1.5em 0pt

JADAVPUR UNIVERSITY

MASTER DEGREE THESIS

Refurbishing Sensor Cloud Infrastructure for Efficient IoT Applications

---

A thesis submitted in fulfillment of the requirements for the degree of
Master of Technology in Distributed & Mobile Computing
in the

School of Mobile Computing And Communication

by
AKASH CHOWDHURY
University Roll Number: 001730501013
Examination Roll Number: M4DMC19015
Registration Number: 141110 of 2017-2018

Under the Guidance of
Dr. NANDINI MUKHERJEE
Professor, Department of Computer Science and Engineering
Faculty of Engineering and Technology
Jadavpur University
Kolkata-700032

2019

Declaration of Originality and Compliance of Academic Ethics

I hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, i.e. me, as part of Master of Technology in Distributed & Mobile Computing studies.

All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name	: AKASH CHOWDHURY
Class Roll No.	: 001730501013
Examination Roll No.	: M4DMC19015
Registration No.	: 141110 of 2017-2018
Thesis Title	: Refurbishing Sensor Cloud Infrastructure for Efficient IoT Applications

Signed:

Date:

To whom it may concern

This is to certify that the work in this thesis entitled "Refurbishing Sensor Cloud Infrastructure for Efficient IoT Applications" has been satisfactorily completed by Akash Chowdhury, University Roll Number: 001730501013, Examination Roll Number: M4DMC19015, Registration Number: 141110 of 2017-2018. It is a bona-fide piece of work carried out under my supervision at Jadavpur University, Kolkata-700032, for partial fulfillment of the requirements for the degree of Master of Technology in Distributed & Mobile Computing from the School of Mobile Computing And Communication, Jadavpur University for the academic session 2017-2019.

Dr. Nandini Mukherjee
Professor
Department of Computer Science & Engineering,
Jadavpur University
Kolkata-700032.

DIRECTOR
School of Mobile Computing
And Communication,
Jadavpur University
Kolkata-700032.

Prof. Pankaj Kumar Roy
Dean, Faculty of Interdisciplinary
Studies, Law and Management
Jadavpur University
Kolkata-700032.

Certificate of Approval

(Only in case the thesis is approved)

This is to certify that the thesis entitled "Refurbishing Sensor Cloud Infrastructure for Efficient IoT Applications" is a bona-fide record of work carried out by Akash Chowdhury, University Roll Number: 001730501013, Examination Roll Number: M4DMC19015, Registration Number: 141110 of 2017-2018, in partial fulfillment of the requirements for the award of the degree of Master of Technology in Distributed & Mobile Computing from the School of Mobile Computing And Communication, Jadavpur University for the academic session 2017-2019. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

_____

(Signature of the Examiner)
Date:

_____

(Signature of the Examiner)
Date:

Jadavpur University

Abstract

Faculty of Interdisciplinary Studies, Law and Management, Jadavpur University
School Of Mobile Computing And Communication

Master of Technology in Distributed & Mobile Computing

Refurbishing Sensor Cloud Infrastructure for Efficient IoT Applications

by
AKASH CHOWDHURY
University Roll Number: 001730501013
Examination Roll Number: M4DMC19015
Registration Number: 141110 of 2017-2018

In IoT, a major application scenario is environmental monitoring. To monitor any given environment, a Wireless Sensor Network (WSN) is needed. But, any standalone WSN possess resource constraints in terms of computation, communication, storage and power supply. Thus, WSN is integrated with the cloud computing technology to create a new infrastructure termed as the Sensor-Cloud. The Sensor-Cloud hosts huge number of sensor nodes, most of which reside behind NATs in a private network. A sensor node should not unnecessarily stream data to a cloud storage all the time. As this will lead to energy drain of the sensor node. To make the sensor nodes stream on demand, the sensor node must be acquired first. To do so, the Sensor-Cloud must be able to propagate an acquiring signal through the NAT of the sensor node. Thus, there must be a mechanism to establish a communication channel through which the acquiring signal can be propagated to the sensor node. In Sensor-Cloud, after acquiring a set of sensor nodes, when that set of sensor nodes stream data together to a sensor gateway it may happen that the bandwidth required at any time instant is higher than that available in the wireless medium. It may also happen at any instant of time that the required channel capacity exceeds the maximum available channel capacity of the access link connecting the sensor gateway to the cloud storage. To overcome this problem, the sensor nodes must be scheduled in such a way that the maximum bandwidth required in the wireless medium and maximum channel capacity required at the access link at any time instant can be minimized. Therefore, this thesis mainly focuses on solving the two aforementioned problems of a Sensor-Cloud by providing efficient solutions for each of them.

Acknowledgements

On the submission of "Refurbishing Sensor Cloud Infrastructure for Efficient IoT Applications", I wish to express gratitude to the School of Mobile Computing And Communication for sanctioning a thesis work under Jadavpur University under which this work has been completed.

I would like to convey my sincere gratitude to **Dr. Nandini Mukherjee**, Professor, Department of Computer Science & Engineering, Jadavpur University for her valuable suggestions throughout the project duration. I am really grateful to her for her constant support which helped me a lot to fully involve myself in this project and develop new approaches in the field of Internet of Things.

I would like to express my sincere, heartfelt gratitude to Mr. Sunanda Basu, Ph.D Scholar, Department of Computer Science & Engineering, Jadavpur University, Kolkata, for suggestions and guidance.

I would also wish to thank Dr. Punyasha Chatterjee, Director of the School Of Mobile Computing And Communication, Jadavpur University and Prof. Pankaj Kumar Roy, Dean, Faculty of Interdisciplinary Studies, Law and Management, Jadavpur University for providing me all the facilities and for their support to the activities of this research.

Lastly I would like to thank all my teachers, classmates, guardians and well wishers for encouraging and co-operating me throughout the development of this project. I would like to especially thank my parents whose blessings helped me to carry out my project in a dedicated way.

Regards,
AKASH CHOWDHURY
University Roll Number: 001730501013
Examination Roll Number: M4DMC19015
Registration Number: 141110 of 2017-2018
School of Mobile Computing And Communication
Jadavpur University

Signed:

Date:

Contents

List of Figures

List of Tables

List of Abbreviations

IoT	**I**nternet **O**f **T**hings
QoS	**Q**uality **O**f **S**ervice
WSN	**W**ireless **S**ensor **N**etwork
JSON	**J**ava**S**cript **O**bject **N**otation
M2M	**M**achine **T**o **M**achine
LAN	**L**ocal **A**rea **N**etwork
XML	**E**xtensible **M**arkup **L**anguage
NAT	**N**etwork **A**ddress **T**ranslation
GSP	**G**lobal **S**ervice **P**rovisioning
SPPS	**S**ervice **P**rovisioning using **p**re-signaling
RNT	**R**equester side **N**AT **T**raversal
SSP	**S**ecure **S**ervice **P**rovisioning
UPnP	**U**niversal **P**lug and **P**lay
NAT-PMP	**N**AT **P**ort **M**apping Protocol
PCP	**P**ort **C**ontrol **P**rotocol
STUN	**S**ession **T**raversal **U**tilities for **N**AT
TURN	**T**raversal **U**sing **R**elays around **N**AT
ICE	**I**nteractive **C**onnectivity **E**stablishment
CAN	**C**ontext-**A**ware **N**AT **T**raversal
GPA	**G**radual **P**roximity **A**lgorithm
ANTS	**A**dvanced **N**AT Traversal **S**ervice
P2P	**P**eer **T**o **P**eer
NAPT	**N**etwork **A**ddress **P**ort **T**ranslation
SOHO	**S**mall **O**ffice **H**ome **O**ffice
ISP	**I**nternet **S**ervice **P**rovider

List of Symbols

cp	Client payload in probe message
sp	Server payload in reply message
ρ	Random number
δ	Variable controlling delay decision
d	Decided delay
p	Period of a sensor (in number of time slots)
T	Number of transactions to be done in a p
f	Size of each transaction (in number of time slots)
BW	Number of available frequency channels
Ch_{max}	Maximum required channel capacity (in bits per time slot)
L	LCM of all the periods of the sensor nodes
θ_{ijk}	Offset for k^{th}transaction (in number of time slots) in j^{th} period of i^{th} sensor node

Chapter 1

Introduction

1.1 Sensor-Cloud Infrastructure

Internet of Things (IoT) is a technological concept that aims to connect huge number of devices via the Internet to manifest an infrastructure that is necessary to provide any service or support any given application scenario. One of the most important component that is needed to support any large scale IoT infrastructure is sensor nodes.

A major application scenario of the IoT paradigm is environmental monitoring [2]. Through environment monitoring huge amount of data is generated that is required for analysis in order to reach to certain conclusions about the state of the environment [3]. Thus, for monitoring of any given environment a network of sensing nodes are required which forms a Wireless Sensor Network (WSN). Snowfort [4],is an open source WSN for data analytics in infrastructure and environmental monitoring.

Any typical WSN comprises of a set of sensor nodes that is deployed in a particular environment for monitoring that environment continuously by sensing the various environmental parameters and stream those sensed data to a destination, say a cloud server. Any WSN gets the capability of being distributed all over the environment and have total coverage of the environment by the means of self-regulated sensor nodes. These sensor nodes can be programmed how to get activated and start sensing, how and when to stream the sensed data and when to again become dormant. A sensor node becomes dormant or gets into a very low power state because it is important to save energy or battery life as the nodes are battery powered and not connected to any external power source.

The sensor nodes of a WSN can sense data about different environmental parameters like, temperature, humidity, atmospheric pressure, pollution, speed of wind, vibration and motion and many more [5] [6].

These sensor nodes are equipped with -

a) A radio transceiver or some means for wireless communication,

b) A small microcontroller,

c) An energy source. In maximum cases, cells/batteries are used as power source.

Any sensor node is comprised of three basic modules [7] -
a) Sensing module,
b) Processing module,
c) Communication module

With the advancements in computer networks and sensing devices, it has gradually become very easy nowadays to deploy a WSN in any environment. Growing demands for smart IoT infrastructures and the required capability prevalence in WSN(s) to provide it has increased the utilization of WSN(s) in many fields like healthcare [8], military and defence services like target tracking and surveillance [9] [10], governmental projects for handling environmental issues like natural disaster management [12], exploration of hazardous environments [11] and seismicity sensing [13].

But there are many challenges and disadvantages of sensor networks and they are as follows.
a) Communication range of the sensor nodes of a WSN are very short.
b) Issues regarding security and privacy.
c) Issues related to reliability of the WSN.
d) Mobility issues.
e) Power constraints
f) Storage constraints
g) Computational constraints
h) Bandwidth availability constraints

On top of this a design of the WSN also varies according to the environment in which it will be deployed. It also depends on the type of monitoring application it is or the parameter (temperature or seismicity) that is to be monitored for the environment.

The size of the environment is also a dependency. For smaller areas less number of sensor nodes are required to have total coverage but for larger areas the number of sensor nodes required is quite large. All these issues degrade the performance of a WSN thus degrading the quality of service (QoS) [14].

To strengthen the WSN(s), it was incorporated with the Cloud Computing technology. Cloud computing provided two basic things to the WSN – a) computational efficiency, by providing highly scalable computational units and software services, b) better control of the sensor nodes of the deployed WSN as the cloud also provides better communication bandwidths and gives provision to host real time interactive mobile applications for on demand remote supervision of the WSN [15]. The integration of the WSN and cloud computing gave birth to the new cloud, termed as the

Sensor-Cloud.

A Sensor-Cloud infrastructure [16] is meant to manage the sensor nodes of a WSN over the cloud. When WSN got integrated to the cloud infrastructure several problems faced by the WSN like computational constraints, communication constraints and storage constraints got relaxed.

Huge amount of data could be gathered and stored in the cloud to perform analytics on the data in order to make valuable conclusions about the status of the monitored environment. With high storage capacity the QoS of the WSN services also got uplifted as the monitoring can be done constantly and data can be gathered at a very high rate. Sensor nodes, as a part of a WSN deployment, can now send data directly to the cloud through a common gateway. The gateway interfaces the WSN and the cloud storage [17].

Sensor-Cloud has many real life applications like in irrigation, monitoring of disaster and environments, health monitoring and telemetric. In healthcare senario, a patient with cardiovascular disease, can be constantly monitored by an application hosted in the cloud that connects to the deployed WSN to continuously collect data of different medical parameters like blood sugar level, sleeping patterns, weight, heart beat rate, pulse rate and many more. The patient himself or his guardians can monitor the parameters continuously through a mobile application that accesses the data storage in the cloud where the sensor nodes of the WSN streams and stores data [18].

1.1.1 Architectural View of Sensor-Cloud

IntelliSys defined [19] [20] SensorCloud as –

> "*An infrastructure that allows truly pervasive computation using sensors as an interface between physical and cyber worlds, the data-compute clusters as the cyber backbone and the internet as the communication medium.*
> "

In [21], the definition of Sensor-Cloud provided by MicroStrains is stated as –

> "*It is a unique sensor data storage, visualization and remote management platform that leverage [sic] powerful cloud computing technologies to provide excellent data scalability, rapid visualization, and user programmable analysis. It is originally designed to support long-term deployments of MicroStrain wireless sensors, Sensor-Cloud now supports any web-connected third party device, sensor, or sensor network through a simple OpenData API.*"

A problem with a standalone sensor network is that the application using the sensor network keeps sole authority of using the sensor network as a single user. Suppose for example, there is a WSN W deployed for serving application A. In this case application A holds the sensor network W and the sensed data in such a way that no other application, say application B, can share the resources of the sensor

network *W*. This causes massive wastage of sensing resources available in *W* as now application *B* needs to have a sensor network of its own causing very low resource utilization. Even when application *A* is not accessing the resources of the sensor network *W*, application *B* cannot access the free resources available at the sensor network *W*. If application *B* needs the same sensed data of application *A*, there is no provision for application *B* to access the same data already sensed by the sensor network *W* for application *A*.

Thus, to facilitate sharing of sensing resources, in Sensor-Cloud infrastructure the physical sensors are virtualized on a cloud platform to create virtual sensors. These virtual sensors get provisioned for multiple WSN applications. A virtual sensor can actually be either a single physical sensor node or a group of physical sensor nodes at the physical layer. Whereas a physical sensor node can be a part of more than one virtual sensors. With sensor virtualization, any two virtual sensors serving two independent WSN applications can actually share physical sensor nodes of a single WSN. Thus, the concept of virtual sensors uplifts the resource utilization of the physical sensing nodes and thus helps to create more innovative, efficient, scalable and diverse WSN applications sharing the same physical sensor nodes.

From a Sensor-Cloud, a requester when requests for a type of sensing service, the Sensor-Cloud investigate its available resources, finds the suitable ones and provisions an appropriate virtual sensor with the required specifications. The requester does not need to think about the infrastructural level details of how the virtual sensor is providing the required services or what are the physical sensor nodes that got virtualized to create the provisioned virtual sensor. In such a scenario it can be realized that a particular physical sensor node can be parts of more than one virtual sensor node [22]. The virtual sensors after being created are continuously monitored by the Sensor-Cloud in order to destroy the unwanted instances of virtual sensors thus increasing QoS of the services provided by the Sensor-Cloud by proper utilization of the physical resources [23]. To choose the appropriate sensor nodes according to the specification of the required services (e.g. type of sensors required, amount of data required, etc.) there exists a mechanism called publish/subscription [33] mechanism that helps to find and select the appropriate sensor nodes [34].

The most demanding sensing scenario in IoT healthcare services is continuous monitoring of the patients in which the sensor nodes continuously sense health parameters and send to the cloud gateway for storage. These sensed data can be continuously monitored by a smart application and according to the parameter values decisions can be taken automatically by the application regarding how to treat the patient or to contact any medical personnel or hospital to reach the patient if immediate medical assistance is required. Such medical services ask for smart WSN(s) and efficient computation of decision making algorithms by the cloud based on the sensed data sent to the cloud, in a completely automated structure involving no third-party intervention [24].

In figure 1.1, an overview of the Sensor-Cloud is depicted.

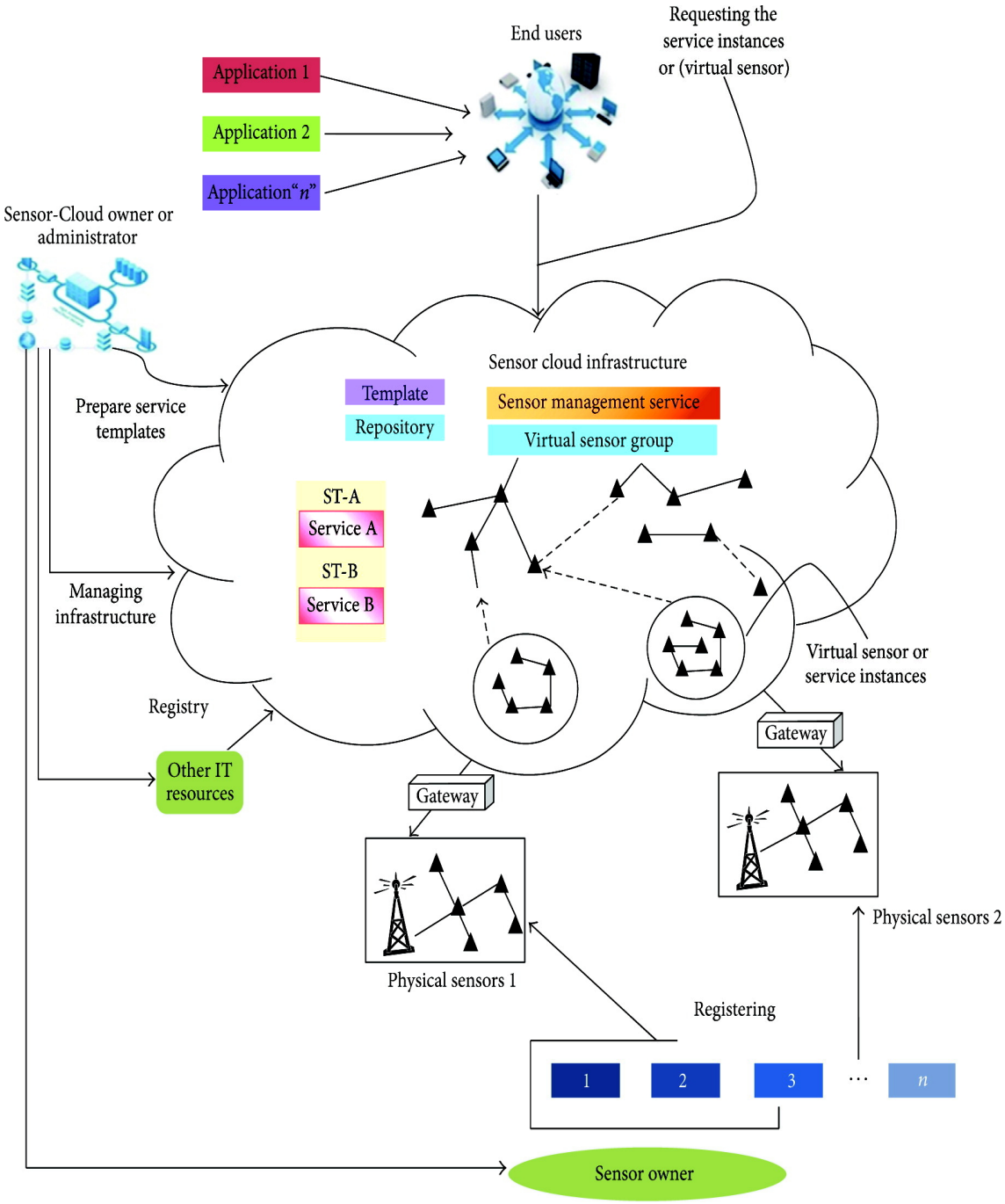


FIGURE 1.1: Overview of the Sensor-Cloud [1]

Sensor-Cloud infrastructure makes the virtual sensor available to the users in such a manner that the virtual sensors act exactly as an integral part of the IT resources like disk storage, main memory and processor [35]. Users when use a virtual sensor of the Sensor-Cloud infrastructure only needs to be concerned about the status of the virtual sensor and not about the physical status of the actual physical sensors from which the virtual sensor has been procreated.

In a Sensor-Cloud infrastructure, the sensor nodes are registered by the sensor node owners free of cost. When not willing to keep the sensor nodes as a part of the Sensor-Cloud infrastructure the owners can even deregister the sensor nodes at any time without any cost. After the sensor nodes are registered to the Sensor-Cloud infrastructure, they go through certain processes to become operational. From the available sensor nodes in the Sensor-Cloud infrastructure, different templates for service instances comprising a virtual sensor or a group of it are created so that the users can avail the services provided by the Sensor-Cloud infrastructure. The various components of the Sensor-Cloud infrastructure are as follows.

1.1.1.1 Resource Host

Resource Host are logical hosts that are remotely located somewhere in any organization, institution or environment.These logical hosts have a pool of remote resources that gets provisioned on demand. The Resource Host consist of two main parts - a) physical sensors, and b) a sensor coordinator.

- *Physical Sensor* – Physical sensors are the devices that actually sense and accumulate data but are not capable of transmitting the data. The physical sensors may not possess independent power source (eg. batteries). Multiple sensor may share the same power resource and get activated when attached to any power source. Each sensor have an unique id for identification.

- *Sensor Coordinator* – The sensor coordinator does the data transmission. It also acts as the power source for multiple sensors but do not have sensing capabilities of its own. It is a non-sensing resource with a unique id for identification. There can be a) globally identifiable coordinator, and b) locally identifiable coordinator. While the globally identifiable coordinator can be communicated directly for the cloud server, the locally identifiable coordinator is communicated through a sensor gateway. Sensor coordinators are made available with on board power supply.

  For data transmission, the coordinator is equipped with either a Wi-Fi module or GSM module or a USB module. The sensor coordinators according to their status of local or global coordinator, communicates with the cloud server either directly or via the sensor gateway.

  There are three main ways in which the physical sensors are attached to the sensor coordinator.

  1. *Fixed* – This type of attachments depicts the sensors that can never be plugged in/out of the sensor coordinator. For example, like TelosB, Iris, MICA2 that are located at specific positions. For example, multiple health sensors attached to a patient in motion.

2. *Mobile* – This type of attachments involve sensors to be connected to the coordinator either through wired or wireless means. There is mobility support for both the sensor and cloud.
3. *Variable* – There is a coordinator located at any fixed location. Multiple sensor are connected to the coordinator and those sensors are supposed to be attached to multiple individuals in a time slotted manner.

- *Sensor gateway* – These devices receive data from the locally unidentifiable coordinators and forwards the same to the cloud server. For efficient transfer of data the sensor coordinators make intelligent routing decisions based on the available network topology [40]

In figure 1.2, the various components present in the architecture of the Sensor-Cloud is depicted.

