A Forward Compatible IoT Protocol and Framework Addressing Concerns due to Internet-outage

A PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:
Manish Singh Saini,
Abhay Chandra,

Supervised by:
Dr. Gourinath Banda,
Assistant Professor,
Computer Science and Engineering,
Indian Institute of Technology Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2016

CANDIDATE'S DECLARATION

We hereby declare that the project entitled **"A Forward Compatible IoT Protocol and Framework Addressing Concerns due to Internet-outage"** submitted in partial fulfillment for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** is an authentic work.

The project was supervised by **Dr. Gourinath Banda, Assistant Professor, Computer Science and Engineering, IIT Indore**.

Further, we declare that we have not submitted this work for the award of any other degree elsewhere.

Manish Singh Saini **Abhay Chandra**

CERTIFICATE by BTP Guide

It is certified that the declaration made by the students is correct to the best of my knowledge and belief.

Dr. Gourinath Banda,
Assistant Professor,
Discipline of Computer Science and Engineering,
IIT Indore

ACKNOWLEDGEMENTS

We would like to express our gratitude towards **Dr Gourinath Banda** for constantly supporting and motivating us throughout the project and for always giving us new ideas and guiding us in the right direction.

We would also like to thank our family, friends and colleagues for always being there with us and for being a constant source of motivation.

Manish Singh Saini
Abhay Chandra
B.Tech IV Year
Discipline of Computer Science and Engineering
IIT Indore

ABSTRACT

Internet of Things (IoT) is one of the most emerging fields in the technology sector today. It has the potential to change and enhance the everyday tasks of people ranging over the sectors such as infrastructure, medical, etc. to everyday living of the people. Soon there are going to be millions of devices communicating with each other and sharing information to automate various tasks, which otherwise are time-consuming and tedious.

Therefore, many big organisations are working on IoT-related guidelines, references, standards and architectures. In the following work, at first, one such protocol i.e One IoT protocol ver1.0 is discussed. This One IoT (ver1.0) protocol defines minimum framework, architecture and guidelines which help in identifying what devices can be called as IoT-enabled, help OEMs make such devices, which remain forward compatible with various upcoming IoT frameworks.

Majority of the existing IoT protocols, frameworks, architectures, etc. do not address certain concerns that are specific to Indian context. We propose improvements to One IoT protocol ver1.0 to address these challenges. An example for such a concern is *Internet outage*. Here the obvious question is “how should the IoT devices behave when they are not connected to the internet? ”. We define the notion of Safe mode operation in which the device operates when internet goes off. Simply not doing anything isn’t always an option as will be seen further by giving examples from sectors like medical. With these modifications we define the Version 2.0 of One IoT Protocol. We discuss the modifications done in the protocol in detail, going on to the implementation and testing of the protocol.

Contents

Candidate's Declaration iii

Certificate by BTP Guide v

Acknowledgements vii

Abstract ix

1 Introduction to Internet of Things 1

2 One IoT Protocol ver1.0 3
- 2.1 Architecture . . . 3
- 2.2 Minimum Framework . . . 4
- 2.3 Minimum Behaviour from an IoT-node . . . 4
- 2.4 Protocol Entities . . . 5
- 2.5 Protocol . . . 5
- 2.6 Dealing with the Dynamic IP addresses of the IP devices . . . 8
- 2.7 Security . . . 8
- 2.8 Shortcomings of the Protocol . . . 9

3 Challenges to One IoT Protocol ver1.0: Inherent to Internet 11
- 3.1 DoS/DDoS Attacks . . . 11
- 3.2 Internet Connection Lost . . . 13
- 3.3 Power failure of the IoT device or the User Agent . . . 13

4 One IoT Protocol ver2.0 15
- 4.1 Major Problems . . . 15
- 4.2 Safe Mode . . . 15
- 4.3 Steps involved in Safe Mode . . . 16
- 4.4 Modifications done in ver2.0 . . . 18

5 Implementation and Testing 21

6 Conclusion and Future Scope 25

Bibliography 27

Appendices 31

A Internet Health Check 31

B Updating local copy of SDF at Thing 33

C Safe Mode Operations 35

D Safe Mode Descriptor File(SDF) Format 37

List of Figures

2.1 IoT Architecture . . . 4
2.2 IoT Node Architecture . . . 5
2.3 Registration Flow . . . 6
2.4 Advertisement Flow . . . 7
2.5 Invocation Flow . . . 7
2.6 Handling IP changes in DAMs . . . 9

3.1 Smart Medical Device(SMD) during Normal Mode . . . 12
3.2 Smart Medical Device(SMD) during Internet Outage . . . 12

4.1 SDF for a device . . . 16
4.2 Safe Mode Triggering . . . 17
4.3 Triggering Mode FlowChart . . . 18
4.4 IoT Node Architecture ver2.0 . . . 19
4.5 Architecture ver2.0 . . . 20
4.6 Periodic Fetching . . . 20

5.1 UTM Cloud Running . . . 21
5.2 Safe Mode Operations under Internet Health Check Failure . . . 22
5.3 Normal Mode Operations under Internet Health Check Success . . . 23
5.4 Client Receiving Response from Thing . . . 23

Chapter 1

Introduction to Internet of Things

Internet of Things (IoT) is a network of physical device - which include automobiles, buildings, basic infrastructure, household devices, etc.- that are embedded with sensors, actuators and/or microcontroller/s together with communication module enabling them to communicate with each other over Internet. IoT allows various objects and devices to be controlled remotely over Internet. The basic idea behind IoT is to make everyday systems act more intelligent. It is a network of connected systems which can handle real time requests as well as provide real time data to perform various tasks conveniently and efficiently. The opportunities with IoT deployments across various domains are endless.

The advancements in IoT as a technology are happening at a lightning speed. It is not just a Machine-to-Machine interaction over a network, many architecture and protocols are being developed all over the world in order to create a standard framework which IoT devices can follow so that they can create a well-balanced and flourishing IoT ecosystem. Many big organisations and alliances are working together to devise such standards to bring out robustness in such systems. For instance, ThingWorx is one such system released by PTC [1]. Open Interconnect Consortium (OIC) has recently released a reference IoT-architecture [2] as well. The Industrial Internet Reference Architecture (IIRA) is a reference architecture for industrial IoT [3] catering for manufacturing sector. Identifying the gravity of this drastically changing IoT paradigm, several nations have released their own IoT policies[4]. There is a great need for such standards because IoT devices need to follow some protocol and architecture so that they can work together seamlessly. However, due to many organisations working on their own set of rules, there is an array of options for the OEMs and the customers as well leading to heterogeneity in the sector. Many such small and disjoint networks of smart devices following their own rules won't serve the purpose. There is a need to have a unified protocol standard which can support and bring together all such networks on a single platform so that all smart devices can come aboard a single large network and thus overcoming the limitations due to heterogeneity. Our One IoT protocol ver2.0, like its predecessor ver1.0, proposes abstractions that not only addresses heterogeneity and forward compatibility, but also addresses the arising concerns due to Internet outage.

Chapter 2

One IoT Protocol ver1.0

One IoT Protocol ver1.0[5] defines a protocol and framework, that is unconditionally forward compatible. It defines:

- the minimum criteria to be followed by a device to qualify to be called as an IoT-enabled device and
- a standard to be followed by the IoT devices to work together efficiently to create a robust IoT ecosystem.

It also provides a reference to the IoT OEMs to create IoT-enabled devices across various domains and applications.

Preliminaries

- **Embedded Systems** : These are micro-controller based electronic control systems, embedded in a physical device, which provides the main IoT functionality to the device.
- **Embedding Entity** : It is the physical device which embeds the embedding system defined above. For example, in a Smart Air Conditioner(AC), controller system is the embedded system and AC is the Embedding entity.
- **Sensors and Actuators** : These devices bridge the physical world with the digital world in an embedded system. Sensors sense the measurable quantities of change in the environment and give this information as a signal back to the actuators which performs physical task based on the input signal.

As per the protocol, **Internet of Things** is a worldwide network of interconnected embedding devices, each having its own IP address and hence is uniquely addressable on the internet. Also, it includes accessing, controlling and availing various services from the devices.

2.1 Architecture

The architecture defined by One IoT protocol ver1.0 comprises of the following components:

1. IoT Node/s
2. Internet Connectivity Layer
3. Cloud Layer
4. User Agents

*Definition(**IoT Node**)*: An IoT node consists of an embedded system, which is a micro-controller based system with IP-enabled interface along with sensors and/or

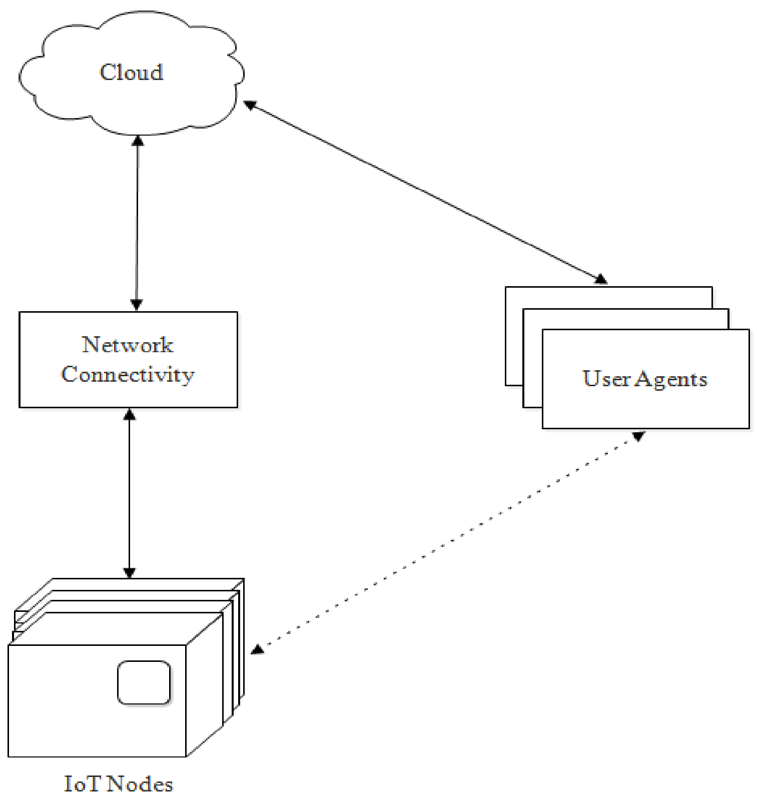


FIGURE 2.1: IoT Architecture

actuator(s), and an embedding entity, which is a physical system, which embeds the embedded system.

For example, a Smart Air Conditioner (AC) is an IoT Node, which has a microcontroller system as its embedded system and AC as an embedding entity.

If a device qualifies to be an IoT node, it is called as IoT enabled device or a Thing. Thus, 'Things' are physical objects consisting of processor/s, sensor/s and/or actuator/s with some intelligence either *in situ* or *ex situ*[6].

2.2 Minimum Framework

Minimum framework of One IoT Protocol ver1.0 includes :

Cloud Server (Remote host/s), Connectivity layer (both IP and other wireless modes) and IoT nodes.

