Patching A Patch - Software Updates Using Horizontal Patching

Milosh Stolikj, Pieter J. L. Cuijpers, and Johan J. Lukkien, Member, IEEE
Dept. of Mathematics and Computer Science, Eindhoven University of Technology,
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands

Abstract—This paper presents a method for optimizing incremental updates of consumer electronic devices running multiple applications, called horizontal patching. Instead of using separate deltas for patching different applications, the method generates one delta from the other. Due to the large similarities between the deltas, this horizontal delta is small in size. In all test cases horizontal patching produced smaller deltas, with compression ratios between 8.02% and 43.38%.

I. INTRODUCTION

Today’s consumer applications, such as a DLNA based media system [1], are running on multiple, networked devices. A more pervasive upcoming system is an adaptive ambient lighting system [2], that employs a network of low capacity nodes with different roles. For instance, while some nodes measure luminance, others are responsible for switching the light actuators.

Updating software is an essential feature of modern CE devices, for the purpose of bringing new functionality, or correcting discovered bugs. Since the number of nodes to be updated can be large, the communication medium has limitations and the update should be swift, a software update is a non-trivial task. This is especially true for sensor networks where the lifecycle of nodes depends on small batteries.

II. BSDIFF DELTA ENCODING

We use the BSDIFF [3] format for delta encoding. An update with BSDIFF is created in two steps (Figure 1). First, a delta (δ) between the two versions is constructed. Then, the delta is compressed using Bzip2 (c(δ)) and sent to the node for update. There, after decompression, the delta is applied to the old version to reconstruct the new version. BSDIFF has a two-pass algorithm to construct optimized deltas. In the first pass, completely identical blocks are found...
in the two versions. Next, these blocks are extended in both directions, such that every prefix/suffix of the extension matches in at least half of its bytes. These extended blocks correspond to the modified code.

The BSDIFF delta is built of three parts (Figure 3): a control block of commands; a diff block of bytewise differences between approximate matches and an extra block of new data. When the old and new version are similar, the diff block consists of large series of zeroes, which are easy to compress.

III. HORIZONTAL PATCHING

Consider a network with two node types (A and B) as in Figure 2. Currently, when the operating system needs to be updated, two separate deltas are created, one for each node type. Both deltas are distributed independently.

In horizontal patching, one delta is used as a basis, and the other delta is an update of the first one (Figure 4). The motivation is that the operating system is the largest part of the software in both cases, and that is the part that is changed. Both deltas hold the same modifications, thus the horizontal patch between them is smaller in size than the vertical one.

The combined delta then consists of the basis and the horizontal delta, compressed together (c(δ₀ + δ₂), or c(δ₁ + δ₃)). E.g., when δ₀ and δ₂ are used, only δ₀ needs to be executed for updating node type A. On node type B, first δ₂ is executed on δ₀, producing δ₁; finally, δ₁ is executed (Figure 5). The savings in space by using the combined delta in the multi-hop part of the network outweighs the loss in using it in the last-hop part.

![Fig. 4. Possibilities for horizontal patching.](image)

![Fig. 5. Horizontal patching in practice.](image)

IV. EVALUATION

We tested on a sample set of seven applications for the Contiki operating system [7]¹, and two consecutive operating system updates (Table I). We considered all combinations of 2 applications, and for each combination we computed the size of all deltas in Figure 4. The first application (node type A) was always larger than the second one (node type B).

We used the compression ratio, i.e., the percentage of data saved from transmission from the original data, as a metric: 

\[ cr = \left(1 - \frac{\text{delta size}}{c(\delta_0) + c(\delta_2)}\right) \times 100. \]

The results in Table II indicate that largest reductions in size are achieved by using horizontal patching from larger to smaller applications (δ₀ + δ₂). Improvements differ depending on the type and size of applications. The gain is the smallest when both applications are larger than the operating system. When both applications are of similar size, horizontal patching gives up to 43% smaller deltas.

V. CONCLUSION

In this paper we presented horizontal patching, a method for optimizing the size of incremental updates in a multi-application environment. Horizontal patching reduces the size of updates by constructing one delta from another. We validated our hypothesis with experiments for two applications.

As future work, we foresee the analysis of impact of the additional processing in horizontal patching on the total delay of an update. Furthermore, when the number of applications increases, interesting combinatorics come into play for choosing the set of possible and optimal updates. Additional analysis would define which delta should be taken as a basis, and how the update should be formed.

Although the method was demonstrated in a sensor network, it can be used for updating CE devices which share the same code base. We showed that updating the firmware of several CE devices, as television sets or smart phones, can be done by a flooding scheme using smaller updates.

REFERENCES


