Configurable Reprogramming Scheme for Over-the-Air Updates in Networked Embedded Systems

Ondrej Kachman
2nd year, full-time study
Supervisor: Ladislav Hluchý, Consultant: Marcel Baláž

Institute of Informatics, Slovak Academy of Sciences
Dúbravská cesta 9, Bratislava, Slovak Republic
ondrej.kachman@savba.sk

Abstract—**Networked embedded systems are nowadays used in various applications and the number of devices used in such systems grows by millions each year. Physically inaccessible, resource constrained low-power devices sometimes require remote over-the-air reprogramming. For the last 20 years, many reprogramming schemes have been developed and built upon. These schemes are required to be fast and energy effective. The amount of operations executed on the target devices should be minimal, delta files shared on the network should be very small. This paper analyzes existing solutions and proposes a new, configurable reprogramming scheme, that can provide more control over the process of the device reprogramming.**

Keywords—networked embedded system, over-the air update, reprogramming, low-power device

I. INTRODUCTION

Recent advances in the internet-of-things technologies enabled fast development of various systems of collaborating computational devices, also called networked embedded systems. Wireless sensor networks (WSNs) are the first example of such systems. WSNs are used to collect various data from their environment and mostly consist of low-power devices. These systems evolved into cyber-physical systems (CPS) in the last 10 years. These systems collect and process data, then take actions based on the results of processed information. CPS structure usually includes small low-power devices as sensors and actuators, various networking devices for networking and powerful computers for data processing [1].

A. Low-power devices and their reprogramming

This paper is focused on the low-power devices often found in the described systems. These devices create an interface between physical and cyber world. The amount of the low-power devices in CPS or WSNs may vary, but it can often reach hundreds. Some of the devices may not be physically accessible after their deployment. These devices are battery powered and communicate through wireless network interface. They are expected to run for months or years after their deployment. Firmware of the devices developed under the test conditions may malfunction in the real environment. In that case, reconfiguration is required [2]. If simple reconfiguration of some parameters does not help, firmware requires reprogramming. To prevent the waste of energy, update data shared on the network and the number of operations executed on the target devices must be minimal. This is the main reason for the development of energy efficient reprogramming schemes for low-power devices.

B. Related work

The fist solutions developed in early 2000's were loading the full firmware image onto the target device, then using bootloader to replace the old firmware version. Incremental reprogramming, loading many firmware versions one after another, resulted in rapid battery depletion. This triggered the development of block based reprogramming schemes. These schemes found updated blocks of the firmware, then sent these blocks to the target devices reducing amount of the data shared on the network and also the number of written memory cells. Latest reprogramming schemes use byte based differencing, providing the best results.

1) Remote incremental linking [3]*:* In 2005, remote incremental linking scheme was proposed. This scheme introduced slop regions – a free space in the program memory between the firmware functions. This approach enabled functions to grow and shrink in their slop regions, reducing the shifts and relocation changes in the firmware, making it possible to generate smaller delta files.

2) Zephyr [4]*:* Another roprogramming scheme proposed function indirection table to reduce the impact of function shifts on the delta generation. This approach created a table that pointed to all functions and the firmware would call functions through this table. However, this table only handled `call` instructions and not relative calls and jumps. Jumping to and from the table also resulted in worse execution time of a firmware.

3) Hermes [5]*:* Built over Zephyr, this scheme allocates fixed addresses for variables, further ruducing the delta files size. This approach scans through the source files before the compiler is invoked and puts initialized and uninitialized variables into assigned structures, preserving their order when

the compiler generates `.bss` (for uninitialized variables) and `.data` (for initialized variables) sections.

4) R3 reprogramming scheme [6]*:* This approach is focused on the relocatable code and object files. The object files for many low-power platforms have standard executable and linkable format (ELF). The files generated by compiler have relocatable format. Reference instructions in `.text` sections need their addresses resolved by linker. R3 sets these relocatable entries to zero, generates small metadata for a loader located on the target device, that resolves relocations during boot. Authors demonstrated improvement in delta size over existing solutions.

