August 28, 2019

A NOVEL TECHNIQUE TO PERFORM LARGE FILE UPDATES ON INTERNET OF THINGS (IOT) DEVICES USING BLOCK CHAIN IN LOW-POWER AND LOSSY NETWORKS (LLNS)

Lele Zhang

Chuanwei Li

Yinfang Wang

Li Zhao

A NOVEL TECHNIQUE TO PERFORM LARGE FILE UPDATES ON INTERNET OF THINGS (IOT) DEVICES USING BLOCK CHAIN IN LOW-POWER AND LOSSY NETWORKS (LLNS)

AUTHORS:
Lele Zhang
Chuanwei Li
Yinfang Wang
Li Zhao

ABSTRACT

Proposed herein is an efficient large file update (e.g., a firmware update) technique that utilizes blockchain technology, rather than a classic centralized solution. This technique adopts a decentralized and distributed architecture to disseminate information, which may greatly reduce bandwidth occupation and accelerate the updating process. Various benefits may also be realized by this technique including, but not limited to: 1) making sure that each piece of a block is correct through a blockchain hash (e.g., a node could be aware of the legality of a file download before the whole file download is completed); 2) as opposed to a Multicast Protocol for Low-Power and Lossy Network (MPL) solution, all nodes do not need to conserve as many packets as possible in Random Access Memory (RAM), rather a block may be saved directly into non-volatile flash, which could mitigate the usage pressure for temporary buffers; and/or 3) as opposed to a centralized solution, this technique does not need to exchange information with a Network Management System (NMS), rather a node may pull missing blocks from its neighbors as long as they have them and, after two rounds of failures, it can turn to the NMS, which may not result in concurrent traffic to the NMS.

DETAILED DESCRIPTION

There are developments involving field area network (FAN) solutions for smart utility applications, such as advanced metering infrastructure (AMI) and Distributed Automation (DA). In some cases, terminal nodes may need to download files with a large size from a Network Management System (NMS), such as firmware and/or configuration updates. For a constrained wireless mesh network (WMN), a large size file may need to

be divided into many small blocks and then distributed. In general, broadcast/multicast propagation is a preferred method for updating a large file for all nodes in a WMN. It is through such propagation that a problem may be identified such as, for example, in large scale WMNs, packet loss is unavoidable and considerable among nodes while broadcast propagations are being performed. Because a large file is often divided into many small pieces to spread, many WMN nodes may fatally miss some packets with high probability, which means they could receive an incomplete file. Figure 1, shown below, illustrates a typical broadcast download environment.