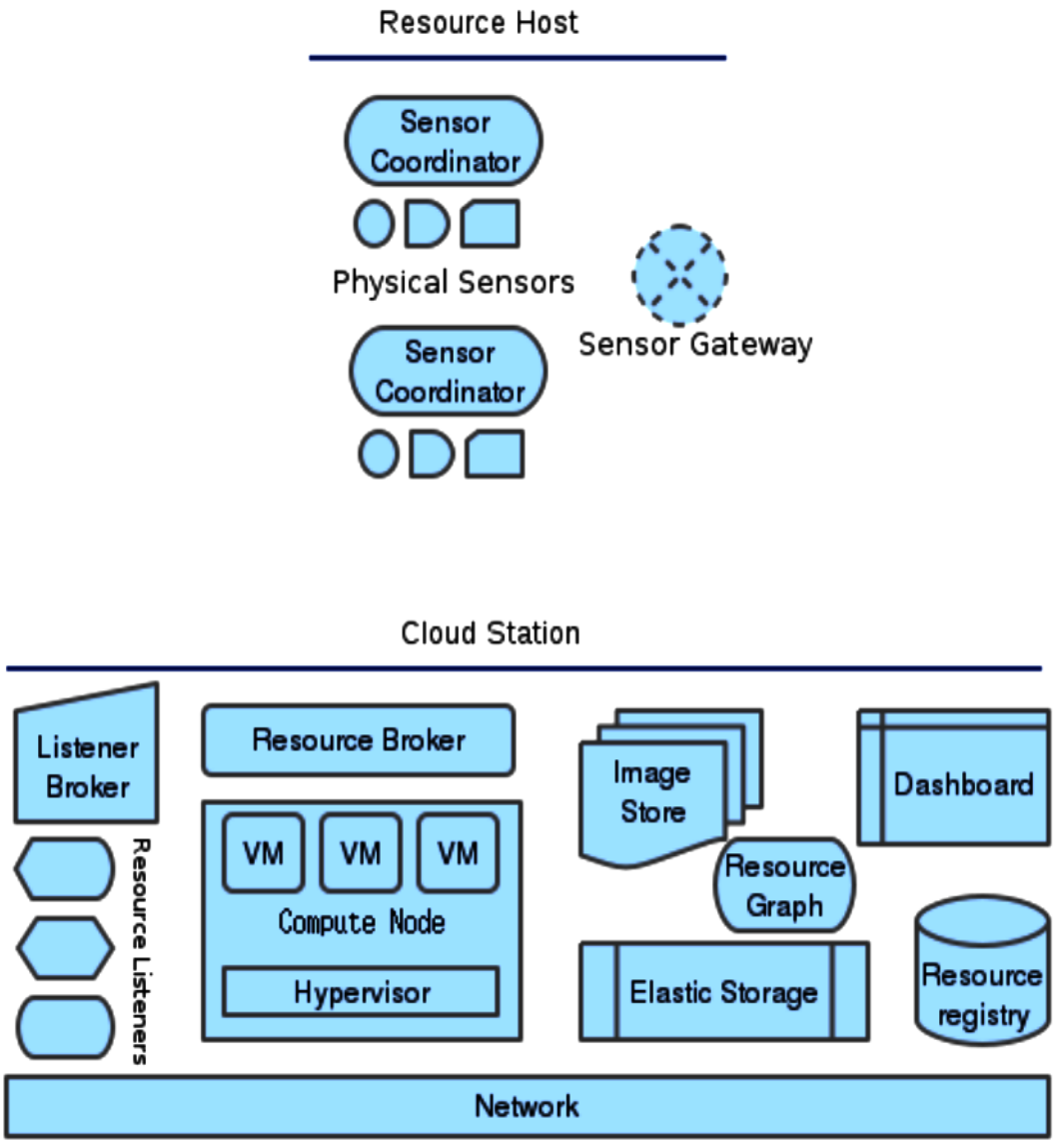


FIGURE 1.2: The architecture of a Sensor-Cloud

1.1.2 Advantages of Sensor-Cloud

With the integration of Cloud and WSN, the new paradigm of Sensor-Cloud were able to provide better QoS in terms of agility, reliability, portability, real-time access and control, scalability and flexibility. In the following, the advantages of the Sensor-Cloud have been enlisted which brought massive success to the concept of Sensor-Cloud.

1. *Analytics* – The Sensor-Cloud infrastructure possess huge resources of vast networks of WSNs to gather sensed data, enormous repositories of high-end computational units and fast communication networks. These infrastructural assets provided huge support to users willing to perform analytics of sensor data for creation of any smart IoT application [35].

2. *Scalability* – Any user can scale his/her own WSN deployment to any extent by asking for the service from the Sensor-Cloud platform. Earlier when there was no cloud computing integration, a user had to invest a huge amount of capital for buying all the equipments needed to extend the WSN [36].

3. *Collaboration* – Sensor-Cloud infrastructure has the capability of sharing huge amount of sensed data, gathered from the collaboration of huge number of physical sensor nodes, among multiple organizations with different WSN applications that requires sensed data.

4. *Visualization* – Sensor-Cloud also provides APIs for visualization of the sensed data that is stored in the cloud and retrieved as per queries. These APIs help to understand better the future aspects of any situation depending on the sensed data [37].

5. *Free Provisioning of Increased Data storage and Processing Power* Large-scale IoT application data gathered form vast WSNs can be stored easily in cloud storage. Storage and management of the huge amount of sensed data has become very easy with the presence of the ubiquitous cloud resources in the Sensor-Cloud [38].

6. *Dynamic resource provisioning* – The users of the Sensor-Cloud can access the Sensor-Cloud platform from any where and at time on demand. The users can even provision new resources on demand [38].

7. *Multi-tenancy* – Multi-tenancy refers to the model of operation in which multiple intances are integrated to create new and innovative services. Cloud platform available in Sensor-Cloud infrastructure makes it very easy to integrate services provided by multiple tenants [36].

8. *Automation* – Automatic delivery of resources or provisioning of services highly improved the resource acquisition time or service delivery time.

9. *Flexibility* – Sensor-Cloud facilitates the acquiring and releasing of services anytime on demand. The sensor nodes are made to serve multiple application of a single user of single application of multiple users, in any extent.

10. *Low Response Time* – Due to the vast high speed routing networks available in the cloud, the WSN applications have become capable of providing real-time services with minimal latency [39].

1.1.3 Applications of Sensor-Cloud

In this section, various Sensor-Cloud platforms and applications prevailing in the contemporary world have been enlisted as the following.

1.1.3.1 Various Sensor-Cloud Platforms and software APIs

1. *Nimbits* – It is free platform available as a social service to gather sensed data and share them though the cloud. It is an open-source platform for IoT and is capable of providing high-end data processing services. It supports data points for diverse input formats like JSON, GPS, XML and text-based [41].

2. *Pachube* – It is a cloud based service for supporting IoT with an available infrastructure which is scalable enough to let the users configure different IoT services and applications. It is one of the oldest online database service providers. It facilitates different IoT service provisioning, storage and sharing of of real-time sensed data from various environments. Pachube also provides an easy API to manage the stored sensed data via a very interactive website.

3. *IDigi* – It is a machine-to-machine (M2M) platform that facilitates building of low cost, secured and highly scalable products to bind te enterprise applications and the remotely situated device assets together. Irrespective of the location of the remote sensor nodes and ability to reach it, *iDigi* platform provides a remote device connectivity software for simplified connectivity with remote devices and their association with enterprise applications [43].

4. *IoT ThinkSpeak* – It is another open source IoT applications that provides an API for efficient storage and retrieval of data from various device assets. It may use any means of LAN or HTTP services over the Internet for communication with the device assets and others [44].

1.1.3.2 Various Sensor-Cloud applications

In this section, there are many emerging Sensor-Cloud application present that have been discussed. Various application scenarios of Sensor-Cloud has been depicted in the following.

1. *Ubiquitous monitoring in Healthcare* – Various wearable sensor nodes like accelerometer, proximity sensors, ambient light sensor, temperature sensor and blood sugar detectors can be used to procreate an application for monitoring the health of a patient [45].

2. *Monitoring of Tunnels* – The amount of light present in the tunnel can be monitored by the means of smart WSNs to avoid accidents inside the tunnel or below long and dark bridges. Distributed and continuous sensing of light levels inside the tunnel will help to create an adaptive lighting system inside the tunnel that will provide lights when the tunnel is sensed to be dark and put off the light when there is enough presence of light to avoid accidents in the tunnels [51].

3. *Wildlife Monitoring* – Sensor-Clouds is very efficient to track animals in a forest and detect forest fires [47]. It helps to continuously monitor endangered species and atmospheric parameters of the forests and wildlife sanctuaries.

4. *Environment monitoring for Emergency and Disaster Detection* – Disasters like earthquakes and volcanic eruptions can be detected beforehand by continuously monitoring them with the help of a smart WSN comprising of sensors like strain sensor, barometric sensors, accelerometers, vibration sensors, temperature sensors, light and image sensors [48]. The sensor nodes deployed in any environment can be used in a shared mode for different monitoring applications running in that environment.

5. *Agriculture and Irrigation Control* – Sensor-Cloud is capable of serving in the field of agriculture as well. For monitoring the crops a *field server* can be developed that consists of camera sensors, temperature and humidity sensors, CO_2 concentration sensors, soil moisture sensor. The server can continuously monitor the field to keep the farmer notified about the health of the crops [49].

6. *Earth Observation* – In [50] several GPS stations have been connected through a sensor grid to collect GPS data and perform analytics on them. This GPS data would help to deal with natural calamities like volcanic eruptions, cyclones, Tsunamis and earthquakes.

7. *Transportation Applications* – Traffic can be tracked continuously and the sensed data can be sent to cloud server for processing, in order to make decision on maintaining the traffic. Applications can be developed to notify vehicles of traffic jam occurred several kilometers away so that the driver can change the route to reach the destination [51].

1.2 Motivation

In the previous section, a brief introduction to Sensor-Cloud Infrastructure has been provided. From the entire aforementioned study, there are two problems that the Sensor-Cloud infrastructure faces. The first problem that has been realized is related to the scenario when the sensor nodes are required to stream data to the cloud storage as per the requirements of the monitoring application.

In the Sensor-Cloud there can be many users those request for sensor services whenever needed. When a sensor node is not required to stream data, it stays in a low power state to save its battery backup while deployed as a part of a WSN. Whenever a request for a particular service is received at the Sensor-Cloud, it starts finding the appropriate sensor node which has the capability of streaming data according to the requirements of the user.

On finding an appropriate match the Sensor-Cloud infrastructure needs to activate the sensor node or the group of sensor nodes (*as per the service requirements*) so that they get back to active state from sleep state and start streaming sensed data. In most cases the sensor nodes are residing in a private address space or in a private network and thus do not possess any public IPv4 address. It is quite evident that the range of IPV4 addresses are not enough to provide all the sensor nodes with unique public IP addresses. Thus, to extend the usability of the IPv4 range, the concept of Network Address Translation (NAT) was introduced. By network address translation, multiple nodes residing in a private domain behind a gateway or router, can communicate with nodes residing in the external network or Internet by sharing a single public IPv4 address. Due to the presence of NAT, the sensor nodes residing behind it can only possess a private IPv4 address but no unique publicly addressable IPv4 address which is required by any cloud server willing to set up a communication channel to the sensor node as and when required. If the sensor nodes residing behind NAT cannot be addressed publicly then it becomes a serious issue for the Sensor-Cloud Infrastructure to propagate the activating signal to the required sensor nodes.Thus there must be a suitable NAT traversal technique to solve this problem of waking up the required sensor nodes residing behind NATs. This is a very important problem that needs to be solved for a Sensor-Cloud infrastructure in order to let it collect sensed data from the available sensor nodes and make it available to the user. Otherwise to make the sensed data available at any time, the sensor nodes will be required to perform non-stop data streaming. This will be highly power consuming for the sensor nodes as they are power constrained devices, having cell/battery as the only power source.

Now, when in active state the sensor nodes starts streaming data to a common gateway, the total traffic generated in a time instant can be either high, average or low, depending on the schedule of the sensor nodes. If at any time instant all the sensor nodes transmit data together then the bandwidth required at the wireless

medium will be very high, whereas if few sensor nodes transmit at that time slot then the bandwidth required will be low.

Now if the sensor nodes send data randomly, obeying any random schedule, then it may happen that at any time instant bandwidth required is not available at the wireless medium. It will be a wastage or under-utilization of the available bandwidth if the bandwidth required for the worst case, in which all the sensor nodes transmit data, is made available at the wireless medium. Thus, to avoid wastage of available bandwidth there must be some means to determine the minimum bandwidth that can be required at the wireless medium. The channel capacity of the access link that propagates the sensed data to the cloud server from the gateway will also get affected in the same way as the bandwidth in the wireless medium. Thus, the minimum channel capacity required at the access link must also be determined by some means so that only the required amount of channel capacity is made available at the access link.

Thus, there exist two problems which are needed to be addressed and solved by some means for having better QoS for the Sensor-Cloud infrastructure.

1.3 Objective

The two problems discussed in the previous section must be solved for better implementations of the Sensor-Cloud.

The thesis thus focuses on two problems. The first problem is propagating the activating signal to the sensor nodes residing behind NATs. The problem that should be addressed is as follows.

A sensing device (client) is behind a NAT and a publicly addressable server needs to signal the client when it wants to execute some operation on it. The client is unaware when it will be signaled from the server. Being inside a NAT the client cannot be directly addressed by the server. Repeated polling from the client will drain its battery. Therefore, an energy efficient NAT traversal solution is required that minimizes the client's overhead while providing a consistent channel of communication that the server can use to wake up the client as and when required. This channel should be maintained by the cloud server and not by the sensor device. The sensor node must only assist in the establishment of the channel as the sensor nodes are energy constrained devices.

For the second problem discussed earlier, there is need to schedule the sensor nodes of a WSN in such a way that at any time instant the maximum bandwidth required at the wireless medium and the maximum channel capacity required at the access link get minimized. This will in turn prohibit wastage of resources and simultaneously decrease establishment cost and maintenance cost of the WSN. Thus,

regarding this problem the objective is to find a schedule that would lead to minimum bandwidth and channel capacity requirements in the wireless medium and access link respectively.

1.4 Contribution

In chapter 3, a communication scheme is proposed to solve the problem of activating sensor nodes on demand. In the proposed scheme sensor nodes as clients communicate with the cloud server through their NATs repetitively until an energy-efficient channel is established. This channel is maintained by the cloud server by sending periodic keep-alive packets to the sensor node.

In chapter 4, the second issue discussed in the section **??** has been addressed. The issue has been realized to have two possible variations. Each of the variations have been studied thoroughly and for each variation a polynomial-time heuristic algorithm is proposed in chapter 4. The working methodologies and results of the algorithmic simulations are also shown comprehensively at the end of chapter 4.

1.5 Organization of the Thesis

The remaining chapters of the thesis are organized as follows. In chapter 2, the related works of the two problems discussed in section 1.2 are discussed in detail. Then in chapter 3, a communication scheme is proposed to solve the problem of activating sensor nodes on demand. The problem is described thoroughly and then the scheme is described in detail. In chapter 4, solutions for the second problem mentioned in section 1.3 are discussed in detail. For each variation of the problem an algorithm and a pseudo code is provided for better understanding. At the end of the chapter, results obtained in case of both algorithms are discussed. Chapter 5 states the conclusion obtained from the entire work and some future aspects related to the works done are depicted.

1.6 Summary of the chapter

This chapter introduces the topic of this thesis. A detailed background of the Sensor-Cloud infrastructure has been depicted in the beginning of the chapter. Then in section 1.2, the problems that can occur in case of the Sensor-Cloud infrastructure are discussed in detail. Then in section 1.3, the main objectives regarding the solution of the two problems are stated. In section 1.4, the contributions done towards solving the problems are discussed and in section 1.5 the organization of this entire thesis has been depicted thoroughly.

Chapter 2

Literature Survey

In this chapter, the related work with respect to both the problems discussed in the previous chapter -

1. Communicating with a device residing behind a NAT from a public cloud server, and
2. Minimization of Bandwidth requirements of the wireless interface and channel capacity of the wired access link, while sensor nodes are streaming data together after being activated from the public server.

have been discussed. This chapter is divided into two sections where section2.1 discusses the various NAT traversal techniques through which devices behind residing behind NAT in different application scenarios communicate with each other and section 2.2 discusses about the various sensor-cloud infrastructures and various scenarios in which there are needs of scheduling sensor nodes.

2.1 Various Network Address Translators traversal techniques

When an internal host in a private network sends a connection request to an external host in the public network, the reply packet from the external host needs to traverse the NAT in order to reach the client. The technique to successfully traverse a NAT is termed as NAT traversal technique. There are different situations when NAT traversal is required. One situation can be when the communicating hosts are behind different NATs. Another situation can be when both the communicating hosts are not behind NATs. The different categories of NAT traversal are enlisted below.

2.1.1 Categories of NAT traversal techniques

There are many NAT traversal techniques available for different application scenarios. Based on the support provided by combinations of requester, service, universally addressable infrastructure nodes and application scenarios, the NAT traversal techniques are categorized into four classes [58].

1. *Requester side NAT traversal* (RNT) - In this class of NAT traversal, the requester or the client side supports NAT traversal only. It gives support to those clients

that has active participation in establishment of the connection but still faces problems due to their existence behind NATs.

2. *Global Service Provisioning* (GSP) - a host hosting a particular service behind a NAT wants to make itself available globally by maintaining a mapping on the NAT.

3. *Service Provisioning using pre-signaling* (SPPS) - It is an extension to the GSP where both the requester and service residing behind NATs supports NAT traversal. Both must have support for at least one of the NAT traversal frameworks.

4. *Secure Service Provisioning* (SSP) - It is a further extension of SPPS, where the authentication and authorization factor is entertained. In this case the hosts need to be authorized to access each others services or the channel established between them. This can be implemented by enforcing restriction rules at the hosts, at the NATs of the hosts, at the firewall or at the data relay node.

2.1.2 Various NAT traversal techniques

1. UPnP [59] - It exploits port forwarding [52] technique for establishing direct communication between peers. But here the NAT must be enabled with UPnP service or else UPnP is helpless.

2. NAT-Port Mapping Protocol(NAT-PMP) - NAT-PMP [60] is a protocols according to RFC 6886 can be used by host behind NATs to get a internal-to-external address translation mapping on the NAT for any destination on the external address realm. The host also can specify the NAT, the lifetime of the NAT table entry.

3. Port Control Protocol(PCP) - PCP [61] is another protocol same as NAT-PMP. While NAT-PMP was limited to consumer-grade devices for deployment, PCP can be also deployed in carrier-grade [62] and dual-stack lite equipments [63]. But still in this case the NAT must support either of the protocols.

4. Session Traversal Utilities for NAT (STUN) [64] - It helps to know the type of NAT the host is residing behind to facilitate NAT traversal. It discovers the public address of the internal host via a STUN server and notifies the external host about it which can then communicate to the client by using that address. However, it does not work for symmetric NATs [65].

5. Traversal Using Relays around NAT (TURN) protocol [66] - It has a relay server that relays packets between two hosts behind NATs that has desire to communicate and send data. In this case both the peer is aware about the communication.

6. Interactive Connectivity Establishment (ICE) [67] - ICE is a more complex NAT traversal technique when both peers are behind NAT. It uses both STUN and TURN to obtain a set of address candidate pairs created by combining address candidates of the two communicating hosts. These pairs are checked for connectivity establishment. But just like STUN and TURN, even ICE is not suitable for the problem discussed in the previous chapter. Because in all these cases both of the communicating parties are actively participating by exchanging messages. However in our case one party (the client) is not pre-informed when the other (server) will try to communicate with the device. A complex infrastructure like TURN, STUN and ICE in this case also becomes an overhead.

7. Context-Aware NAT traversal (CAN) [68] - CAN believes in not trying all forms of connectivity checks for establishing connection between hosts behind NATs. Both the hosts behind NATs find information about their addresses in both private and public address realms, the type of NAT they are residing behind and whether the NAT implements hair-pin translation [69] and connection tracking or not. Both hosts exchange this information with each other through a SIP [70] based protocol. On the basis of these exchanged information the hosts find the appropriate communication paths that further goes through the connectivity check.

8. Gradual Proximity Algorithm (GPA) [71] - It is based on Relay policy like TURN. It attempts to find the closest Relay server for relaying message between hosts behind NAT. Firstly it searches for a Relay in the autonomous system, then one present in the same country otherwise it tries the one present in the host's continent. If still not found it chooses a Relay randomly.

9. Advanced NAT Traversal Service (ANTS) [72] - It involves sending signals to establish direct connection between two mobile clients without involving any intermediate server as in case of STUN and TURN. ANTS incorporates UPnP to acquire NAT configuration data needed to traverse the NAT. The input module takes inputs like behaviour of the NAT, type of application and proof of being a potential requester, i.e. supports ANTS or not. Whenever an applications needs to have NAT traversal, based on the NAT behavior, application type and the fact that the application supports ANTS or not, ANTS decides a NAT traversal technique dynamically from it's NAT Traversal module. For example if the the application is a web server behind NAT then it will use GSP type NAT traversal techniques.

10. NUTSS [73] - It exploits packet spoofing to enable NAT traversal between two hosts behind NAT for both TCP and UDP. But all ISPs do not allow packet spoofing.

11. MIDCOM [74] - This protocol allows hosts behind a NAT or firewall to change the behaviors of the middle-box to enable necessary connections. But this is not possible every time as all hosts are allowed to alter the middle-box.

12. NatTrav [75] - In this technique, a connection broker is present between the initiator and the recipient. The recipient registers itself with the broker with a URI to uniquely identify itself and accept connections from the initiator. The broker provides NAT traversal service for the recipient if it is behind a NAT.

13. NATng [76] - It is a framework that uses a *Domain Name System Application Layer Gateway* (DNSALG) for supporting private address name resolutions and controlling hole punching functionality.

14. NATBLASTER [77] - Unlike NUTSS, NATBLASTER does not rely on packet spoofing and is a technique to traverse the NAT and establish a direct TCP connection between two hosts residing behind NATs.

None of the above mentioned works are suitable to effectively solve the NAT traversal problem discussed in the previous chapter 1. In the following section a literature survey on various components of sensor-cloud infrastructure has been done.

2.2 Instantaneous traffic minimization in a Sensor-Cloud Infrastructure

Throughout the years, since the evolution of Sensor-Cloud Infrastructure there have been many works done on the Sensor-Cloud in the fields of routing protocols [25], techniques of processing data [26], operating systems [28], coverage issues [29], management of energy consumption [30], localization [31], clock synchronization [27] and programming [32] In Sensor-Cloud infrastructure, good QoS measures and proper resource utilization is very important for creating smart, efficient and cost-effective services IoT services. It is quite evident that efficient traffic management is necessary in order to achieve better QoS and efficient resource allocation and utilization [85]. Efficient traffic management is very crucial for bandwidth optimization and minimization of the required channel capacity at the access link that transfers data bits from the sensor gateway to the cloud. Thus, traffic shaping [86] or traffic policing [87] are two ways for instantaneous traffic minimization by altering the outbound gateway characteristics [88] [89].

Traffic shaping is a way to optimize the overall performance of the network by delaying certain types of packets travelling through the networks [90]. In [91], it is seen that Bell Canada throttles the traffic generated from peer-to-peer(P2P) file sharing applications from travelling through their access links. The technique of traffic shaping is also used by Comcast to limit the aggregated traffic load generated from users that consumes higher portions of their available bandwidth, by allowing each users a time-window of 5 minutes [92].

These approaches though serves the purpose of minimizing the instantaneous aggregated traffic or smoothing of the aggregated traffic, requires the gateway to monitor the incoming traffic continuously. This is quite an onerous job for the gateway to carry out. As traffic is controlled by the gateway, the gateway is required to have infrastructure for queuing, especially deep queues, that adds an extra amount of delay in data transmission.

Whereas, in traffic policing, the traffic management is done by maintaining a maximum rate for sending and receiving of data bits on interfaces of the gateway. The traffic within the bounded sending rate are transferred straight away but the data streams with higher data rate are either dropped or in some cases are transmitted with a different priority [93]. This approach is inefficient and very much power consuming for sensor nodes as large number of packet drops occur while transmission. This further degrades the overall output rate of a data stream.

In [94] the technique mentioned aims to change the quality of the transmitted data at real time. Though suitable for video transmissions or voice transmissions, this technique is not suitable for sensor data transmissions.

In [78], the technique used to smooth the aggregated traffic of all sensors is to schedule the sensor nodes with fixed offsets [95] for starting their transmissions. The main motivation behind the approach is to distribute the aggregated traffic from sensor nodes as evenly as possible on the access link. For sensor nodes performing periodic transmissions the offsets specifies the sensor nodes when to start sending the data frames in each period where the period is comprised of a fixed number of time slots. This approach schedules the sensor nodes so that the aggregated traffic received at the gateway always remains within a minimum determined range of channel capacity. Thus, the gateway need not take the burden of traffic shaping. To schedule the sensor nodes in that manner periodic offsets were set for all sensor nodes to obey in every period. M.Grenier et. al. proposed a scheme to enhance controller area network (CAN) network performance by scheduling messages with offsets [96]. The offset of each stream is chosen such that the release of its first frame is as far as possible from the other frames already scheduled.

The application scenario and the problem depicted in chapter 4 does not bear any significant resemblance to any of the aforementioned literature.

