September 13, 2019

SECURE INTERNET OF THINGS (IOT) FIRMWARE UPGRADES IN A WIRELESS NETWORK

Vinay Saini

Jerome Henry

Sowbhagya H. S

Robert Barton

SECURE INTERNET OF THINGS (IOT) FIRMWARE UPGRADES IN A WIRELESS NETWORK

AUTHORS:

Vinay Saini
Jerome Henry
Sowbhagya H S
Robert Barton

ABSTRACT

Internet of Things (IoT) devices/sensors/endpoints (collectively and generally IoT devices) are becoming part and parcel of many establishments. With the inbuilt support for Wi-Fi 6, also known as “AX Wi-Fi" or "802.11ax Wi-Fi,” in many IoT devices, the number of deployed IoT devices will continue grow. Wi-Fi® is trademark of the Wi-Fi Alliance. These IoT devices come from various manufacturers and will require network/Internet access for firmware upgrades. As such, there is a need to create a secure and tested firmware upgrade environment for such devices. Presented herein are techniques to track the firmware of IoT devices and use Manufacturer Usage Descriptions (MUD) to recognize the recommended firmware and settings. Further, the techniques presented herein include a mechanism to optimize firmware upgrades using Target Wake Time (TWT) scheduling.

DETAILED DESCRIPTION

It is expected that, within the next two years, more than twenty (20) billion IoT devices will be network connected. In particularly, it is expected that more and more enterprises and industries will onboard IoT devices, such as cameras, Smart Lights, sensors, *etc*., and that most of these devices will connect over various wireless networks. With Wi-Fi 6, large amounts of devices can connect and be spread across a large geographic area (large cells and multiple cells).

Additionally, with Uplink Orthogonal Division Multiple Access (UL OFDMA) capabilities in Wi-Fi 6, access points are able to sustain a much greater number of active associations, relative to previous versions of Wi-Fi (e.g., up to 4K endpoints may be associated to a single 11ax access point, and the protocol was designed to allow for that many active clients). The IoT devices can be deployed across large areas (like smart cities) and in networks with reduced or no direct connectivity to the Internet.

Ensuring that all deployed IoT devices run the most update to date firmware is mandatory to avoid, for example, potential security attacks. However, such update logic comes with various challenges, such as:

1. How to identify the availability of firmware;

2. How to distribute upgrades in a safe and secure manner to a large number of endpoints; and

3. How to validate that the new firmware is valid (authentic) and is not causing issues.

In a pull model (where IoT devices contact an update server at regular intervals), the attack surface is wide (e.g., between the discovery of a new exploit and the next update-pull cycle, a large number of clients may be exposed). In a push model (where a server pushes the update to the individual clients), Man-in-the Middle (MItM) attacks are possible and, as such, fake firmware may be distributed to clients. In both the push and pull models, the requirement to 'keep the pipe open' between all clients and all possible update servers makes large scale management impossible.

Presented herein are techniques to address the above challenges by combining novel MUD and Wi-FI 6 capabilities in complementary ways. MUD is well-known, but has conventionally been intended to carry policy information. In contrast, the techniques presented herein extend MUD intent by allowing it to carry implementation and state information (e.g., about the device, the firmware it runs, device needs, the tolerance to delay, the change of behavior after upgrade, *etc*.).

The concepts of IoT firmware updates and the updating of large quantity of devices have been explored. However, Wi-Fi 6 devices present specific constraints and optimization opportunities that have not been explored yet, and that are specifically addressed by the techniques presented herein.

Background:

The updating firmware on IoT devices requires the following:

- Identifying if there is an update for the IoT device and assessing its criticality (e.g,. is it a security patch or just a feature update).
- Accessing the verified copy of the firmware update.
- Distributing the firmware in an efficient manner and quickly ensuring devices are awake and ready to process the image when they are ready to go.

In the world of IoT, there are two main models to fulfil the above requirements:

- In a pull model, the IoT device attempts to connect to update servers at regular intervals. This model is problematic for emergency or security updates (as the device may not attempt the update until its next update pull cycle).
- In a push model, the management platform can initiate an update at any time when the IoT device is awake and connected. A limitation of this model, for emergency updates, is that the vendor may need to update 'now' 'all' devices on the planet, which presents obvious scalability issues (and it is difficult to explain to a customer that their IoT device was compromised because it was lower on the list of devices to update urgently, as other customers were 'more important').