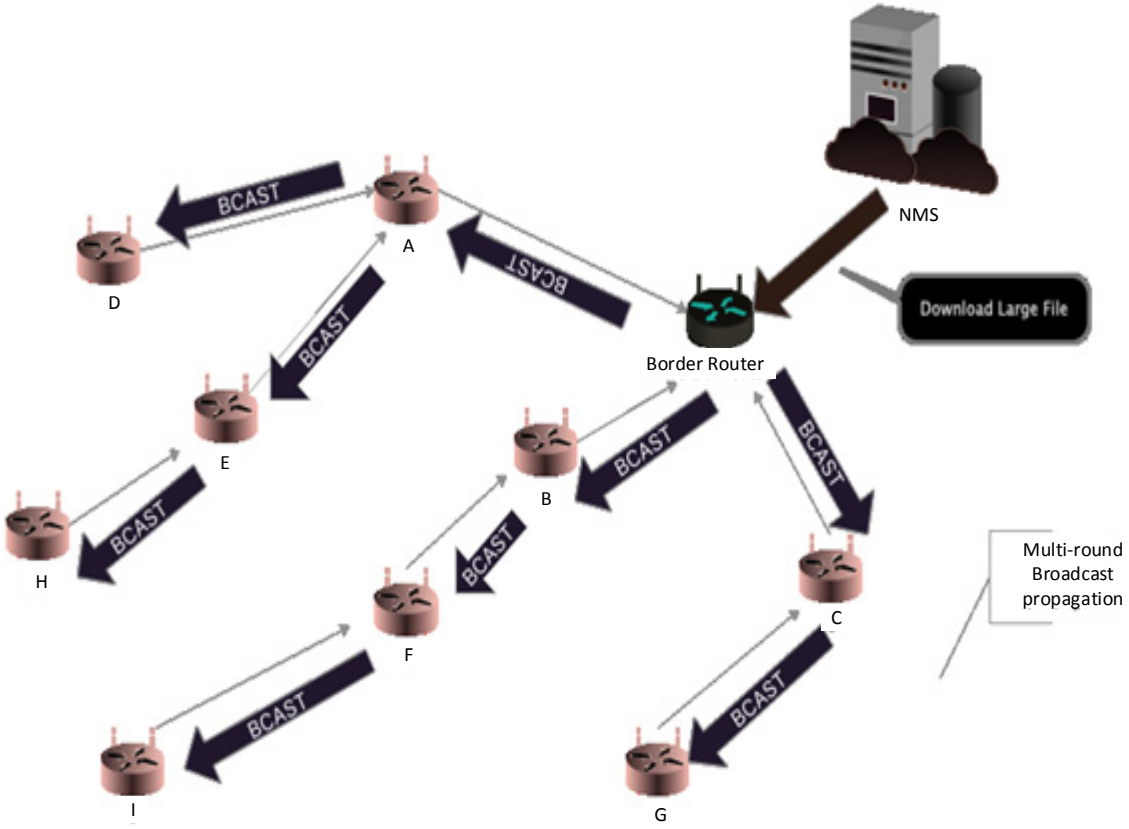

Figure 1 – Broadcast Download

In order to ensure all nodes can receive the integrated file, the source (e.g., NMS) may allow several rounds of packets pushing and, thus, a large period of time may be involved for propagating in the whole WMN (e.g., several days or weeks for firmware upgrading in field, depending on the quantity of nodes, etc.). Further, there may be some nodes that might not receive the file completely and, therefore, may need to pull re-transmissions from the NMS via a unicast method, as shown in Figure 2, below, which can further increase throughput and duration.