2.3 Summary of the chapter

In this chapter, section 2.1 discusses the related prevalent NAT traversal techniques used in different application scenarios. The various categories of NATs present have also been discussed briefly. After studying the literature illustrated in the section 2.1, it was concluded that none of the mentioned techniques were suitable to effectively solve the problem of waking up a sensor node that resides behind a NAT and is a power constrained device.

Whereas, in the section several works related to different application scenarios of smoothing of instantaneous traffic of sensor nodes or other media has been discussed. Based on the studied literature it was concluded that none of the works bore any significant resemblance with the application scenario defined in chapter 4.

Chapter 3

Energy Efficient Passive NAT Traversal through UDP

3.1 Introduction

Sensing devices are the key components in most IoT infrastructures. Their job is to sense and transmit the data to any given destination as and when required. Environment monitoring and remote supervision of various smart devices deployed in smart homes, factories and other environments are two prevalent IoT application scenarios in the contemporary world. In this chapter a problem faced in case of a sensor-cloud infrastructure has been addressed. The problem mainly arises when the sensor nodes of a sensor-cloud serves as an integral part of an IoT infrastructure serving an IoT application. The first problem that was discussed in section 1.3 was about waking up a sensor node when it has been deployed in any environment while it resides behind a NAT.

In many scenarios, a sensor node may not be required continuously for indefinite amount of time. In a Sensing As A Services based delivery model sensing devices are acquired by the remotely located cloud based resource manager as and when required. This requires the sensors to be addressable by the cloud server. However while using IPv4 based network, sensing devices often reside behind NAT(s).

NATs [53] extend IPv4 address space and provide security to the private network beneath it by discarding any unsolicited network packet that it receives and also by acting as a firewall in many cases. The most common NAT prevalent in the contemporary world are Traditional NATs [54], specifically Network Address Port Translators (NAPT) [55], in which multiple clients residing in a private network behind a NAPT can initiate an outbound connection by sharing the single registered public ipv4 address of that NAPT. This type of NAT is widely used by Small Office Home Office (SOHO) [56] groups to connect to the Internet sharing a single registered IP address provided by Internet Service Provider (ISP). Herein, each connection is distinguished by a distinct port number.

In this scenario, there is a need to signal the sensor devices with necessary supervisory commands to execute different tasks for which they are configured to perform. In that case, the devices can remain dormant until they are required to perform

any task. This type of communication is termed as *passive*, as they are acquired without their prior knowledge.

For example, a smart device deployed in a smart home can be instructed from an external server to get switched on or off or perform any particular task either independently or in cooperation with other devices present in that smart home. If the device is a sensor then it can be instructed to start sensing data and transmit them to a desired destination.

However, when an IoT application running on a cloud server desires to collect sensor data and attempts to acquire the sensor devices, it needs to propagate the activating signal to those smart devices that are behind a NAT and are not publicly addressable. The devices behind NAT, being privately addressable cannot be addressed spontaneously by a remotely located publicly addressable server. However a server can respond to a connection that is initiated by the client. The client's NAT allows the client to receive packets through the same port that was previously used to initiate the connection. This happens due to NAT hole punching, which the NAT implements by creating temporary NAT entries that comprises a mapping between the private and public addresses of the client. Once a hole is punched its lifetime is limited by a fixed amount of time determined by the NAT. Hence to establish a consistent channel of communication from a cloud server to a client, a connection has to be reinstated at certain time interval either by the client or by the server. This can be done from the client side by repeated transmission of keep alive packets. But this leads to energy drainage in those small resource constrained devices.

This chapter thus aims to describe a scheme that involves UDP hole punching along with minimal polling to detect the translation table entry lifetime in any given NAT and maintain the consistent channel of communication.

The proposed scheme is applicable for Basic NATs [54], NAPTs [54], Bidectional NATs [54], Twice NATs [54] and Multihomed NATs [54]. It also works in case of Full cone NAT [57], Restricted cone NAT [57], Port-restricted cone NAT [57] and Symmetric NATs [57].

3.2 Scheme Overview

The proposed scheme requires two parties, a client behind a NAT and a publicly addressable server that can communicate over UDP. Both parties exchange messages in order to establish a consistent channel for bidirectional communication. At the beginning, the server listens on two UDP ports for incoming connections. The client initiates the conversation by sending messages to both ports. We refer the first port as trial port and the second port as live port. The conversation follows the four operational stages mentioned in Figure 3.1.

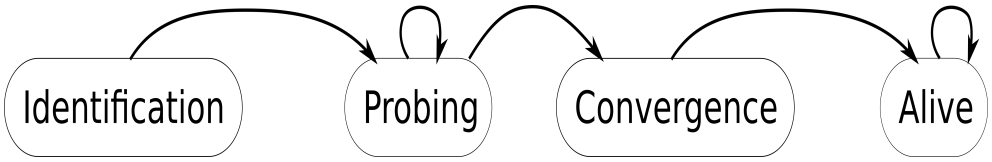


FIGURE 3.1: Scheme Overview

3.2.1 Identification

The client sends a handshake message to the server along with a token to identify itself. The server validates and responds with a cookie if the client identity is valid. For the future communications, the client glues this cookie with all its messages. This is done in order to stop any fraudulent device to enter the network and cause DOS attacks [98] say by flooding requests to the cloud server in order to make it busy or go down.

3.2.2 Probing

In this operational stage, the maximum permissible delay of the NAT is determined. For determining the maximum sustainable delay, the client sends a probe message to server's trial port. On receiving this message, the server immediately sends an acknowledgement along with a delay, after which the server promises to send a reply. Client on receiving the acknowledgement starts a timer with the mentioned time duration and expects to receive a delayed reply just before the timer completes. Whether it fails or succeeds in receiving the server's delayed reply the client sends another probe message to the server.

The server, before sending the acknowledgement message, starts a timer with the same interval that it sends in the acknowledgement message. If, at the client, the timer completes before receiving a reply, it is considered that the reply sent by the server failed to traverse the NAT.

From the next probe message, the server figures out whether the previous reply did reach the client or not. If the previous message fails to reach the client, the server decreases the delay for the next probe, otherwise increases. Both the client and the server keep on repeating this process to converge to a suitable maximum delay that is permissible by the client's NAT.

On the other hand, on live port, the server keeps sending a message on a time interval which is known to be successful through the previous probe on trial port. While sending the first probe, the client simultaneously sends a message to the live port to initiate this conversation. When the dynamic delay on the trial port increases, the stable delay on the live port is also increased. So the delay used in communication through the live port is always below the client NAT's threshold.

Initially the delay is always increased exponentially until it reaches such a delay which is beyond the maximum idle time permitted by the NAT. As a result the NAT entry is removed and the connection is lost. However a lower and an upper bound is

found at the end of this stage, between which the actual delay is present. We call this stage as exponential stage. After exponential stage is finished a localization stage is started that finds the actual delay using bisection method.

3.2.3 Convergence

Throughout the conversation the stable delay on live port keeps increasing until it reaches the maximum permitted delay. Server detects that scenario as convergence and instructs the client to stop sending probing messages. The communication moves towards the alive phase.

3.2.4 Alive

After convergence stage, the server starts sending only delayed reply packets to the client at a fixed time interval. In ideal scenario the client receives all the messages sent by the server because the delay used are all just below the NAT's permissible delay. However if an alive message is not received by the client within the expected time duration client sends a failure message to re-establish communication.

The Sequence diagram shown in Figure 3.2 details the communication between the client and the server on trial port.

3.3 Protocol Operation

To implement the above mentioned communication scheme a protocol has been designed. The protocol consists of 7 different messages. In this section each of these messages are explained.

3.3.1 Handshake

An handshake message is sent by the client which contains the unique identification credentials. On receiving this message, server verifies whether the client is allowed by the server or not. Upon acceptance, a handshake reply message is sent by the server. If no handshake reply is received by the client within a short period of time, the client re-transmits the handshake message.

3.3.2 Handshake reply

An handshake reply consists of a 32 bit unique cookie that the server maintains to uniquely identify that connection. The server maintains a table that maps that cookie with the client connection endpoint (pair of IP address and port). The client glues the same cookie with all its subsequent messages. If the server gets the same cookie from a different connection endpoint, it assumes that the client endpoint has changed due to some connectivity problem on client side and replaces the client's endpoint with the current one.

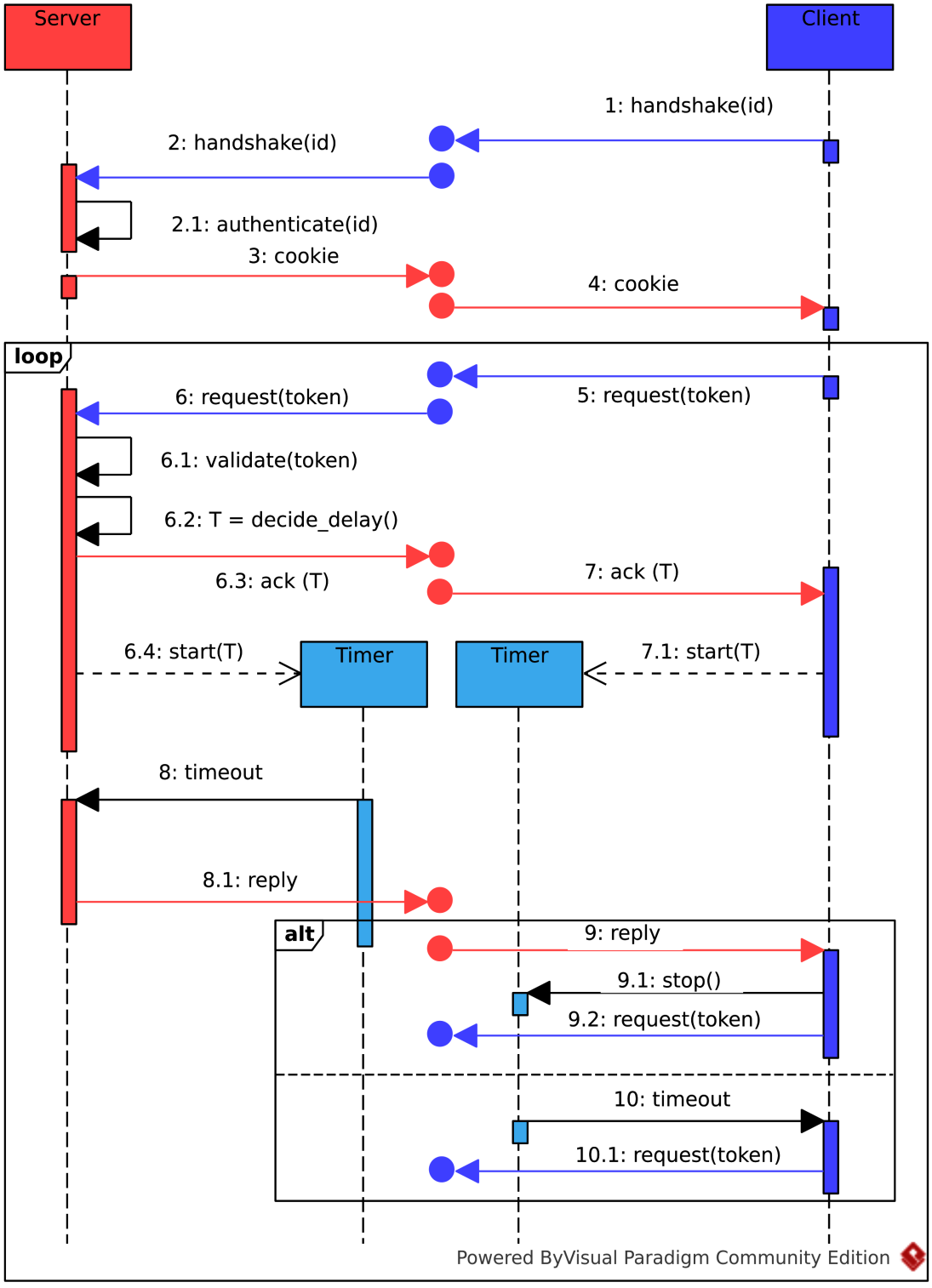


FIGURE 3.2: The protocol sequence diagram

3.3.3 Probe

In the probing stage, the client sends a *probe* message expecting a delayed *reply* message from the server. The probe message contains two fields, the cookie and a payload. The cookie field contains the cookie received in handshake reply message. The

payload is required to validate the probe message sent to the server by the client. This validation is done by the server. The payload field contains a 32 bit integer that is generated by the client, based on the previous *reply* message received from the server. The payload is generated in such a way that helps the server to decide whether that client has received the last reply or not as the generated payload for a probe depends on the payload of the last reply message received by the client. Our implementation uses Equation 3.1 to generate the client payload (cp) for the n^{th} probe as cp_n. sp_{n-1} and cp_{n-1} are respectively the previous server payload received in reply to the previous probe and the last client payload sent. The Initial values sp_0, and cp_0 are set to 0 and a random value respectively, which are used to generate cp_1 and sp_1. ρ_n is a random number used at the n_{th} probe. Server uses the ρ_n for generating a unique payload for the *reply* message.

$$cp_n = sp_{n-1} \oplus (cp_{n-1} + 1) \tag{3.1}$$

$$sp_n = cp_n \oplus \rho_n \tag{3.2}$$

$$cp_n \oplus sp_{n-1} == cp_{n-1} + 1 \tag{3.3}$$

3.3.4 Acknowledgement

On receiving a probe message, the server decides a suitable delay after which the expected delayed *reply* message will be sent. That delay is sent to the client as an immediate acknowledgment of the client's probe. For the first probe, delay (d) of 1 second is decided. For the subsequent probes the delay is gradually increased or decreased depending on whether the client has received the last *reply* or not. This delay decision is controlled by a variable δ shown in Equation 3.4 which is initially set to 2. On success this delta is multiplied with the current delay (d) to yield the next delay, which leads to an exponential growth. The first failure terminates the exponential growth phase and the upper bound of the permissible delay is found. A localization stage is started that finds the actual maximum permissible delay using bisection method. In case of failure δ is modified as shown in Equation 3.4 and is multiplied with d^* which is the last successful delay. In case of success in localization phase δ is not modified, but d is modified and set to the current one.

$$d = \begin{cases} d \times \delta & \text{success} \\ d^* \times \delta & \text{failure} \end{cases} \quad \text{where } \delta = \begin{cases} 2 & \text{success at exponential stage} \\ \delta & \text{success at localization stage} \\ \frac{\delta+1}{2} & \text{failure} \end{cases} \tag{3.4}$$

If the client has received the server's payload sp_{n-1} of $(n-1)$'th probe before sending the n'th probe then Equation 3.3 is satisfied as shown in Equation 3.5. Hence the server detects whether the client has failed or succeeded in receiving the previous reply message by evaluating Equation 3.3.

$$\begin{aligned} & cp_n \oplus sp_{n-1} \text{ substitute } cp_n \text{ using Eq 3.1} \\ & = ((sp_{n-1} \oplus (cp_{n-1} + 1))) \oplus sp_{n-1} \\ & = (cp_{n-1} + 1) \end{aligned} \tag{3.5}$$

An example is shown in Table 3.1 assuming the actual limitation is around 180 seconds. The first row is the initial stage. The exponential growth stage is continued till the 8^{th} row. The successful probes are marked with green color and the failed rows are marked with red color. Although the example takes 20 probes to converge, we can eliminate some probes that go beyond previously marked upper bounds. In Table 3.1 the previous upper bounds are marked with # symbol. The rows marked with a $*$ go beyond the last known upper bound, thus skipped to save number of probes in our implementation.

A randomly generated 16 bit integer identifier is attached with the *ack* message which is carried in the subsequent *reply* message to associate the reply with the current probe.

TABLE 3.1: Example of Delta Decision

		d	d^*	δ	
0		1	1	2	
1	1*2	2	1	2	
2	2*2	4	2	2	
3	4*2	8	4	2	
4	8*2	16	8	2	
5	16*2	32	16	2	
6	32*2	64	32	2	
7	64*2	128	64	2	
8	128*2	256	128	2	#
9	128*1.5	192	128	1.5	#
10	128*1.25	160	128	1.25	
11	160*1.25	200	160	1.25	*
12	160*1.125	180	160	1.125	#
13	160*1.0625	170	160	1.0625	
14	170*1.0625	180	170	1.0625	*
15	170*1.03125	175	170	1.03125	
16	175*1.03125	180	175	1.03125	*
17	175*1.015625	177	175	1.015625	
18	177*1.015625	179	177	1.015625	
19	179*1.015625	181	179	1.015625	*
20	179*1.0078125	180	179	1.0078125	*
21	179*1.00390625	179	179	1.00390625	

3.3.5 Reply

The server after sending the *ack*, waits for d time and sends the *reply* to the client. This marks the end of a single probe comprising the probing and delay decision stage. The *reply* message contains the identifier which was sent in the *ack* and a payload. The identifier is used to validate relevance of the *reply* message in context of the previous *ack*. The payload of n^{th} probe sp_n is calculated using Equation 3.2 where cp_n is the payload received in the client's probe message and ρ_n is a random number generated on iteration n.

However, as UDP is unreliable, it can lead to message loss. So, instead of sending a single reply, a set of k redundant replies are sent from the server. Delay between each of these replies are set to μ time units such that $k\mu < 1$ second. The last redundant reply is sent on d time unit and the first reply is sent on $d - k\mu$ time. The client matches the identifier received from the last *ack* message with the identifier in the reply message and discard redundant replies, after receiving the first reply.

A NAT may associate different timeouts for an UDP port depending on the type of conversation carried on that port. NATs detect whether the conversation is bidirectional or unidirectional and assigns a larger timeout for bidirectional communications. In most NATs the default timeouts for unidirectional and bidirectional communication are respectively 30 seconds and 180 seconds [97]. However, it can be altered by the network administrator. In case of failures, a new conversation session is started that is initially unidirectional. So a 1 second probe is performed to make that conversation session bidirectional. After the 1 second probe is finished the next probe is performed and the delay decided for it happens to be according to Equation 3.4 as shown in Table 3.1.

3.3.6 Alive Request

This message is sent by the client to the server for establishing the *Alive* connection. It comprises of two fields, cookie and safe delay. The safe delay field stores the last successful delay on the trial port conversation. After sending a *alive request* with safe delay d, the client expects it will receive an *alive reply* at every d seconds interval without sending another *alive request*. However if the alive reply is not received, client sends another *alive request* to reopen the port.

3.3.7 Alive Reply

In trial port communication whenever a probe is successful, the delay assumed in that probe is updated as safe delay. Thus this safe delay is either incremented or kept same by the server. Once a delay is evident to be safe it implies that UDP datagrams sent on that interval can pass through the client's NAT without requiring the client to send anything. The *alive reply* message is repeatedly sent using the current safe delay as time interval. This message consists of the safe delay after which time the

next alive reply will be sent. So if the safe delay is updated then the client is also notified through this message.

3.4 Results and Discussion

To experiment with the proposed scheme we have implemented a client and a server and tested under different network configurations. The entire scheme is implemented in C++ on linux based environments. The server has been set up on a remotely located virtual machine having publicly addressable IPv4 address. The client is always behind a NAT. At first we have experimented by putting the client under an ADSL based broadband connection. Routers from different manufacturers including TP-Link, Netgear, Huawei, ASUS have been used in the experiments. In Figure 3.3 the orange line shows probe delays and the blue line shows the alive delays.

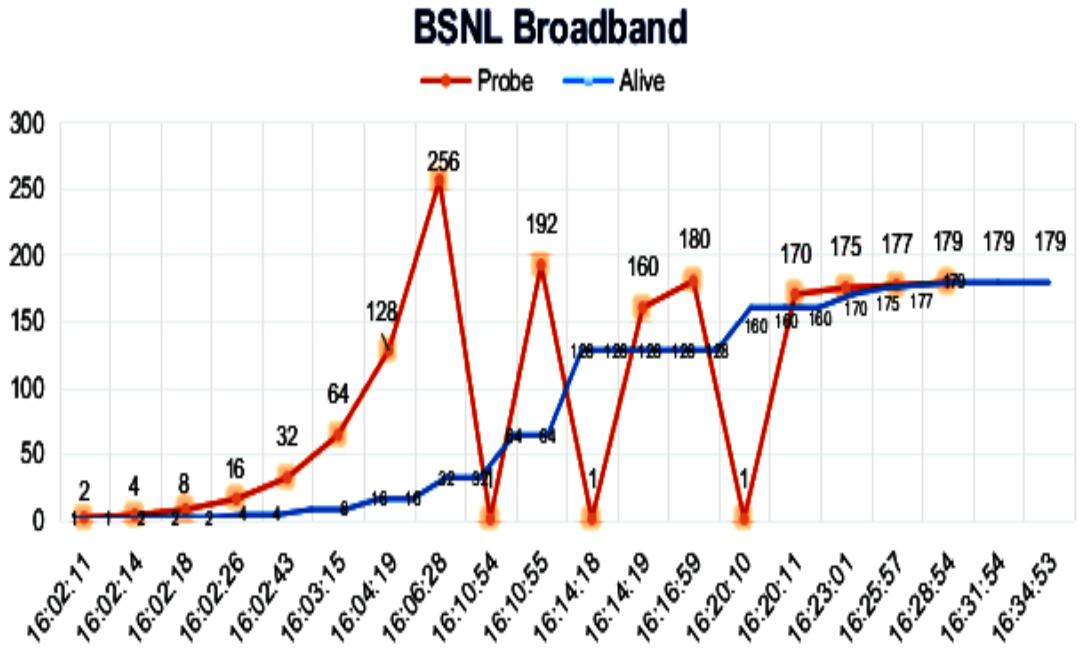


FIGURE 3.3: Results ADSL broadband

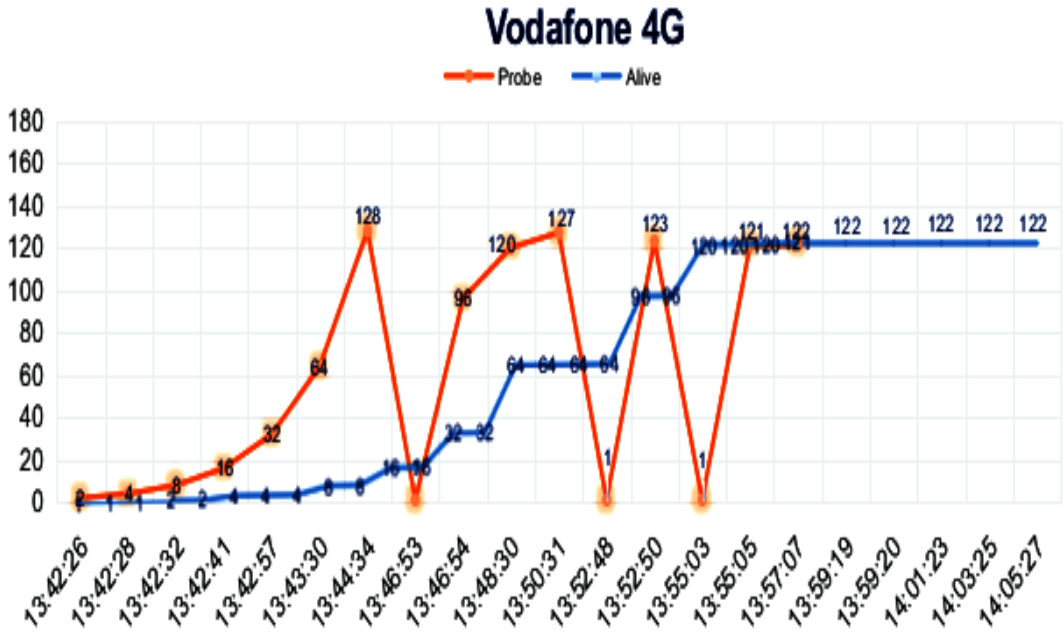


FIGURE 3.4: Scheme Results in Vodafone 4G network

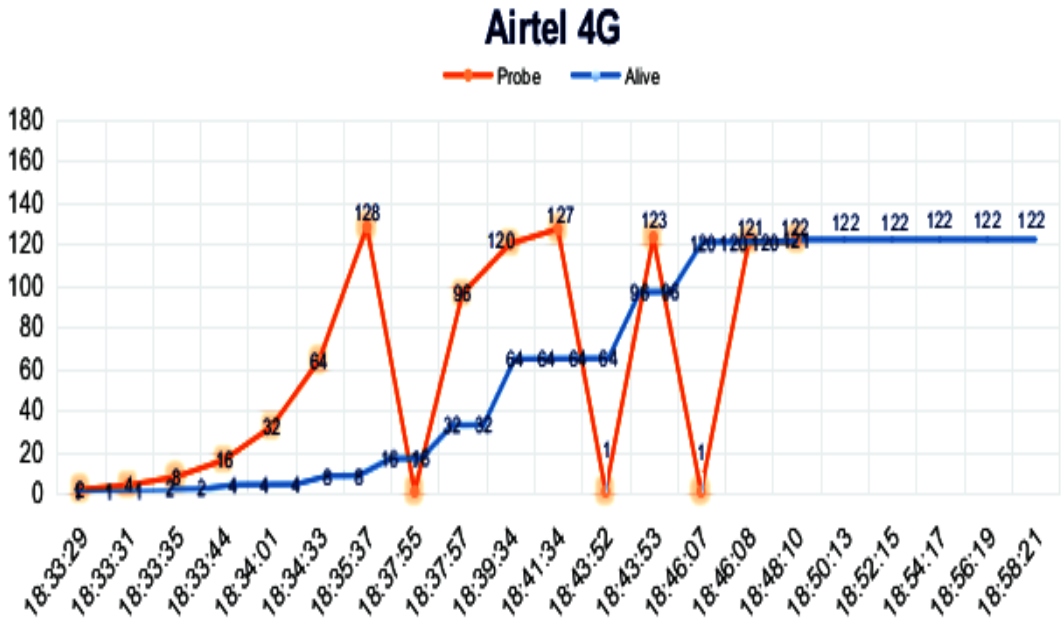


FIGURE 3.5: Scheme Results in Airtel 4G network

To experiment with GSM based internet services we have used mobile hotspot with various manufacturers including, Samsung, Apple, Motorola under two network providers, Vodafone and Airtel both providing 4G services. It has been observed that both providers sustain 122 seconds delay as shown in Figures 3.4 and 3.5. For the broadband based experiments it took about 29 minutes and 6 seconds to

converge. Whereas, in the GSM based experiments a total time of about 18 minutes and 14 seconds were taken to attain convergence.