5) Q-diff scheme [7]*:* This scheme is also focused on the object files. It also takes advantage of slop regions and improves the technique with possibility of placing a function and its slop region into a different part of the memory. This scheme changes layout of .bss and .data sections, requiring changes to instructions with indirect addressing of variables. To prevent changes to relative jumps, Q-diff adds new code blocks to the end of the memory, then points `call` instructions to those blocks. This may add more execution time to the firmware. Solution claims to be platform independent, however, authors do not address the problem of platform specific relocation types and how they identify relative or fixed reference instructions. One of the main benefits of Q-diff is that the update process does not require external memory and reboot to finish successfuly.

II. Definition of a problem

Previous chapter provides some insight into some of the most relevant reprogramming schemes for constrained low-power devices. This chapter sums up the state-of-the art and describes, what can be done to improve the existing solutions.

A. *Compilers, linkers and object files summary*

Most of the analyzed literature avoids direct changes to compilers or linkers. This is very important, as the manufacturers of the devices usually provide compiler and linker for their platforms that is able to generate the most optimized code. There are approaches that altered registers allocated during compilation, but they worsened firmware execution time after every update.

The best solution is to work with the object files. These files have standard ELF format and can be examined in their relocatable form, before linking, or in their executable form, after linking. By examining the relocatable files, linking process can be managed better.

B. *Platform independency summary*

Even solutions that claim to be platform independent cannot be fully independent. The authors of such solutions may have focused on the same family of devices only, or avoided explanation of their definition of platform independent.

Every microcontroller family uses different relocation types. In order to be able to work with relocations, we must obtain the definitions of relocation types for chosen platforms. Every relocation type is calculated differently, some relocations have the same value at every memory position (`call` instructions, `load` from and `store` to RAM), some may change their value at different address (`rjmp` and `rcall` instructions).

Furthermore, if the software alters the source code directly in the object files, it must include the complete instruction set of a chosen platform, making it even more platform specific. This is the case of Q-diff approach, that changes indirect instructions. To sum up, no solution can be fully platform independent.

C. *Memory fragmentation summary*

Slop regions fragment program memory. Different approaches argue that it is a waste of space and inefficient use of program space. There are also some speculations, that fragmented memory consumes more energy.

If the linker is configured to provide slop regions to functions, it places them to the different parts of the program memory, resolves relocations correctly and generates executable file. No additional instructions are generated and the code is optimized by the compiler in the previous stage. As the most low-power devices currently use NAND flash memory for their firmware, the access time is the same. The point of slop regions is to use the program space to the full potential and enable to generate smaller deltas. It is not a waste of space nor inefficient.

We carried out some experiments and evaluated energy consumption of a fragmented firmware. These experiments and their results are described in the following chapter.

D. *Proposed solution*

We propose configurable reprogramming scheme. Various methods have their own advantages and for different reprogramming strategies, different solutions may perform better. If there are not many incremental updates, there is no need to make unnecessary changes to function or variable placements. Requirements for our scheme:

- Enable memory fragmentation and defragmentation. Fragmented memory with slop regions is better for frequent incremental updates with small deltas. Defragmented memory is the default state and has slightly better energy consumption.
- Do not alter the source code. The source code is optimized for the best performance by the compiler, added instructions cause worse execution time. The reprogramming scheme also does not need to include processor specific instruction sets.
- Take advantage of the relocatable entries. These can be read from the object files before linking. Their final address can be found out from the executable file. These entries can be used for memory fragmentation and can be stored on the target device, consuming memory but enabling small deltas, or they can be sent to the device saving memory but using larger deltas.
- Enable updates that do not require external memory of a device. Apply the update on-the-fly and do not reboot

the device. Only clear the stack and registers if necessary.

- The solution must enable differencing algorithms to generate as small delta files as possible to prevent network congestion and overall waste of energy on the network communication.

III. USING RELOCATABLE CODE FOR MEMORY FRAGMENTATION

This chapter describes, how will our solution use relocatable entries in object files to fragment and defragment program memory. First, we perform and experiment, that measures energy consumption of a chosen device with defragmented and fragmented memory. Then, we describe how relocations are resolved and which relocations have to be altered in a function when it is shifted.

A. Energy consumption of a fragmented memory

We perform an experiment on an ATmega32u4 microcontroller with 32KB of NAND flash program memory. We base this experiment on the energy consumption estimation model for NAND flash memories [8]. The model suggests that the energy consumed during activation of a memory cell depends on how far from the previously activated region the cell is. The memory regions are created by 2^k bytes and activation of each region consumes E_k energy. Formal representation:

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

i and j represent memory address. Term N(i,j) represents the largest changed region – $2^{N(i,j)}$ bytes.

$$N(i,j) = \lfloor \log_2(i \oplus j) \rfloor \quad (2)$$

The energy E_k varies. The most energy is consumed by activation of a different page. Activation of cells within the same page does not consume as much energy.