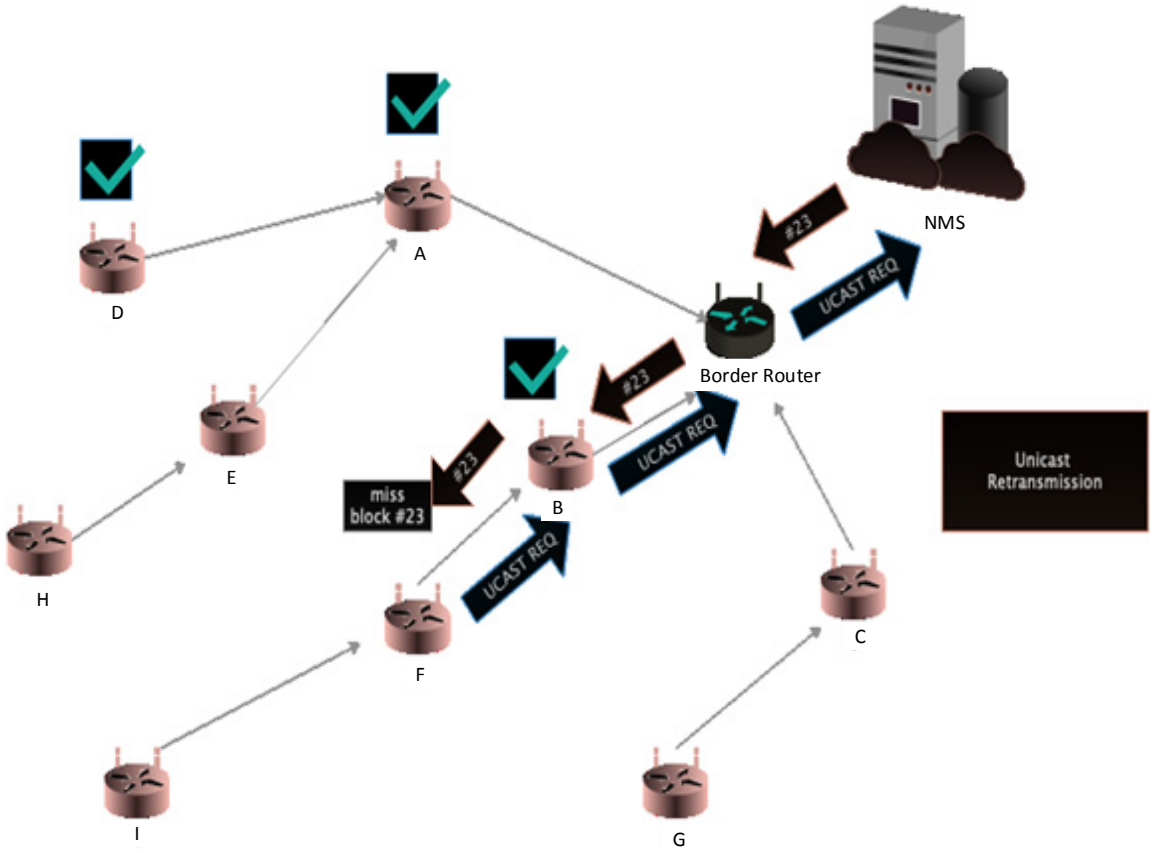

Figure 2 – Unicast Re-Transmission Progress

Given the above issues, proposed herein is a novel technique to speed-up large file updates in an LLN using blockchain technology. This technique is not intended to utilize a whole blockchain progress to facilitate a WMN large file distribution (e.g., firmware update). Rather, this technique involves a node preferring to pull missing blocks from neighbors rather than from central equipment such as the NMS, as illustrated in Figure 3, below. This technique does not derive missing blocks through MPL and it does not need to exchange information from NMS in advance.