Both of the push and pull models also suffer from the device update orchestration problem. The server can know which devices have been updated, but does not know when non-updated devices are going to update next (in a pull model), or why they haven't been updated yet (in push/pull models where IoT devices are unreachable by the server). Similarly, local OT management systems may know which devices have received an update, but network elements do not have that knowledge, thus potentially allowing unsafe devices to continue operating without the required update. For example, in 2017, five Nissan Renault factories were crippled due to the Wannacry ransomware attack which froze HMI workstations on their factory floor. These were HMI stations running older, unpatched versions of Windows in their industrial automation system.

The techniques presented herein addresses this issue, especially in the context of 802.11ax environments, as an illustration of the method in wireless environments where data exchange can be scheduled.

Acquiring Information about Device Firmware Update:

In the examples presented herein, each object is associated to an object group that the group documents the object type. Additionally, every IoT device adhering to the MUD architecture must send a URI after every reboot and ask for a new MUD file. This is proxied via the MUD controller. This URI can be cached for a group of IoT devices (Grouping of IoT devices based on MUD is prior art). The techniques presented propose an enhancement to the MUD exchange with several additional elements which take into account 11ax devices.

First, the MUD URI is augmented to include the current firmware version. This is an extension to RFC 8520, but is different from the current art. RFC 8520 allows the URI to contain in the identifier of the MUD file a path that can indicate the firmware version requested by the IoT device (e.g., myvendor.com/object/v43/file or myvendor.com/object/filev43). As the MUD URI is hard coded in the device, this provision forces the vendor to install, in that location, a new version of the firmware (e.g. v44) when updates are needed, which creates complexity. In the techniques presented herein, the version of the firmware is appended to the MUD URI, separated by a well-known special character (e.g., myvendor.com/object/file**<br>**v43). This information is stripped by the MUD controller and retained for future use.

Other embodiments may provide the firmware version in a different structure, such as: an 802.11 IE at association time or in a null frame; a second frame following the MUD URI, *etc*. In all cases, the firmware version is obtained along with the MUD file, but does not reflect the version that the IoT device is attempting to obtain, but rather the version that the IoT device is currently running.

Next, the MUD file returned by the vendor includes several new components. First, the MUD file returned by the vendor includes the current recommended firmware version, size, release date and criticality level. In the current art, the MUD file defines access rules, but not firmware versions. The techniques presented herein use MUD in a new way to carry this information and to indicate the manufacturer's recommenced firmware (and therefore configuration) state. As a result, the MUD controller has a record of the discrepancy. As such, MUD is no longer a policy tool, but instead becomes a tool to exchange state information (state of the device, mentioning its current firmware version in the request, expected state returned by the vendor).

Second, the MUD file returned by the vendor includes the location of the server from where the firmware update can be obtained. Third, the MUD file returned by the vendor includes Wi-Fi6 delay tolerance and RU information, as per the below.

The change of device behavior expected after the update is now addressed. In the context of the examples presented herein, the behavior describes the TWT and RU profile that will be used for the IoT device. It is noted that, in 802.11ax, IoT devices can conserve battery and only wake up at Target Wake Times. The IoT device can request a TWT interval from the AP. The AP also implements a scheduler and can override the IoT device requested TWT interval. It follows that an IoT device may send several types of traffic. As such, the IoT device may request TWTs matching the traffic type and configured communication interval. Depending on the traffic, delaying TWT (due to AP override) may be tolerable only to a certain delay value.

The current art related to MUD describes access rules that the network elements can use to determine the IoT device's requirements. The techniques presented herein extend MUD to also insert an override tolerance element that includes the expected traffic volume at each TWT, but also the IoT device tolerance to TWT delay or traffic starvation. Such tolerance depends on the vendor application and the vendor chipset. This new dimension informs the network as to how much the network can deviate from the elements provided by the vendor. The combination of firmware size and delay tolerance is used in two ways:

1. The AP can now allocate the proper RU structure for the IoT device firmware update, having knowledge both of the update size and the IoT device tolerance to additional transmission delay for each next packet.
2. The AP can now build an upstream OFDMA scheduler that not only accounts for the expected traffic volume for each client, but also the individual tolerance to override and delay (thus preventing layer 7 failures for excessive delays).