The test firmwares consisted of two or five jumps between the different pages of memory. The microcontroller was powered by a stable source with the voltage of 5V. We observed current consumption of each test scenario. The results of experiments are in the Table 1.

TABLE I. AVERAGE CURRENT CONSUMPTION FOR FRAGMENTED MEMORY

Firmware	**Max. activated regions**	**Average current (mA)**	**Description**
1	2	28,61	Loop on a single page
2	7	29,80	Loop on a two consecutive pages
3	11	30,06	Loop on a two shifted consecutive pages
4	14	30,20	2 pages, 16KB jump
5	15	30,28	2 pages, 32KB jump
6	14	30,27	5 jumps throughout the whole 32KB

We observe the increase in the current when we need to activate more regions for jumps. The worst case scenario is 5% worse than the best case scenario. For expected lifetime of 5 years, this could prolong the device's battery life by 3 months. However, it is not possible to keep the whole firmware on a single page. Second scenario, with 2 consecutive pages, saves only 1,4% - 2 weeks out of 5 years. This is more realistic. This means that fragmented memory does require more energy, but not unacceptably more. Temporary fragmentation of a memory should not deplete the battery too fast and it can help to generate smaller delta files and save energy on a network communication.

B. Shifting functions in the program memory

This is currently in a development stage and has not been tested yet. For each function that is shifted, we must change relative instructions within this function and also all relative instructions pointing to this function. Fig. 1. illustrates a simple scenario with a shift of a function_2. Two relative jumps must be changed in the firmware in order for it to work correctly, a relative call to itself does not have to be changed.

To find the relocations and their values, we must explore both relocatable and executable object files:

- For every relocatable file, list all the `.text` sections with relocatable entries and store their offsets, sizes and types.
- Resolve the final addresses of `.text` sections from the executable file. These addresses are assigned during linking.
- Using the offsets of relocatable entries, find all final values of the generated relocations.

Now, when the function is to be shifted, we know exactly which relocations must be altered. Note, that we do not add any additional instructions to the code, only alter the existing relative instructions (and possibly some calls to the shifted function).

Currently, the tool that extracts all relocatable entries from the object files is complete. It requires platform specific relocation types to determine which relocations are relative. We have yet to program an update agent responsible for function shifts on the target devices.

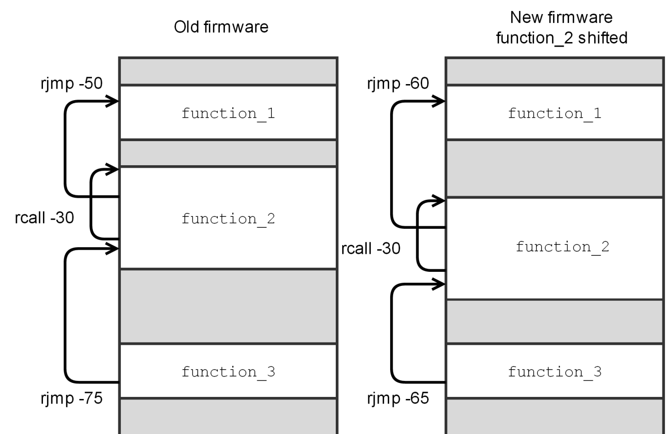

Fig. 1. Changes to some relocatable entries after a function shift

IV. Generating delta files

The proposed technique will grant the programmers full control over the layout of a firmware. It will also be possible to store the table with relocations on the device. Providing single functions or whole modules with slop regions aims to generate small deltas for frequent, incremental updates. Differencing algorithms compare firmware images and encode data shifts as COPY operations and new data as ADD operations.

We developed a differencing algorithm that can update a device without use of an external memory, does not require reboot and does not use RAM memory, it only requires 4 generic purpose registers [9]. The algorithm is called Delta Generator and performed better than R3diff [6] (up to 19%) in 6 out of 7 firmware change cases.

Table 2 shows the sizes of the delta files generated by two differencing algorithms – R3diff and Delta Generator. R3diff sets all relocations to zero, Delta Generator uses slop regions. Column ‘Changed’ shows the amount of bytes that changed between the old and the new firmware. Once we improve our scheme to fully handle relocations and function shifts, delta files of Delta Generator should be even smaller.