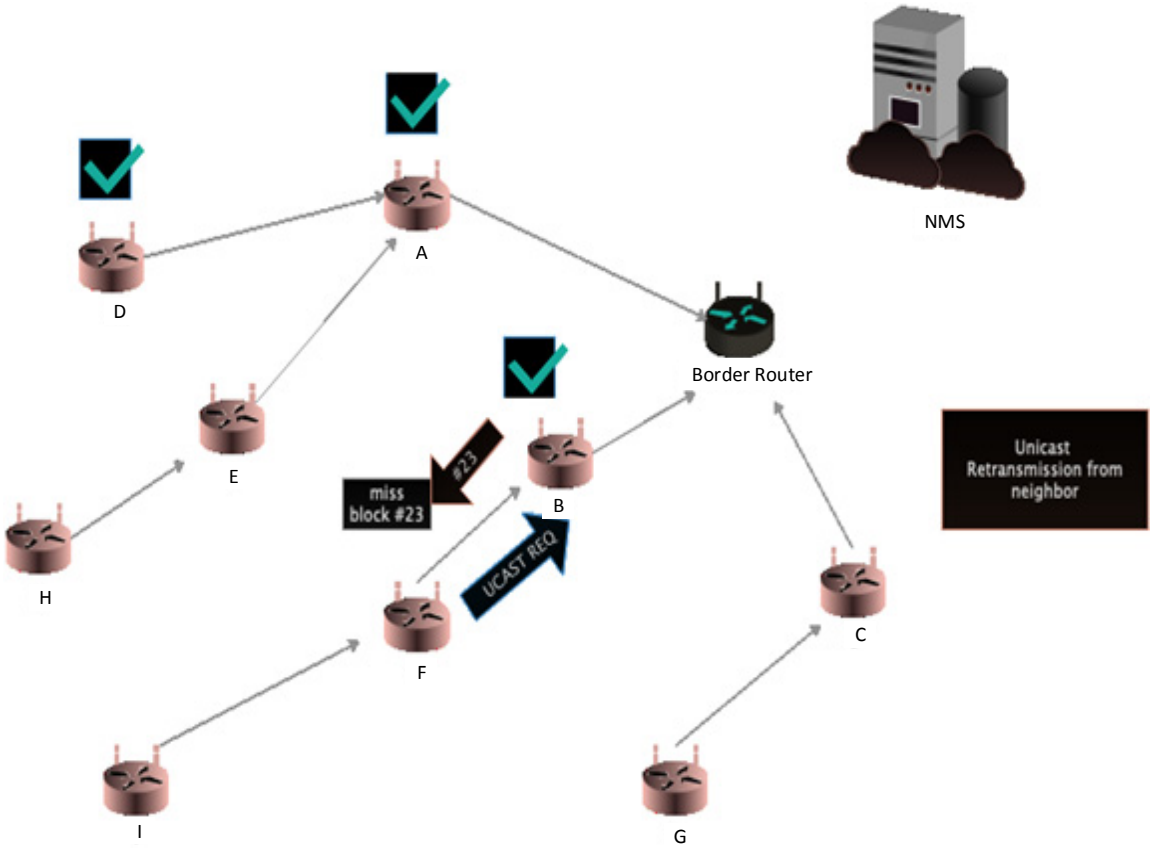

Figure 3 – Unicast Re-Transmission from a Neighbor

In at least one implementation, operations for realizing the technique of this proposal may include the following steps:

Step 1: A large file (e.g., firmware update) can be split into small blocks by the NMS (e.g., 650 bytes per block in the firmware update case) and the NMS can generate blockchain information for each block. An example block structure is detailed in Table 1, below.

TABLE 1

Attributes Name	Description	Length (Bytes)
fileHash	This denotes this block belongs to which file	32
hashPrevBlock	The previous block hash value	32
hashCurrentBlock	The current block hash value	32
hashNextBlock	The next block hash value	32
blockNum	The sequence number of the block	4
blockData	The content of the block	650

Step 2: The NMS performs sending all blocks for the file in a first round in which each node will record all received blocks into a bitmap to count/identify which blocks may be missed.

Step 3: Since a node knows which blocks it has missed in accordance with the bitmap, the node can send a request to its preferred parent with relevant elements, such as {fileHash, hashRequiredBlock, hashExistingBlock, blockNum} in order to obtain the missing block. For example, if a node, say Node A, as shown in Figure 4, below, lacks block #23 but it has block #22, then it can send a request to Node B with the following elements:

1. fileHash – The fileHash element can be used to identify which file a requesting node desires (e.g., if node A wants block #23 in file A, the fileHash element can be used to ensure that node B does not send block #23 of file B to node A).
2. hashRequiredBlock – The hashRequiredBlock element can be used to identify a requested block's hash value (e.g., block #23's hash value).
3. hashExistingBlock – The hashExistingBlock element can be used by a requesting node to provide an adjacent block's hash value for which the requesting node has received (e.g., if node A desires to obtain block #23, it can provide a block #22 or #24 hash value using the hashExistingBlock element for node B to check. Once node B has verified the hash value successfully, it will response with block #23 if it has the block).
4. blockNum – In order to find a requested block quickly, a requesting node is to provide the block number (blockNum) for a requested block (e.g., in the example involving Node A, the block number would be 23).

For the present example, once node A has received block #23 from node B, it can check the block data to determine whether the hash value for the block is correct. The received block may be acceptable as long as the calculated hash is equal to the value in the blockchain table.

Step 4: The requesting node may ask for missing block(s) from its parents (up to three) in advance and, if the request fails, it can consider other candidates in order to obtain

missing block(s). If a responder does not have a requested block, the request will fail regardless of whether the responder misses the same block or does not have the file.

Step 5: If the requesting node fails with all of its neighbors, it can set a back-off window and then perform another round of requests in case one of the responders obtains the block during this period.

Step 6: If the second round still results in failures from all of requesting node's neighbors, the requesting node can send a request to the NMS directly.

Figure 4, below, illustrates features of the technique provided by this proposal.

