PROJECT TITLE

**“FIRMWARE UPDATE OF IoT DEVICES VIA OTA”**

*SUBMITTED BY*

Mr. Piyush Nindane, Mr. Bhagwat Suryawanshi and Ms. Shubhada Deshmukh

*UNDER THE GUIDANCE OF*

Prof. Nabegha Masood

(Academic year: 2020-21)

**CONTENT**

**1) Introduction………………………………………….1**

**2) Project Idea Mapping……………………………….8**

**3) Network Architecture……………………………….9**

**4) File Transfer………………………………………...11**

**5) Layered Architecture………………………………13**

**6) Resource Estimates………………………………...14**

**1) Introduction**

A firmware is set of files (program) which is written in the ROM of any physical device, these files define the functionality of hardware present in the device. Any updates or rectification in the existing hardware can be done by upgrading the present firmware of the device by an updated one. The authenticated files are transferred using different modes of transportation over an update manager server, by the OEM (Original Equipment Manufacturer). These files are then downloaded by the host device using various methods. In our project we will be using various set of protocols and mechanism like TCP, SFTP, MQTT, REST. The project involves Host, Server, Gateway, and MQTT broker configuration and setup.

**1.1 OTA Firmware Updates**

* Over-the-air (OTA) Firmware updates are a vital component of any IoT system. Over-the-air firmware updates refers to the practice of remotely updating the code on an embedded device.
* Over-the-air (OTA) firmware updates facilitates us with the following elements:

1. The ability to add new firmware features to a product after a device has been deployed in the field to improve functionality over time.
2. The opportunity to rapidly respond to bugs and security vulnerabilities without the need for physical recalls of devices or truck rolls.
3. Ensuring embedded developers can quickly prototype and seamlessly roll out new versions of device firmware, speeding up innovation cycles.

* A successful OTA update requires complex coordination between IoT hardware, device firmware, network connectivity, and an IoT device cloud. Over-the-air (OTA) updates offer many benefits for Internet of Things (IoT) devices as it enables remote patching of bugs or security flaws.

**1.2 Benefits of OTA**

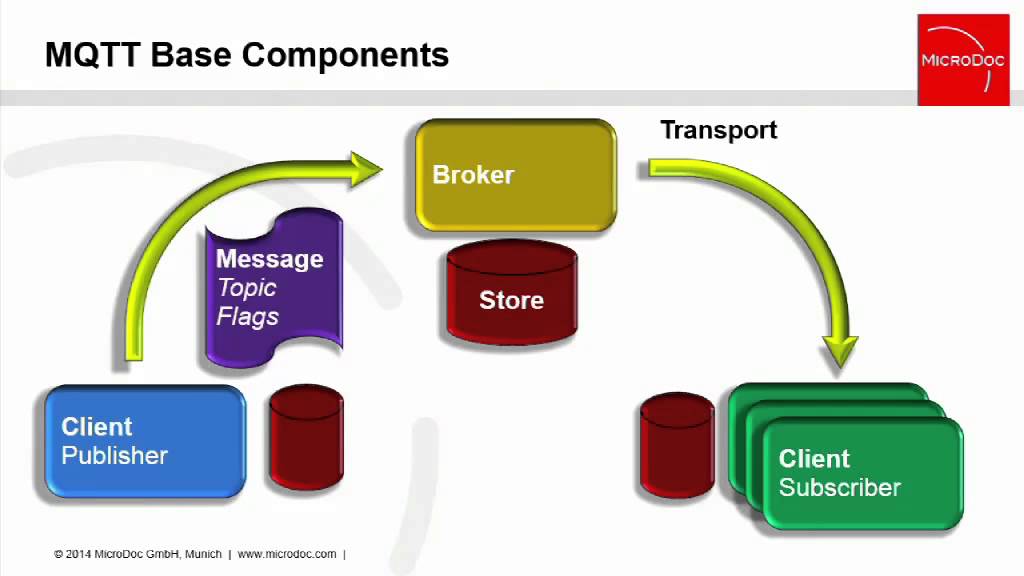
* It allows continuous improvement of devices even after they are deployed.
* It increases functionality through updates to one or more devices.
* It saves cost as you manage the firmware updates from a remote platform.
* OTA updates allow to quickly correct security vulnerabilities.
* It increases scalability by adding new features and infrastructure to the products after their launch.

**1.3 Document Overview**

* This document describes the software design of “Firmware update at the end node using mqtt protocol”.
* It also briefs about the hardware design of “Firmware update at the end node using mqtt protocol”.
* This document is intended for primary use by the developers and the Quality assurance team.