2.3 Minimum Behaviour from an IoT-node

The IoT node must incorporate a minimum behaviour to be a part of the protocol's framework. This includes:

1. Receiving commands from the remote host/s and user agent/s through any wireless media.
2. Executing the commands accordingly and,
3. Reporting its operational and configurational status and sensor information periodically to the authorised remote host/s.

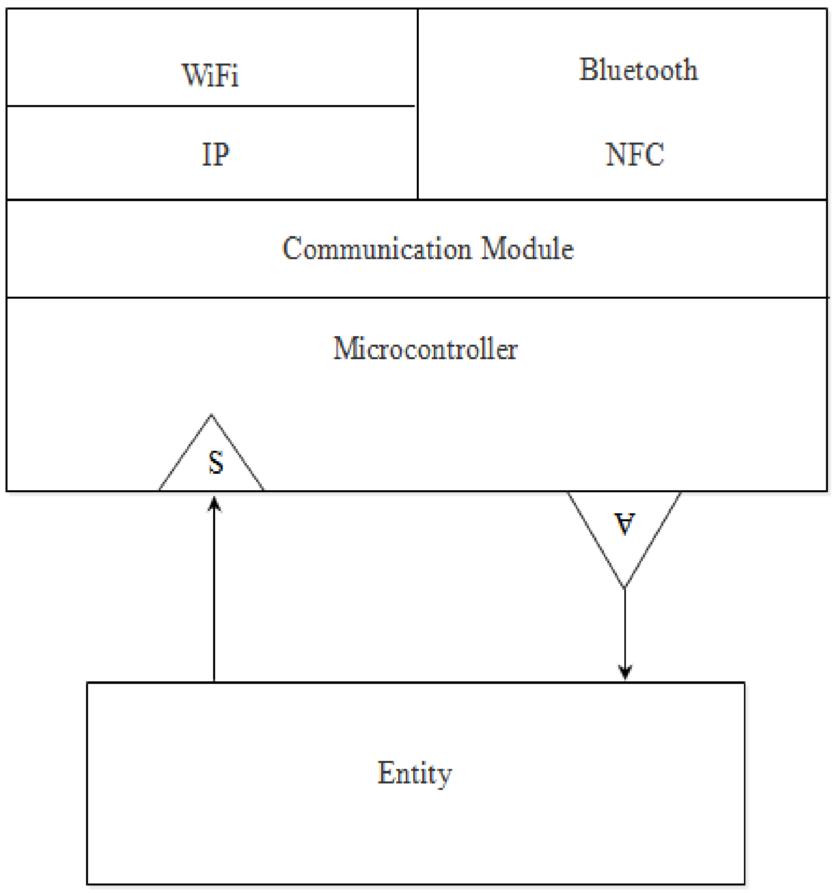


FIGURE 2.2: IoT Node Architecture

2.4 Protocol Entities

The protocol runs on Application layer and consists of three entities: User Agents (Client), Cloud and IoT Nodes.

User Agent or Client is a host machine which can either be a web app, mobile app or any other device used to access IoT nodes remotely with the help of communication media such as WiFi or Bluetooth.

Cloud is the place where the remote host is running. It plays an important role in keeping track of the large and growing number of IoT devices connected to the network.Thus it maintains user accounts of various user agents (Clients) which are mapped with the IoT devices to be respectively used by them. The cloud thus grants or revokes access to the IoT devices and provides a seamless communication to the user agents even in case of dynamic IP addresses of the devices.

2.5 Protocol

The protocol operates on three entities – User Agents, Cloud and IoT nodes.

The protocol runs on Application layer and is independent of the communication media. However, better and uniform communication experiences can be obtained by minor tweaking for various media.

The various functionalities/steps involved in the protocol are:

1. **Discovery** The IoT nodes need to be discovered before any communication can actually happen between various entities.Only then will the user agents be able to communicate with them.

   This is done by performing a local scan via WiFi, Bluetooth or other short range protocols.

   Also this phase isn't applicable for IP based communication.

2. **Registration** An User Agent can communicate with an IoT device if it has it's IP address and key. However, the IP address isn't static and may change timely. The remote host at Cloud maintains a many-to-many relationship between User Agents and Things. If a User Agent wants to access a certain Thing, it must log in with its credentials on the remote host on Cloud and then provide the Thing ID and Thing Key. This is called as REGISTER request. (Thing ID is the unique identifier provided by the OEM. Thing Key is a private secret code of sufficient length.)

   The Cloud then grants user access to that particular Thing. Also, the User can now communicate with the Thing even in case of IP address change as it can always send a REQUEST-ADDRESS to obtain the new IP address.

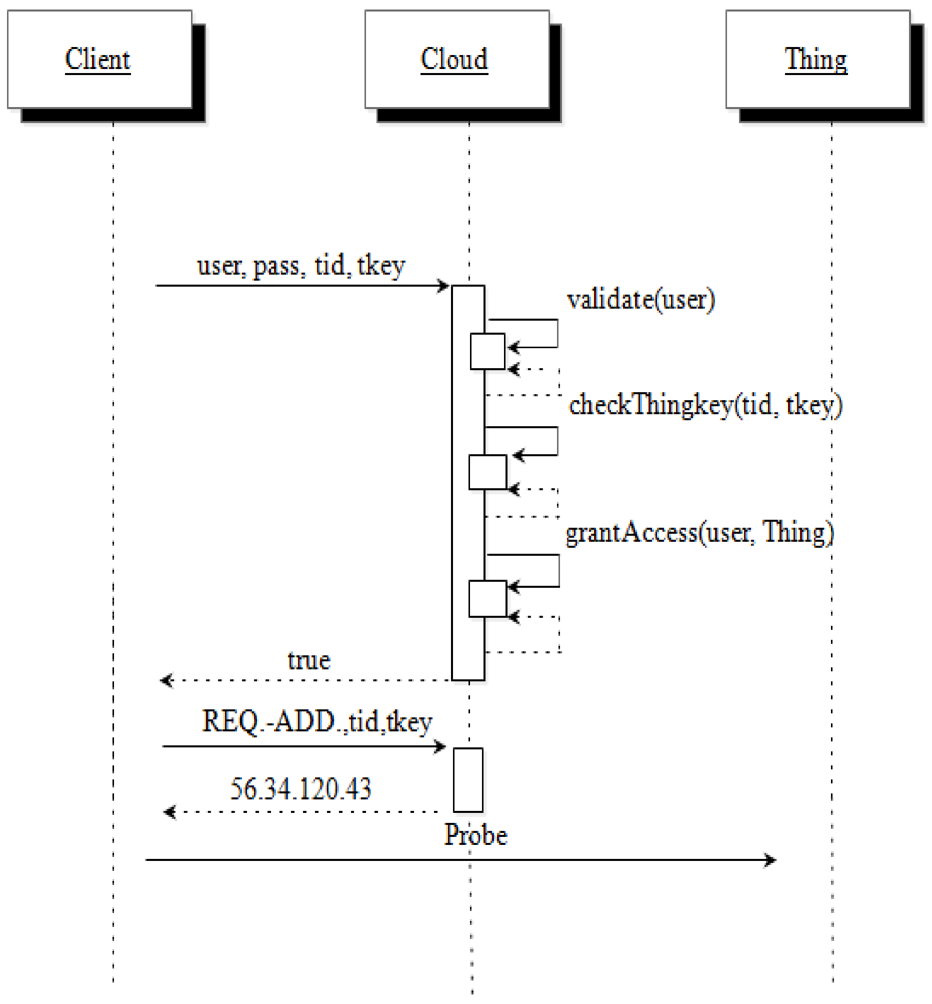


FIGURE 2.3: Registration Flow

3. **Advertisement** After a successful registration of the Client with the Thing,the Client is now ready to avail services of the Thing. However, for that the Client needs to know two things:

   - What all services are offered by the Thing and,
   - What are the methods to avail those services?

For that, Client sends a PROBE request to the Thing, which returns a parse able manifest in response (can be ineither XML or JSON format). This response contains a list of services/operations offered by the Thing. This list includes identifiers to invoke those operations, expected I/O format for each service and a small human readable description of each service.

This type of response format also makes it possible to create dynamic user-interfaces on the fly specific to each Thing.

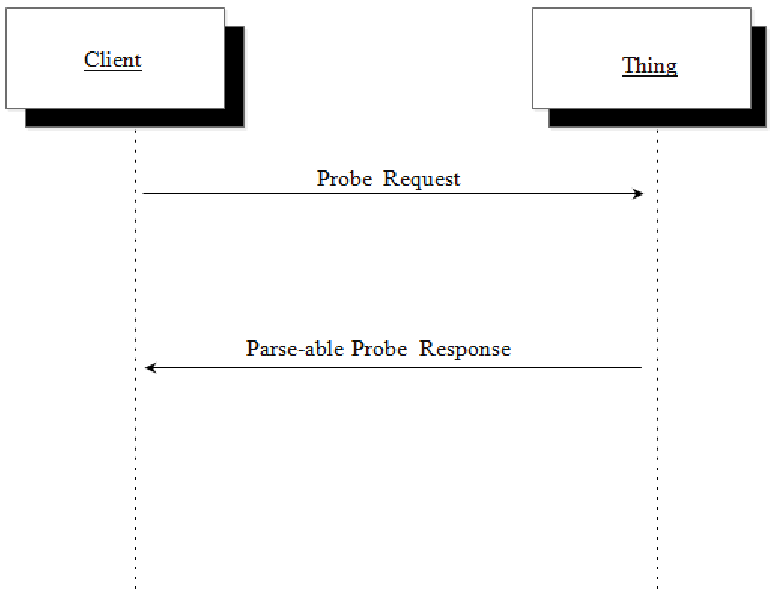


FIGURE 2.4: Advertisement Flow

4. **Invocation** After the registration and advertisement processes are done, the user agent can now avail the services by invoking the Thing.

   The parse able manifest obtained in the previous step helps in creating a dynamic interface which can easily be used by the user to invoke a number of services of the Thing.The output can also be displayed on the interface built upon the top of the output format taken from the manifest. Thus, allowing customer-friendly implementations.

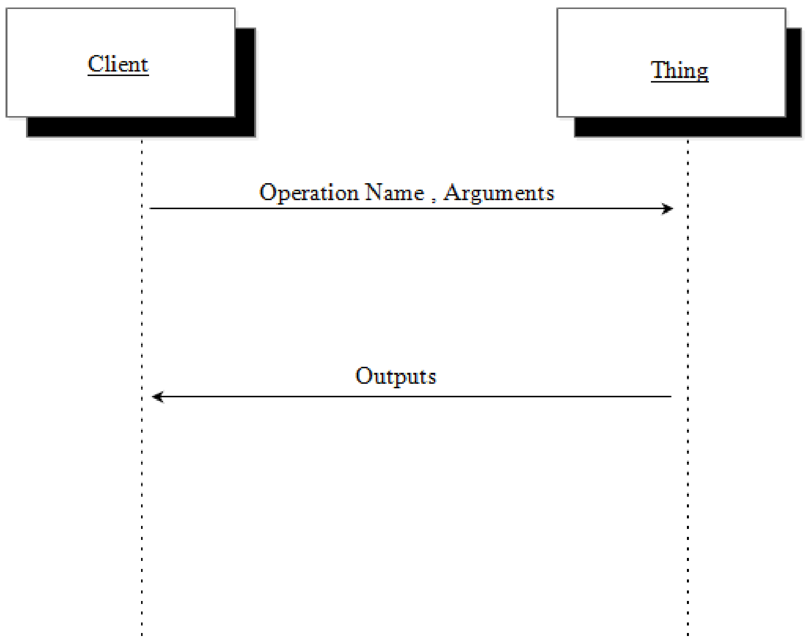


FIGURE 2.5: Invocation Flow

5. **Configuration Updates** As the number of IoT devices increase, there will be varying usage of such numerous IoT devices. Some being time critical will consume more power while others not so.Therefore, a number of protocol variables and server parameters will require updation. These changes include variation in tick rates, security parameters and different protocol variables. The main reason of changing these parameters is to enhance the performance metrics of devices as per their usage.

