Technical Disclosure Commons

Defensive Publications Series

January 03, 2019

FIRMWARE COMPRESSION MECHANISM FOR SPEEDING UP FIRMWARE UPDATING IN A RESOURCE RESTRICTED NETWORK

Yinfang Wang

Chuanwei Li

Feiliang Wang

Yajun Xia

FIRMWARE COMPRESSION MECHANISM FOR SPEEDING UP FIRMWARE UPDATING IN A RESOURCE RESTRICTED NETWORK

AUTHORS:
Yinfang Wang
Chuanwei Li
Feiliang Wang
Yajun Xia

ABSTRACT

Techniques are described herein for adopting a substring list creation and extraction mechanism as well as a variable-length patch mechanism to generate an Over-The-Air (OTA) image path file having a small size. This saves network resources and reduces side effects on data transmission / network service when executing image / firmware upgrading. As a result, OTA updating can be sped up, and the peak network traffic during the firmware updating may be narrowed.

DETAILED DESCRIPTION

Field Area Network (FAN) solutions are being developed for smart utility applications, such as Advanced Metering Infrastructure (AMI) and Distribution Automation (DA). A Network Management System (NMS) manages and operates thousands of nodes in multiple Personal Area Networks (PANs). The firmware updating also is controlled by the NMS. Existing solutions for firmware updating in a mesh network use a simple mechanism for transmitting the entire image from the NMS.

Typically, the entire updating image size can be 100KB or more on different platforms. Moreover, the wireless environment is strongly affected by environmental conditions that change over time (e.g., temporal changes in interference, physical obstruction, propagation characteristics of the physical media, etc.). That causes missing packets. Existing solutions for firmware upgrades in mesh networks use the simple mechanism of re-transmitting all the packets from the NMS in several rounds in case of missing packets. As a result, the entire upgrade process usually takes days or weeks in a mesh network with thousands of nodes. But users require the firmware updates to complete quickly and have minimal side effects on daily business.

One solution is to provide a fixed-length (e.g., 20KB) firmware difference (diff) compression patches to reduce the transfer size during firmware updating. However, the existing mechanism has significant defects, particularly when code addition or deletion on the image occurs.

Figure 1 below illustrates an example of code addition. As shown, there is an old image with 100KB. The new image for updating is 110KB, with the image increment of 10KB shown in red. The image is divided into blocks each with a fixed size of 20KB. Using the existing method, the final image diff is shown in blue. Here, the image patch size is up to 90 KB. Compared to the origin size of 110 KB, the optimization effect is not obvious.

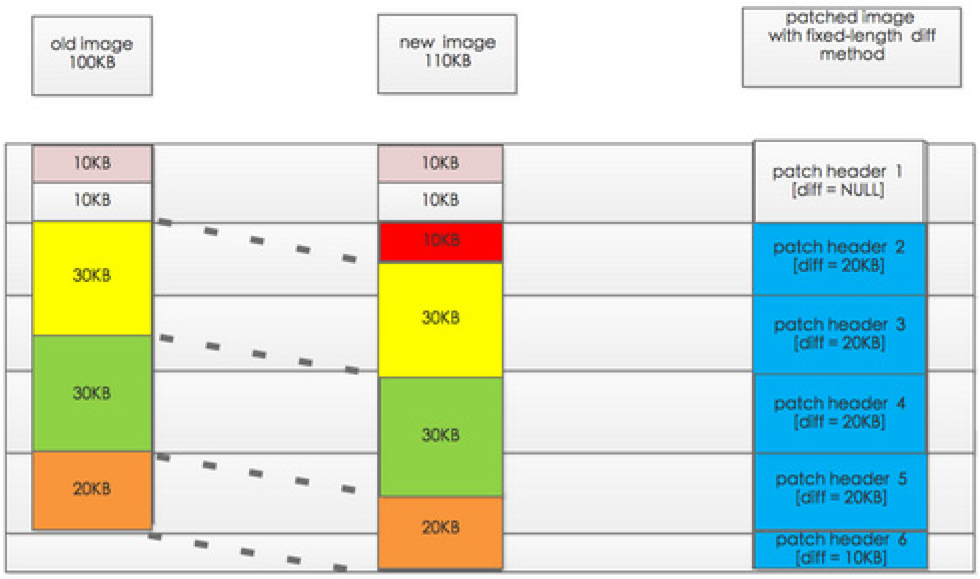
old image 100KB
new image 110KB
patched image with fixed-length diff method
10KB
10KB
30KB
30KB
20KB
10KB
10KB
10KB
30KB
30KB
20KB
patch header 1 [diff = NULL]
patch header 2 [diff = 20KB]
patch header 3 [diff = 20KB]
patch header 4 [diff = 20KB]
patch header 5 [diff = 20KB]
patch header 6 [diff = 10KB]

Figure 1: Fixed-length split mechanism

Techniques described herein provide a better mechanism for generating the transferred image patch size in order to save mesh network resources.