As detailed above, the techniques presented herein focus on 802.11ax, where the need for such a solution is clearly emerging. However, the same method could be applied to other scheduled wireless technologies.

In a pull model, for each known MUD URI, the MUD controller periodically triggers the MUD file request on behalf of the IoT Device Group and looks for any recommended firmware updates. In a push model, the MUD controller registers as a virtual IoT device in order to receive the updates. This is shown below in FIG. 1.

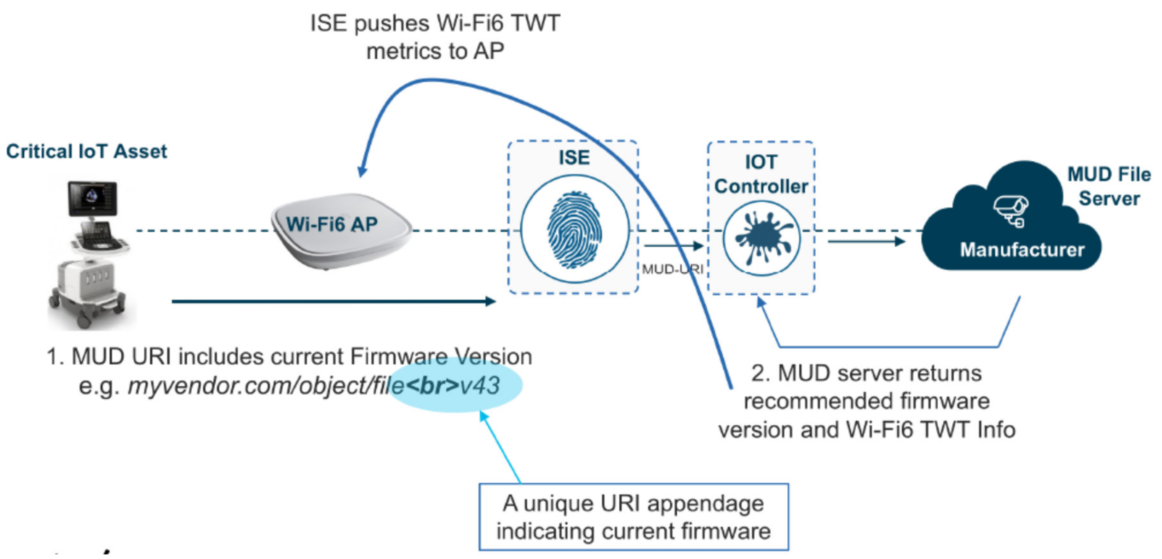
ISE pushes Wi-Fi6 TWT metrics to AP
Critical IoT Asset
Wi-Fi6 AP
ISE
IOT Controller
MUD-URI
MUD File Server
Manufacturer
1. MUD URI includes current Firmware Version e.g. myvendor.com/object/file
v43
2. MUD server returns recommended firmware version and Wi-Fi6 TWT Info
A unique URI appendage indicating current firmware

FIG. 1

Distributing the Firmware:

Certain network management system can maintain a list of the identified IoT devices, their current firmware version, connection state, Target Wake up Time and currently associated AP. In one embodiment, a network management system publishes to the AP (or WLC) a list of IoT clients with their firmware status (in a simplified format, this could be limited to 'current vs update due').

As the MUD controller signals a firmware update for a group of objects, the network management system downloads a copy of the firmware on behalf of the device and keeps a local copy. This mechanism avoids any issues arising due to non-availability of the Internet or possible congestion when the IoT devices (or tens of thousands of them) wake up at the same time. However, this mechanism is optional.