2.6 Dealing with the Dynamic IP addresses of the IP devices

The protocol works on all kind of communication media, be it Static or Dynamic.

For, **Static Addressed Media** such as Bluetooth, NFC or static IP, addresses of IoT devices don't change.

However, for **Dynamic Addressed Media** like the Internet which operates on IP, addresses are leased. These IP addresses keep on changing. The User Agent thus needs to update its local cache with the updated IP addresses of the Things it's mapped with. This is achieved by keeping the IP address of Cloud static.

All IoT nodes keep on updating their new IP addresses on the Cloud using the UPDATE-ADDRESS method, and the process is called as **pinging**.The rate at which this process occurs is called as **tick rate**. The Cloud is hardwired to listen to such methods and update the address of the Things in its local cache.

The higher the real time demand of IoT devices, the higher are the tick rates for those IoT devices.

In case a User Agent's request to access a Thing gets unauthorised because of the latter's change in IP address, the User Agent can now obtain the new updated address from the Cloud.

2.7 Security

As the number of IoT devices increase and reach to the order of billions, security will become a big issue[7]. There will be numerous devices sharing highly sensitive information with each other. Therefore, some level of robustness is needed in the IoT based solutions.

Security can be provided as a service from the cloud in case if the processing abilities of the IoT devices are limited.

In One IoT protocol ver1.0, there is a great need to encrypt the raw packet data, from end to end, with the help of accepted encryption standards having sufficient encryption entropy.

Also, a "salt" can be added to the encrypted packets in order to make the encryption entropy more random and security of data even more higher. Using salt, attacker won't be able to find out any pattern between the encrypted packet and the physical outcomes.

Attacks like DoS/DDoS can be rerouted to the Cloud by identifying them using several existing DoS detection services to prevent direct attack on IoT devices, which have limited hardware and hence have higher chances of crashing.

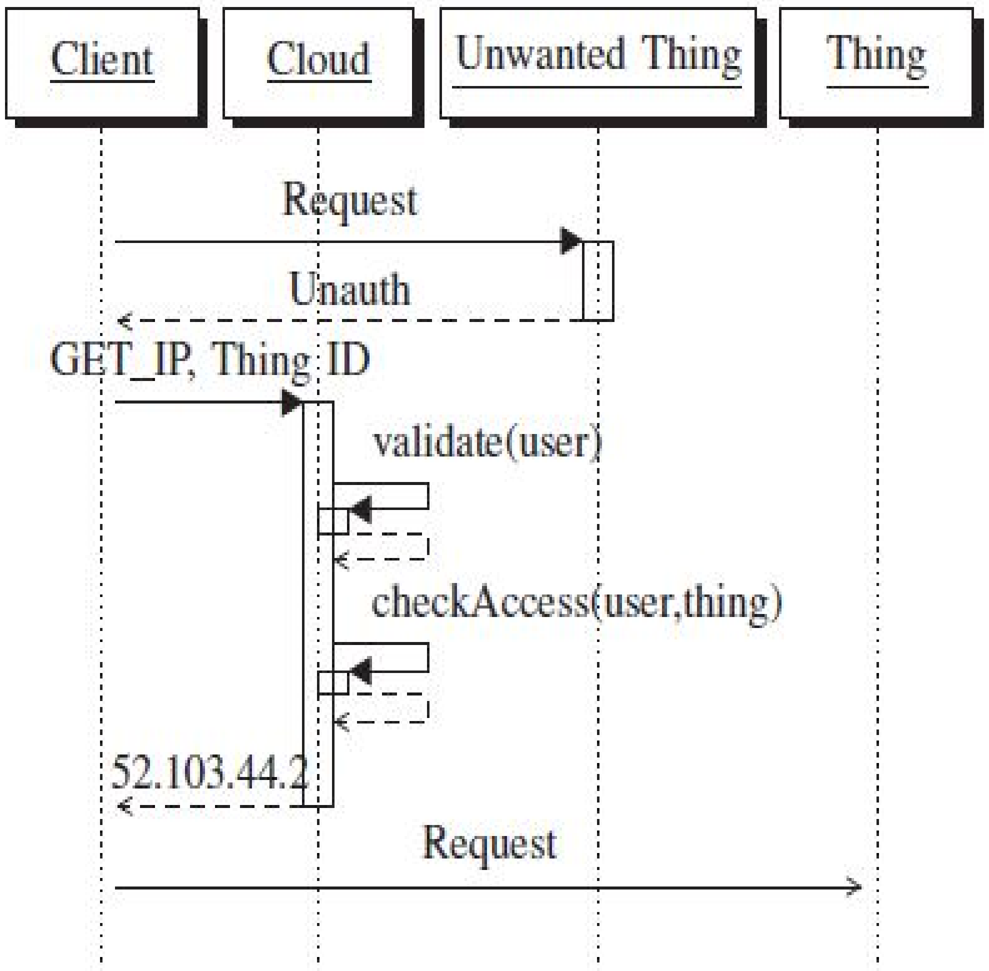


FIGURE 2.6: Handling IP changes in DAMs

2.8 Shortcomings of the Protocol

There are other several hurdles and challenges which come across the implementation of the IoT ecosystem. Cyber attacks, unexpected power failures, internet outage are to name a few. In most such cases, the core problem is Internet outage.

If an IoT ecosystem follows One IoT protocol ver1.0, then what are the IoT nodes supposed to do if the internet goes off suddenly? Should they just wait for the connection to revive? Maybe, but what if the requirement of the IoT node is immediate and critical? That is, waiting isn't option. Then, should the node take some action in the mean time between which connection revives? But, what action should it take?

These are some important questions which are not answered by One IoT Protocol ver1.0. Thus, there is a necessity to modify the protocol to account these challenges which prevail all over the internet space today, especially in the Indian landscape[8].

The following section discusses the challenges faced in detail and shows how One IoT Protocol ver1.0 doesn't account them. The section afterwards proposes One IoT Protocol ver2.0 as a modification of ver1.0 to deal with those challenges.

Chapter 3

Challenges to One IoT Protocol ver1.0: Inherent to Internet

Along with the general challenges to IoT a few other hurdles in consumer space are also reckoned. These are:

3.1 DoS/DDoS Attacks

A **denial-of-service (DoS)** attack is an attempt to make a machine or network resource unavailable to its intended users, such as to temporarily or indefinitely interrupt or suspend services of a host connected to the Internet.

A **distributed denial-of-service(DDoS)** is where the attack source is more than one, often thousands of, unique IP addresses.

Consider the picture of our One IoT protocol, where the clients access remotely located Things using their User Agents (Apps/Websites) via the Cloud Server. Now, think what will happen if the IoT nodes or Things in an IoT system are exposed to such attacks.There can be two possibilities :

1. **IoT device gets flooded with attacks** If this happens, the client registered on the Cloud to that Thing wouldn't be able to avail its services.
2. **Cloud Server gets flooded with attacks** In this case, the client will never be able to reach the Thing it wants to connect to as it will first need IP address of that Thing(if it has changed during the last tick...otherwise client can use stale IP to directly connect with it). This will affect almost all the clients registered on that cloud server.

In both the above cases, what if the services which the client wants to avail are of critical and/or immediate importance. Due to poor reliability on the services it would act as a major problem for the client to perform actions.

For example,based on some medical prescription,consider a Smart Medical Device (SMD), which automates the dosage of Insulin to be administered depending on the blood sugar level of a person. The remote application could be some algorithm to calculate appropriate insulin level to be administered and sends this dosage value back to the SMD, which in turn sets the insulin dosage to this value. Such SMD per se could be an IoT-device as per our definition. **If the Cloud or the Thing is under attack, then the medical device would never know the amount of dosage to be given to the user and our IoT setup will fail in such circumstances.** Often the response time related requirements in other sectors could be much more immediate and critical. Especially, with safety critical systems in the health-care area such situation could lead to loss of life.