**Technology Used:**

**1. MQTT (Message queuing telemetry transport)**



Publish/Subscribe

MQTT PROTOCOL

Fig. 1

1. MQTT (Message Queuing Telemetry Transport) protocol is protocol specifically designed for “machine to machine” communication.
2. MQTT protocol runs over TCP/IP and has a data packet size with low overhead minimum (> 2 bytes) so that consumption of the power supply is also small enough.
3. This protocol is a data-agnostic protocol that can transmit data in various forms such as binary data, text, XML, or JSON and this protocol uses a publish/subscribe model rather than client-server model.
4. Stack TCP/IP is also widely supported by Raspberry Pi and many other common device boards. There are so many options for implementing MQTT protocol on devices.

5) A common system of MQTT requires two main software components:

* MQTT Client is to be installed on device. Client PAHO library of Eclipse can be used.
* MQTT Broker serves to handle publish and subscribe data. A Linux platform can use broker that is available free such as mosquitto, HiveMQ etc.

The advantage of publish/subscribe system is that the data sender (publisher) and the data receiver (client) do not know each other because there is a broker between both. In addition, there is time decoupling which makes publisher and client unable to be connected simultaneously so that client will stay to receive delayed data previously.

**1.1 Control Signal**

MQTT has 14 types of control signal, namely:

* CONNECT - Client request to connect to Server
* CONNACK - Connection Acknowledgement
* PUBLISH - A message which represents a new/separate publish
* PUBACK - QoS 1 Response to a PUBLISH message
* PUBREC - First part of QoS 2 message flow
* PUBREL - Second part of QoS 2 message flow
* PUBCOMP - Last part of the QoS 2 message flow
* SUBSCRIBE - A message used by clients to subscribe to specific topics.
* SUBACK - Acknowledgement of a SUBSCRIBE message
* UNSUBSCRIBE – A message used by clients to unsubscribe from specific topics.
* UNSUBACK - Acknowledgement of an UNSUBSCRIBE message
* PINGREQ - Heartbeat message
* PINGRESP - Heartbeat message acknowledgement
* DISCONNECT – Graceful disconnect message sent by clients before disconnecting.
* From those signals, there are only four main signals which are used directly by the client, namely PUBLISH, SUBSCRIBE, UNSUBSCRIBE, CONNECT. Other signals are part of the publish/subscribe mechanism.

**1.2 Topic & Quality of Service (QoS)**

In MQTT, it is as known topic that serves as filter for broker in sending message to every client connected and subscribed. MQTT provides quality level of service, which is called QoS. This level guarantees the reliability of message delivery. Level 0 message is sent only once. Messages are sent depending on the existence of network, and there is no attempt to transmit a message back. Level 1 messages are sent at least once so that if the subscriber does not recognize (acknowledge) the message, then the broker will send a message to publisher to receive the message acknowledgement status from the client. Level 2 is to make sure that message was received. With this level, it can be ensured that the message is certainly conveyed and avoided from duplication of messages sent.

**2. Network Address Translation (NAT)**

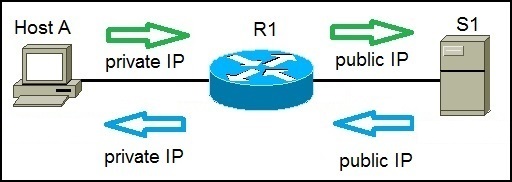


Fig.2

**NAT (Network Address Translation)** is a process of changing the source and Destination IP addresses and ports. Address translation reduces the need for IPv4 public addresses and hides private network address ranges. This process is usually done by routers or firewalls.

For e.g. In the figure above, Host A request a web page from an Internet server. Because Host A uses private IP addressing, the source address of the request has to be changed by the router because private IP addresses are not routable on the internet. Router R1 receives the request, changes the source IP address to its public IP address and sends the packet to server S1. Server S1 receives the packet and replies to router R1. Router R1 receives the packet, changes the destination IP addresses to the private IP address of Host A and sends the packet to Host A.

There are three types of address translation:

1. **Static NAT –** translates one private IP address to a public one. The public IP address is always the same.
2. **Dynamic NAT –** private IP addresses are mapped to the pool of public IP addresses.
3. **Port Address Translation (PAT) –** one public IP address is used for all internal devices, but a different port is assigned to each private IP address. Also known as **NAT Overload**.

