 128 bytes. We compare our solution

to R3diff differencing algorithm [13], one of the best differencing algorithms that generate deltas for firmware updates in an external memory. For this purpose we created tool that calculated the amount of required read and write operations encoded in deltas based on Eq. (3). We also calculated the approximate number of region activations (Eq. (5)) that update process required. Results of our analysis are shown in Table 1.

Table 1. Comparison of R3diff [13] differencing algorithm to proposed Delta Generator (DG)

Change case	Changed bytes	Delta size (bytes)		Read bytes		Written pages		Regions activated	
		R3diff	DG	R3diff	DG	R3diff	DG	R3diff	DG
1	2	15	12	6	4	1	1	28	28
2	2668	966	954	1370	1361	45	37	922	772
3	1222	100	106	628	623	18	12	348	236
4	3054	1522	1448	1621	1674	138	121	2990	2636
5	3150	2051	1986	1675	1722	140	120	3036	2618
6	648	1193	970	563	408	120	85	2936	2198
7	3136	2057	1990	1672	1709	138	122	3026	2724

The experimental results show upgrade over existing solution – R3diff. The amount of read operations is roughly the same, but the proposed optimizations reduced the amount of page writes. The calculation of activated flash regions is not precise, byte by byte readings were excluded from the calculations. We were not yet able to determine exact power consumption of read and write operations for the flash memory used. The exact numbers could better show how not only size, but also encoding influences the energy efficiency of over-the-air updates. We also have not yet evaluated similarity improving techniques. We provided basic model that is a base for future improvements.

5 Conclusion and Further Work

We identified partial problems in the area of over-the air updates. In addition to this, we created the basic energy consumption model of reprogrammed memories and investigated the effect of delta file encoding on update energy performance. The current model is basic. It should be upgraded to a more detailed model and experimentally evaluated with real data. The executed experiments have shown good results. The vision is to create a precise model that takes into account similarity improvements, differencing algorithms, energy consumption of wireless interfaces, energy consumption of update agents and updated firmware. These are the main problems that influence effectivity of over-the-air updates. Perfection of solutions to these problems in combination with good energy consumption model can lead to fast, secure and energy effective updates for low-power devices in cyber-physical systems.

Acknowledgment. This work has been supported by Slovak national project VEGA 2/0192/15.

References

1. Levis, P., Culler, D.: Maté: a tiny virtual machine for sensor networks. In: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X), pp. 85–95. ACM Press, New York (2002)
2. Wu, F.-J., Kao, Y.-F., Tseng, Y.-C.: From wireless sensor networks towards cyber-physical systems. Pervasive Mob. Comput. **7**, 397–413 (2011). Elsevier B.V., Philadelphia
3. Stathopoulos, T., Heidemann, J., Estrin, D.: A Remote Code Update Mechanism for Wireless Sensor Networks. Technical report, Center for Embedded Networked Sensing (CENS). University of California (2003)
4. Hui, J. W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: Proceedings of the 2nd International Conference on Information Processing in Sensor Networks (IPSN 2008), pp. 81–94. ACM Press, New York (2004)
5. Aschenbruck, N., Bauer, J., Bieling, J., Bothe, A., Schwamborn, M.: Selective and secure over-the-air programming for wireless sensor networks. In: 21st International Conference on Computer Communications and Networks (ICCCN), pp. 1–6. Munich (2012)
6. Shi, J., Wan, J., Yan, H., Suo, H.: A Survey of cyber-physical systems. In: International Conference on Wireless Communications and Signal Processing (WCSP). Nanjing (2011)
7. Zhang, Y., Yang, J., Li, W.: Towards energy-efficient code dissemination in wireless sensor networks. In: International Symposium on Parallel and Distributed Processing (IPDPS), pp. 1–5. Miami (2008)
8. Huang, Y., Zhao, M., Xue, C. J.: WUCC: Joint WCET and update conscious compilation for cyber-physical systems. In: 18th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 65–70. Yokohama (2013)
9. Koshy, J., Pandey, R.: Remote incremental linking for energy-efficient roprogramming for sensor networks. In: Proceedings of the Second European Workshop on Wireless Sensor Networks, pp. 354–365. Istanbul (2005)
10. Pallister, J., Eder, K., Hollis, S. J., Bennet, J.: A high-level model of embedded flash energy consumption. In: International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), ACM Press, New York (2014)
11. Dong, W., Liu, Y., Chen, C., Bu, J., Huang, C., Zhao, Z.: R2: incremental reprogramming using relocatable code in networked embedded systems. IEEE Trans. Comput. **62**, 1837–1847 (2013)
12. Shafi, N. B., Ali, K., Hassanein, S.: No-reboot and zero-flash over-the-air programming for wireless sensor networks. In: 9th Annual IEEE Communication Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), pp. 371–379. Seoul (2012)
13. Dong, W., Mo, B., Huang, C., Liu, Y., Chen, C.: R3: optimizing relocatable code for efficient reprogramming in networked embedded systems. In: IEEE INFOCOM Proceedings, pp. 315–319. Turin (2013)
14. Jeong, J., Culler, D.: Incremental network programming for wireless sensors. In: 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON), pp. 25–33. Santa Clara, California (2004)
15. Hu, J., Xue, C. J., He, Y., Sha, E. H.-M.: Reprogramming with minimal transferred data on wireless sensor network. In: 6th International Conference on Mobile Adhoc and Sensor Systems (MASS 2009), pp. 160–167. Macau (2009)
16. Panta, R.K., Bagchi, S., Midkiff, P.: Efficient incremental code update for sensor networks. ACM Trans. Sens. Netw. (TOSN) **7**, 30 (2011). ACM Press, New York

17. Mo, B., Dong, W., Chen, C., Bu, J., Wang, Q.: An efficient differencing algorithm based on suffix array for reprogramming wireless sensor networks. In: International Conference on Communications (ICC), pp. 773–777. Ottawa (2012)
18. Jurković. G., Sruk, V.: Remote firmware update for constrained embedded systems. In: 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1019–1023. Opatija (2014)
19. Bazzaz, M., Salehi, M., Ejlali, A.: An accurate instruction-level energy estimation model and tool for embedded systems. IEEE Trans. Instrum. Meas. **62**, 1927–1934 (2013)