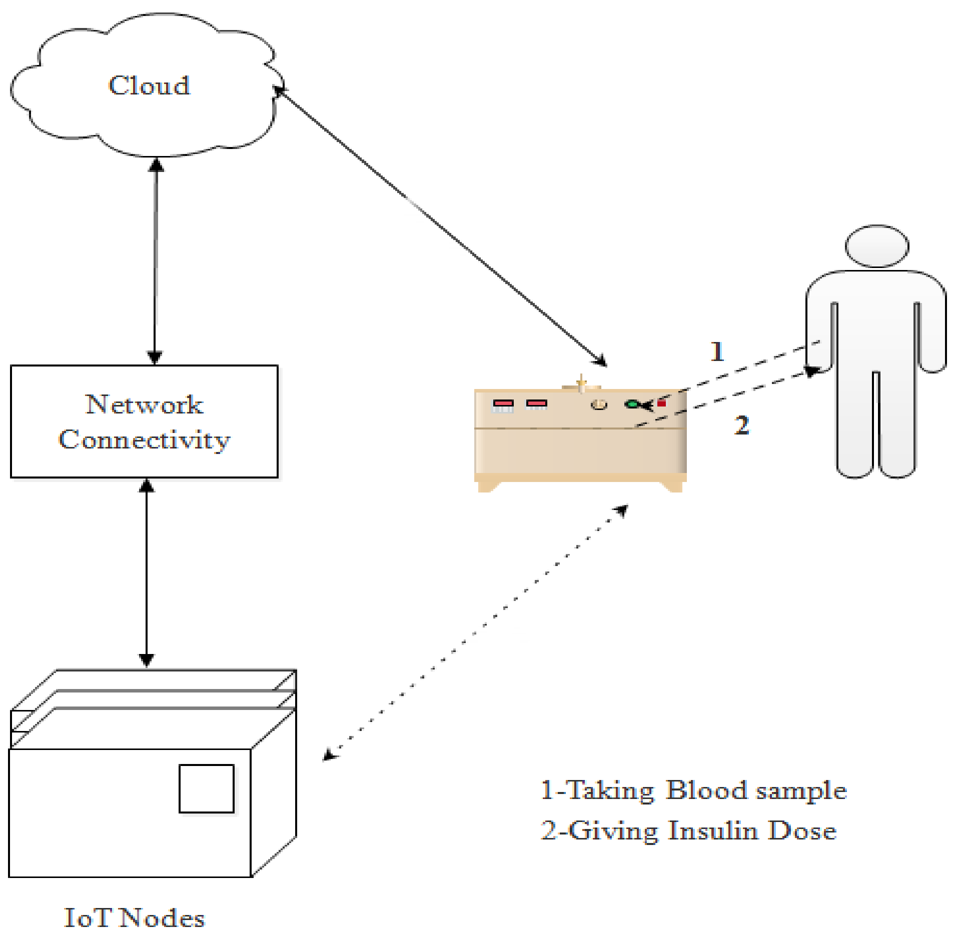


FIGURE 3.1: Smart Medical Device(SMD) during Normal Mode

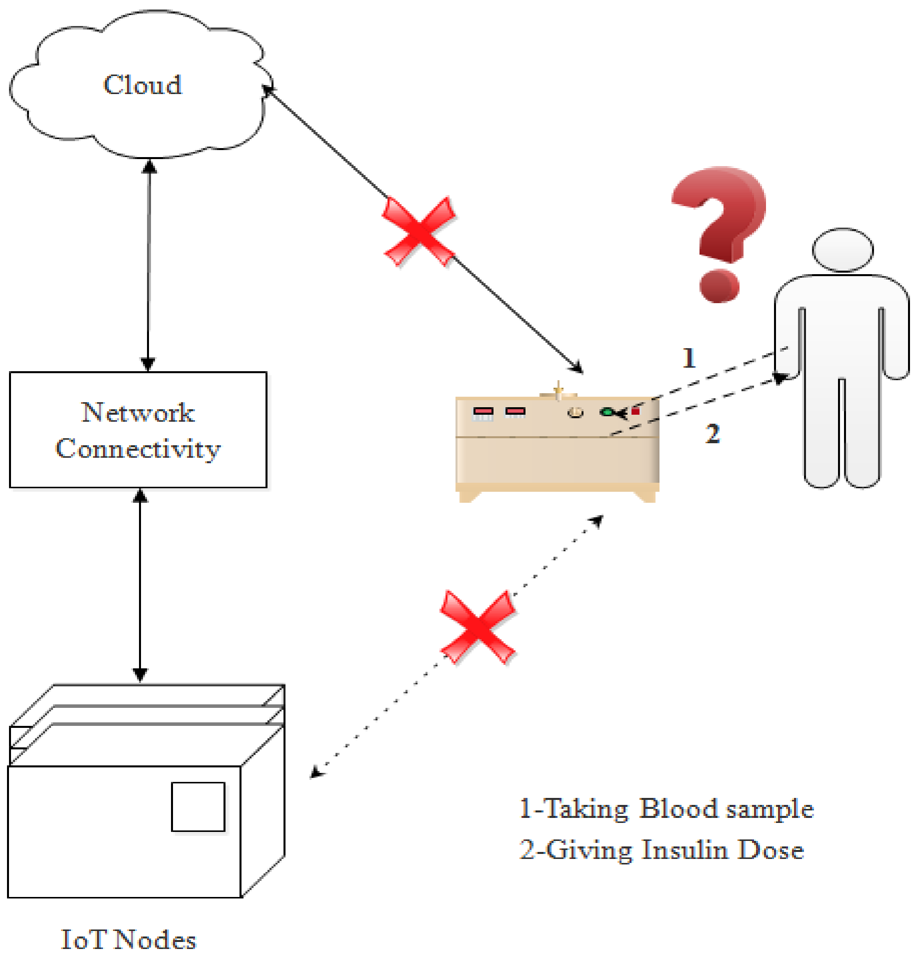


FIGURE 3.2: Smart Medical Device(SMD) during Internet Outage

3.2 Internet Connection Lost

Even today, Internet remains a major problem in India. Reliability, bandwidth and efficiency issues still prevail in the Indian Internet space. Such issues make it very difficult for the IoT ecosystem to thrive and unleash its full potential.

For instance, take the scenario of the medical device discussed before. If due to some reason there is no Internet connectivity, then there will be no option for the device to obtain the appropriate amount of Insulin dosage. Such failures occurring in commercial space where most of the manufacturing work is IoT controlled would lead to waste of both time and money.

Thus, it can be a high risk to deploy IoT systems in such unpredictable environment.

3.3 Power failure of the IoT device or the User Agent

If power failure occurs leading to shut down of the Thing, one won't be able to avail the services from that Thing leading to disruption of services. To ensure last mile connectivity of IoT in India, there are bigger problems of providing continuous and reliable power supply first before even thinking of successful deployment of IoT network.

Chapter 4

One IoT Protocol ver2.0

We are now going to deal with the issues of Internet outage. To overcome such issues a modified version of One IoT Protocol i.e **One IoT Protocol ver2.0.** is introduced.

4.1 Major Problems

Any IoT ecosystem is likely to face the following situations.

- Internet connectivity lost
- DoS/DDoS attacks
- Power failure(Connection related)

Therefore, in such situations if the Thing goes offline it won't operate and significant losses would incur. So whenever such Internet outage happens the device operates in **Safe Mode**.

4.2 Safe Mode

In One IoT protocol ver1.0, a minimum behaviour was proposed, which is to be expected of an IoT-node to fit in our framework and follow our protocol.These includes:

- Receiving the commands from authorised remote host/s and user agent/s (through any wireless media)
- Executing the commands accordingly and
- Periodically reporting operational status and sensor information to authorised remote host/s.

In addition to above behaviour, in version2.0 we have added one more behaviour,which is

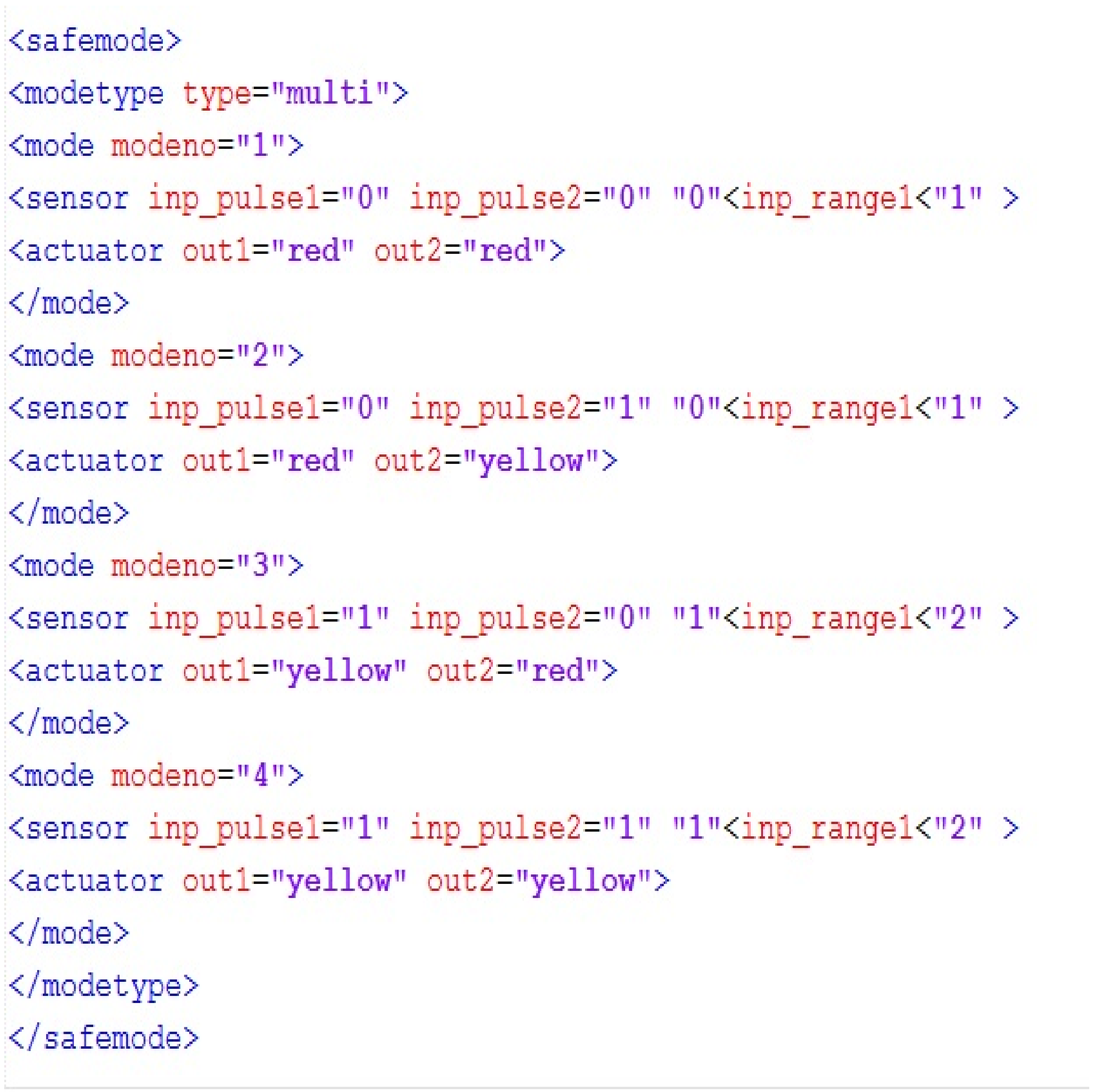
- Incorporating a Safe Mode, which would get activated when device suffers from Internet outage.

*Definition(**Safe Mode**)*: An IoT device consists of several sensors and actuators. These device depends on sensor values to set the actuation controls to certain values. **Safe Mode** for a given IoT device is defined to provide these set of values during Internet outage. The Safe Mode is further characterised by a Safe Mode Descriptor File:

- *Definition(**Safe mode Descriptor File(SDF)**)*: The Safe mode Descriptor File defines various standard sensor input values which correspond to their actuating output values in a predefined XML format.It can contain multiple Safe modes of operation depending on various combinations of input-output values and a Default Safe mode of operation under no inputs.

Following is an example on how the Safe mode Descriptor File(SDF) looks for a simple device showing various combinations of LED outputs to display the degree of danger with 'red' being the signal to the highest danger value for various signal inputs. The node has:

- three total inputs consisting of two switches and one value knob
- two LED outputs,and
- four safe modes of operation based on varying input values.



```
<safemode>
<modetype type="multi">
<mode modeno="1">
<sensor inp_pulse1="0" inp_pulse2="0" "0"<inp_range1<"1" >
<actuator out1="red" out2="red">
</mode>
<mode modeno="2">
<sensor inp_pulse1="0" inp_pulse2="1" "0"<inp_range1<"1" >
<actuator out1="red" out2="yellow">
</mode>
<mode modeno="3">
<sensor inp_pulse1="1" inp_pulse2="0" "1"<inp_range1<"2" >
<actuator out1="yellow" out2="red">
</mode>
<mode modeno="4">
<sensor inp_pulse1="1" inp_pulse2="1" "1"<inp_range1<"2" >
<actuator out1="yellow" out2="yellow">
</mode>
</modetype>
</safemode>
```

FIGURE 4.1: SDF for a device

4.3 Steps involved in Safe Mode

1. At first, the node will check whether there is an Internet outage and the process is called as **Internet Health Check**. It can either be a success (True) or a failure (False). There is a lot of already existing literature on how to detect the connectivity status.