We have observed for 24 hours, that the live channel is consistent for server to client communication. During that time the server was able to wake up the client at any time. It is also observed that the consistent live channel may fail to receive the UDP alive replies from the server due to network problems. But number of such occurrences are significantly low. In that case the client re-transmits an alive request to re-initiate the alive channel. For our experiments on wired broadband connection no re-transmission was necessary for 24 hours. Therefore after convergence, a consistent server to client communication channel was established without the client sending any UDP packet. However for the GSM based experiments some packet loss were encountered. Though the number of such occurrences was significantly low.

3.5 Summary of the Chapter

In this chapter, the problem of successfully traversing the NAT of the sensor nodes in order to instruct them to stream data, have been addressed. The chapter first defines the problem scenario which depicts the need to traverse the NAT of the sensor nodes. Then in section 3.2, the overview of scheme designed to solve to the problem have been discussed. And then in section 3.3, a protocol have been proposed in order to implement the designed scheme illustrated in section 3.1 with the comprehensive operational details have been discussed. Finally in 3.4, the proposed protocol have been tested in three different networks. For GSM based networks Vodafone 4G and Airtel 4G network have been considered and for ADSL based broadband connection BSNL broadband network have been considered. The experiments have also been performed in case of different router vendors which includes routers of TP-Link, Netgear, Huawei, and ASUS. The results mentioned in 3.4, depicts that ADSL based broadband networks sustain the established channel better than the GSM based networks.

Chapter 4

Sensors Scheduling for Aggregated Traffic Minimization

4.1 Introduction

In a sensor-cloud infrastructure, a set of sensor nodes comprising a Wireless Sensor Network (WSN) is required to stream sensed data at a certain rate to a particular destination. In a typical sensor-cloud infrastructure IoT sensor nodes are connected to the cloud servers through routing gateways.

These IoT Sensor nodes when deployed in any given environment, forming a WSN, is required to sense and send data to a cloud storage or an application hosted in the cloud, on demand.

In the previous chapter the addressed issue was all about how to activate a standalone sensor node or a network of sensor nodes on demand from a public server or cloud server, so that they wake up and start streaming sensed data while residing in a private network behind a routing gateway. The proposed mechanism stated in the previous chapter solves the issue of activating sensor nodes in private network, as and when required.

In this chapter, the problem that is addressed, arises when a set of active sensor nodes starts streaming data from a common starting time instance. Herein, the maximum aggregated traffic load generated at any time instance, due to the simultaneous data streaming of the active sensor nodes, must be minimized.

In large scale IoT scenarios like environment monitoring in smart homes or smart cities [83] [84], sensor nodes generate huge amount of traffic when they actively stream sensed data simultaneously according to their respective schedules to any particular application or cloud storage through a common gateway. This huge traffic must be efficiently accommodated [79] in both the communication interfaces, the one between the sensor nodes and the gateway and the one between the gateway and the cloud server. As the IoT sensor devices are growing exponentially with time [80] [81], the amount of data transfers between the sensor nodes and cloud servers are also increasing. Thus, increasing the amount of generated traffic that the IoT networks must be able to accommodate efficiently [82].

But unfortunately, IoT networks fail to do so as at any time instant the aggregated traffic load from all the sensor nodes may require a transmission bandwidth that is higher than that available in the IoT networks and thus may cause instant congestion and data loss. In order to solve this problem of exceeding bandwidth requirements, congestion and probable data loss, sensor nodes must be scheduled with offsets so that the maximum aggregated traffic load that gets generated from all the sensor nodes at any time instance can be minimized.

For more comprehensive understanding of the problem, a wireless IoT sensor network model is considered in section 4.2, where figure 4.1 depicts the wireless IoT sensor network model.

4.2 Wireless IoT Sensor Network Model

In figure 4.1, the wireless IoT sensors represent a set of sensor nodes deployed in any given environment with an objective to collect sensed data. The gateway is connected to two communication interfaces, a wireless network between the sensor nodes and the gateway and a wired medium between the gateway and the Internet leading to the cloud server. The total available bandwidth of the wireless medium is divided into number of frequency channels each having a bit rate enough to propagate the sensed data. Each sensor node requires a wireless frequency channel to stream data to the gateway through the wireless interface. The gateway further transfers the received data bits to the cloud server through the wired interface comprising the single wired access link which possesses a particular channel capacity that is measurable in terms of bit rate.

All the wireless IoT sensor nodes having specific periods, duties and transaction sizes when transmit data to the gateway through different wireless frequency channels the traffic from all the sensors received at the gateway during each time slot is aggregated and transferred to the cloud server by the gateway via the wired access link. Thus, the capacity of the access link must be enough to transfer all of the aggregated data bits in each time slot.

On the other hand, at any time slot if n sensor nodes are to transmit data then n frequency channels will be required. This n must not be more than the total number of frequency channels available at the wireless interface.

If n is large enough to demand a bandwidth (i.e. range of frequencies) that is not available at the wireless interface then it may happen that in a particular time slot more than one sensor nodes attempt to transmit data through the same frequency channel. This will lead to collision among all the transmissions happening via same frequency, causing data loss and power wastage of the sensor nodes.

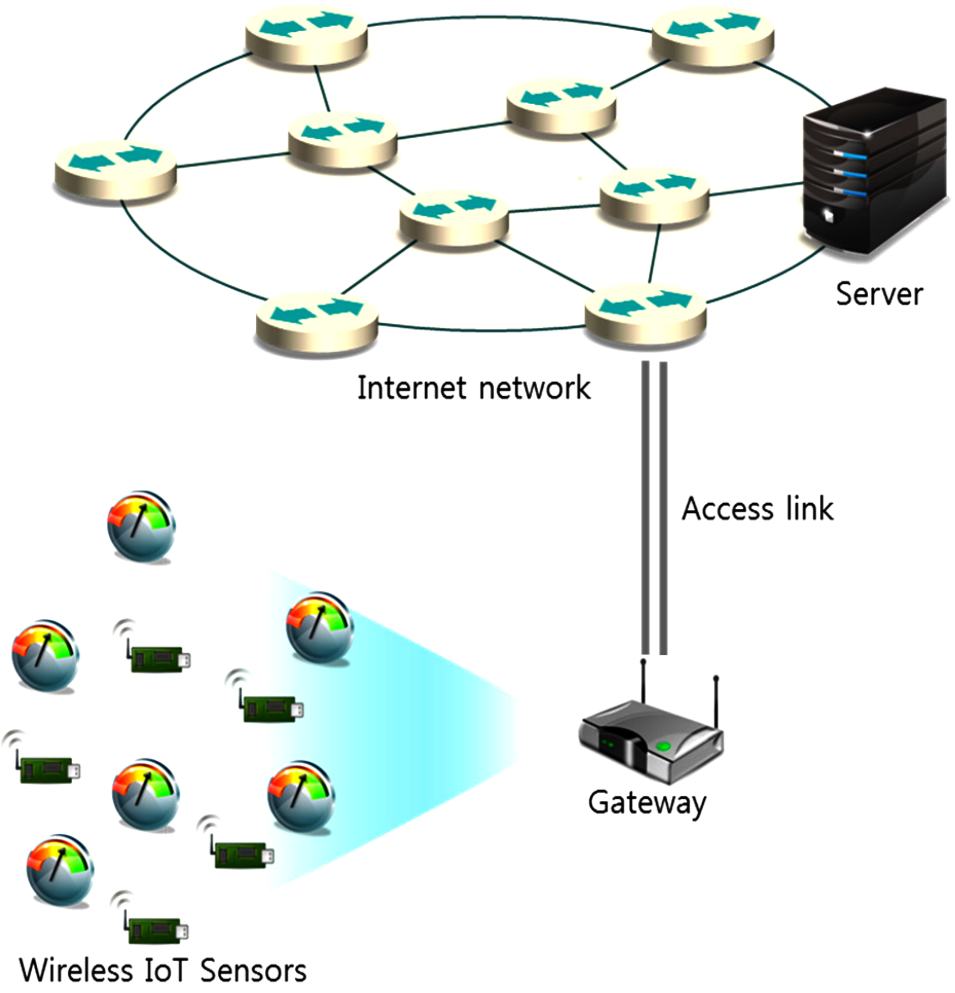


FIGURE 4.1: Wireless Sensor Network model in Internet of Things(IoT) scenario [79]

Thus, the bandwidth at the wireless interface of the gateway is measured in terms of range of available frequencies or number of available frequency channels, whereas in the access link the channel capacity can be measured in terms of bit per time slot.

4.3 Application Scenario

According to the application scenario, there is a set S, where $|S| \in \mathbb{Z}^+$, of n ($n \in \mathbb{Z}^+$) IoT sensor nodes, comprising a WSN for collecting data from any given environment.

Each i^{th} sensor node in set S, represented as s_i, where $0 \leq i \leq n-1$, in the set S must perform a certain number of transactions, say T_i ($T_i \in \mathbb{Z}^+$) number of transactions, once per a certain amount of time, termed as period p_i of the i^{th} sensor node, where p_i ($p_i \in \mathbb{Z}^+$) is measured in terms of number of unit time slots. The T_i number of transactions, that is required to be performed in a period p_i is termed as *duty*, of the sensor node s_i.

In each of the T_i number of transactions sensor node s_i streams w_i ($w_i \in \mathbb{Z}^+$) number of readings or samples. Therefore, size of a transaction, denoted as f_i ($f_i \in \mathbb{Z}^+$), is defined as the number of time slots required by the sensor node s_i to fully complete all the processes required to stream the w_i number of readings.

Now, according to the application scenario, the size of a period p_i happens to be always higher enough to accommodate T_i number of transactions completely. Therefore,

$$p_i \geq (T_i \times f_i), \tag{4.1}$$

Now at any time slot t if c_t number of sensor nodes are scheduled to transact then for possible transmission at the wireless medium there must be at least c_t number of frequency channels available to avoid collision and data loss at the time slot t.

Thus to avoid collision and data loss in any time slot t the maximum number of frequency channels , denoted by BW, that must be available for wireless interface at any instance of time can be defined as

$$BW = max(c_t) \forall t \in [0, \infty) \tag{4.2}$$

It is also given that each sensor node s_i is capable of transmitting a maximum of b_i ($b_i \in \mathbb{Z}^+$) bits of data in one time slot. Set B comprises of the set of b_i(s), of every i^{th} sensor node $\forall i$ $0 \leq i \leq n-1$ in set S.

Now suppose if in any time slot t ($t \in \mathbb{Z}^+$) a set X_t of sensor nodes, where $X_t \subseteq S$ and $|X_t| \leq |S|$, are scheduled to transmit data bits at their maximum rate then at

that time slot t total bits that can be transmitted to the gateway, denoted as μ_t ($\mu_t \in \mathbb{Z}^+$), can be defined as

$$\mu_t = \sum_{m=0}^{|X_t|} b_m, \tag{4.3}$$

where X_t is the set of sensor nodes transmitting at time slot t and b_m is the maximum number of bits that the m^{th} sensor node in $set X_t$ can transmit per time slot. Therefore, maximum required channel capacity at the access link, denoted as Ch_{max}, measured in bits per time slot, can be defined as in equation 4.4.

$$Ch_{max} = max(\mu_t) \forall t \in [0, \infty) \tag{4.4}$$

Now, in each period p_i, each of the T_i transactions is performed by the sensor node s_i after waiting for a certain number of time slots from the start of the period p_i, termed as offset, which can range from (0, ..., p_i - f_i) and is denoted by θ_i. Therefore, θ_i is a two-dimensional matrix of sensor node s_i, where each of the elements of the matrix is a θ_{jk} which is the offset for starting the k^{th} transaction in j^{th} period of i^{th} sensor node. Hence, in the application scenario, the WSN is assumed to be comprised of n sensor nodes of which each sensor node s_i is characterized by a six-tuple sensor node as $s_i = (i, p_i, T_i, f_i, b_i, \theta_i)$ where i represents the identification number of the sensor node.

4.4 Problem Definition

If all the n IoT sensor nodes in set S, each of which is characterized by a six-tuple as $s_i = (i, p_i, T_i, f_i, b_i, \theta_i)$, performs their respective k^{th} transaction $\forall\, k \in [0, T_i)$, in each of their respective j^{th} period $[t_0 + \mathrm{j} \times p_i \rightarrow t_0 + (\mathrm{j}+1) \times p_i)\ \forall\, \mathrm{j} \in \mathbb{Z}^+$ with offset θ_{jk}, where t_0 is the common start time for all sensor nodes in S, then the maximum aggregated traffic load c_t ($c_t \in \mathbb{Z}^+$), on any time slot t can be as high as $|S|$, in the worst case, where c_t is defined as the total number of sensor nodes in set S that are scheduled with offset θ_{jk} to transact on that t^{th} time slot. If $c_t = |S|$, then it denotes that all the sensor nodes in set S are scheduled to transact on the t^{th} time slot. This worst case scenario occurs because of a worst combination of all $\theta_i\ \forall$ i, ($0 \leq i \leq n-1$) of n sensor nodes.

Therefore, the objective is to find out the respective θ_i of every sensor node s_i in set S, such that the combination of all those θ_i will lead to the minimization of the maximum $c_t, \forall t \in [0, \infty)$.

To examine the resultant aggregated traffic from n sensor nodes it is necessary to consider a specific number of time slots for observing the traffic, say L number of time slots. As L is considered to be the LCM of the periods of the n sensor nodes, the

traffic load pattern observed in each of the L time slots will exactly repeat in the next L number of time slots.

Therefore any sensor node s_i will have $\frac{L}{p_i}$ number of periods in L time slots and the traffic load in any t^{th} time slot in L time slots will be same as in any $(m \times L + t)^{th}$ time slot, where $m \in \mathbb{Z}^+$ and $0 \leq m \leq \infty$.

Thus, the objective is to find that combination of all $\theta_i \; \forall$ i, $0 \leq i \leq n-1$) of n sensor nodes for L time slots, due to which the maximum aggregated traffic load c_t on any time slot $t \; \forall \; t \in [0, L-1]$, will be minimized, where θ_i of sensor node s_i for L time slots is a two-dimensional matrix in which each element is θ_{jk} where θ_{jk} is the offset which the sensor node s_i uses to perform the k^{th} transaction in its j^{th} while θ_{jk} $\in$ [0, p_i - f_i], $\forall$ i $\in$ [0, n), j $\in$ [0, $\frac{L}{p_i}$) and $\forall$ k $\in$ [0, T_i).

The minimization of the maximum $c_t \; \forall \; t \in [0, L-1]$ leads to the minimization of

1. bandwidth requirements (*or, number of frequency channels*) at the wireless interface for data transmission in the WSN and,

2. the maximum channel capacity Ch_{max} (*in, bits per time slot*) in equation 4.4, necessary to be possessed by the access link.

4.4.1 Problem Variation

The problem discussed so far can have two variations with respect to how the T number of transactions are to be executed in a period, where $T > 1$. Both the variations are enlisted below.

1. *Consecutive execution of transactions* - There is a restriction that all the T number of transactions are required to be done consecutively.
2. *Non-consecutive execution of transactions* - There is *NO* such restriction that the T number of transactions are required to be done consecutively.

To have a deeper insight of both the problem variations mentioned above, an example has been considered in table 4.1 that depicts the specifications of a set of four sensor nodes.

TABLE 4.1: Four IoT sensor nodes - sensor 0 ($p_0 = 2$, $T_0 = 1$, $f_0 = 1$, $b_0 = 16$), sensor 1 ($p_1 = 3$, $T_1 = 2$, $f_1 = 1$, $b_1 = 8$), sensor 2 ($p_2 = 6$, $T_2 = 2$, $f_2 = 2$, $b_2 = 16$), sensor 3 ($p_3 = 6$, $T_3 = 1$, $f_3 = 1$, $b_3 = 8$)

i	p_i	T_i	f_i	b_i
0	2	1	1	16
1	3	2	1	8
2	6	2	2	16
3	6	1	1	8

In table 4.1, there is a set $|S|$ of four IoT sensor nodes that have been considered to comprise a WSN. According to the periods of sensor node 0 ($p_0 = 2$), sensor node

1 ($p_0 = 3$), sensor node 2 ($p_2 = 6$) and sensor node 3 ($p_3 = 6$) the number of time slots for which the maximum aggregated traffic must be monitored is the LCM of all the periods which is six. Thus in fig 4.2, six time slots ($t_0 \rightarrow t_5$) is shown.

For sensor node 0, there are three periods that are present among the six time slots. Similarly, for sensor node 1 there are two periods each of three time slots and for sensor node 2 and 3 there is a single period each of which comprises of six time slots. In table 4.2 the periods of each sensor is shown by the two-tuple (x_{ij}, y_{ij}), where time slot x_{ij} and y_{ij} represents the starting point and the finishing point of the j^{th} period in case of the i^{th} sensor node.

TABLE 4.2: The starting and ending point of each period in each sensor node represented by the two-tuple (x_{ij}, y_{ij})

i	j	x_{ij}	y_{ij}
0	0	0	1
	1	2	3
	2	4	5
1	0	0	2
	1	3	5
2	0	0	5
3	0	0	5

Suppose all the four IoT sensor nodes starts performing each of their T transactions in each of their periods with a random offset combination of θ_{ijk} that is mentioned in table 4.3. Herein, θ_{ijk} represents the offset at which the i^{th} sensor node starts the k^{th} transaction of the j^{th} period.

If the four IoT sensor nodes obey the offset configuration mentioned in table 4.3 then the maximum aggregated traffic load per time slot will look as depicted in figure 4.2.

Now, in the 0^{th} time slot, as all the four sensor nodes transmit data bits to the gateway shown in figure 4.1, $c_0 = 4$ which is the maximum c_t, $\forall t \in [0 \rightarrow 5)$, according to equation 4.2, BW = 2.

As the four sensor nodes can send data bits at their maximum capacity in the 0^{th} time slot, so at the access link the Ch_{max} required in the worst case will be 48 bits per time slot according to equation 4.4.

Now, with respect to the example discussed so far that was considered in 4.1, the comprehensive explanation of the aforementioned problem variants 1 and 2 are as follows.

TABLE 4.3: Random Offset configuration for the four IoT sensor nodes

i	j	k	θ_{ijk}
0	0	0	0
	1	0	0
	2	0	0
1	0	0	0
	0	1	1
	1	0	0
	1	1	1
2	0	0	0
	0	1	2
3	0	0	0

In figure 4.2, s0, s1, s2, s3 represents the four sensor nodes as mentioned in table 4.1. As per the offset configurations mentioned in table 4.3, all the four IoT sensor nodes are scheduled to transact at the 0^{th} time slot. Thus this configuration of offsets can be considered as the worst configuration because there exist at least one time slot in which all the sensor nodes are scheduled to transact together. As in the 0^{th} time slot it can be seen that all the four sensor nodes can transmit data together the number of frequency channel required at that time slot for the wireless medium is four.

4.4.1.1 Consecutive execution of transactions

With respect to this problem variant in which every i^{th} sensor node must execute T_i number of transactions consecutively in each of their j^{th} period a better configuration of offsets can be the one depicted in table 4.4.

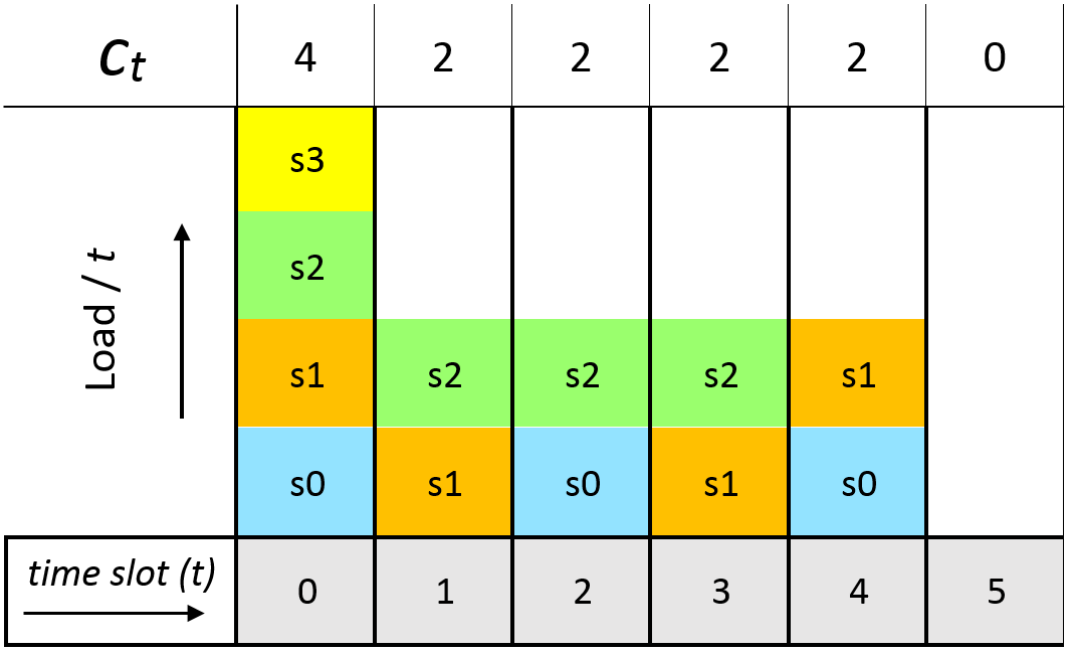


FIGURE 4.2: Aggregated traffic load per time slot from all the four sensor nodes with offset configuration mentioned in table 4.3

TABLE 4.4: A better offset configuration for the four IoT sensor nodes when transactions in each period are performed consecutively

i	j	k	θ_{ijk}
0	0	0	0
	1	0	0
	2	0	0
1	0	0	0
	0	1	1
	1	0	0
	1	1	1
2	0	0	2
	0	1	4
3	0	0	1

Now if the same four sensor nodes in 4.1, obey the offset configurations mentioned in table 4.4 then as depicted in figure 4.3, the maximum aggregated traffic load on any time slot or maximum c_t happens to be 3, where t is the 4^{th} time slot in the six time slots.

