Poster Abstract: Incremental firmware update using an efficient differencing algorithm

Konstantinos Arakadakis*
konarak@ics.forth.gr
Institute of Computer Science, Foundation for Research and Technology-Hellas (FORTH)
Heraklion, Greece

Alexandros Fragkiadakis
alfrag@ics.forth.gr
Institute of Computer Science, Foundation for Research and Technology-Hellas (FORTH)
Heraklion, Greece

ABSTRACT

Modern IoT solutions require minimisation of the transmitted data during a firmware update, in order to save energy for the constrained devices. To accommodate this requirement, IoT nodes can be updated incrementally using only parts of the current firmware, which is already stored in their flash memory in order to reconstruct the new firmware locally. This the role of the so-called differencing algorithm that executes in a firmware server and aims to detect common segments between the current and the new firmware, producing an encoded small *delta script*, which is finally transmitted to the IoT nodes. In this work, we present a differencing algorithm that operates in byte-level and can compute optimal, in terms of size, *delta scripts* in $O(nlogn)$ time and $O(n)$ space complexity.

CCS CONCEPTS

• **Theory of computation → Design and analysis of algorithms**; **Theory and algorithms for application domains**.

KEYWORDS

Internet–of–Things (IoT), delta script, differencing algorithm, firmware update, over–the–air–programming

ACM Reference Format:
Konstantinos Arakadakis and Alexandros Fragkiadakis. 2020. Poster Abstract: Incremental firmware update using an efficient differencing algorithm. In *The 18th ACM Conference on Embedded Networked Sensor Systems (SenSys '20), November 16–19, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3384419.3430471

1 INTRODUCTION

Firmware updates are frequently released to introduce new functionality and fix bugs of previous versions. Instead of transmitting the whole firmware each time a new version is released, IoT devices can be updated incrementally, a method in which a firmware server creates a *delta script* that constitutes of a set of commands that the nodes have to execute to reconstruct the new firmware locally.

Usually, these commands can either be *COPY* or *ADD*. A *COPY* command is used to encode a segment of bytes that does not need to be transmitted, as the receiving node already has it and can copy it e.g. from the flash memory the current firmware is stored. Additionally, each *COPY* command contributes β bytes to the final delta script, for the opcode itself and information regarding the initial address, the length of the segment, etc. An *ADD* command encodes the non-matching segments of bytes, contributing α bytes plus the number of the bytes in the payload. As these bytes contribute mostly to the size of the final delta script, the differencing algorithm used should be able to encode the majority of the bytes of the new firmware in *COPY* commands, matching them with the corresponding segments of the current one.

Several differencing algorithms are presented in the literature (see [1]). For example, *RMTD* [2] is able to detect the following types of common segments: (i) segments between the new and the current firmware in forward and reverse order, and (ii) segments between the new and the partially reconstructed new firmware in both forward and reverse order. For example, a segment S = "ABCD" in the new firmware can be matched with a segment "ABCD" or "DCBA" in the current firmware version. However, *RMTD* is criticised for its excessive time and space complexity ($O(n^3)$ and $O(n^2)$) that render it unusable for complex applications. Another differencing algorithm, *DASA* [4], is able to find the common segments between the new and current images in forward order, but with a lower time and space complexity ($O(nlogn)$ and $O(n)$), using suffix arrays [3] (*SA*s) and longest common prefix (*LCP*) arrays. *SA*s are data structures that store the starting indexes of all suffixes of a string, sorted in lexicographical order, while *LCP* arrays are auxiliary data structures that store the lengths of the longest *LCP* between adjacent suffixes stored in suffix arrays. The differencing algorithm proposed here is based on *DASA*, having the same complexity, also being able to detect the same types of common segments as *RMTD*, however resulting to smaller delta scripts compared to *DASA*. We note here that this work is currently in-progress.

2 METHODOLOGY

Initially, the current and the new firmware images are used for the construction of the byte array T (Figure 2), and then the *SA* is computed, sorting the resulted l_4 suffixes. Additionally, the *LCP* array is computed, calculating the length of the longest common prefix of each suffix with its previous one in the suffix array. Next, the algorithm finds the optimal delta script that encodes all the bytes of the new firmware (see Algorithm 1).

The algorithm follows a greedy approach and for each byte i of the new firmware, it checks if it is more efficient (size-wise) to use an *ADD* or a *COPY* as the last command of the script that

*Also with Department of Computer Science, University of Crete.

SenSys '20, November 16–19, 2020, Virtual Event, Japan

ACM ISBN 978-1-4503-7590-0/20/11…$15.00
https://doi.org/10.1145/3384419.3430471

Corrected Version of Record. V.1.1. Published November 25, 2020

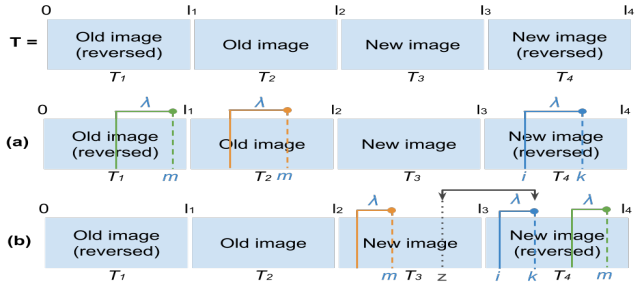

Figure 2: The construction of byte array T and the common segments between: (a) new and current firmware, (b) new and partially reconstructed new image. The segment in blue is the corresponding segment starting at the reference suffix in T_4, while the orange and green are matching ones in forward and reverse order.