   In case of a failure, it will perform Internet Health Check again for a fixed number of times in order to reconnect to the Internet before Safe mode is triggered in the device.

2. In case the Safe mode is triggered, the Safe mode Descriptor File (SDF) is read and actuation output values are implemented.

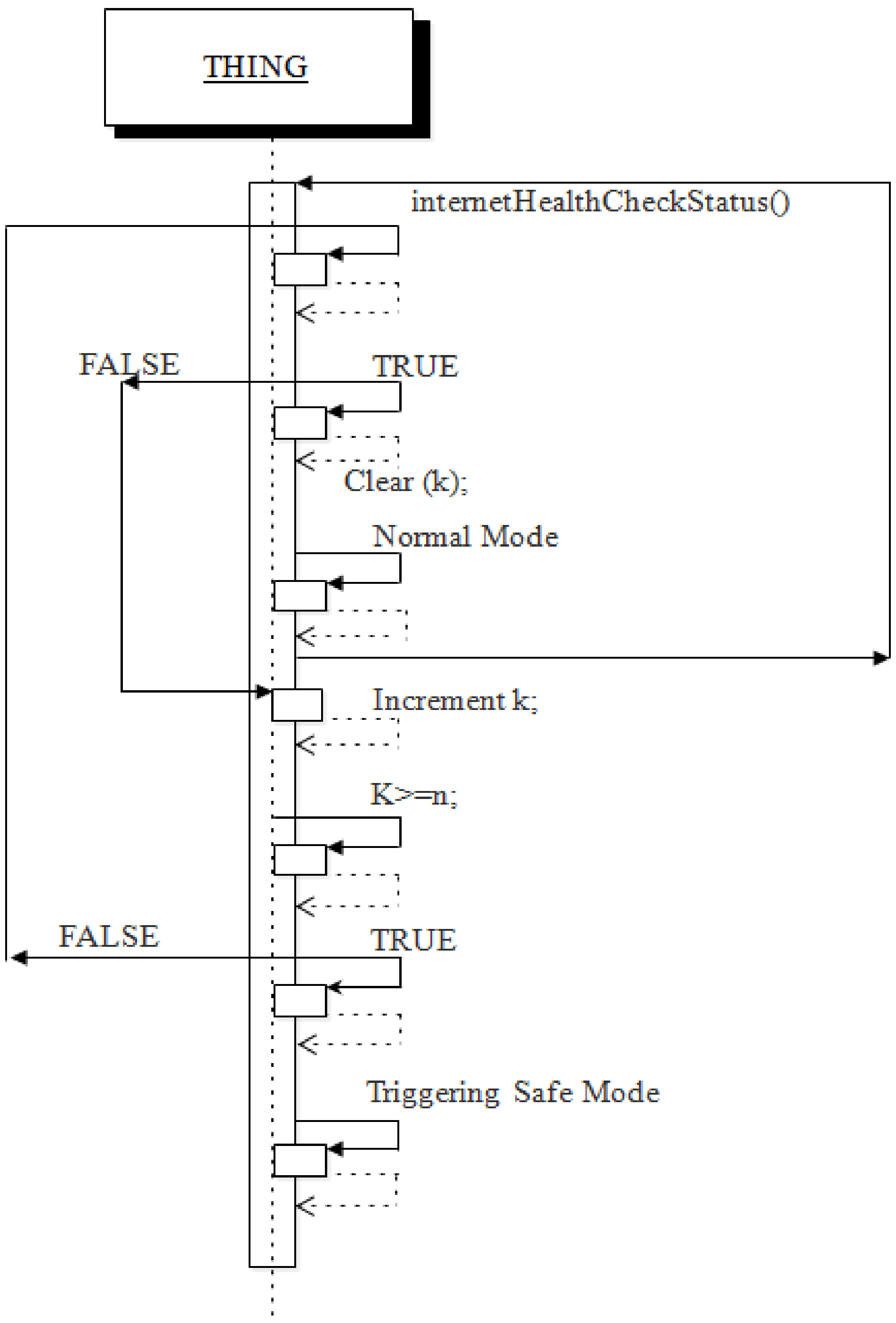


FIGURE 4.2: Safe Mode Triggering

3. The device remains in the the Safe mode for a fixed duration of time called as **Safe Mode Period**. A **Watch Dog Timer** takes care of this duration. When this duration lapses, Internet Health Check is done. If the connection is diagnosed to be normal, the device leaves the Safe mode and again moves into the Normal mode. In normal mode the device operates as per the One IoT protocol ver1.0 meaning that other protocol methods being same.

Recall the SMD example, where in case if the device can't get Insulin dosage value, it will transit from Normal mode to Safe mode. The naive Safe mode could either do nothing or administer a dosage as per the output value in SDF corresponding to match with the current input sugar level.

The above proposed additions to One IoT Protocol ver1.0 leads to the One IoT protocol ver2.0.

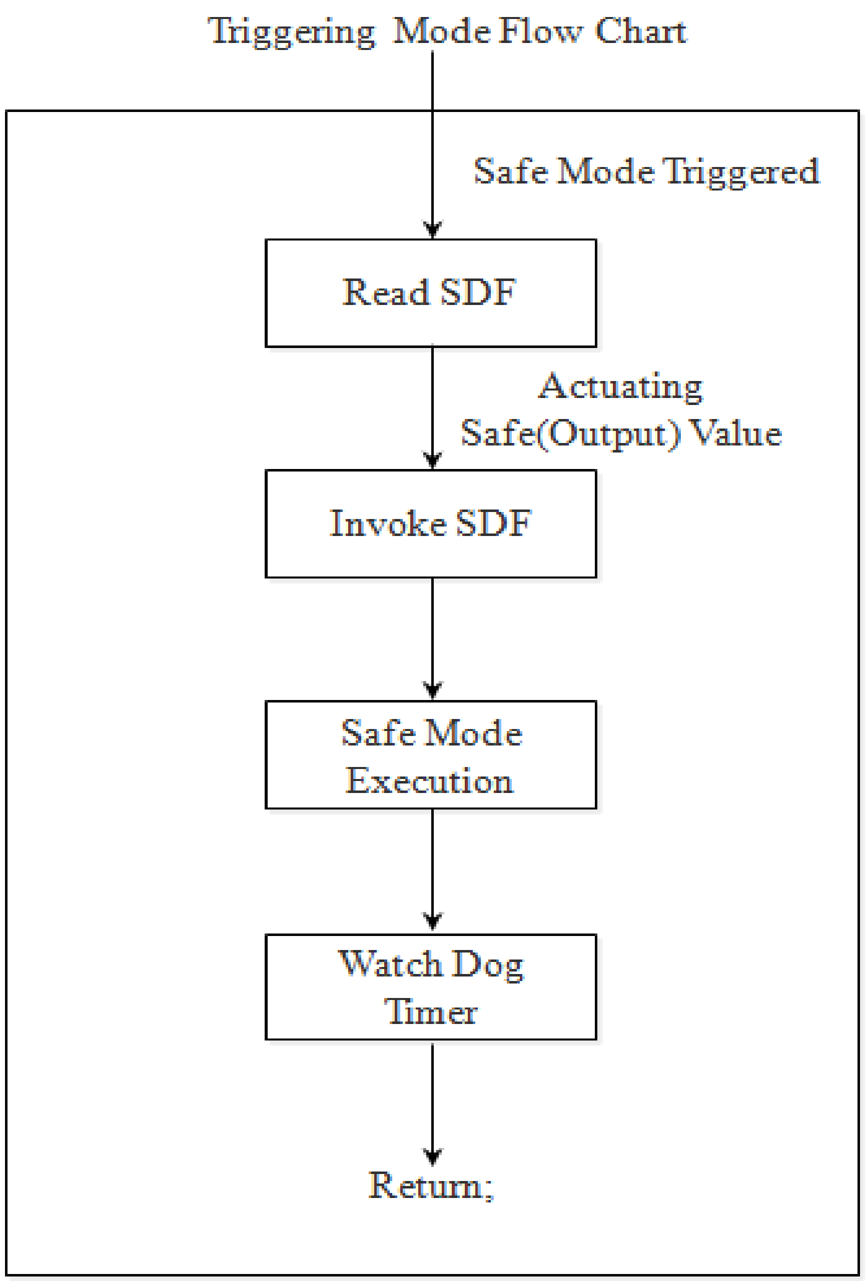


FIGURE 4.3: Triggering Mode FlowChart

4.4 Modifications done in ver2.0

Because of the introduction of Safe Mode, some modifications both in the protocol as well in the architecture are done.

The OEM now has to make and incorporate a SDF (Safe mode Descriptor File) into the IoT device in a predefined XML format (given by the protocol) which will be used during the Safe Mode Period. Thus, the following figure shows the modified IoT node architecture.

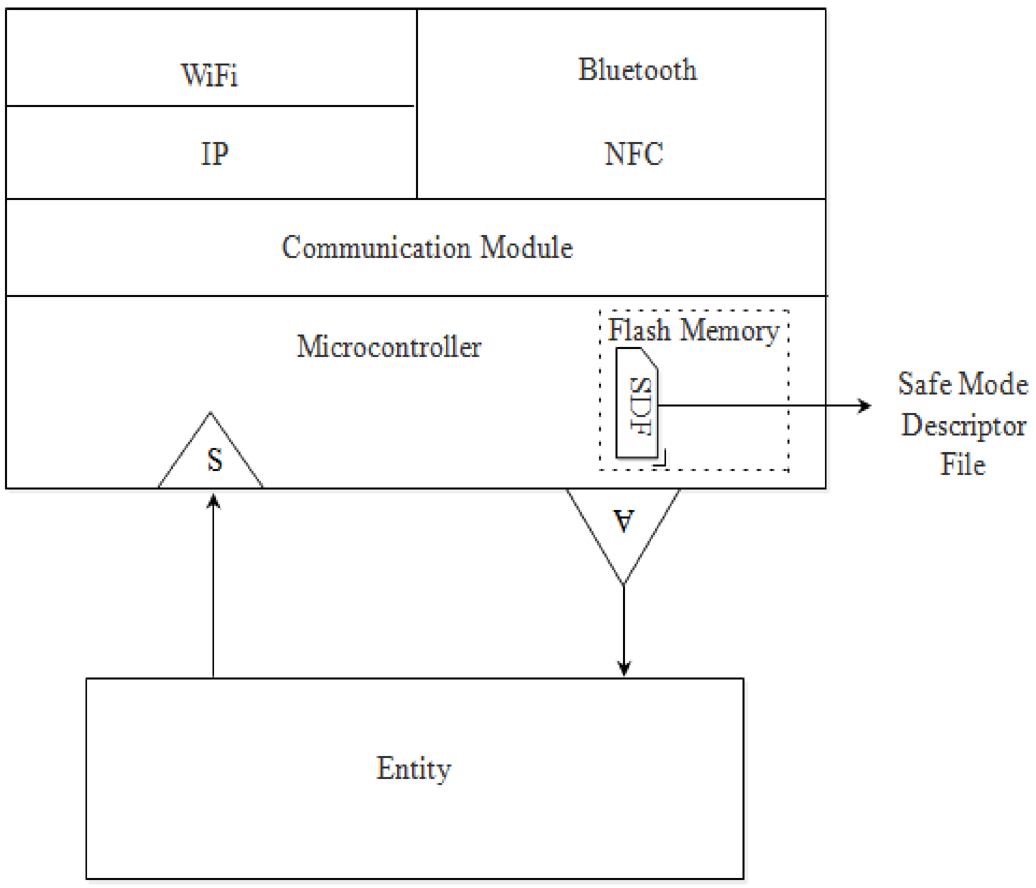


FIGURE 4.4: IoT Node Architecture ver2.0

With respect to the protocol, a few important modifications are done in version2.0.These are:

- The OEM must maintain an updated SDF of their IoT devices on their own server in a predefined format to be given by the protocol. This updated SDF is stored on the OEM Thing Utility (OTU) Cloud.
- Also, the remote host at Cloud, which we now call as the User Thing Management (UTM) Cloud has to maintain an updated SDF of IoT devices connected to it. This is important because Safe mode values of the devices may change over time and thus require modifications. The Cloud must periodically check the OTU Cloud to update its local storage of SDF.