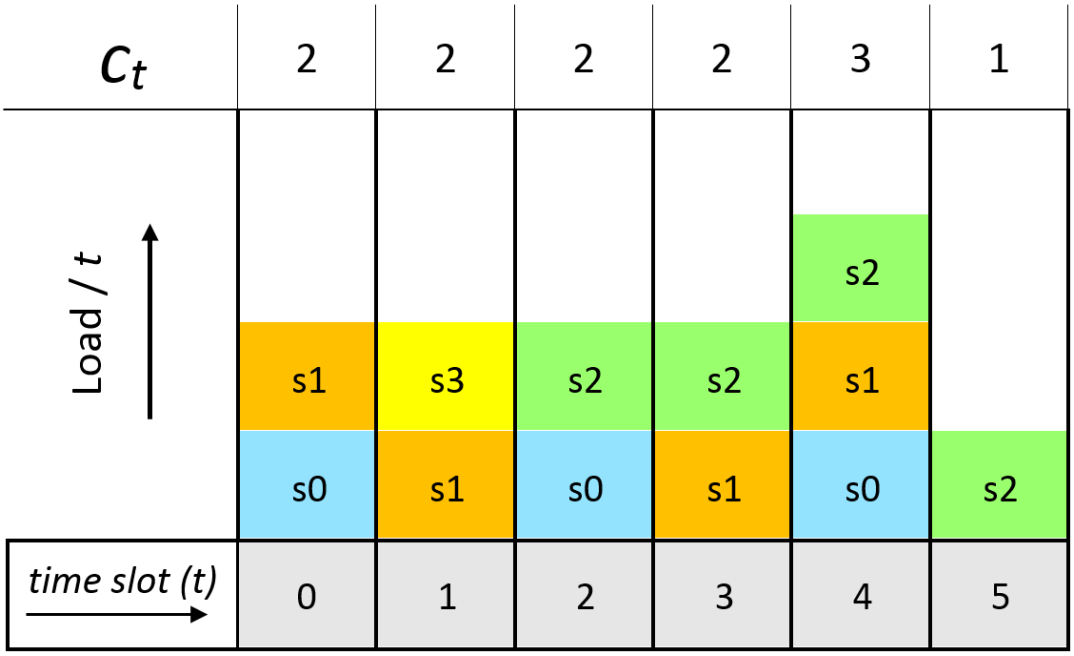


FIGURE 4.3: Aggregated traffic load per time slot from all the four sensor nodes with offset configuration mentioned in table 4.4. The Sensor nodes executes multiple transactions in periods (*if present*) consecutively.

According to the offset configuration mentioned in table 4.4 the aggregated traffic pattern of the four sensor nodes happens to be so that at the fourth time slot sensor node s0, s1 and s2 are scheduled to perform their transaction.
Thus in the 4^{th} time slot it can be seen that as three sensor nodes might transmit data together the number of frequency channel required at the wireless medium in that time slot is three and this further concludes that $BW = 3$, c_t is maximum in t=4.

It is also realized from figure 4.3 that in the 4^{th} time slot, as three sensor nodes s0, s1 and s2 transmit data bits to the gateway shown in figure 4.1, at their maximum capacity, μ_t is found to be maximum for $t = 4$. Thus, the maximum channel capacity Ch_{max} (*in, bits per time slot*) that will be required in the worst case can be calculated as follows.

According to equation 4.3,

$\mu_4 = 40$ bits/time slot,

It is evident from fig 4.3, that μ_t is maximum for $t = 4$

Then, from equation 4.3 and 4.4,

$Ch_{max} = 40$ bits per time slot

Thus, it is evident that in case of the worst configuration in table 4.3, the four sensor nodes requires a maximum of four frequency channels for communication at the wireless interface and for the access link the maximum channel capacity required is 48 bits per time slot. Whereas, in case of the the better offset configuration in table 4.4 the four sensor nodes requires a $BW = 3$ for communication at the wireless

medium and $Ch_{max} = 40$ bits/time slot for the access link.

Thus, it can be concluded that an optimal configuration of the offsets for the first problem variant will surely minimize the bandwidth and channel capacity requirements in the wireless interface and in the access link respectively. A straight forward optimization for the first problem variant can require checking of $\left(\frac{L}{p} \times \left((p - f \times T) + 1\right)\right)^n$ combinations where n is the total number of sensor nodes to be scheduled. This will take an exponential time in the order of $\mathcal{O}(L^n)$.

In Section 4.5.1, an heuristic algorithm is proposed that incorporates a greedy approach to solve this problem variant. Unlike the straight forward optimization of exponential time that checks $\left(\frac{L}{p} \times \left((p - f \times T) + 1\right)\right)^n$ combinations, the proposed heuristic algorithm checks only $\left(n \times (p - f \times T)\right)$ combinations and works in polynomial time in the order of $\mathcal{O}(Ln)$.

4.4.1.2 Non-Consecutive execution of transactions

Now, for the second problem variant in which there is no restriction of scheduling transactions together in a period, the worst case scenario possible in this case will look as depicted in figure 4.2, which shows the aggregated traffic generated per time slot for performing transactions according to offsets mentioned in table 4.3.

According to the offset configuration mentioned in table 4.5 the aggregated traffic pattern of the four sensor nodes happens to be so that in all the $L = 6$ time slots maximum c_t is two.

In all time slots as maximum c_t = 2 there can be at most two sensor nodes that can transmit data together at any time slot and thus for the wireless medium in any time slot $BW = 2$.

It is also realized from figure 4.4 that in the time slot 2 and 4, sensor nodes s0 and s2 transmit data bits to the gateway shown in figure 4.1, at their maximum capacity, μ_t is found to be maximum for t = 4 and t = 2. Thus, the maximum channel capacity Ch_{max} (*in, bits per time slot*) that will be required in the worst case can be calculated as follows.

According to equation 4.3, considering $t = 2$
μ_2 = 32 bits/time slot,
It is evident from fig 4.4, that μ_t is maximum for t = 2 and t=4
Then, from equation 4.3 and 4.4,
Ch_{max} = 32 bits per time slot

Thus, it is evident that in case of the worst configuration in table 4.3, the four sensor nodes requires a maximum of four frequency channels for communication at the wireless interface and for the access link the maximum channel capacity required is 48 bits per time slot.

TABLE 4.5: A better offset configuration for the four IoT sensor nodes when transactions in each period are NOT performed consecutively

i	j	k	θ_{ijk}
0	0	0	0
	1	0	0
	2	0	0
1	0	0	0
	0	1	1
	1	0	0
	1	1	2
2	0	0	1
	0	1	3
3	0	0	5

Now if all the four sensor nodes obey the offset configurations mentioned in table 4.5 instead of the one mentioned in table 4.3 then the resultant aggregated traffic load pattern per time slot will be as depicted in figure 4.4.

Whereas, in case of the the better offset configuration in table 4.5 the four sensor nodes requires,
$BW = 2$, i.e., two frequency channels for communication at the wireless medium and
$Ch_{max} = 32$ bits/time slot for the access link.

Thus it can be concluded that an optimal configuration of the offsets for the second problem variant will surely minimize the bandwidth and channel capacity requirements in the wireless interface and in the access link respectively. A straight forward optimization for this problem variant can require checking of $\left(\frac{L}{p} \times \left((p - f \times T) + 1\right) \times \mathrm{T}\right)^n$ combinations where n is the total number of sensor nodes to be scheduled. This will take an exponential time in the order of $\mathcal{O}\left(L \times \mathrm{T}\right)^n\big)$.

In Section 4.5.2, an heuristic algorithm is proposed that incorporates a greedy approach to solve this problem variant. Unlike the straight forward optimization of exponential time that checks $\left(\frac{L}{p} \times \left((p - f \times T) + 1\right) \times \mathrm{T}\right)^n$ combinations, the proposed heuristic algorithm checks only $\left(n \times (p - f \times T) \times \mathrm{T}\right)$ combinations and works in polynomial time in the order of $\mathcal{O}(nLT)$.

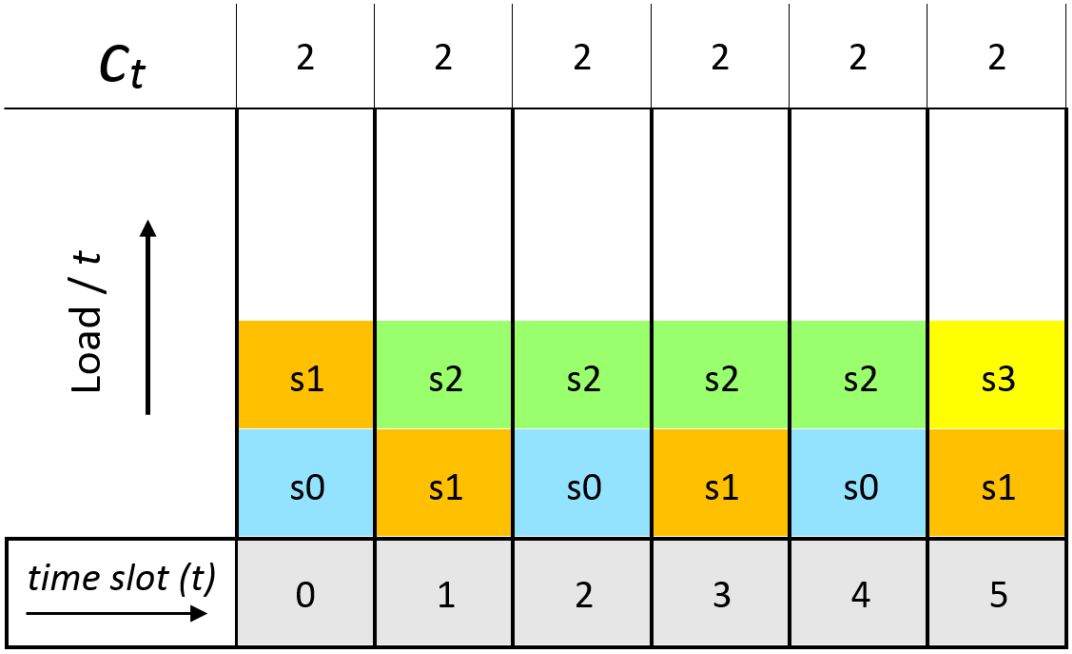


FIGURE 4.4: Aggregated traffic load per time slot from all the four sensor nodes with offset configuration mentioned in table 4.5. The Sensor nodes DOES NOT execute multiple transactions in periods (*if present*) consecutively.

4.5 Proposed Heuristics

This section is divided into two subsections,

- Heuristic solution for consecutive execution of transactions
- Heuristic solution for non-consecutive execution of transactions

4.5.1 Heuristic solution for consecutive execution of transactions

The notations mentioned in table 4.6, are used to describe the algorithm. In this section, first a proper description of the proposed heuristic approach has been depicted and then in 4.5.1.2 a pseudo code based on the proposed algorithm has been illustrated.

TABLE 4.6: Description of notations used in algorithm and pseudo code

No.	Symbol	Description
1	i	Iterator for sensor nodes
2	j	Iterator for period of a sensor node
3	k	Iterator for transaction in a period
4	θ_{ijk}	Appropriate offset for the k_{th} transaction in j_{th} period of i_{th} sensor node
5	L	LCM of all the period of all the sensor nodes
6.	**ATL**	A set of size L to store the final aggregated traffic from all the sensor nodes after local optimization process ends
7.	**AGGREGATE_ONE**	A set of size L to store traffic of any sensor node for any particular offset while finding the best offset
8.	**TEMPORARY**	A set of size L to store aggregate of **ATL** and **AGGREGATE_ONE** for any particular offset while finding the best offset
9.	**max(x)**	function that returns the maximum load in any position of list **x**
10.	**sd(x)**	function that returns the standard deviation of **x**
11.	**circularShift(x)**	function that returns the list **x** after circularly shifting its elements one position to the right

4.5.1.1 Description of Proposed Heuristic Algorithm

The following is a brief description of the algorithm that can be implemented to solve the problem mentioned in 4.4 for the variation of consecutive execution of transactions.

4.5.1.1.1 Brief Overview of the algorithm - The main backbone behind the algorithm is local optimization in each period. For a sensor node i, the optimal offset for all of its periods are found. The process involves the following.

- Local periodic optimization - Firstly, two one-dimensional list called **TEMPORARY**[q] and **ATL**[q] of size L is considered, where $0 \leq q \leq L-1$ and L is the LCM of the period of the n sensor nodes as mentioned in the previous section. Each of the L positions represents each of the L time slots. All the L position in **TEMPORARY**[q] and **ATL**[q] is initialized to be zero. **ATL** stores the total aggregated traffic load generated up till the last locally optimized sensor node.

AGGREGATE_ONE is another list of size L that stores the traffic generated for any particular offset. In **TEMPORARY**, the summation of the **AGGREGATE_ONE** generated for any of the offset choices and the current **ATL** is saved. Thus, **TEMPORARY** depicts the impact of a particular offset on the **ATL**. All the available offset choices ranging between [0, (p-fT)] are checked one by one. The most appropriate offset θ_{ij} found for the j^{th} period is selected as the offset according to which the i^{th} sensor node will start the T number of transactions in the j^{th} period. As the transactions are to be executed consecutively, it is enough just to know when the transactions should be started in any period. Thus, an offset choice θ_{ij} is considered to be the most appropriate if -

1. on adding traffic to the **TEMPORARY**[q] according to that offset θ_{ij}, the maximum aggregated traffic load among all the positions in **TEMPORARY**[q] results to be minimum and,
2. the standard deviation of all the loads in L positions of **TEMPORARY**[q] also results to be minimum.

The standard deviation is also checked because it depicts how evenly the load is distributed among the L positions in **TEMPORARY**[q]. The standard deviation of any list, say W can be calculated as in equation 4.5.

$$std(W) = \sqrt{\frac{1}{|W|}\sum_{q=0}^{L-1}(W[j] - \frac{1}{|W|}\sum_{q=0}^{|W|-1}W[q]^2)} \tag{4.5}$$

- Every j^{th} period of the i^{th} sensor node is locally optimized by finding the best offset choice for it. According to the best set of offsets obtained for all the periods, the best generated **TEMPORARY** is made the new **ATL**.
- For the next sensor node, again for every j_{th} period, the traffic generated in **AGGREGATE_ONE** for each of the offsets choices is added to the new **ATL** in **TEMPORARY**[q] in order to find the best set of offsets. The offset for which the maximum aggregated traffic load among all the positions in **TEMPORARY**[q] results to be minimum along with minimum standard deviation value of **TEMPORARY**[q], is selected as best offset.
- This entire process is done for all the n sensor nodes. And at the end of the local optimization process for all the sensor nodes, we get an offset configuration either same as the optimal configuration or the one very very close to it in terms of maximum BW and Ch_{max} requirements.

4.5.1.1.2 Comprehensive Description of the algorithm - To have a better insight of the proposed algorithm, a more detailed description of the proposed algorithm, step by step, is mentioned below.

Every θ_{ij} value returned after the following operations will be the most appropriate offset selected for j^{th} period of i^{th} sensor node.

1. Sort the every i_{th} sensor node of S in such a way that, $p_i \leq p_{i+1}$ **AND** $(f_i \times T_i) \leq (f_{i+1} \times T_{i+1})$, if $p_i = p_{i+1}$.

2. for i=0 $\rightarrow$ n-1 : // for all n sensor nodes
   a. for j=0 $\rightarrow$ $\frac{L}{p_i}$: // for all $\frac{L}{p_i}$ periods of i_{th} sensor node
      I. For every j^{th} period that starts with j^{th} time slot in L time slots, find the θ_{ij} $\in$ [0, p_i - ($f_i \times T_i$)] for which *max*(TEMPORARY) **AND** *std*(TEMPORARY) is found to be minimum after performing the following operations.
         A. for w=0 $\rightarrow$ ($f_i \times T_i$):
            i. **TEMPORARY**[($j \times p_i$) +w + θ_{ij}] = **ATL**[($j \times p_i$) + w + θ_{ij}] + 1 //incremental traffic of j^{th} period of i_{th} sensor node with offset θ_{ij}.
            ii. **end** for w
      II. Add traffic to the j^{th} period of ATL according to the most appropriate offset θ_{ij} found in the previous step.
         A. for w=0 $\rightarrow$ ($f_i \times T_i$):
            i. **ATL**[($j \times p_i$) + w + θ_{ij}] = **ATL**[($j \times p_i$) + w + θ_{ij}] + 1
            ii. **end** for w
      III. **Return** θ_{ij}
      IV. **end** for j
   b. **end** for i

4.5.1.2 Pseudo Code for the proposed algorithm

In this section, a pseudo code have been provided for better comprehensive understanding of the entire algorithm.

1. **main():**
   a. *begin* **main**
      i. **Input:** A set of n sensor nodes with values their characteristics mentioned in the tuple (i, p_i, T_i, f_i, b_i)
      ii. Sort the every i_{th} sensor node of S in such a way that, $p_i \leq p_{i+1}$ **AND** $(f_i \times T_i) \leq (f_{i+1} \times T_{i+1})$, if $p_i = p_{i+1}$.
      iii. L : *(Global variable)* Initialized with the LCM of p_i for all i=0 $\rightarrow$ n-1
      iv. **ATL**[q]:*(Global variable)* List of size L for all q=0 $\rightarrow L-1$
      v. θ[i]: List of size n, where each of index stores a list of size $\frac{L}{p_i}$ that stores the best found offset for each of the $\frac{L}{p_i}$ periods of the i^{th} sensor node.
      vi. *for* i=0 $\rightarrow$ n-1:
         θ[i] = **findBestSetOfOffsets**(p_i, $(f_i \times T_i)$)
         ATL = **addTraffic**(θ[i], p_i, $(f_i \times T_i)$)
      vii. **end** *for* i
      viii. **RETURN** θ[0 $\rightarrow$ n-1]
   b. *end* **main**

2. **findBestSetOfOffsets(**p_i**,** $(fT)_i$**:**
   a. *begin* **findBestSetOfOffsets**
   b. o[0 $\rightarrow$($\frac{L}{p_i}$ - 1)] : List of size $\frac{L}{p_i}$, stores the respective offsets for each of the $\frac{L}{p_i}$ periods. **Initialized** with 0
   c. for j=0 *to* $\frac{L}{p_i}$-1:
      i. $x = j \times p - i$ //start of period
      ii. y = ((j + 1)$\times$ $p - i$) - 1 //end of the period
      iii. o[j] = **findBestOffset**(p_i, $(fT)_i$, x, y)
      iv. **end** *for* j
   d. **RETURN** o[0 $\rightarrow$($\frac{L}{p_i}$ - 1)]
   e. *end* **findBestSetOfOffsets**

3. **findBestOffset**(p_i, $(fT)_i$, x, y):
   a. *begin* **findBestOffset**
   b. **AGGREGATE_ONE**[q] : List, the length is L, initialized to 0 $\forall$ q = 0 $\rightarrow$ *L-1*
   c. **TEMPORARY**[q] : List, the length is L, initialized to 0 $\forall$ q = 0 $\rightarrow$ *L-1*
   d. for w=0 $\rightarrow$ $(fT)_i$-1:
      i. **AGGREGATE_ONE**[x + w] = 1
      ii. **end** *for w*
   e. for q=0 $\rightarrow$ $L-1$:
      i. **TEMPORARY**[q] = **AGGREGATE_ONE**[q] + **ATL**[q]
      ii. **end** *for q*
   f. *maxLoad* = **max**(**TEMPORARY**)
   g. *sd* = **sd**(**TEMPORARY**)
   h. *Offset* = 0
   i. for t=1 $\rightarrow$ (p_i - $(fT)_i$):
      I. **AGGREGATE_ONE** = **circularShift**(**AGGREGATE_ONE**)
      II. for q=0 $\rightarrow$ $L-1$:
         i. **TEMPORARY**[q] = **AGGREGATE_ONE**[q] + **ATL**[q]
         ii. **end** *for q*
      III. *maxLoadTemp* = **max**(**TEMPORARY**)
      IV. *sdTemp* = **sd**(**TEMPORARY**)
      V. **IF** (*maxLoadTemp* $\leq$ *maxLoad* **AND** *sdTemp* < *sd*)
         THEN
         i. *maxLoad* = *maxLoadTemp*
         ii. *sd* = *sdTemp*
         iii. *Offset* = *t*
      VI. **end** for *t*
   j. **RETURN** *Offset*
   k. *end* **findBestOffset**

4. **addTraffic**(θ[i], p_i, $(fT)_i$):
   a. *begin* **addTraffic**
   b. **TRAFFIC[q]**: A list of size L to hold traffic of L slots. Initialized to 0.
   c. for j=0 $\rightarrow$ ($\frac{L}{p_i}$):
      I. for w=0 $\rightarrow$ $(fT)_i$-1:
         i. **TRAFFIC**[($j\times$ p_i) + θ[i][j]+ w] = 1
         ii. **end** for w
      II. **end** for j
   d. for q=0 $\rightarrow$ $(L-1)$:
      I. **ATL**[q] = **ATL**[q] + **TRAFFIC**[q]
      II. **end** for q
   e. **RETURN ATL**
   f. *end* **addTraffic**

4.5.1.3 Evaluation of the proposed algorithm

In this section, an example depicted in 4.7 has been considered. The traffic of sensor nodes 0 and 1 have been represented by blue and orange colour respectively. According to the algorithm depicted in the previous section, the evaluation of the

TABLE 4.7: Two IoT sensor nodes - sensor 0 ($p_0 = 3$, $T_0 = 1$, $f_0 = 1$) and sensor 1 ($p_1 = 6$, $T_1 = 1$, $f_1 = 2$)

i	p_i	T_i	f_i
0	3	1	1
1	6	1	2

example shown in table 4.7 is as follows.

The LCM of the periods of the sensor nodes 0 and 1 happens to be 6. Thus in figure 4.5 six times positions have been shown for ATL, $TEMPORARY$, and $AGGREGATE\_ONE$. In figure 4.5, the process of local optimization is depicted for the period 0 of sensor node 0. After checking all the offset choices offset 0 was found to be most appropriate for period 0 of the sensor node 0. In figure 4.6, the process of local optimization is depicted for the period 1 of sensor node 0. After checking all the offset choices offset 0 was found to be most appropriate for period 0 of the sensor node 0. Thus, the traffic of the sensor node 0 is added to the **ATL** as depicted in figure 4.10.