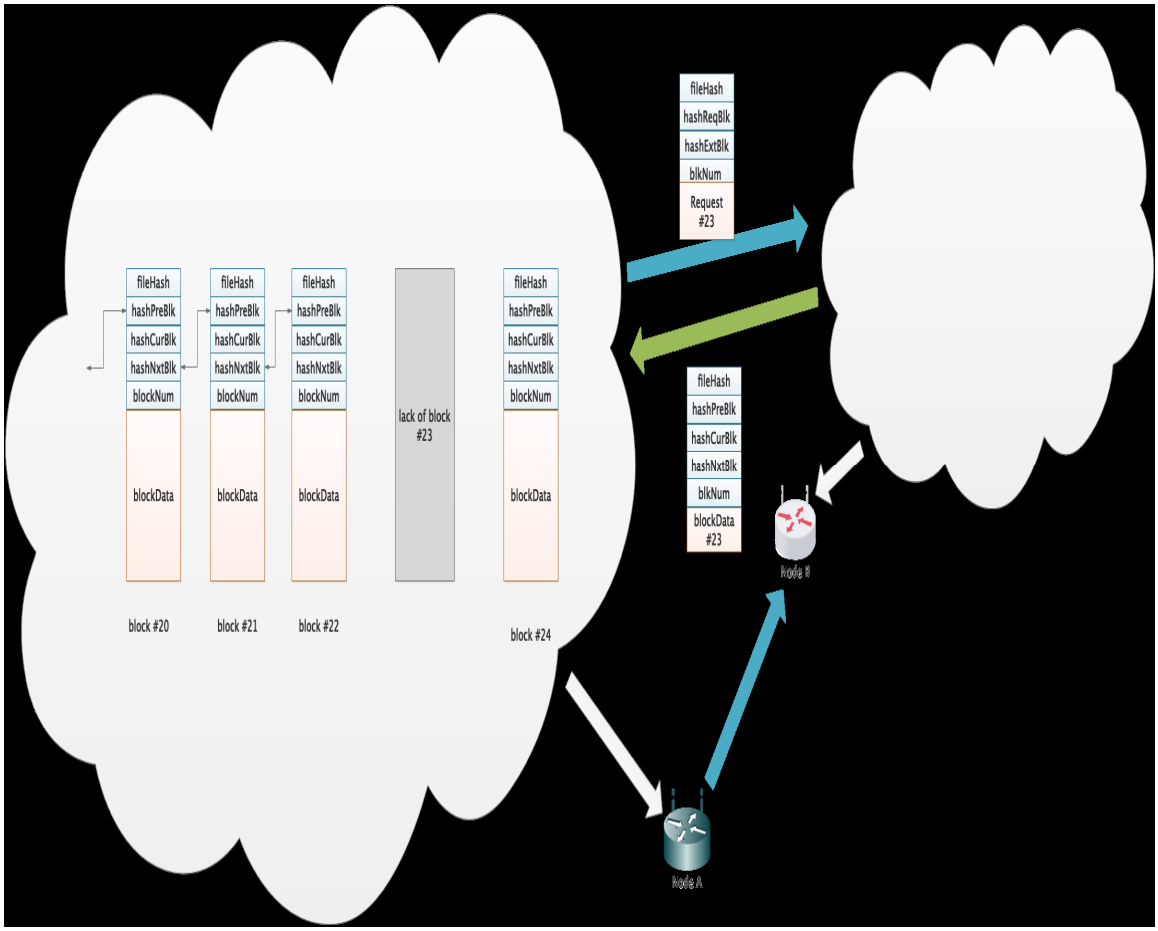

Figure 4

In summary, proposed herein is an efficient large file update (e.g., a firmware update) technique that utilizes blockchain technology, rather than a classic centralized solution. This technique adopts a decentralized and distributed architecture to disseminate information, which may greatly reduce bandwidth occupation and accelerate the updating process. Various benefits may also be realized by this technique including, but not limited

to: 1) making sure that each piece of a block is correct through a blockchain hash (e.g., a node could be aware of the legality of a file download before the whole file download is completed); 2) as opposed to an MPL solution, all nodes do not need to conserve as many packets as possible in RAM, rather a block may be saved directly into non-volatile flash, which could mitigate the usage pressure for temporary buffers; and/or 3) as opposed to a centralized solution, this technique does not need to exchange information with a NMS, rather a node may pull missing blocks from its neighbors as long as they have them and, after two rounds of failures, it can turn to the NMS, which may not result in concurrent traffic to NMS.