Algorithm 1: *Delta script generation*

$opt_0 \leftarrow 3, opt_0^C \leftarrow \infty, opt_0^A \leftarrow 3$
$commands_0 \leftarrow$ "*ADD* Byte 0"
for *i=1 to new_firmware_size - 1* **do**
  $k \leftarrow finder(i)$
  if k = *None* **then**
    $opt_i^C \leftarrow \infty$
  else
    $opt_i^C \leftarrow opt_{k-1} + \beta$
  end
  $opt_i^A \leftarrow min(opt_{i-1}^A + 1, opt_{i-1}^C + \alpha + 1)$
  $opt_i \leftarrow min(opt_i^A, opt_i^C)$
  if $opt_i = opt_i^A$ **then**
    $commands_i \leftarrow$ "*ADD* Byte i"
  else
    $commands_i \leftarrow$ "*COPY* for Bytes k to i of the update"
  end
end

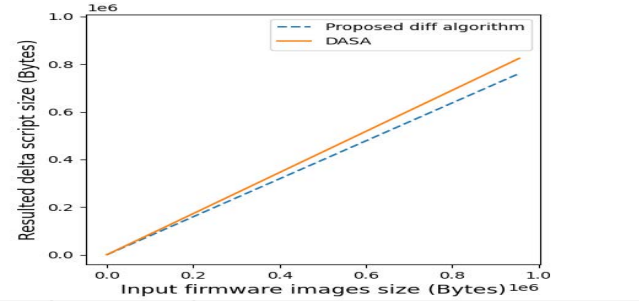

Figure 1: *Delta script* size for various firmware sizes

could encode the first i bytes of the image. In order to solve this optimisation problem, three notations are used: opt, opt^A and opt^C. Opt^A refers to size of the delta script, having an *ADD* as the last command, while opt^C assumes a *COPY* as the last command. In order to determine whether there is an index k, so that the segment of the new firmware $[k...i]$ can be reconstructed copying another segment e.g. from the current firmware, the routine *finder* is called for byte i (described in Section 2.1). The *delta script* is a subarray of the array *commands[]*. Some preliminary results (Figure 1 show that the proposed algorithm outperforms *DASA* in terms of the *delta script* size (shown in x-axis).

2.1 Finder

The *finder* routine is called for each byte i of the new firmware and returns the indexes k and m, so that the segment $[k...i]$ of the new image can be reconstructed, copying a segment starting with byte m. This can be done efficiently using the *SA* and *LCP* array that were computed previously. For this reason, the T_4 part of the string T is used as reference, as for each byte i, the algorithm tries to find a matching segment based on the suffix that starts at the corresponding index in T_4 (namely $l_4 - i$). Then, the algorithm visits the suffixes that are ranked close to the suffix in T_4 and finds the one that shares the longest *LCP* (of length λ) and can be used to encode a segment of the new firmware as shown in Figure 2. If λ is not zero, k is given as $i - \lambda$, otherwise a *COPY* command cannot be used. Obviously, the indexes of the bytes in T for both k and m must be mapped back to the correct ones in the new and the current firmware images. It must be noted that for the first two cases of common segments, as shown in (a) of Figure 2, the matching segments just need to be contained in T_1 and T_2, respectively. However, for the other two types, it must be ensured that the two segments do not overlap in the update image, and that the matching segment precedes the reference one in order. Hence for the orange segment in (b), m should always be less than z, the mirroring index of k in T_3.

3 CONCLUSION-FURTHER WORK

Differencing algorithms should produce the smallest possible delta scripts in order to minimise the required transmitted data over a network. In this work, we presented a differencing algorithm that is able to detect a variety of common segments, producing an optimal *delta script*, in terms of size, in $O(nlogn)$ time and $O(n)$ space complexity. Further work includes the implementation of the proposed algorithm in constrained IoT nodes and evaluate its performance in terms of the execution time and energy required.

4 ACKNOWLEDGMENTS

This research has been financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE (project code: T1EDK-03389).

REFERENCES

[1] K. Arakadakis et al. *Firmware over-the-air programming techniques for IoT networks – A survey.* Accessed: 2020-10-02. 2020. URL: https://arxiv.org/abs/2009.02260.
[2] J. Hu et al. "Reprogramming with Minimal Transferred Data on Wireless Sensor Network". In: *Proc. of the 6th International Conference on Mobile Adhoc and Sensor Systems.* IEEE, 2009.
[3] Udi Manber and Gene Myers. "Suffix Arrays: A New Method for on-Line String Searches". In: *SIAM J. Comput.* (1993).
[4] B. Mo et al. "An efficient differencing algorithm based on suffix array for reprogramming wireless sensor networks". In: *Proc. of the 2012 IEEE International Conference on Communications (ICC).* 2012, pp. 773–777.