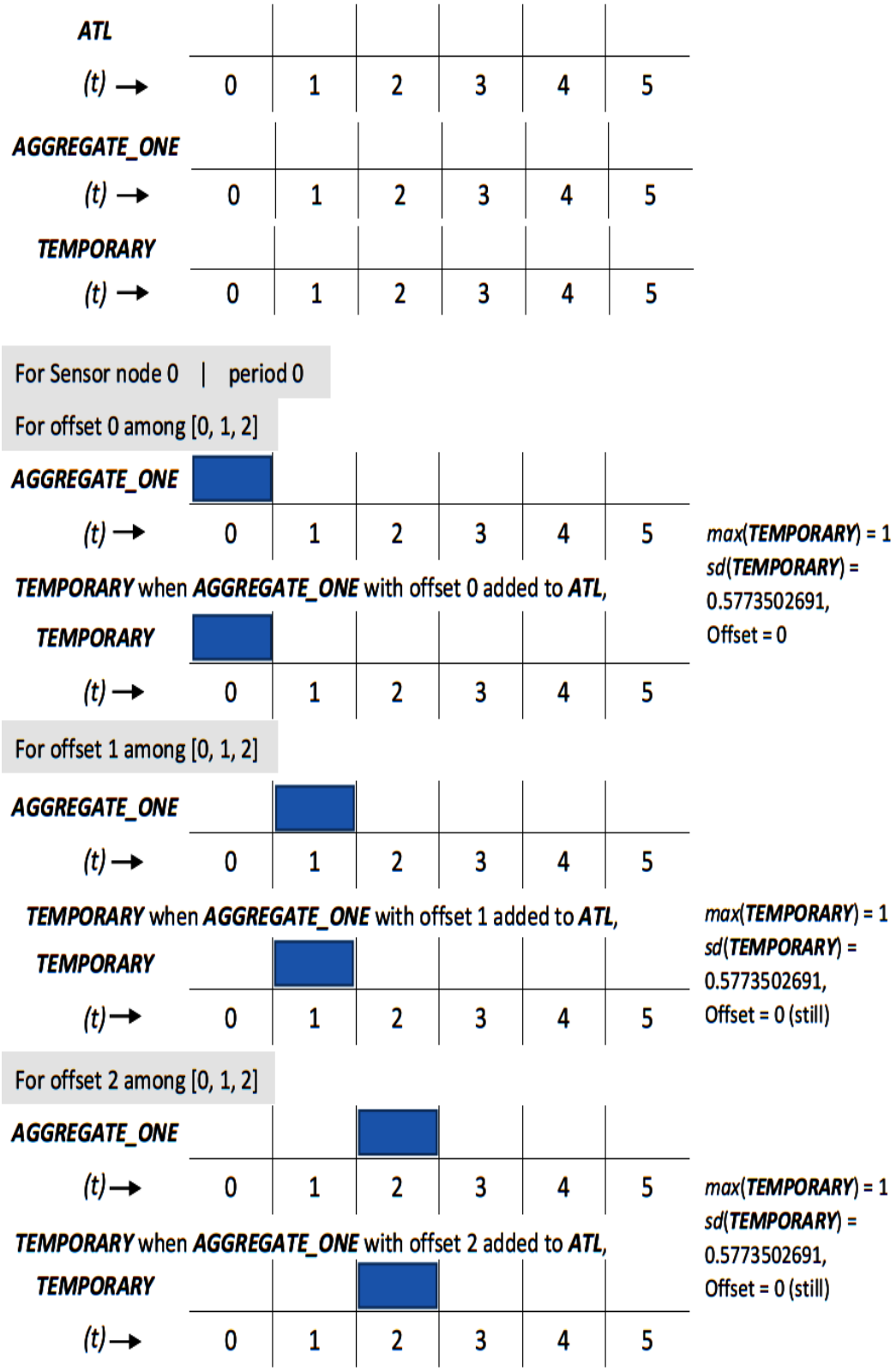


FIGURE 4.5: Local optimization of period 0 of sensor node 0

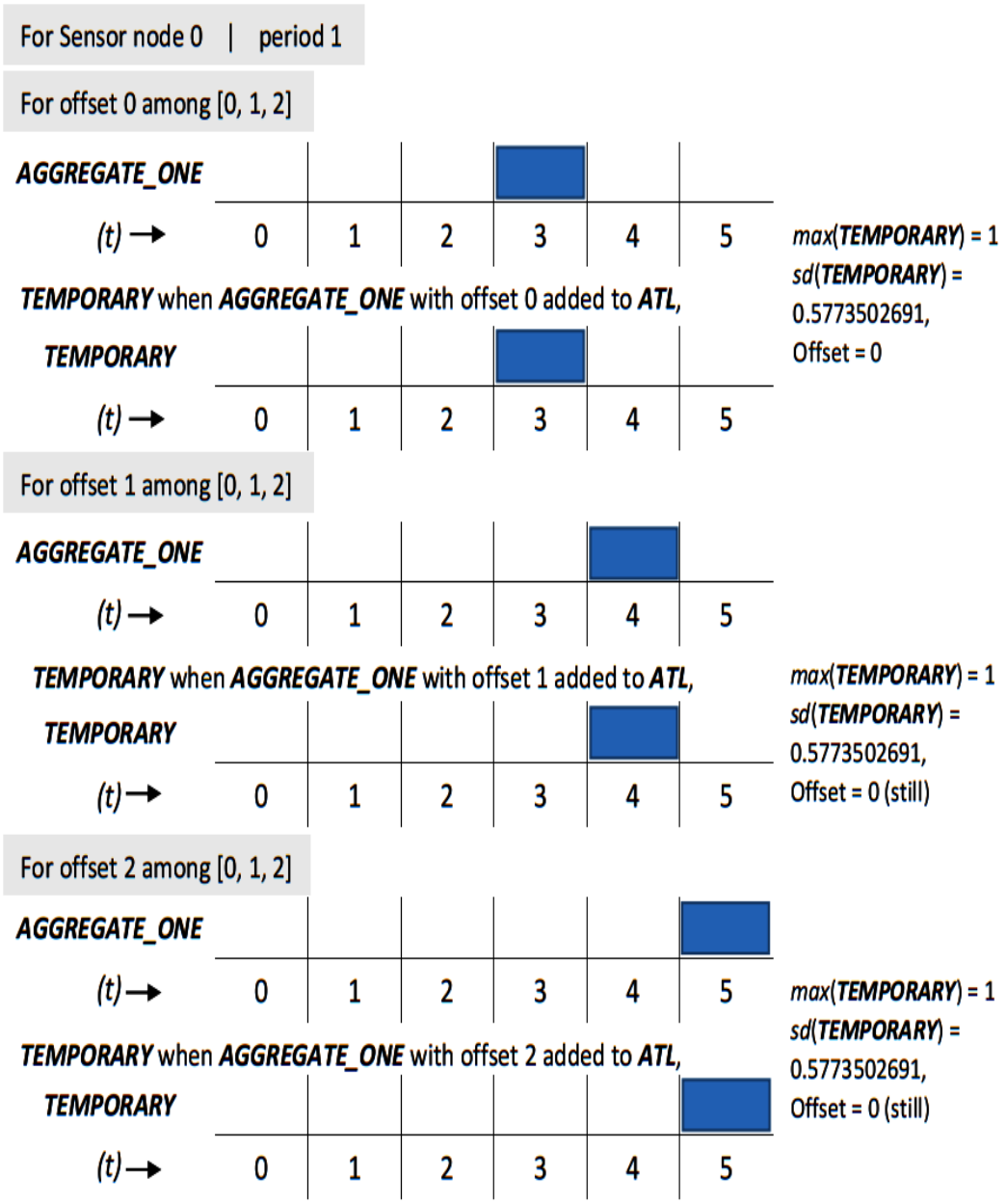


FIGURE 4.6: Local optimization of period 1 of sensor node 0

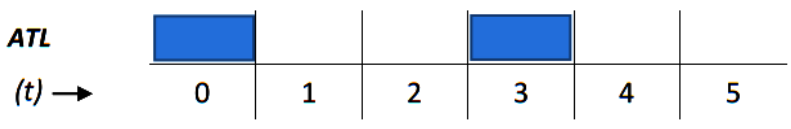


FIGURE 4.7: ATL after sensor node 0 optimized

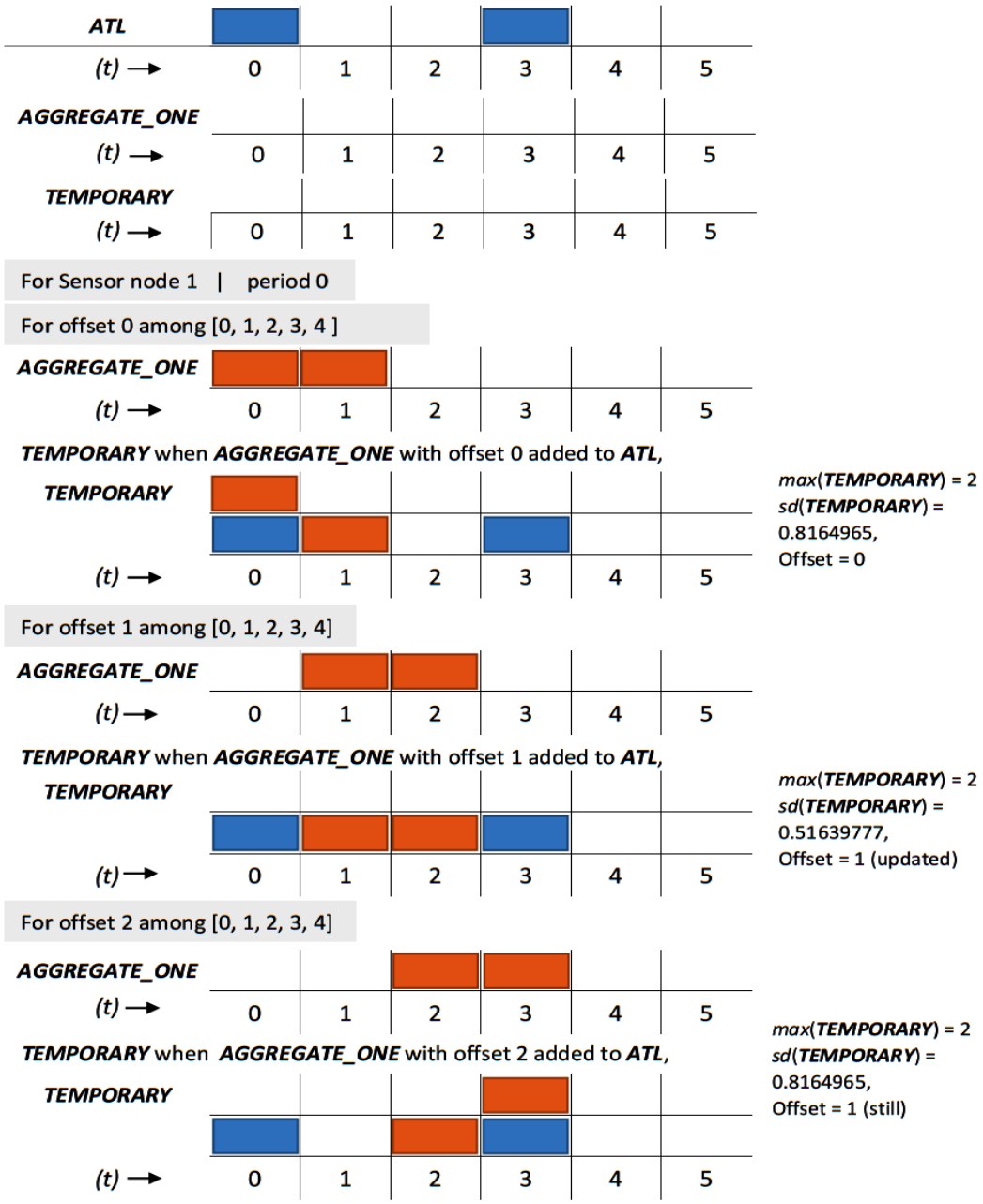


FIGURE 4.8: Local optimization process in period 0 of sensor node 1 in case of offsets 0, 1 and 2

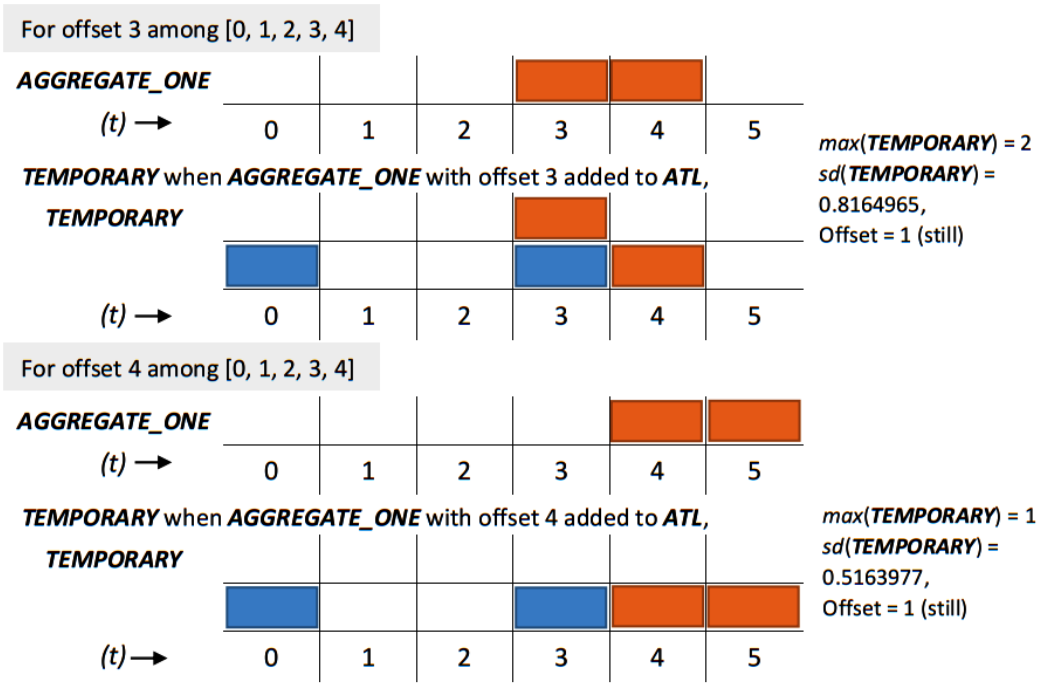


FIGURE 4.9: Local optimization process in period 0 of sensor node 1 in case of offsets 3 and 4

In figure 4.6, the local optimization process for the period 1 of sensor node 0 has been depicted. In figure 4.8, the local optimization process for the period 0 of sensor node 1 has been represented while the offsets 0, 1 and 2 are checked. In figure 4.9, the local optimization process for the period 0 of sensor node 1 has been represented while the offsets 3 and 4 are checked.In figure 4.10 the ATL status is depicted after optimizing sensor node 0 and sensor node 1.

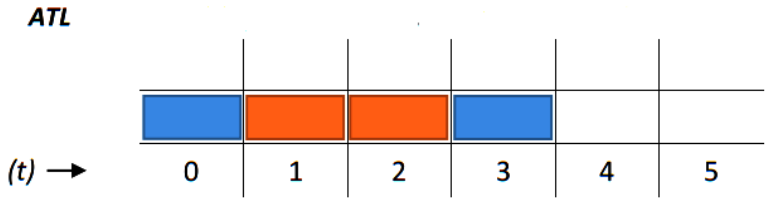


FIGURE 4.10: ATL after sensor node 0 optimized

Thus in the table 4.8, the set of offsets achieved from the algorithm for each period of each sensor node has been enlisted.

Sensor node	period	offset
0	0	0
0	1	0
1	0	1

TABLE 4.8: The set of offsets achieved from the evaluation in figure 4.5, 4.6, 4.8 and 4.9

4.5.2 Heuristic solution for non-consecutive execution of transactions

The notations mentioned in table 4.6, are used to describe the algorithm. In this section, first a proper description of the proposed heuristic approach has been depicted and then in 4.5.2.2 a pseudo code based on the proposed algorithm has been illustrated.

4.5.2.1 Description of Proposed Heuristic Algorithm

The following is a brief description of the algorithm that can be implemented to solve the problem mentioned in 4.4 for the variation of non-consecutive execution of transactions.

4.5.2.1.1 Brief Overview of the algorithm - The main backbone behind the algorithm is local optimization for each transaction in each period. For a sensor node i, the most appropriate offset (in number of time slots) for every k^{th} transaction in every j^{th} period can be found through the algorithm described below.

- *Local optimization for each transaction in a period* - This process is meant to find the best possible offset for any transaction in a period. It focuses on locally optimizing each transaction in each period one by one. Firstly, two one-dimensional list called **TEMPORARY**[q] and **ATL**[q] of size L is considered, where $0 \leq q \leq L-1$ and L is the LCM of the period of the n sensor nodes as mentioned in the previous section. Each of the L positions represents each of the L time slots. All the L position in **TEMPORARY**[q] and **ATL**[q] is initialized to be zero. **ATL** stores the total aggregated traffic load generated up till the last locally optimized sensor node. **AGGREGATE_ONE** is another list of size L that stores the traffic generated for any particular offset. To optimize a transaction in a period the summation of the **AGGREGATE_ONE** generated for an offset choice and the current **ATL** is saved in **TEMPORARY** and checked if that offset choice is best in terms of the maximum amount of load it generates per position of the **TEMPORARY**. Thus, **TEMPORARY** depicts the impact of a particular offset on the **ATL**. All the available offset choices ranging between [0, (p-fT)] are checked in this way, one by one. The most appropriate offset θ_{ijk} found for the k^{th} transaction in j^{th} period is selected as the offset according to which the i^{th} sensor node will start the k^{th} transaction among the T transactions, in the j^{th}

period. As the transactions are to not to be executed consecutively, it must be known that when a k^{th} transaction should be started in any period.

An offset choice θ_{ijk}, is considered to be the most appropriate if -

1. on adding traffic generated in **TEMPORARY**[q] according to that offset θ_{ijk} to the **ATL**, the maximum aggregated traffic load among all the positions in **ATL** results to be minimum and,
2. the standard deviation of all the loads in L positions of **ATL** also results to be minimum.

The standard deviation, according to equation 4.5, is also checked because it depicts how evenly the load is distributed among the L positions in **TEMPORARY**. After checking all the available options for θ_{ijk} the most appropriate offset θ_{ijk} found for the k^{th} transaction of j^{th} period, from the available range, is selected as the offset according to which the i^{th} sensor node will start the k^{th} transaction among the T number of transactions to be done in the j^{th} period. This process is repeated for all the transactions in all the periods of any sensor node. After finding the best θ_{ijk} for a transaction, the **TEMPORARY** so generated is made new **ATL**.

- For the next sensor node, again for every k^{th} transaction in every j_{th} period, the traffic generated for each of the offsets choices is added to the **TEMPORARY**[q] and checked one by one by adding **TEMPORARY**[q] with the **ATL**. This entire process is done for all the n sensor nodes. And at the end of the local optimization process for all the sensor nodes, we get an offset configuration either same as the optimal configuration or the one very very close to it in terms of maximum BW and Ch_{max} requirements. It is quite evident that if a transaction k of size f_i time slots is scheduled to start at a time slot t in any period then the next transaction of that period can only start at $(t + f_i)^{th}$ time slot.

4.5.2.1.2 Comprehensive Description of the algorithm - To have a better insight of the proposed algorithm, a more detailed description of the proposed algorithm, step by step, is mentioned below.

Every θ_{ijk} value returned after the following operations will be the most appropriate offset selected for the k^{th} transaction in j^{th} period of i^{th} sensor node.

1. Sort the every i_{th} sensor node of S in such a way that, $p_i \leq p_{i+1}$ **AND** $(f_i \times T_i) \leq (f_{i+1} \times T_{i+1})$, if $p_i = p_{i+1}$.

2. for i=0 $\rightarrow$ n-1 : // for all n sensor nodes
   a. for j=0 $\rightarrow$ $\frac{L}{p_i}$: // for all $\frac{L}{p_i}$ periods of i_{th} sensor node
      I. for k=0 $\rightarrow$ T_i: //for each of the transactions
         A. For every k^{th} transaction, find the
         $\theta_{ijk} \in \left[0, \left(p_i - \left((T_i - k) \times f_i\right)\right)\right]$ for which max(TEMPORARY) **AND** sd(TEMPORARY) is found to be minimum after performing the following operations.
            1. for w=0 $\rightarrow$ f_i:
               i. **TEMPORARY**$[(j \times p_i) + w + \theta_{ijk}] =$
               ATL$[(j \times p_i) + w + \theta_{ijk}] + 1$ //incremental traffic of j^{th} transaction in j^{th} period of i_{th} sensor node with offset θ_{ij}.
               ii. **end** for w
         B. Add traffic of the k^{th} transaction to the j^{th} period of ATL according to the most appropriate offset θ_{ijk} found in the previous step.
            1. for w=0 $\rightarrow$ f_i:
               i. **ATL**[$(j \times p_i) + w + \theta_{ijk}$] = **ATL**[$(j \times p_i) + w + \theta_{ijk}$] + 1
               ii. **end** for w
         C. **Return** θ_{ijk}
         D. **end** for k
      II. **end** for j
   b. **end** for i

4.5.2.2 Pseudo Code for the proposed algorithm

In this section, a pseudo code have been provided for better comprehensive understanding of the entire algorithm.

1. **main():**
   a. *begin* **main**
      i. **Input:** A set of n sensor nodes with values their characteristics mentioned in the tuple (i, p_i, T_i, f_i, b_i)
      ii. Sort the every i_{th} sensor node of S in such a way that, $p_i \leq p_{i+1}$ **AND** $(f_i \times T_i) \leq (f_{i+1} \times T_{i+1})$, if $p_i = p_{i+1}$.
      iii. L : *(Global variable)* Initialized with the LCM of p_i for all i=0 $\rightarrow$ n-1
      iv. **ATL**[q]: *(Global variable)* One-dimensional list of size L for all q=0 $\rightarrow$ $L-1$
      v. θ[i]: List of size n, where each of index stores a list of size $\frac{L}{p_i}$ that further stores a list of T_i offset for each of the T_i transactions in each of the $\frac{L}{p_i}$ periods of the i^{th} sensor node.
      vi. for i=0 $\rightarrow$ n-1:
         A. θ[i] = **findBestSetOfOffsetsForSensorNode**(p_i, f_i, T_i)
         B. for j=0 $\rightarrow$ $\frac{L}{p_i}$-1:
            I. **ATL** = **addTraffic**(θ[i][j], p_i, f_i, T_i, j)
            II. **end** for j
         C. **end** for i
      vii. **RETURN** θ[0 $\rightarrow$ n-1]
   b. *end* **main**

2. **findBestSetOfOffsetsForSensorNode**(p_i, f_i, T_i):
   a. *begin* **findBestSetOfOffsetsForSensorNode**
   b. o[0 $\rightarrow$($\frac{L}{p_i}$ - 1)] : List of size $\frac{L}{p_i}$, stores the respective set of offsets for each of the $\frac{L}{p_i}$ periods. **Initialized** with 0
   c. for j=0 *to* $\frac{L}{p_i}$-1:
      i. $x = j \times p - i$ //start of period
      ii. $y = ((j+1) \times p - i)$ - 1 //end of the period
      iii. o[j] = **findBestSetOfOffsetForPeriod**(p_i, f_i, T_i, x, y)
      iv. **end** for j
   d. **RETURN** o[0 $\rightarrow$($\frac{L}{p_i}$ - 1)]
   e. *end* **findBestSetOfOffsetsForSensorNode**

3. **findBestSetOfOffsetForPeriod**(p_i, f_i, T_i, x, y):
   a. *begin* **findBestSetOfOffsetForPeriod**
   b. o[0 *to* T_i-1] : List of size T_i that stores the respective offsets for each of the T_i transactions. **Initialized** with 0
   c. for k=0 *to* T_i-1:
      i. *offsetForPreviousTransaction* ← Set the offset found for the previous transaction as the current transaction must be allocated after the previous transaction
      ii. $o[k]$ = **findBestOffset**(*offsetForPreviousTransaction*, p_i, f_i, T_i, x, y, k)
      iii. **end** for k
   d. **RETURN** o[0 *to* T_i-1]
   e. *end* **findBestSetOfOffsetForPeriod**

4. **findBestOffset**(*offsetForPreviousTransaction*, p_i, f_i, T_i, x, y, k):
   a. *begin* **findBestOffset**
   b. **AGGREGATE_ONE**[q] : one-dimensional list, the length is L, initialized to 0 $\forall$ $q = 0 \rightarrow L\text{-}1$
   c. **TEMPORARY**[q] : one-dimensional list, the length is L, initialized to 0 $\forall$ $q = 0 \rightarrow L\text{-}1$
   d. for w=0 $\rightarrow$ f_i-1:
      i. **AGGREGATE_ONE**[$x + w +$ *offsetForPreviousTransaction* $+ f_i$] = 1
      ii. **end** *for w*
   e. for q=0 $\rightarrow L-1$:
      i. **TEMPORARY**[q] = **AGGREGATE_ONE**[q] + **ATL**[q]
      ii. **end** *for q*
   f. *maxLoad* = **max**(**TEMPORARY**)
   g. *sd* = **sd**(**TEMPORARY**)
   h. *Offset* = *offsetForPreviousTransaction* + f_i
   i. for t=(*offsetForPreviousTransaction* + f_i + 1) $\rightarrow \left(p_i - \left((T_i\text{-}k) \times f_i\right)\right)$:
      I. **AGGREGATE_ONE** = **circularShift**(**AGGREGATE_ONE**)
      II. for q=0 $\rightarrow L-1$:
         i. **TEMPORARY**[q] = **AGGREGATE_ONE**[q] + **ATL**[q]
         ii. **end** *for q*
      III. *maxLoadTemp* = **max**(**TEMPORARY**)
      IV. *sdTemp* = **sd**(**TEMPORARY**)
      V. **IF** (*maxLoadTemp* $\leq$ *maxLoad* **AND** *sdTemp* < *sd*)
         THEN
         i. *maxLoad* = *maxLoadTemp*
         ii. *sd* = *sdTemp*
         iii. *Offset* = t
      VI. **end** for t

j. **RETURN** *Offset*
k. *end* **findBestOffset**

4. **addTraffic**($\theta[i][j]$, p_i, f_i, T_i, j):
a. *begin* **addTraffic**
b. **TRAFFIC[*q*]**: A list of size L to hold traffic of L slots. Initialized to 0.
c. for k=0 $\rightarrow$ (T_i-1): // for each transaction
I. for w=0 $\rightarrow$ (f_i-1):
i. **TRAFFIC**[($j\times$ p_i) + θ[i][j][k]+ w] = 1
ii. **end** for w
II. **end** for k
d. for q=0 $\rightarrow$ ($L-1$):
I. **ATL**[q] = **ATL**[q] + **TRAFFIC**[q]
II. **end** for q
e. **RETURN ATL**
f. *end* **addTraffic**

4.5.2.3 Evaluation of the proposed algorithm

In this section, an example depicted in 4.9 has been considered. The traffic of sensor nodes 0 and 1 have been represented by blue and orange colour respectively. According to the algorithm depicted in the previous section, the evaluation of the

TABLE 4.9: Two IoT sensor nodes - sensor 0 ($p_0 = 3$, $T_0 = 1$, $f_0 = 1$) and sensor 1 ($p_1 = 6$, $T_1 = 1$, $f_1 = 2$)

i	p_i	T_i	f_i
0	3	1	1
1	6	2	2

example shown in table 4.9 is as follows.