**2) Project Idea Mapping**

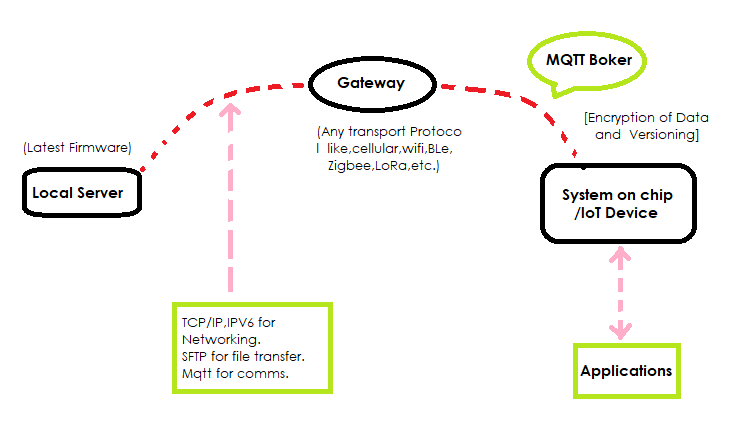
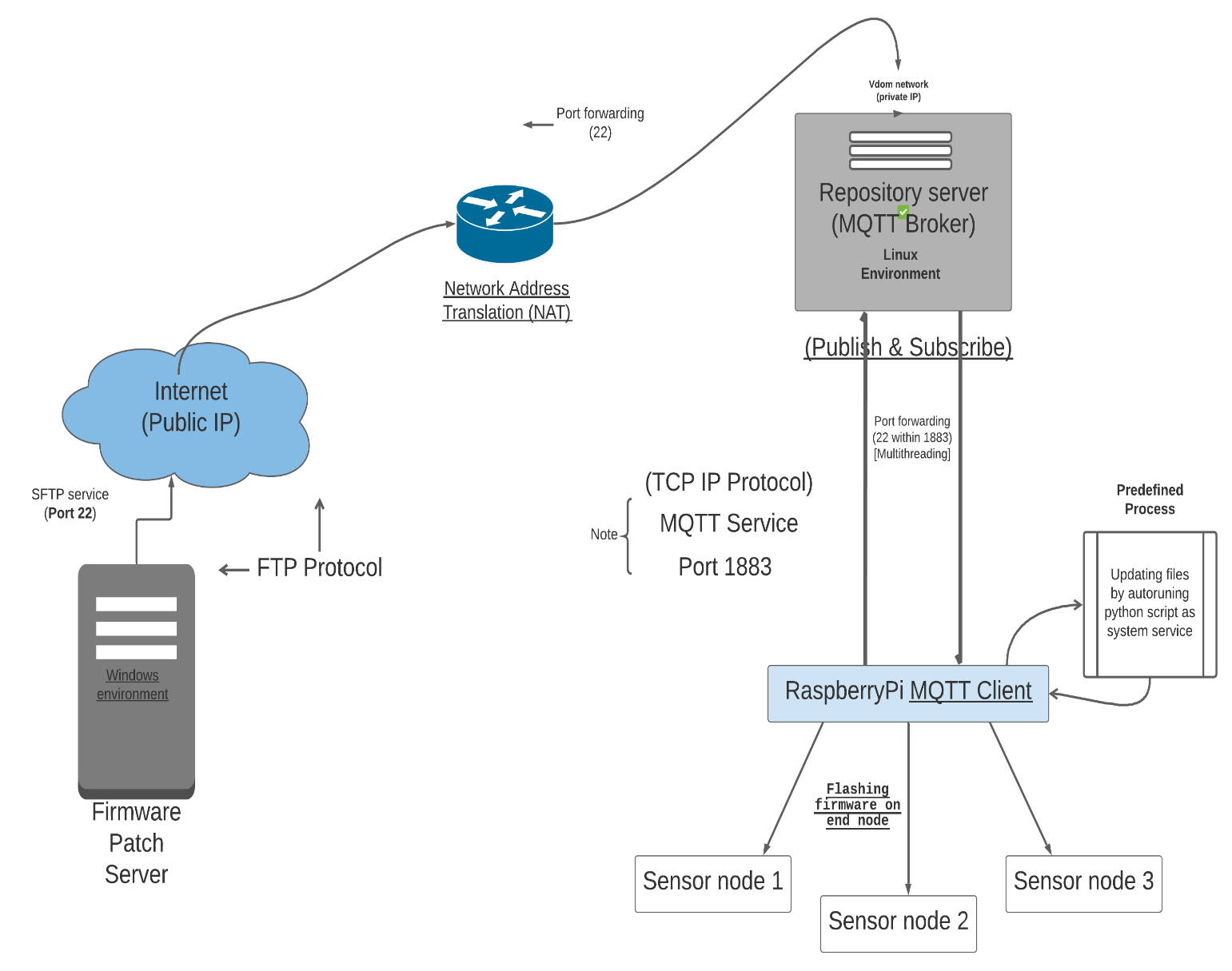


Fig.3

* The Project Idea is based on the Elemental Model shown in the Figure. 3.
* In this generalised diagram, the local server which will act as a Source Machine, sends the latest firmware files to the remote server via a gateway.
* Here, Gateway acts as a bridge between IoT Devices and the Cloud.
* The MQTT broker has been Set up at the Remote Server, which is responsible for publishing availability of latest firmware to SoC.
* Then, SoC will send back the Acknowledgement message, which will eventually lead to the downloading of the latest firmware file.
* And finally, the latest firmware will be transferred to the end nodes.