To resolve the current defects in firmware updating, a flexible mechanism is provided to compress the transferred image size Over-The-Air (OTA) in a resource restricted network. An example firmware updating process is provided in four steps.

The first step involves substring creation and extraction. In a mesh network, the NMS controls the firmware updating process. Before updating new firmware on nodes, the NMS first extracts the substring list between the updating image and the existing old image on flash. The substring list may be extracted by adopting existing algorithms such as a suffix array. In one example, it applies the binary diff program Bsdiff. Its enhanced algorithms may be developed to minimize the additional memory usage including an optimal time and memory algorithm. Second, based on the extracted substring list, the updating image is divided into serval blocks for subsequent processing.

Two cases are considered. The first case involves continuous and small split blocks (e.g., serval bytes on each block). This may lead to production of more patch headers such that the transferred bytes will increase. The *Jaccard* similarity coefficient algorithm may be used on the continuous and small split blocks to produce one suitable size block. The size may be decided by resources on the different platform.

The second case involves a bigger split block. In one example, the size of each split block is more than the maximum Random Access Memory (RAM) on the nodes, and therefore the nodes cannot handle it. Considering the RAM limitations on the mesh node, the maximum transferred size may be adjusted on different platforms. When the split block is greater than the maximum transferred size, the existing split block may be divided again with the maximum transferred size.

Figure 2 below illustrates an old image with substring list [A, B, ..., E] and new image with substring list [A, B, ..., E, F]. The common substring list is [A, B, ..., E]. Based on the common substring list, the image may be divided into serval parts.

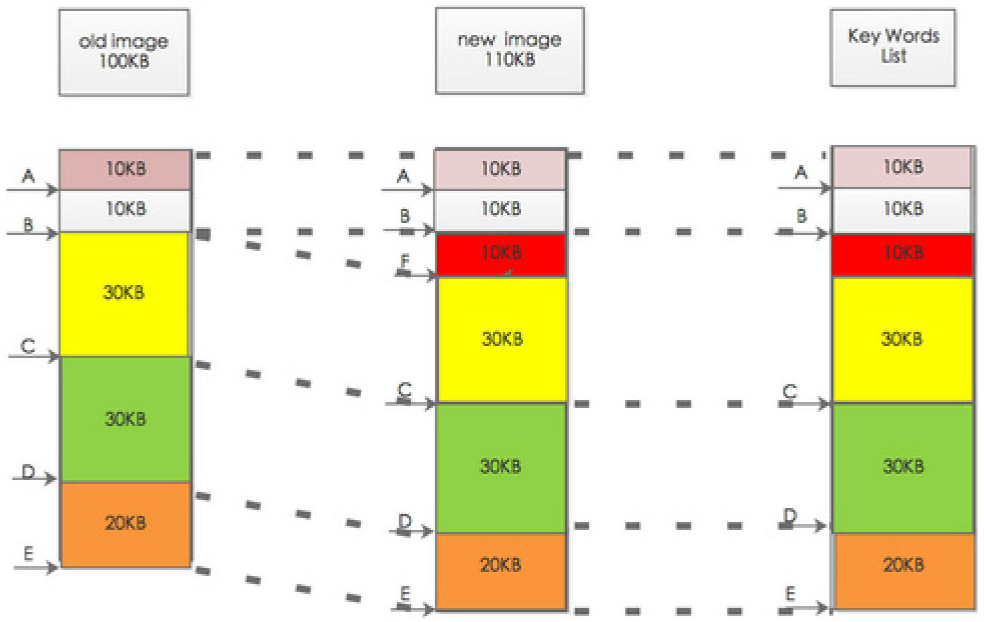

Figure 2: Substring List Creation and Extraction

The second step involves a flexible and variable-length patch mechanism. Based on the aforementioned common key words list, the updating image is divided into several blocks. The diff of each block between the updating image and the existing old image may be obtained, and the variable-length patches generated. Each patched block contains the patch header and diff contexts so that the endpoint can rebuild the upgrading image based on the patch. The patch header includes the old block address, old block size, new block address offset, and new block size. The diff context may be the differences of each block between the old and new images.