The LCM of the periods of the sensor nodes 0 and 1 happens to be 6. Thus in figure 4.11 six times positions have been shown for ATL, $TEMPORARY$, and $AGGREGATE\_ONE$. In the figure 4.11, 0^{th} transaction of the 0^{th} period of the 0^{th} sensor node has been locally optimized and it is found that the 0^{th} transaction should start with an offset of 0.

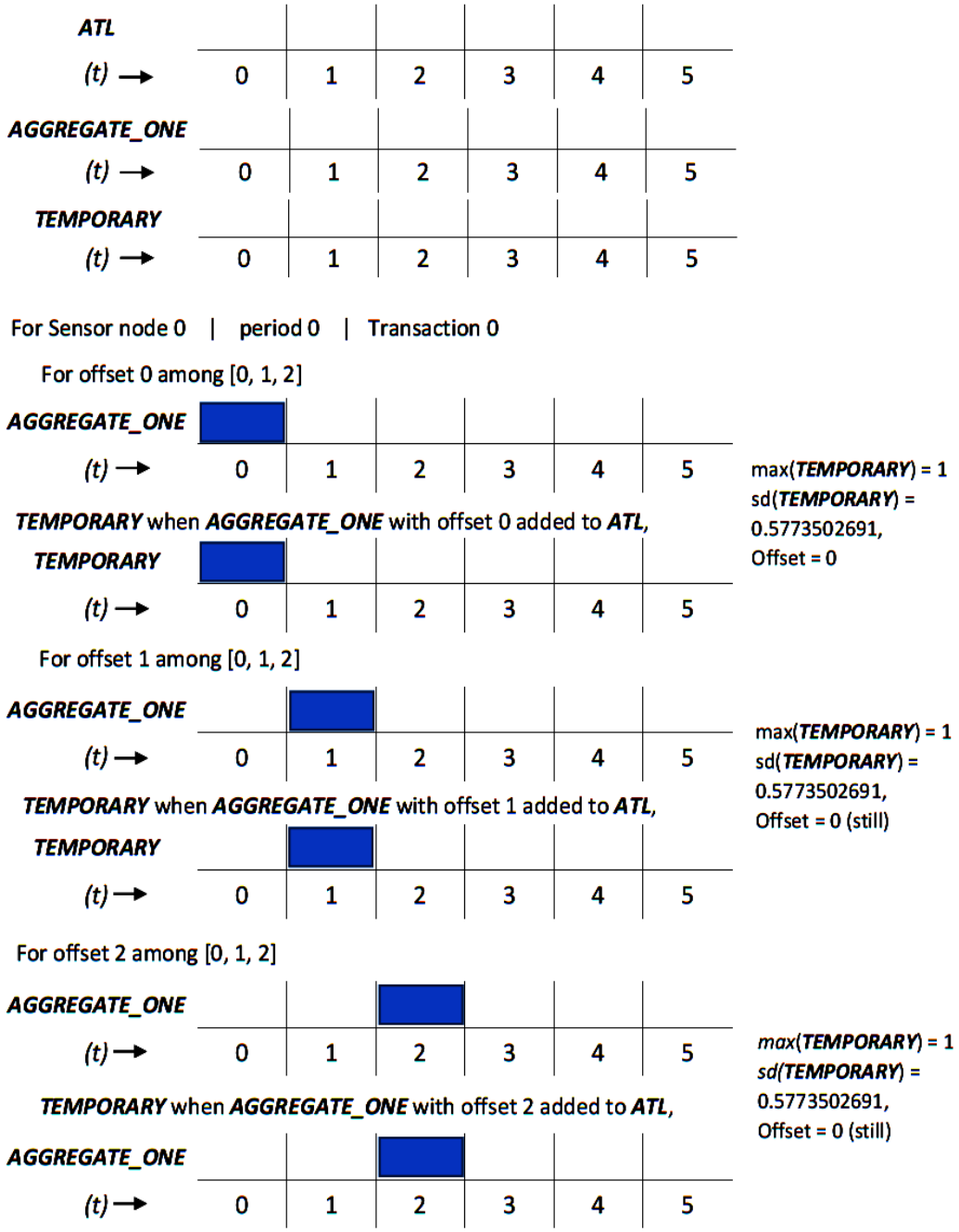


FIGURE 4.11: Local optimization process for 0^{th} transaction of 0^{th} period of 0^{th} sensor node

For the next 1^{th} period of 0^{th} sensor node the local optimization process for the 0^{th} transaction has been shown in figure 4.12.

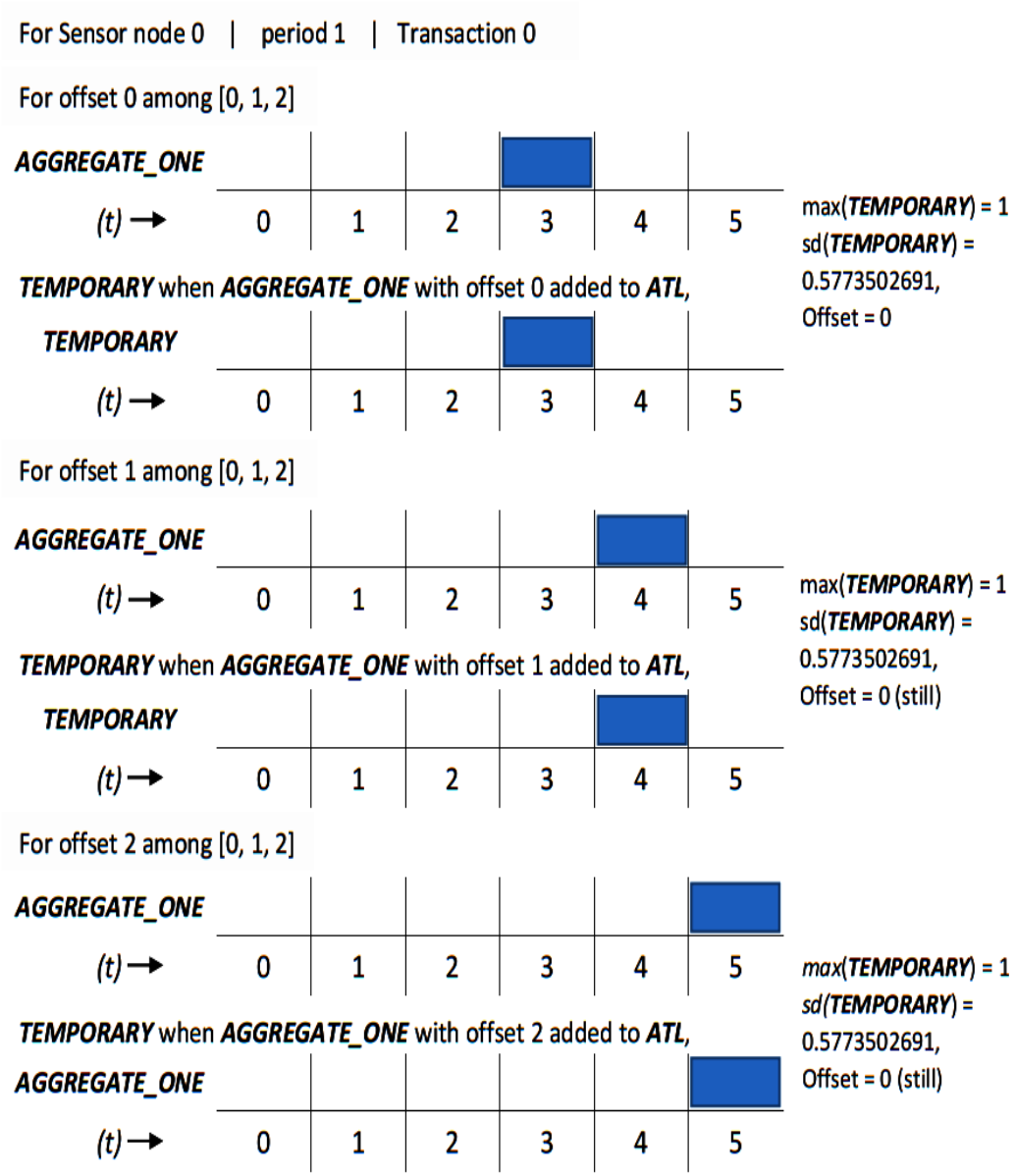


FIGURE 4.12: Local optimization process for 0^{th} transaction of 1^{th} period of 0^{th} sensor node

At the end of the local optimization process for 0^{th} sensor node, the offsets found for the 0^{th} transaction in 0^{th} period and 0^{th} transaction in 1^{th} period is same, i.e. 0. When the **ATL** gets updated by the traffic pattern of the 0^{th} sensor node it looks like the one depicted in figure 4.13.

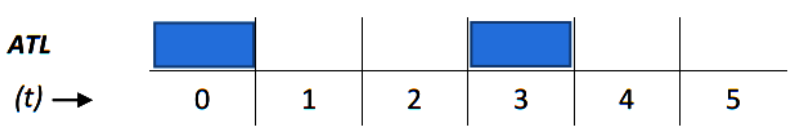


FIGURE 4.13: ATL after transaction 0 of period 0 and 1 of sensor node 0 have been locally optimized

For the sensor node 1, the local optimization process for the 0^{th} transaction has been depicted in figure 4.14. For the 0^{th} transaction, the offset values that should be checked only are 0, 1 and 2. Offset 3 or 4 cannot be an offset for transaction 0 as there is another transaction 1 that is also required to be allocated. If transaction 0 is scheduled with offset 3 to start at the time slot 3 among the six time slots then it is evident that transaction 0 will end at time slot 4. This will leave only one time slot,time slot 5, at the end which is not sufficient to allocate the second transaction that is also required to be done. Again if transaction 0 is scheduled with offset 4 to start at the time slot 4 among the six time slots then it is evident that transaction 0 will end at time slot 5. This will leave no time slot at the end to allocate the second transaction that is also required to be done. Thus, for the 0^{th} transaction of sensor node 1, the best offset found is 1. For the second transaction to be done there are two choices for offsets that are available (3 and 4). The local optimization process for allocating the second transaction is done in figure 4.15.

For offset 3, the maximum load on the six time slots becomes 2 whereas for offset option 4, the transaction gets allocated at the time slot 4 and 5 and thus the maximum load on the 6 time slots is found to be 1. Thus for the second transaction offset 4 is selected as the most appropriate offset.

In figure 4.16, the aggregated traffic of sensor node 0 and 1 has been depicted for the best set of offsets found for each transaction as illustrated in 4.10.

Sensor	period	transaction	offset
0	0	0	0
0	1	0	0
1	0	0	1
1	0	1	4

TABLE 4.10: The set of offsets achieved from the evaluation in figure 4.11, 4.12, 4.14 and 4.15

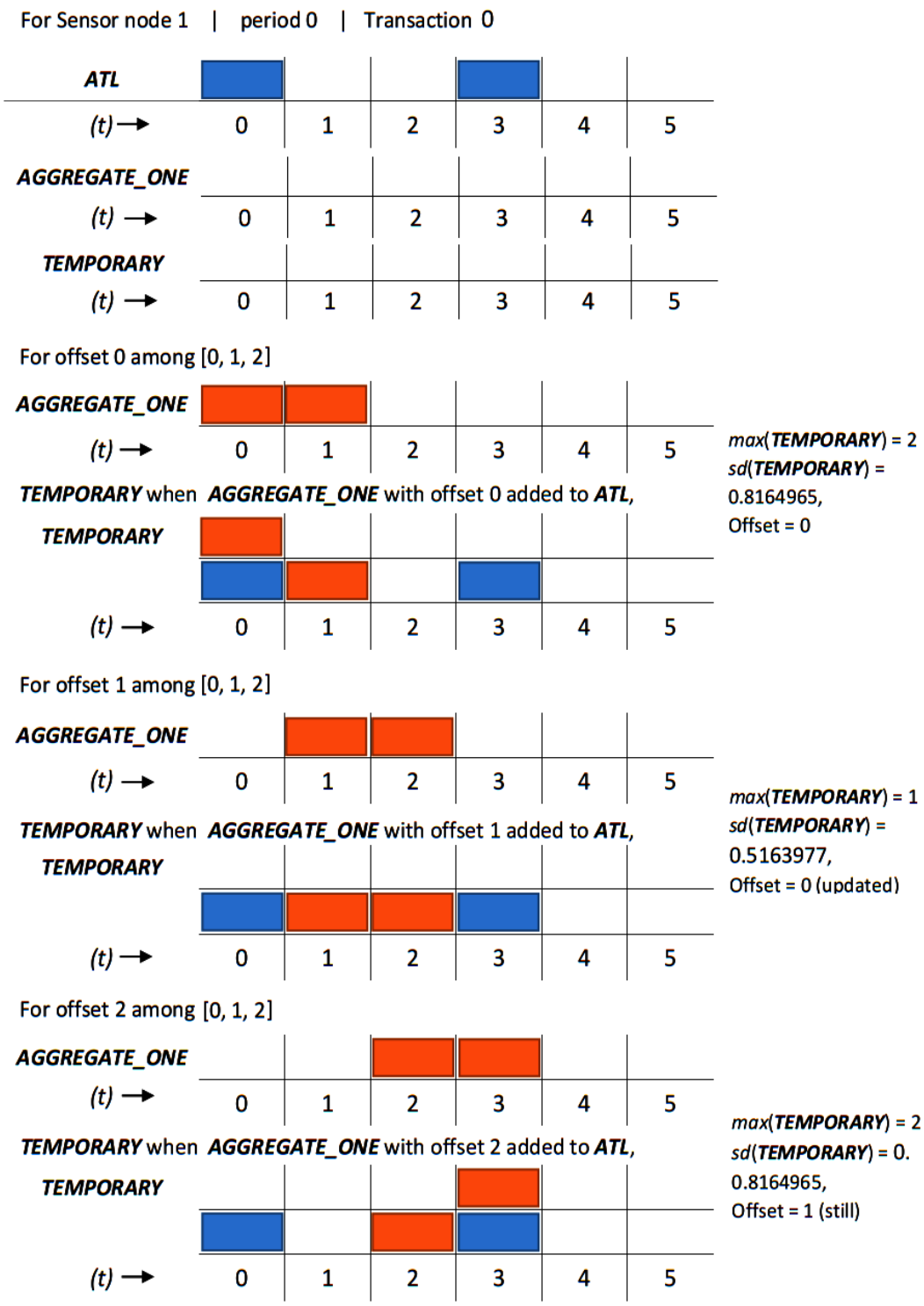


FIGURE 4.14: The local optimization process for the 0^{th} transaction of sensor node 1

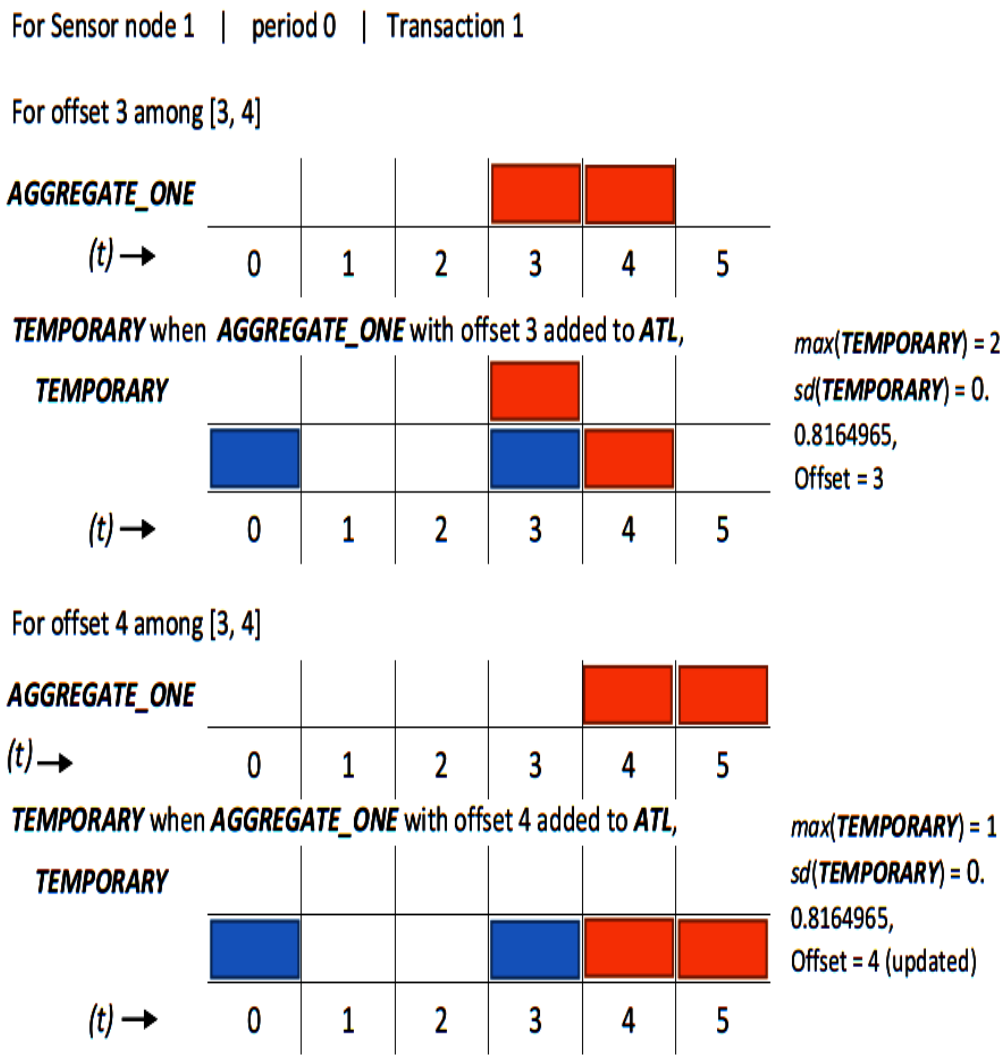


FIGURE 4.15: The local optimization process for allocating the second transaction of sensor node 1

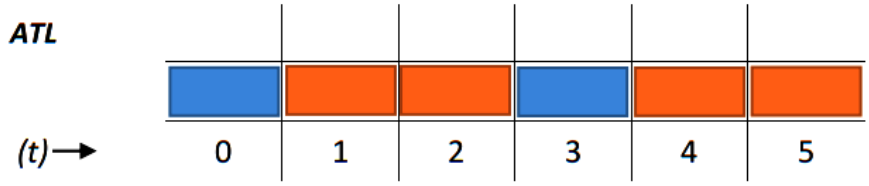


FIGURE 4.16: The aggregated traffic in *ATL* at the end of the local optimization process of sensor node 0 and 1

4.6 Results And Discussions

In this section, there are two subsections, 4.6.1 and 4.6.2. In section 4.6.1,

4.6.1 Results in case of consecutive execution of transactions

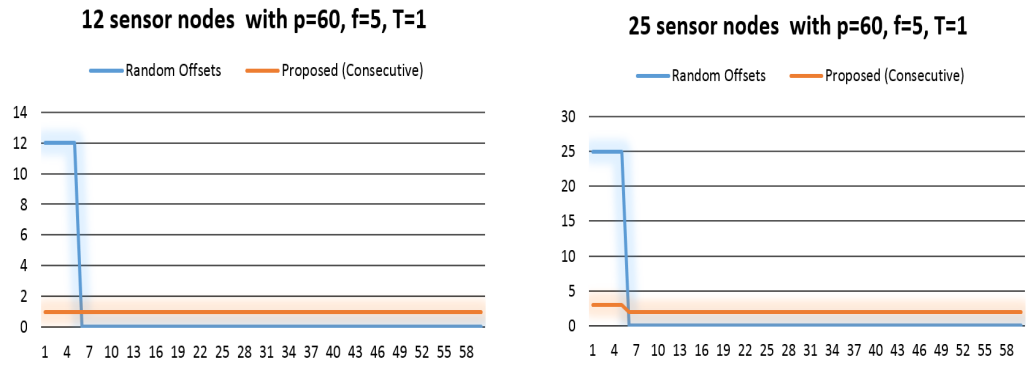
Figure, 4.17a, 4.17b, 4.18a, 4.18b, 4.19a, 4.19b, 4.20a, 4.20b depicts comparison of the proposed (Pro.) with the case of random offset (Rand.) for 12, 25, 50, 100, 250, 500, 1000 and 2000 sensor nodes having p=60, f=5, T=1 and b=8 bits, respectively. The proposed algorithm (*in blue*) has been compared with the case when random set of offsets (*in orange*) are chosen. With growing size of the sensor network, the increase in resource requirements for both Ch_{max} and BW in case of Proposed and Random Offset has been depicted in figure 4.21 and 4.22 respectively.

In figure 4.23, a set of sensor nodes with the specifications mentioned in the table 4.11 have been considered. Herein, the sensor nodes are heterogeneous to each other in nature.

Sensor Nodes	p	T	f	b
$1-12$	12	1	5	8
$12-50$	15	2	5	8
$51-100$	30	4	5	8
$101-500$	60	6	5	8
$501-1000$	60	7	5	8
$1001-2000$	120	8	10	8

TABLE 4.11: A set of 2000 IoT sensor nodes

FIGURE 4.17: Proposed V/s Random Offset in terms of increasing sensor nodes for 12 and 25 sensor nodes



(A) Pro./Rand.: $Ch_{max}(8/96)$, $BW(1/12)$

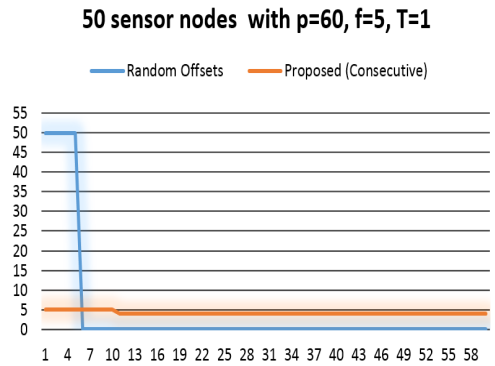
(B) Pro./Rand.: $Ch_{max}(24/200)$, $BW(3/25)$

Figure 4.17a shows the value of Ch_{max} = 8 bits per time slot and BW = 1 which means that a maximum of one frequency channel is required according to the schedule obtained by means of the proposed algorithm. But, on the other hand it can be

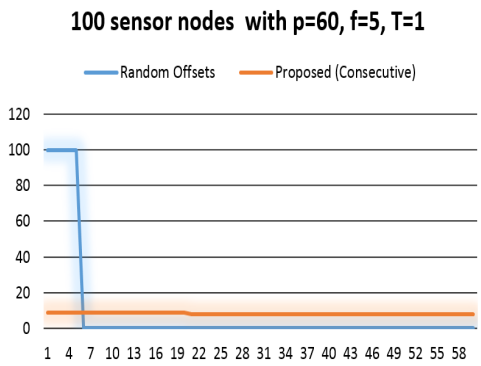
seen that due to a random offset combination the maximum number of sensor nodes that has been scheduled to transmit in any particular time slot is twelve. Which means all the 12 sensor nodes have been scheduled to transmit together and thus twelve frequency channel would be required. Therefore, for the case of random offsets Ch_{max} = 96 bits per time slot and BW=12.