Table II. Delta file sizes generated by the differencing algorithms

Firmware change case	Changed (bytes)	R3diff [6] Delta (bytes)	Delta Generator [9] Delta (bytes)
1	2	15	12
2	2668	966	954
3	1222	100	106
4	3054	1522	1448
5	3150	2051	1986
6	648	1193	970
7	3136	2057	1990

V. Aims of the dissertation thesis, conclusion

This chapter lists aims of the dissertation thesis and provides short commentary on how they will be accomplished. The aims are listed in the following subchapters.

A. *Definition of parameters that influence performance and energy consumption of over-the-air firmware updates*

The thesis will provide in-depth analysis of the chosen problem. It will list all known challenges in this area along with their solutions. Some of these problems have been mentioned throughout this paper.

B. *Proposal of an energy consumption estimation model for over-the-air updates of low-power devices*

Energy consumption estimation models can help evaluate any reprogramming scheme. With the more possible configurations of the firmware, these models can help choose the most effective update strategy. We published paper that describes how these models can help evaluate the energy efficiency of the reprogramming schemes – [10].

C. *Design of a reprogramming scheme for fast and energy effective over-the-air updates of low-power devices*

We developed the differencing algorithm that generates small delta files. We also developed an update agent for target devices that does not use external memory, RAM memory and does not require reboot. We are currently working with relocatable code to give developers more control over the firmware layout, thus make proposed reprogramming scheme configurable.

D. *Implementation of a proposed scheme and its evaluation on a chosen hardware platforms*

Once the reprogramming scheme is complete, we will evaluate it on the ATmega microcontroller family, the MSP430 microcontroller family, and one other platform that has not been chosen yet. We aim at the low-power devices often used as sensors or actuators.

E. *Conclusion*

Our work is showing promise, but the real challenge is making our firmware reprogramming scheme more configurable. We published 2 papers from the completed work. Solution proposed in this paper is yet to be implemented on the target platforms and evaluated. This work has been supported by Slovak national project VEGA 2/0192/15.

VI. References

[1] F.-J. Wu, Y.-F. Kao and Y.-C. Tseng, "From wireless sensor networks towards cyber-physical systems," in *Pervasive and Mobile Computing*, vol. 7, Elsevier B.V., 2011, pp. 397-413.

[2] J. Shi, J. Wan, H. Yan and H. Suo, "A Survey of Cyber-Physical Systems," in *International Conference on Wireless Communications and Signal Processing (WCSP)*, Nanjing, 2011.

[3] J. Koshy and R. Pandey, "Remote Incremental Linking for Energy-Efficient Reprogramming fo Sensor Networks," in *Proceeedings of the Second European Workshop on Wireless Sensor Networks*, 2005.

[4] R. K. Panta, S. Bagchi and S. P. Midkiff, "Efficient incremental code update for sensor networks," *ACM Transactions on Sensor Networks (TOSN),* vol. 7, no. 4, February 2011.

[5] R. K. Panta and S. Bagchi, "Hermes: Fast and Energy Efficient Incremental Code Updates for Wireless Sensor Networks," in *IEEE INFOCOM 2009*, Rio de Janeiro, 2009.

[6] W. Dong, B. Mo, C. Huang, Y. Liu and C. Chen, "R3: Optimizing relocatable code for efficient reprogramming in networked embedded systems," in *IEEE INFOCOM Proceedings*, Turin, IEEE, 2013, pp. 315 - 319.

[7] N. B. Shafi, K. Ali and H. S. Hassanein, "No-reboot and Zero-Flash Over-the-air Programming for Wireless Sensor Networks," in *9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, Seoul, 2012.

[8] J. Pallister, K. Eder, S. J. Hollis and J. Bennett, "A high-level model of embedded flash energy consumption," in *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Jaypee Greens, 2014.

[9] O. Kachman and M. Balaz, "Optimized Differencing Algorithm for Firmware Updates of Low-Power Devices," in *19th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, Kosice, 2016.

[10] O. Kachman and M. Balaz, "Effective Over-the-Air Reprogramming for Low-Power Devices in Cyber-Physical Systems," in *Technological Innovation for Cyber-Physical Systems*, Lisbon, 2016.