  Similarly, IoT nodes should also periodically update their own local storage of SDF by checking it from the Cloud (just like pinging IP addresses). This process is called as **Periodic Fetching** and the rate at which this occurs is called as **Fetch Rate.**

Thus, the modified IoT Architecture for ver2.0 looks like the following figure.

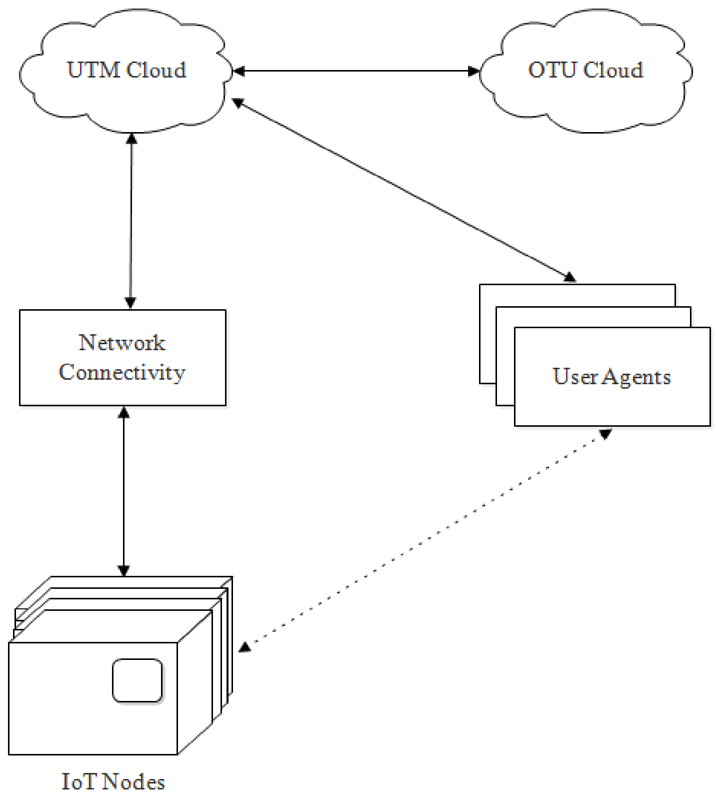


FIGURE 4.5: Architecture ver2.0

Periodic Fetching occurring between OTU and UTM, UTM and IoT Nodes:

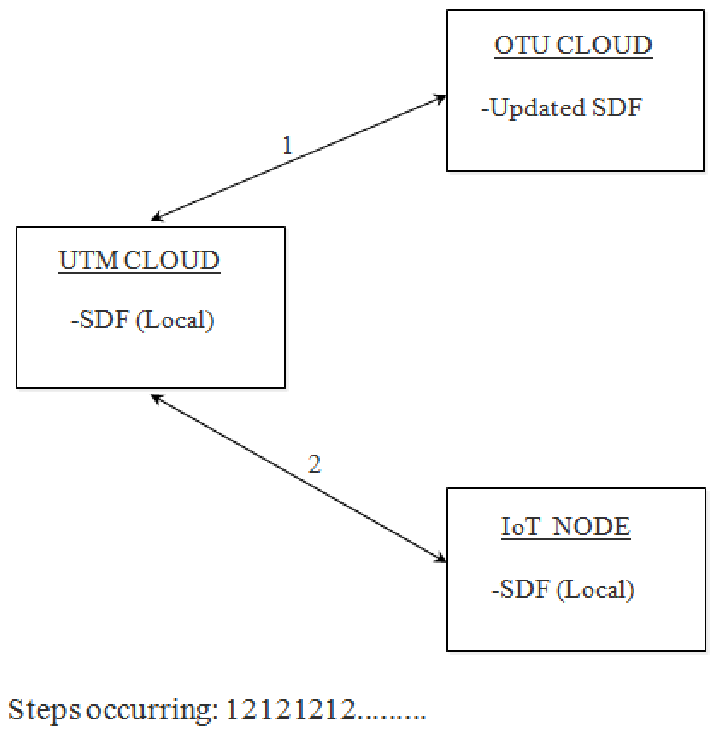


FIGURE 4.6: Periodic Fetching

Chapter 5

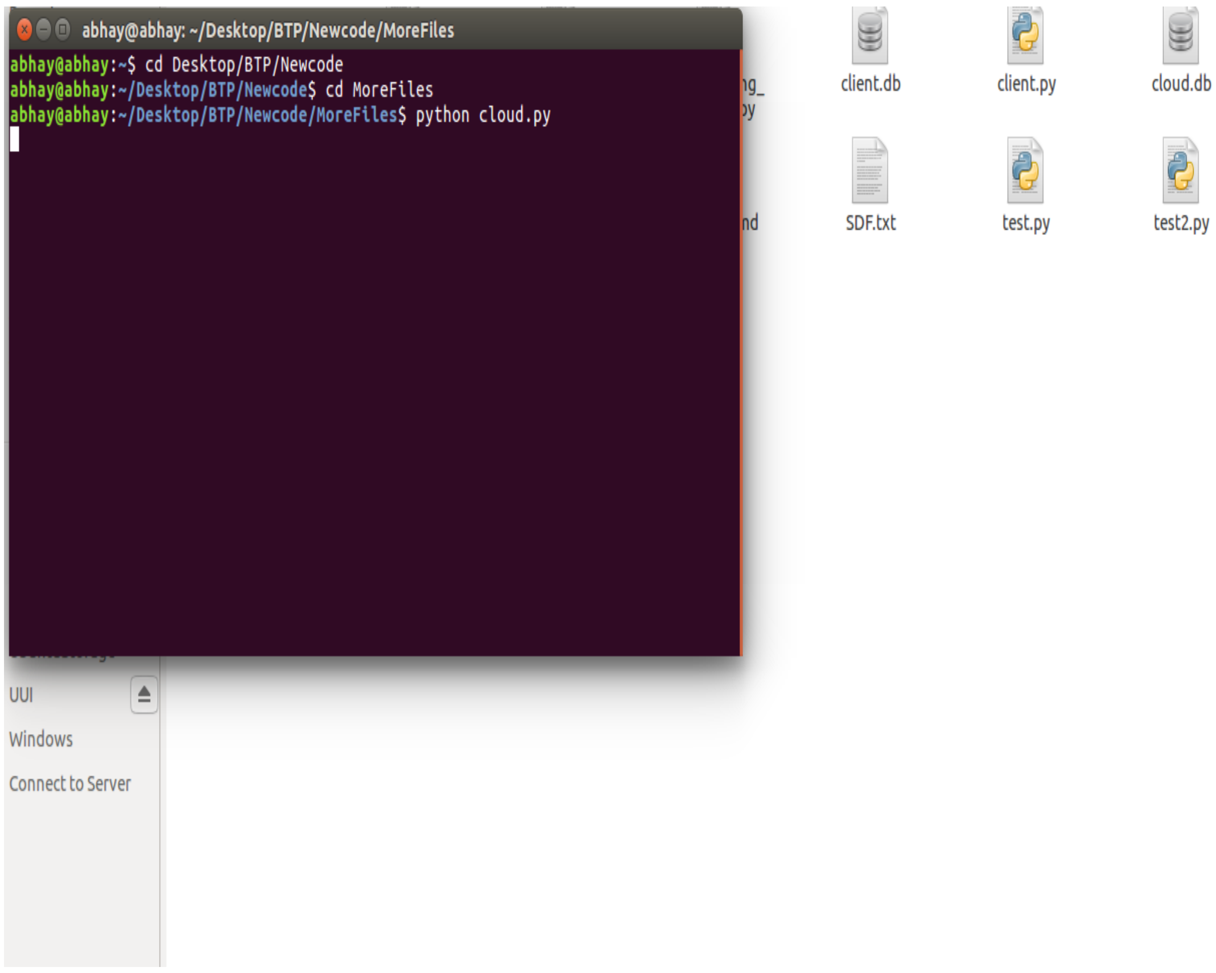
Implementation and Testing

Implementation and testing of One IoT protocol ver2.0 was done on a Ubuntu 16.04 operating system in Python 2.7.12.

All three different processes representing UTM Cloud, IoT node/Thing and Client run on the localhost but on different ports to simulate different IP addresses. OTU Cloud can be represented by a storage location on the machine.

Steps involved during implementation and testing are:

1. Firstly, the UTM Cloud server represented by Cloud.py is run. It is programmed to listen to the incoming requests from the client and to receive the heartbeat messages from the IoT nodes. It keeps a mapping of all Users and Things and verifies the Clients who try to get access to a particular thing. Also, it keeps an updated copy of the SDF for all the Things connected to it so that the latter can obtain an updated copy in case it operates during Safe mode. SDF files at UTM Cloud can be updated by getting them from the



```
abhay@abhay:~$ cd Desktop/BTP/Newcode
abhay@abhay:~/Desktop/BTP/Newcode$ cd MoreFiles
abhay@abhay:~/Desktop/BTP/Newcode/MoreFiles$ python cloud.py
```