Figure 3 below illustrates example code addition on the image. As shown, there is an old image of 100KB. The updating image is 110KB with a code insertion size of 10KB as shown in red. The substring list is [A, B, C, D, E] between the old image and the updating image. The patched image may thereby be obtained. Here, only 10KB of diff context (blue) is generated. Comparing the entire image transferring, the OTA image size can be reduced by 91%. Compared to prior techniques with a patch size of 90KB, the OTA image size can be reduced by 88%.

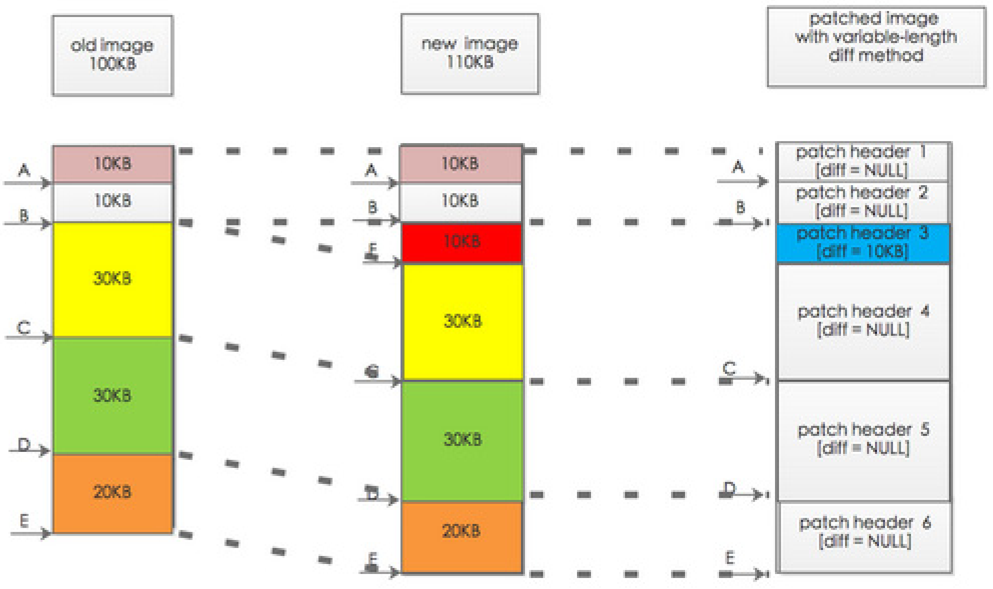
old image 100KB
new image 110KB
patched image with variable-length diff method
A
B
C
D
E
10KB
10KB
30KB
30KB
20KB
10KB
10KB
10KB
30KB
30KB
20KB
patch header 1 [diff = NULL]
patch header 2 [diff = NULL]
patch header 3 [diff = 10KB]
patch header 4 [diff = NULL]
patch header 5 [diff = NULL]
patch header 6 [diff = NULL]

Figure 3: Variable-length split mechanism

Comparing with previous solutions, the techniques presented herein enable smaller-sized patch files to be generated when there is code addition/deletion on an image.

In the third step, the NMS combines all diffs into one patch file, and transfers the patch file to the mesh nodes. But considering the RAM limitations on the mesh nodes, the maximum transferred size may be adjusted.

In the fourth step, once the nodes receive the joint patch file, the mesh nodes decompress the file into pieces, and then patch each to an old image. Finally the nodes generate the new image.

In case of a first time firmware upgrading on the mesh node, there may be no image on the flash slot which needs to be updated. As such, the running or backup image slot may be configured as the old image.

In summary, techniques are described herein for adopting a substring list creation and extraction mechanism as well as a variable-length patch mechanism to generate an OTA image path file having a small size. This saves network resources and reduces side effects on data transmission / network service when executing image / firmware upgrading. As a result, OTA updating can be sped up, and the peak network traffic during the firmware updating may be narrowed.