**3) Network Architecture**



(Linux Environment)

Fig.4

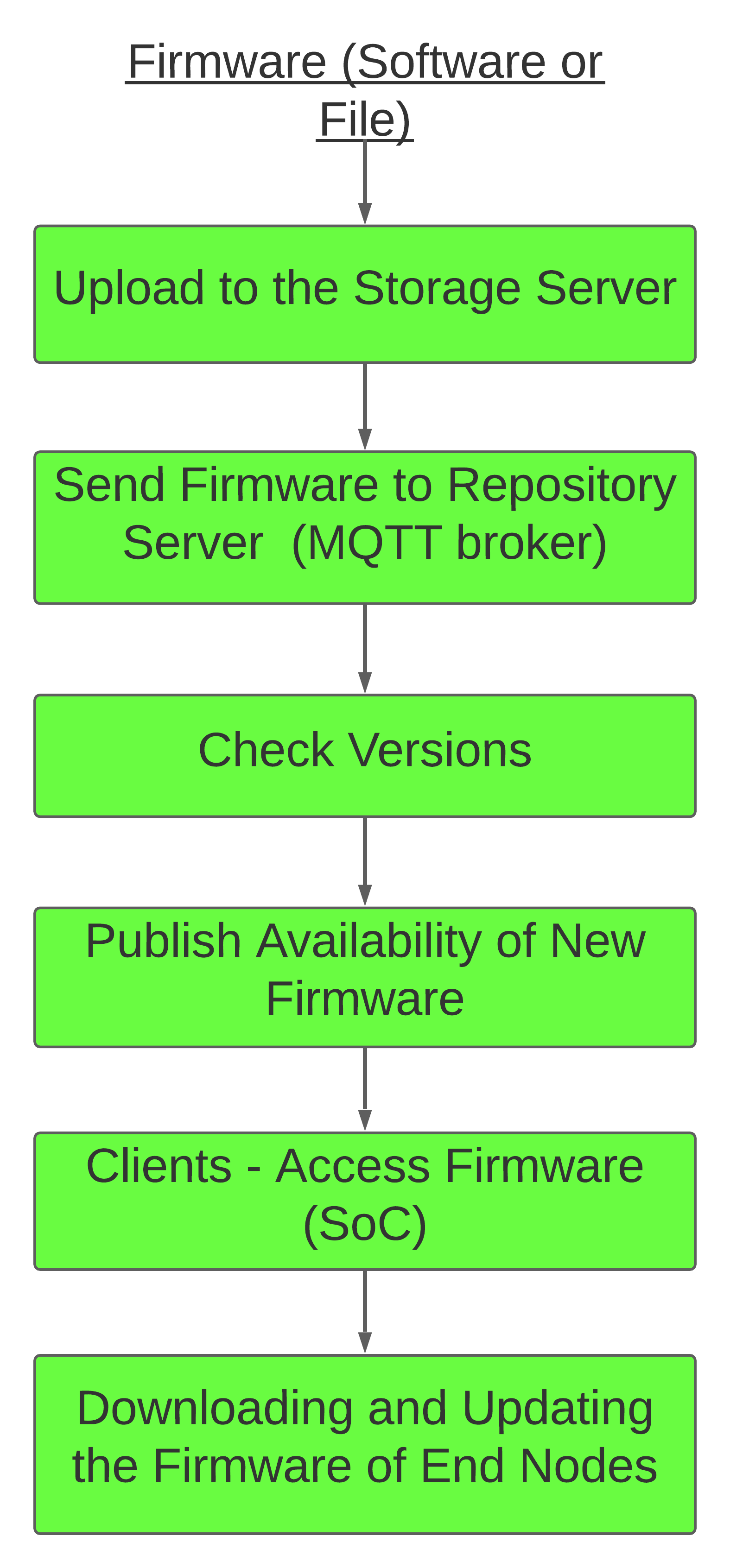
* The Network Architecture as shown in the fig.4 is designed based on the Elemental Model.
* In the Network Architecture diagram, the Firmware Patch Server i.e., Machine1 or the source machine will push the files to the Repository Server using port 22. The files are transferred using FTP protocol.
* On the Local Machine SFTP Server has been Setup in the Linux Environment and files are being transferred over SFTP Service.
* As we are working with the Private Network, we are using NAT technology for translating our private IP address to the public IP Address.
* At the Repository Server, the MQTT Broker has been set up in the Linux Environment. On the Broker side, Virtual Domain Network is present. So, as on the Broker side, we have Private Network, therefore We’ve local private IP Address on the Repository Server (i.e., Source Address), A static public IP address on the Router and then a specific destination address.
* The NAT translates the Private IP Address, which is not unique globally to the Public IP Address and Vice versa. In other words, NAT helps to connect the Private and Public Network together, due to which network devices have access to the Internet.
* Here, the Port Forwarding has also been enabled, so that we can Direct our request from the Repository Server to our Local Server.
* The File transfer is taking place over the SFTP Service. So, port 22 is been used.
* We are using Paramiko Framework of Python for Secured File Transfer, to the Repository Server.
* Once the files are pushed from the Source Machine i.e., Local Server to the Repository Server, The Repository Server will publish the availability message to the SoC. In our case we are using Raspberry Pi.
* As Raspberry Pi has subscribed to the MQTT Broker for New Firmware Availability, it functions as MQTT Client and Receives availability message for New firmware files.
* Then as MQTT Client, it sends Acknowledgement message to the broker- in the .get format, for eventually downloading the files.
* Here, As MQTT service is been used, it is TCP/IP Protocol based and we’ll use Port 1883.We are using SFTP for file transfer, so it will take place through Port 22.
* The Raspberry Pi, then updates the firmware at Sensor Nodes with the New Firmware. This updating of files will be done by autorunning python script as System Service, which is the predefined process.