FIGURE 5.1: UTM Cloud Running

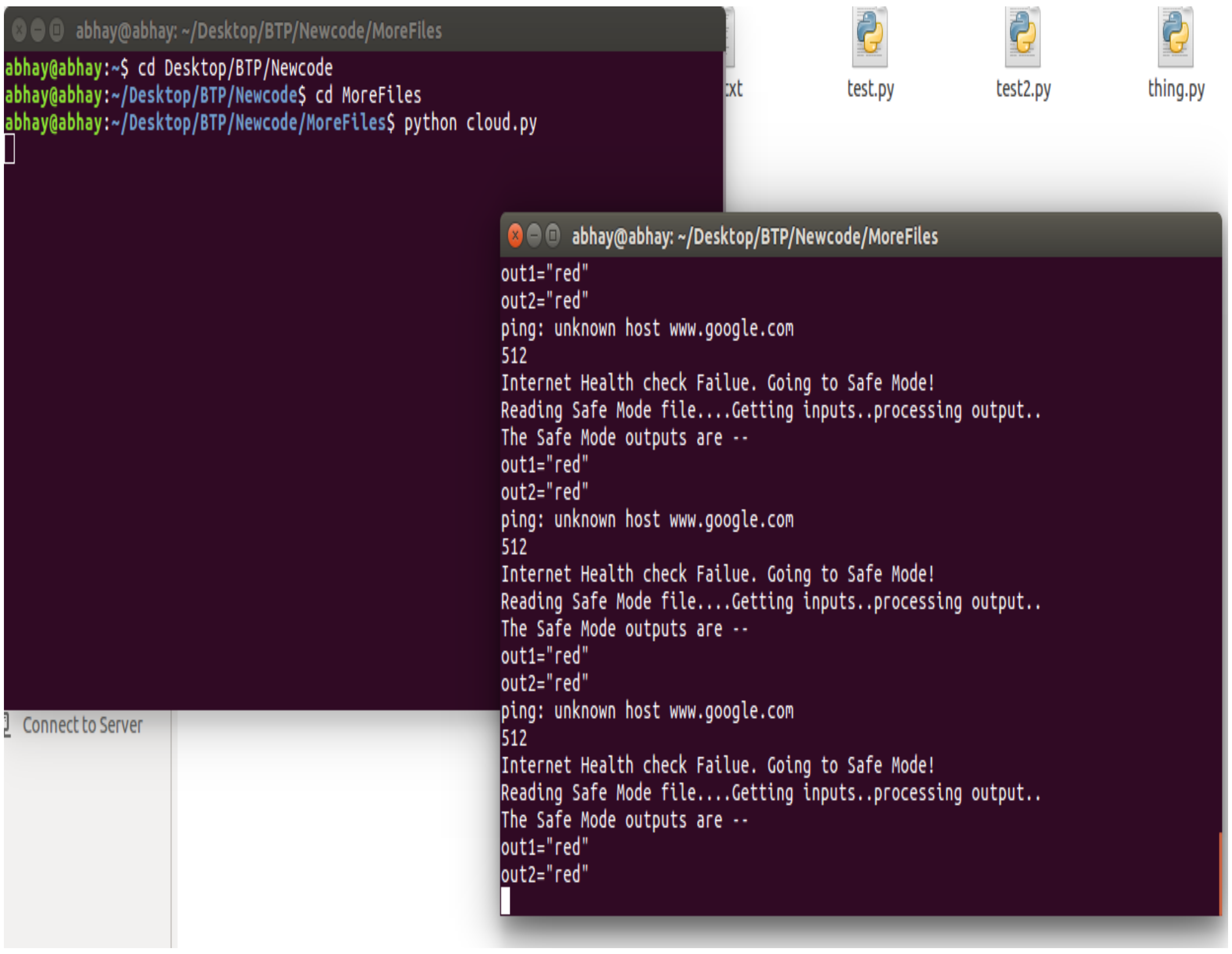
   OTU Cloud which can be a storage location on the local machine.

2. The IoT node represented by Fridge.py is run which executes the functions of Thing.py. The main functionalities of a Thing are represented in the Thing.py

file. It is programmed to continuously ping its IP address to the UTM Cloud and listen to the requests of the Client. However, it keeps on performing the Internet Health Check periodically to check the connectivity statusof the Internet and performs above functions only if Internet Health Check succeeds.

In this implementation, Internet Health Check is performed by pinging an almost all time up server like www.google.com. If the ping response is negative i.e the server is down, it is a failure, else a success.

Under Internet Health Check failure, Safe Mode is triggered i.e function SafeMode() is called. Under this mode, the program fetches the SDF for the device from the latter's local storage , reads it and get the safe output values to be actuated according to the given input sensor values. In case when there aren't any particular input sensor values, the default output values are actuated. Under


abhay@abhay: ~/Desktop/BTP/Newcode/MoreFiles
xt
test.py
test2.py
thing.py
Connect to Server
abhay@abhay: ~/Desktop/BTP/Newcode/MoreFiles

```
abhay@abhay:~$ cd Desktop/BTP/Newcode
abhay@abhay:~/Desktop/BTP/Newcode$ cd MoreFiles
abhay@abhay:~/Desktop/BTP/Newcode/MoreFiles$ python cloud.py
```

```
out1="red"
out2="red"
ping: unknown host www.google.com
512
Internet Health check Failue. Going to Safe Mode!
Reading Safe Mode file....Getting inputs..processing output..
The Safe Mode outputs are --
out1="red"
out2="red"
ping: unknown host www.google.com
512
Internet Health check Failue. Going to Safe Mode!
Reading Safe Mode file....Getting inputs..processing output..
The Safe Mode outputs are --
out1="red"
out2="red"
ping: unknown host www.google.com
512
Internet Health check Failue. Going to Safe Mode!
Reading Safe Mode file....Getting inputs..processing output..
The Safe Mode outputs are --
out1="red"
out2="red"
```

FIGURE 5.2: Safe Mode Operations under Internet Health Check Failure

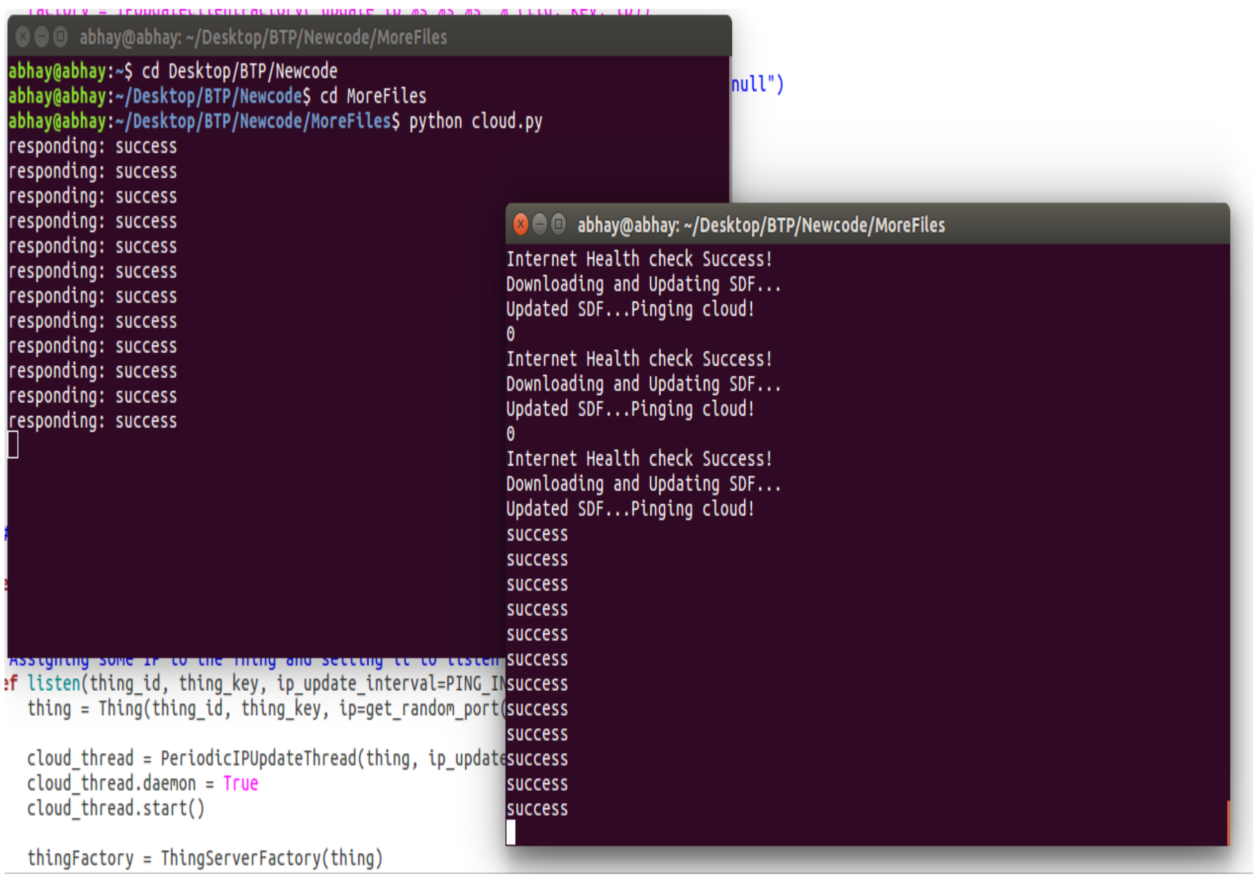
Internet Health Check Success, the thing keeps on pinging UTM cloud i.e Cloud.py and gets a ping success response.

Also, it keeps on updating its local storage of SDF i.e SDF.xml by getting the latter from the UTM Cloud storage which is actually a folder on the local machine representing the UTM Cloud's storage location.

3. The Client represented by Client.py is run. The client can send a message to available things like fridge.py, washing_ machine.py etc. In case, local IP of a Thing when Client is stale gets an updated IP from the UTM Cloud or Cloud.py.It can then send a message like "Hello" to a Thing and can get response from it as a "response of Hello".

The above implementation of One IoT Protocol ver2.0 works fine and is in accordance with the expected behaviour.

Similarly, the protocol can be tested on different machines representing different components i.e Thing, UTM Cloud, OTU Cloud and Client and shall work fine.



```
abhay@abhay:~$ cd Desktop/BTP/Newcode
abhay@abhay:~/Desktop/BTP/Newcode$ cd MoreFiles
abhay@abhay:~/Desktop/BTP/Newcode/MoreFiles$ python cloud.py
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
```