During the next transmission from such a client for which firmware is available, the AP will modify the IoT device's TWT profile to remain awake and receive the firmware (forcing the device to stay awake). When more than one IoT device awakens at the same time, the AP uses the MUD TWT override tolerance value to distribute the IoT devices across the next wake time and limit the count of concurrent requests to the update server. The implementation of this method becomes a combinatorics problem, where *m* objects of *n* different types need to download firmware files of individual size *k* over bursts separated by at most *i* milliseconds (/seconds, /minutes etc.). Each firmware may have a QoS value and be integrated in the standard QoS distribution model. The distribution scheme can use known mechanisms such as permutation routing on RN(p,k) (AKA Datta-Zomaya distribution). However, the application of combinatoric distribution (such as the Datta-Zomaya) in the context of IoT firmware distribution is new, as the original intent of such distribution was energy efficiency on a shared link for the transmitter, not a time efficiency for the receivers in the time domain. It should be noted that other, possible more rudimentary schemes, are possible (e.g., implementing a Lovasz-Schrijver relaxation of the scheduler around the median of the TWT tolerance).

In all cases, the contribution of the techniques presented herein is to organize the devices in groups, and organize a TWT rotation cycle, where groups TWTs are progressively expanded over time until the update cycle starts, so that the start of the next TWT group (at update time) corresponds to (after) the time at which the update push to the previous group has completed. Then, during each group update period, the AP allocates the RUs to each device that match the delay tolerance maximum. It is possible that the delay tolerance would allow for inter-TWT bleed, where a first group would receive a first update burst, go to sleep while a second group receives the first update burst, then the first group receives the second burst (firmware update continuation) while the second group sleeps.

In another embodiment, the AP also restricts the communication of devices with 'update due' status to the update server. This mode can be adapted depending on the update criticality level.

The MUD controller also distributes the TWT tolerance value to the WLC, where this information is used, in combination with the number of IoT devices of each type within each cell, to build a new longer term TWT schedule that is optimized for the overall traffic managed by the AP while respecting the TWT constraint tolerance. This is shown below in FIG. 2.

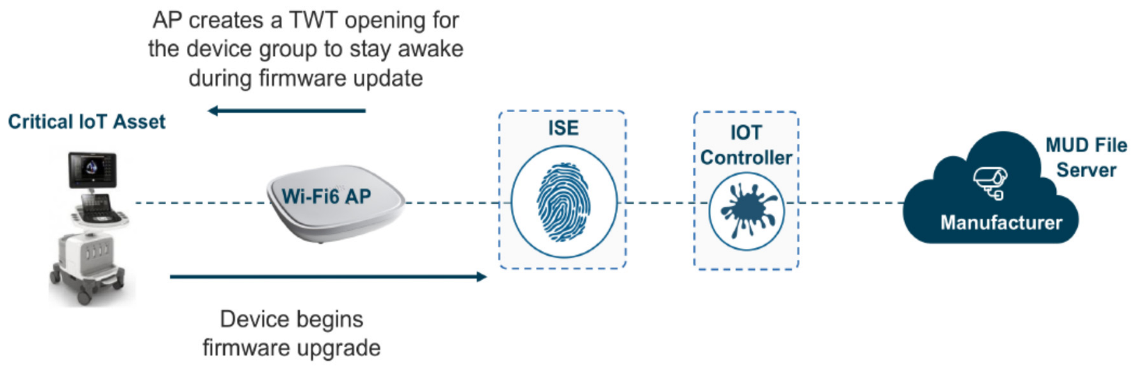
AP creates a TWT opening for the device group to stay awake during firmware update
Critical IoT Asset
Wi-Fi6 AP
ISE
IOT Controller
MUD File Server
Manufacturer
Device begins firmware upgrade

FIG. 2

Firmware Issue Prediction:

A typical device will reboot after the firmware upgrade and re-connect. The AP implements the schedule as above. In parallel, the AP forwards, to the network management system, the TWT requests made by the IoT client. The network management system compares the TWT schedule before and after the firmware upgrade and monitors the device for its behavior and data upload patterns. This will be compared against the MUD file content which also defines any new or updated behavior.

If the device is not sending any data or sending spurious data which is not compliant to MUD file description, then the firmware upgrade to other devices can be stopped and an admin can be notified (if deviation follows the upgrade). A device can also be isolated in a specific VLAN, if the device starts displaying TWT behaviors outside of the MUD description.