**4) File Transfer**

* The files are been uploaded from Machine 1 (Local Server) to Machine 2 (Repository Server) through SFTP, and the protocol used here is FTP. These files are transferred through .put .
* The modules that are developed will provide the following functionality:

1. Hashing of files, Hashing provides with the Encryption of file and helps to uniquely identify a file and Replacing Old files with the new one.
2. Compression of files in a particular format.

* All the tasks i.e., The Hashing, Replacing and Compression of the files will take place at Machine 1 itself even before the files are transferred to Machine 2.
* After file transfer, compressed files will be present on Machine 2.
* Firstly Machine 2 will publish Availability message to Machine 3 using MQTT protocol.
* After publishing the message, MQTT Port will be closed and immediately after that, at the same time SFTP Port will be opened for file transfer i.e., First message will be published through the port 1883 and then the port will be closed, and after that Port 22 will be opened for file transfer.
* When the file is transferred to Machine 3 (Raspberry Pi), it will update the firmware at the Sensor Nodes with the New Firmware.
* To access the Mqtt functionalities through python program, we need to install the Mqtt Python libraries on Raspberry Pi.
* To read the received data in python program, we need python mqtt library.
* The most popular mqtt library for python is Paho-mqtt library.

**5) LAYERED ARCHITECTURE**

Firmware at Sensor nodes is updated wirelessly via serial data communication (UART). It happens at Physical Layer. At Physical Layer data is converted into bits and transmitted to other path via wired or wireless medium.

Availability Message is sent using MQTT Service, via Port 1883.It is TCP Based. It is being carried out over Transport Layer. Transport Layer is responsible for End-to-End Communication and Error free Delivery of Data.

The SoC (Raspberry-Pi) acts as Gateway. Here Internet Protocol is functional. IP responsible for successful File Transfer from Source Host to Destination Host. It is achieved at the Network Layer.

Client (Raspberry - Pi) receives and downloads the file. It takes place at Data link layer. Data link layer looks out for Hardware Address i.e., MAC Address. The 3 main functions of the Data link layer are to deal with transmission errors, regulate the flow of data, and provide a well-defined interface to the Network Layer.

File Transfer is been done via FTP. It takes place at Application Layer. Application Layer provides services to the end users to work over network or internet

**6) Resource Estimates**

* **Hardware Requirements:**
* Firmware Patch Server with Linux Operating System.
* Repository Server with Linux Operating System.
* Raspberry Pi (Model B+) with Raspbian Operating System.
* 16 GB Memory Card.
* DHT – 11 Sensor.
* **Software Requirements:**
* Paramiko Framework of Python for Secure File Transfer Protocol.
* Paho-Mqtt library to access the Mqtt functionalities through Python program.
* Mqtt Broker – Eclipse Mosquitto
* Python version 3.7 or above
* Linux Environment for File Transfer.