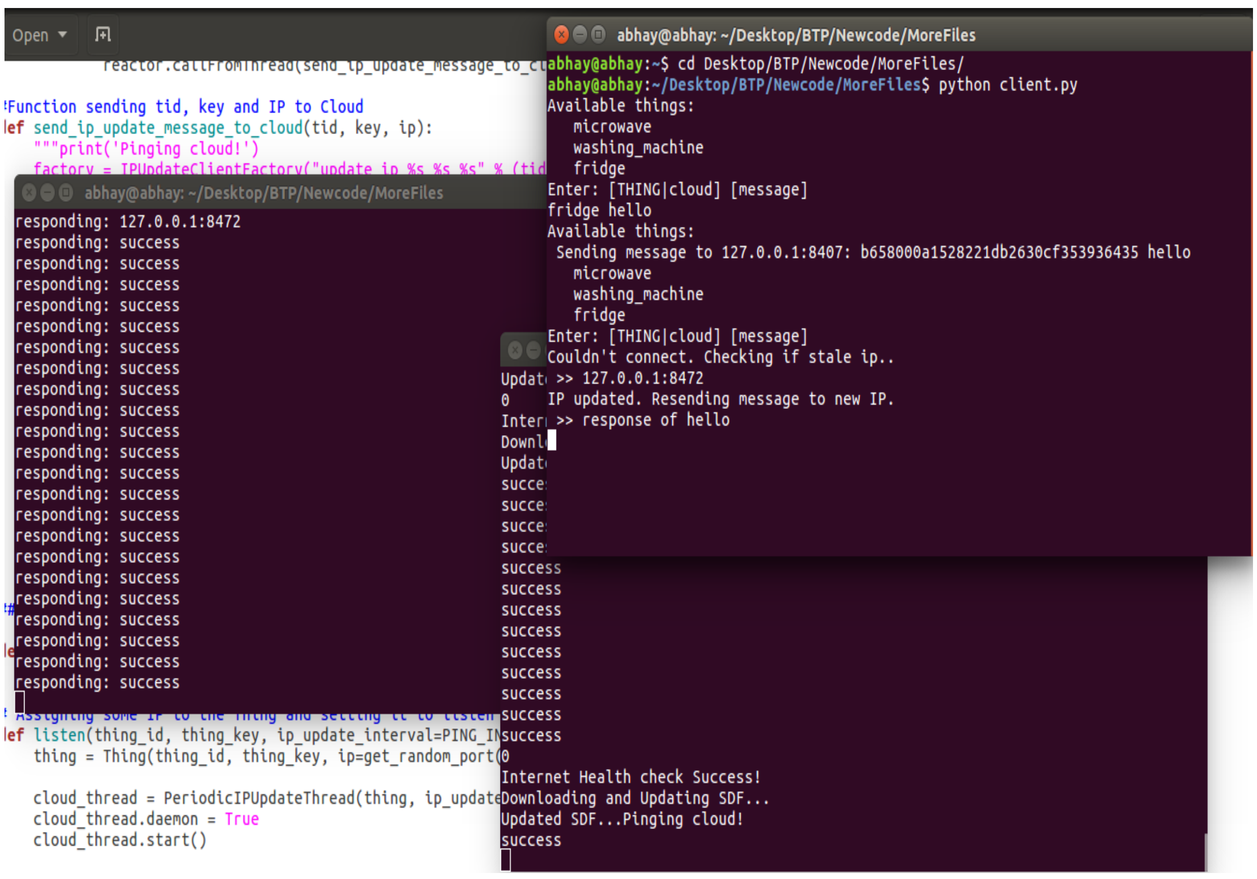
```
Internet Health check Success!
Downloading and Updating SDF...
Updated SDF...Pinging cloud!
0
Internet Health check Success!
Downloading and Updating SDF...
Updated SDF...Pinging cloud!
0
Internet Health check Success!
Downloading and Updating SDF...
Updated SDF...Pinging cloud!
success
success
success
success
success
success
success
success
success
success
success
success
success
```

```
def listen(thing_id, thing_key, ip_update_interval=PING_I
    thing = Thing(thing_id, thing_key, ip=get_random_port(

    cloud_thread = PeriodicIPUpdateThread(thing, ip_update
    cloud_thread.daemon = True
    cloud_thread.start()

    thingFactory = ThingServerFactory(thing)
```

FIGURE 5.3: Normal Mode Operations under Internet Health Check Success



```
abhay@abhay:~$ cd Desktop/BTP/Newcode/MoreFiles/
abhay@abhay:~/Desktop/BTP/Newcode/MoreFiles$ python client.py
Available things:
   microwave
   washing_machine
   fridge
Enter: [THING|cloud] [message]
fridge hello
Available things:
 Sending message to 127.0.0.1:8407: b658000a1528221db2630cf353936435 hello
   microwave
   washing_machine
   fridge
Enter: [THING|cloud] [message]
Couldn't connect. Checking if stale ip..
>> 127.0.0.1:8472
IP updated. Resending message to new IP.
>> response of hello
```

```
responding: 127.0.0.1:8472
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
responding: success
```

```
success
success
success
success
success
success
success
success
success
0
Internet Health check Success!
Downloading and Updating SDF...
Updated SDF...Pinging cloud!
success
```

```
#Function sending tid, key and IP to Cloud
def send_ip_update_message_to_cloud(tid, key, ip):
    """print('Pinging cloud!')
    factory = IPUpdateClientFactory("update ip %s %s %s" % (tid

def listen(thing_id, thing_key, ip_update_interval=PING_I
    thing = Thing(thing_id, thing_key, ip=get_random_port(

    cloud_thread = PeriodicIPUpdateThread(thing, ip_update
    cloud_thread.daemon = True
    cloud_thread.start()
```

FIGURE 5.4: Client Receiving Response from Thing

Thus, the protocol can even be implemented on a large scale network with multiple Clients and Things, Big UTM and OTU Cloud, all placed several kilometres apart but connected by internet and work seamlessly together.

Chapter 6

Conclusion and Future Scope

The One IoT Protocol ver2.0 proposed in the work seems to be very promising to deal with the challenges faced by One IoT Protocol ver1.0 and IoT in general. The root problem of all the challenges discussed before is the Internet outage since it leaves an IoT node without any external input or support from a node or a user agent. In today's scenario large number of physical devices are connected to the Internet thus leading to the formation of IoT ecosystems. In future, IoT is going to become an ubiquitous part of our life and so the modified protocol provides us with a solution to the problem of Internet outage which could lead to disruption of IoT functionalities..The modified protocol provides a Safe mode functionality to cope with the stated problem.

The Safe Mode for an IoT node can be Application Specific, Deployment Specific and Device Specific. One can always argue the need of such Safe Mode in the IoT ecosystems of developed countries where Internet Outage or Power failure is negligible. However, taking no precautions because of the unusual nature of the problem can lead to great losses. The best example is Northeast Blackout of 2003 which occurred for three continuous days having an adverse impact on the health-care sector. Thus, One IoT Protocol ver2.0 leads to the creation of a robust IoT ecosystem by taking into account various possible challenges and proposing ways to deal with them.

However, there are some further possibilities of improvements and modifications in the protocol. Right now, defined xml template of SDF covers two basic types of sensor inputs – range and pulse. There can be further additions of some more input types covering a wider variety of inputs. The format can be simplified and improved to make it easier for the OEMs to make SDF for their devices. This will also help in reading and invoking the SDF easily. Some smartness might also be added to the embedded system of a device to store previous logs of actions taken by it and actuate intelligent outputs using them under Safe mode operation. Other than these, there can be other improvements too, bounded only by creativity and imagination.

Bibliography

[1] *Thingworx Platform. [Online]. Available: http://www.thingworx.com.*

[2] *Open Interconnect Consortium, "The Open Interconnect Consortium and IoTivity," July 2015. [Online]. Available: http://openinterconnect.org/wp-content/uploads/2015/07/OICIoTivity White-Paper Final.pdf.*

[3] *Industrial Internet Consortium, "Industrial internet reference architecture," Industrial Internet Consortium, Tech. Rep., June 2015. [Online]. Available: http://www.iiconsortium.org/IIRA-1-7-ajs.pdf.*

[4] *D. Ministry of Communication and Information Technology India, "Draft policy on internet of things," Government of India, Tech. Rep.,2015.*

[5] *Gourinath Banda, Krishna Chaitanya and Harsh Mohan, "An IoT Protocol and Framework for OEMs to Make IoT-Enabled Devices forward Compatible" in 11th International Conference on Signal-Image Technology Internet Based Systems, SITIS 2015.*

[6] *IEEE, Towards a definition of the Internet of Things (IoT), May 2015.*

[7] *J. Oltsik, "The internet of things: A ciso and network security perspective," Cisco Systems, Tech. Rep., October 2014..*

[8] *FIRSTPOST.BUSINESS, "Internet of Things – An Indian Perspective", Jan 21, 2015. [Online]. Available: http://www.firstpost.com/business/internet-things-indian-perspective-2057171.html .*

Appendices

Appendix A

Internet Health Check

```
response = os.system("ping -c 1 www.google.com 2>&1 >/dev/null")

if response==0 or response==256:
        print "Internet Health check Success!"

elif response==512 || response==1:
        print 'Internet Health check Failue. Going to Safe Mode!'
```

To perform Internet Health Check, the above logic is used where www.google.com is pinged and then the return value is checked. Accordingly, success or failure of the process is obtained which decides the mode of the device.

Appendix B

Updating local copy of SDF at Thing

```
src = "CloudStorage/SDF.txt"
dst = "SDF.txt"

print "Internet Health check Success!"
print "Downloading and Updating SDF..."
copyfile(src,dst)
```

Upon Internet Health Check Success, local storage of SDF of a Thing is updated by using the logic defined. The device gets the updated copy of SDF by copying it from Cloud storage which is a different folder on the local machine.

Appendix C

Safe Mode Operations

```
filename = "SDF.txt"

def Safe_Mode():
        print "Reading Safe Mode file..."
        catch = 0
        fileobject = open(filename, 'r')
        print "Processing SDF..."
        print "The Safe Mode outputs are --"
        for line in fileobject:
          words = line.split()
          if catch==1:
             for word in words:
                if word!="<" and word!=">" and word!="actuator":
                        print word
                        catch = 0
                        break
            else:
               for word in words:
                 if word == 'modeno="default"':
                       catch = 1
        fileobject.close()
```

Upon Internet Health Check Failure, the device transits to Safe mode. In Safe mode operation, the device fetches the Safe mode Descriptor File SDF.xml from its storage, reads it and interprets it. Upon no sensor input values, as displayed in the above code, the device actuates the default output values.

Appendix D

Safe Mode Descriptor File(SDF) Format

Below is the predefined XML format for a SDF file of a Thing. The thing has total n+m sensor inputs. n of them are pulse type inputs(0 or 1) and m of them are range type inputs(lie in range). There are total k actuating outputs. The Safe Modes of operation for a Thing can be either Single or Multiple(p), both having Default mode of operation as well.

```
<safemode>
<!−−When the device has only 1 safe mode and a default safe mode−−>
<modetype type="single">
<mode modeno="default"> <!−−Default safe mode−−>
<!−−(n+m) sensor inputs. n are pulse type. m are range type.
Values of all not known−−>
<sensor inp1="nil" inp2="nil ".........inpnplusm="nil">
<!−− k actuating outputs, all having default values under no sensor
inputs−−>
<actuator out1="default1" out2="default2 ".......outk="defaultk">
</mode>

<mode modeno="1"><!−−Safe Mode No. 1−−>
<!−−n pulse type inputs with values as valp1, valp2...valpn. And m
range type inputs with values ranging b/w valr1 and valrm−−>
<sensor inp_pulse1="valp1"  inp_pulsen="valpn"  "valr1"<inp_range1
<"valr2"  "valrm"<inp_rangem<"valrmplus1">
<!−− k actuating outputs with values as valo1, valo2...valok−−>
<actuator out1="valo1" out2="valo2 ".......outk="valok">
</mode>

</modetype>

<!−−When the device has p no. of safe modes and a default safemode−−>
<modetype type="multi">

<mode modeno="default"><!−−Default safe mode−−>
<sensor inp1="nil" inp2="nil ".........inpnplusm="nil">
<actuator out1="default1" out2="default2 ".......outk="defaultk">
</mode>

<mode modeno="1"><!−−Safe Mode No. 1−−>
<sensor inp_pulse1="valp1"  inp_pulsen="valpn"  "valr1"<inp_range1
<"valr2"  "valrm"<inp_rangem<"valrmplus1">
<actuator out1="valo1" out2="valo2 ".......outk="valok">
</mode>

<mode modeno="p"><!−−Safe Mode No. p−−>
```

```
<sensor inp_pulse1="valp1"  inp_pulsen="valpn" "valr1"<inp_range1<
"valr2" "valrm"<inp_rangem<"valrmplus1">
<actuator out1="valo1" out2="valo2 ".......outk="valok">
</mode>
</modetype>

</safemode>
```