Figure 4.17b shows the value of Ch_{max} = 24 bits per time slot and BW = 3 which means that a maximum of 3 frequency channel is required according to the schedule obtained by means of the proposed algorithm. But, on the other hand it can be seen that due to a random offset combination the maximum number of sensor nodes that has been scheduled to transmit in any particular time slot is 25. Which means all the 25 sensor nodes have been scheduled to transmit together and thus 25 frequency channel would be required. Therefore, for the case of random offsets Ch_{max} = 200 bits per time slot and BW=25. Similar picture have been depicted in 4.18a, 4.18b, 4.19a, 4.19b, 4.20a and 4.20b in case of 50, 100, 250, 500, 1000 and 2000 sensor nodes respectively.

FIGURE 4.18: Proposed V/s Random Offset in terms of increasing sensor nodes for 50 and 100 sensor nodes

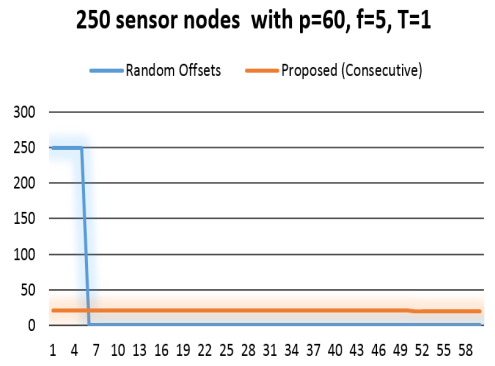


(A) Pro./Rand.: Ch_{max}(40/400), BW(5/50)

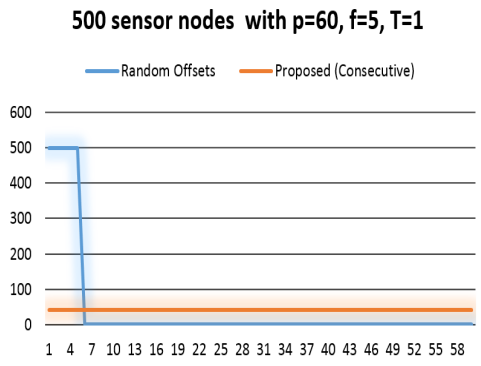


(B) Pro./Rand.: Ch_{max}(72/800), BW(9/100)

FIGURE 4.19: Proposed V/s Random Offset in terms of increasing sensor nodes for 250 and 500 sensor nodes

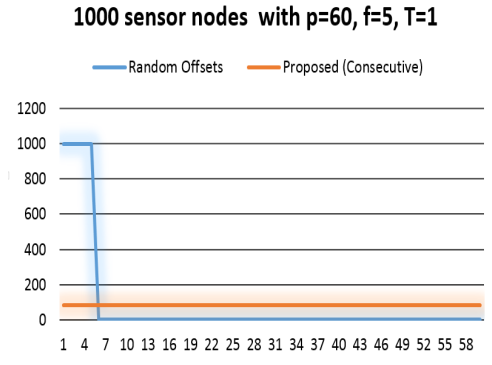


(A) Pro./Rand.: Ch_{max}(168/2K), BW(21/250)

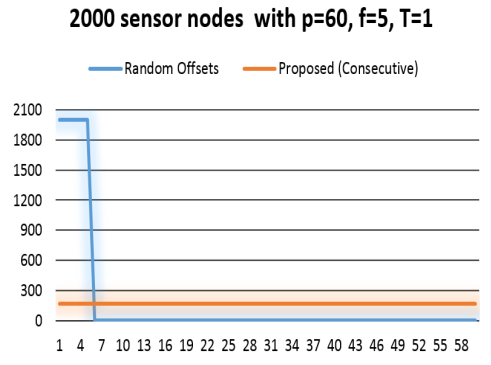


(B) Pro./Rand.: Ch_{max}(336/4K), BW(42/500)

FIGURE 4.20: Proposed V/s Random Offset in terms of increasing sensor nodes for 1000 and 2000 sensor nodes



(A) Pro./Rand.: Ch_{max}(672/8K), BW(84/1K)



(B) Pro./Rand.: Ch_{max}(1336/16K), BW(167/2K)

In figure 4.20, 4.19, 4.18 and 4.17 all the sensor nodes are of homogeneous type. Such an experiment is done to observe the performance of the proposed algorithm if the number of sensor nodes n is increased exponentially.

In figure 4.21, the Ch_{max} has been observed in case of the of the increasing size of the sensor network. The number of sensor nodes n has been increased exponentially to observe the results of the proposed algorithm. From this experiment it was found that the proposed algorithm schedules way better than the one in case of random offset combination. In the same way the behavior of the BW have also been observed in figure 4.22.

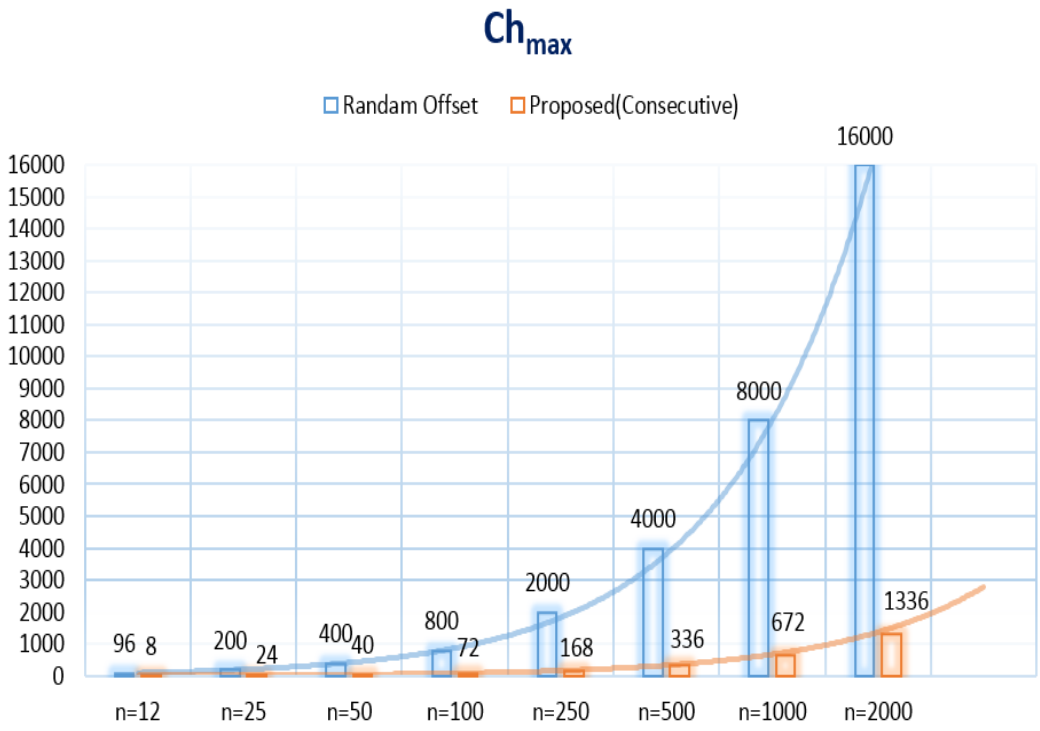


FIGURE 4.21: Increase in Ch_{max} with increasing number of sensor nodes

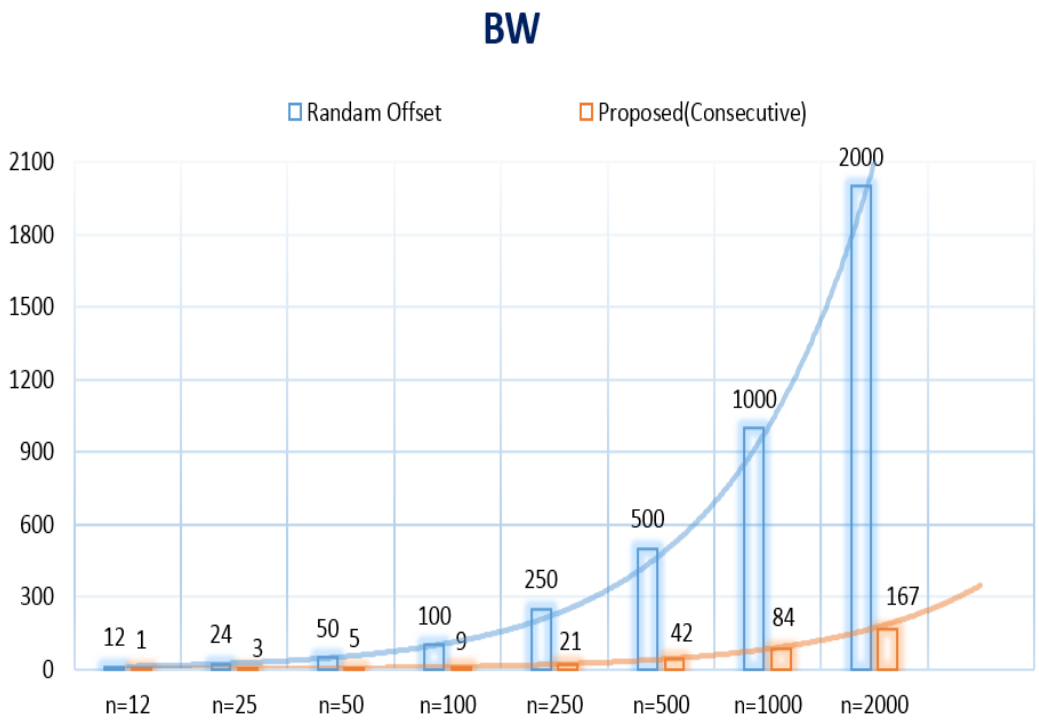


FIGURE 4.22: Increase in *BW* with increasing number of sensor nodes

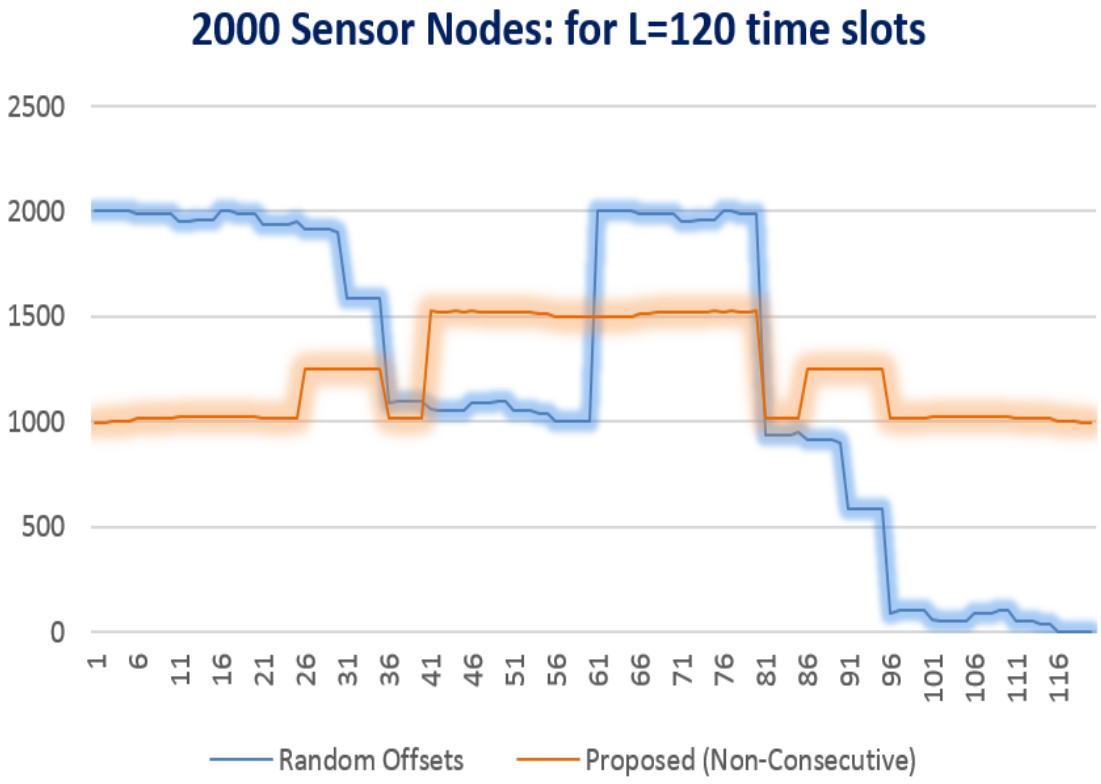


FIGURE 4.23: Aggregated traffic load per time slot in case of consecutive execution of transactions

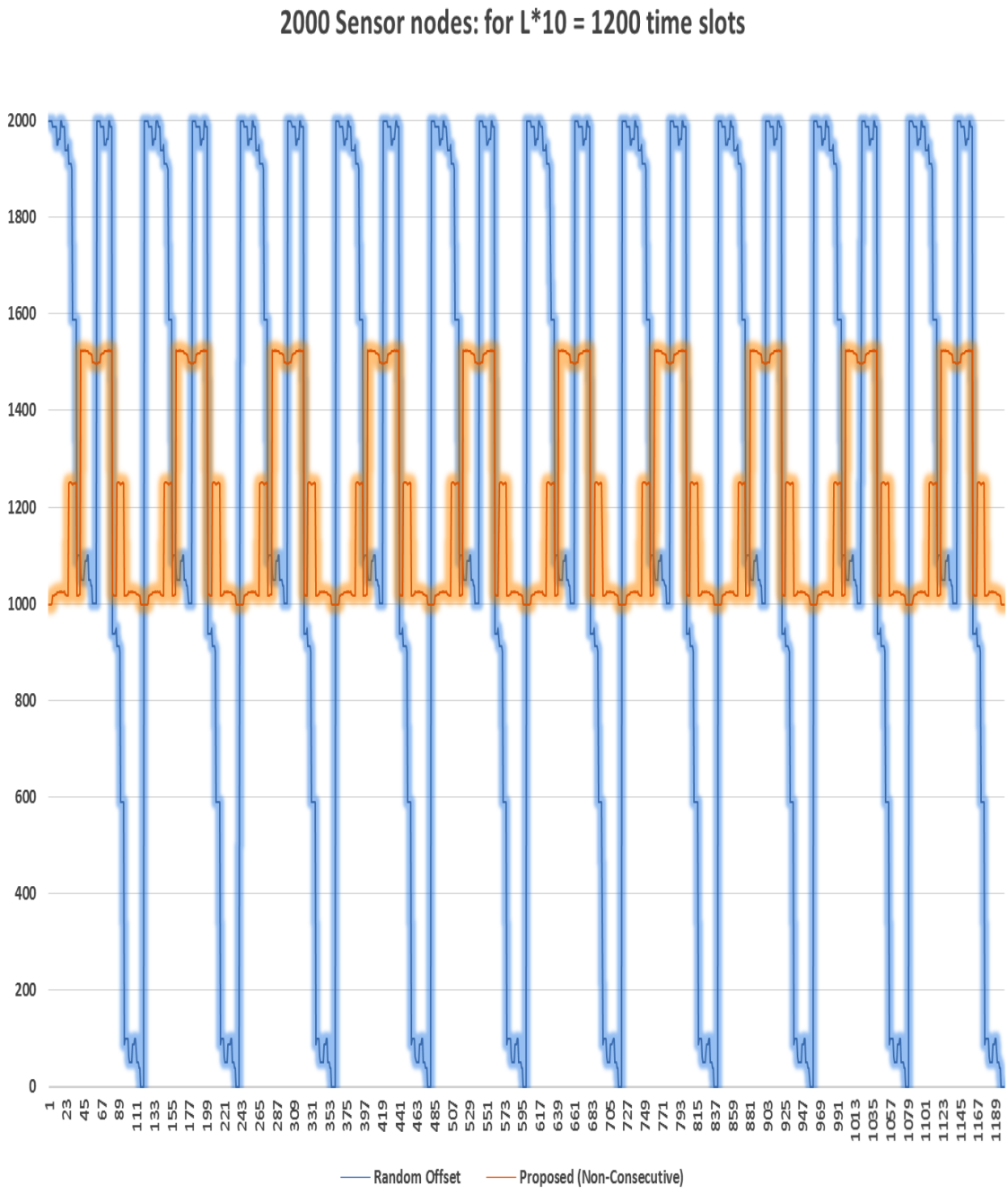


FIGURE 4.24: Aggregated traffic load per time slot in case of consecutive execution of transactions

In figure 4.23, a set of sensor nodes with the specifications mentioned in the table 4.11 have been considered. Herein, the sensor nodes are heterogeneous to each other in terms of p, f and T.

In the figure 4.23, L = 120, as the LCM of the periods of the Sensor nodes considered in table 4.11. Thus in the x-axis there are 120 time slots. The traffic load per time slot in case of Random Offset is depicted in blue and the resultant aggregated traffic according to the proposed algorithm is depicted in orange. The value of Ch_{max}= 12208 bits per time slot. And the value of BW = 1526, which means that

there exist a time slot among the 120 time slots in which a 1526 sensor nodes have been scheduled to transact which requires a maximum of 1526 frequency channels. And as Ch_{max}=12208 bits per time slot the access link must have a channel capacity of at least 12208 bits per time slot.

Thus for a span of 1200 time slots the resultant aggregated traffic per time slot looks like as depicted in figure 4.24, where the x-axis depicts each of the 1200 time slots and the y-axis shows the number of sensor nodes transacting in a time slot. In figure, 4.24, if each of the 2000 sensor nodes, streams data with random offsets in each period for all of the T number of transactions, then the shape of the aggregated traffic load per time slot can look like the one represented by the blue line, in the worst case. The worst case is a schedule that leads all the 2000 sensor nodes to transmit data together in a time slot. For any point in x-axis at which the blue line touches the highest peak of 2000 on the y-axis, depicts a time slot in which all the 2000 sensor nodes are scheduled to transmit data. Whereas, the orange line shows the aggregated traffic generated according to the offsets obtained by means of the proposed algorithm which takes 250.57045197486877 seconds to compute. The lowest and highest peak value in case of random offset is 0 and 2000. whereas in case of the proposed algorithm the lowest and the highest peak is 997 and 1526 respectively. Thus, collectively from the comparisons found according to the difference in peak values (highest peak - lowest peak) and in figure 4.24, it is quite evident that the load distribution is more even in case of the proposed algorithm having a highest peak value of 1526, instead of 2000 as in case of random offsets.

4.6.2 Results in case of non-consecutive execution of transactions

In figure 4.25, a set of sensor nodes with the specifications mentioned in the table 4.11 have been considered. Herein, the sensor nodes are heterogeneous to each other in terms of p, f and T.

In the figure 4.25, L = 120, as the LCM of the periods of the Sensor nodes considered in table 4.11. Thus in the x-axis there are 120 time slots. The traffic load per time slot in case of Random Offset is depicted in blue and the resultant aggregated traffic according to the proposed algorithm is depicted in orange. The value of Ch_{max}= 9792 bits per time slot. And the value of BW = 1224, which means that there exist a time slot among the 120 time slots in which a 1224 sensor nodes have been scheduled to transact which requires a maximum of 1224 frequency channels. And as Ch_{max}=9792 bits per time slot, the access link must have a channel capacity of at least 9792 bits per time slot.

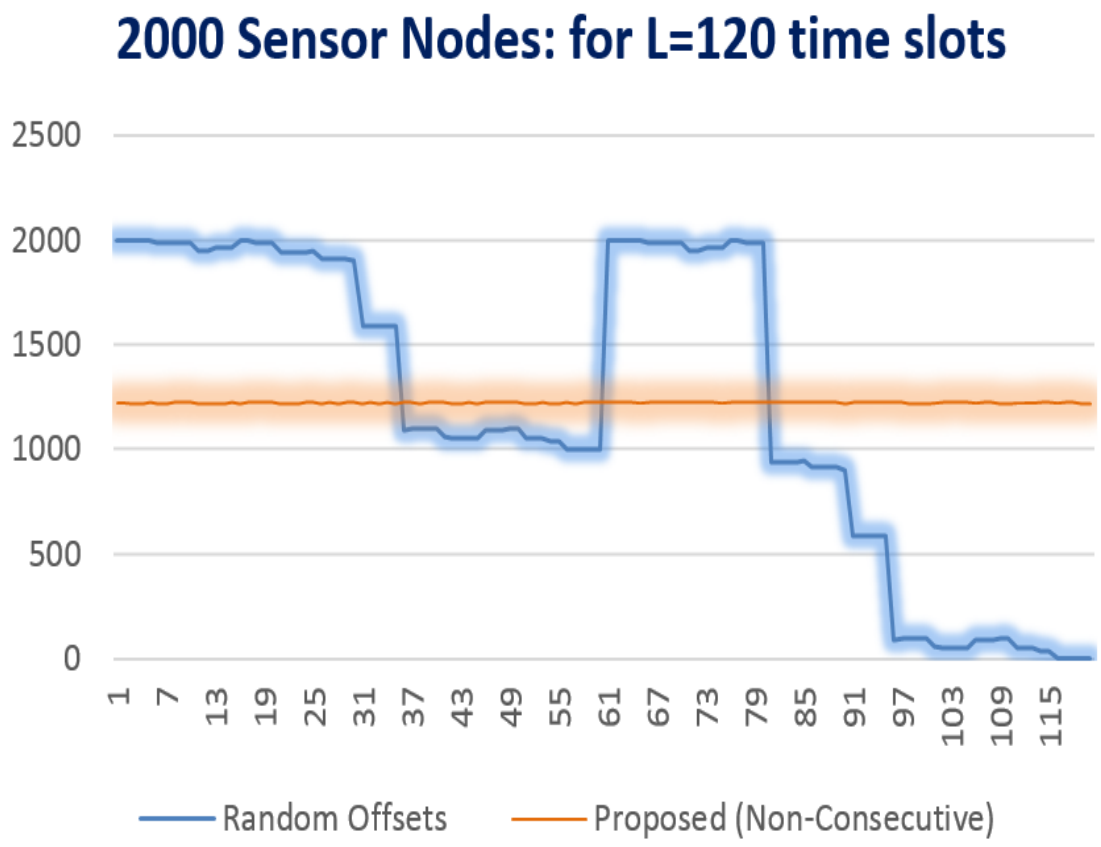


FIGURE 4.25: Aggregated traffic load per time slot in case of non-consecutive execution of transactions

Thus for a span of 1200 time slots the resultant aggregated traffic per time slot looks like as depicted in figure 4.26, where the x-axis depicts each of the 1200 time slots and the y-axis shows the number of sensor nodes transacting in a time slot. In figure, 4.24, if each of the 2000 sensor nodes, streams data with random offsets in each period for all of the T number of transactions, then the shape of the aggregated traffic load per time slot can look like the one represented by the blue line, in the worst case. The worst case is a schedule that leads all the 2000 sensor nodes to transmit data together in a time slot.

For any point in x-axis at which the blue line touches the highest peak of 2000 on the y-axis, depicts a time slot in which all the 2000 sensor nodes are scheduled to transmit data. Whereas the orange line shows the aggregated traffic generated according to the offsets obtained by means of the proposed algorithm which takes 936.4897356033325 seconds to compute.

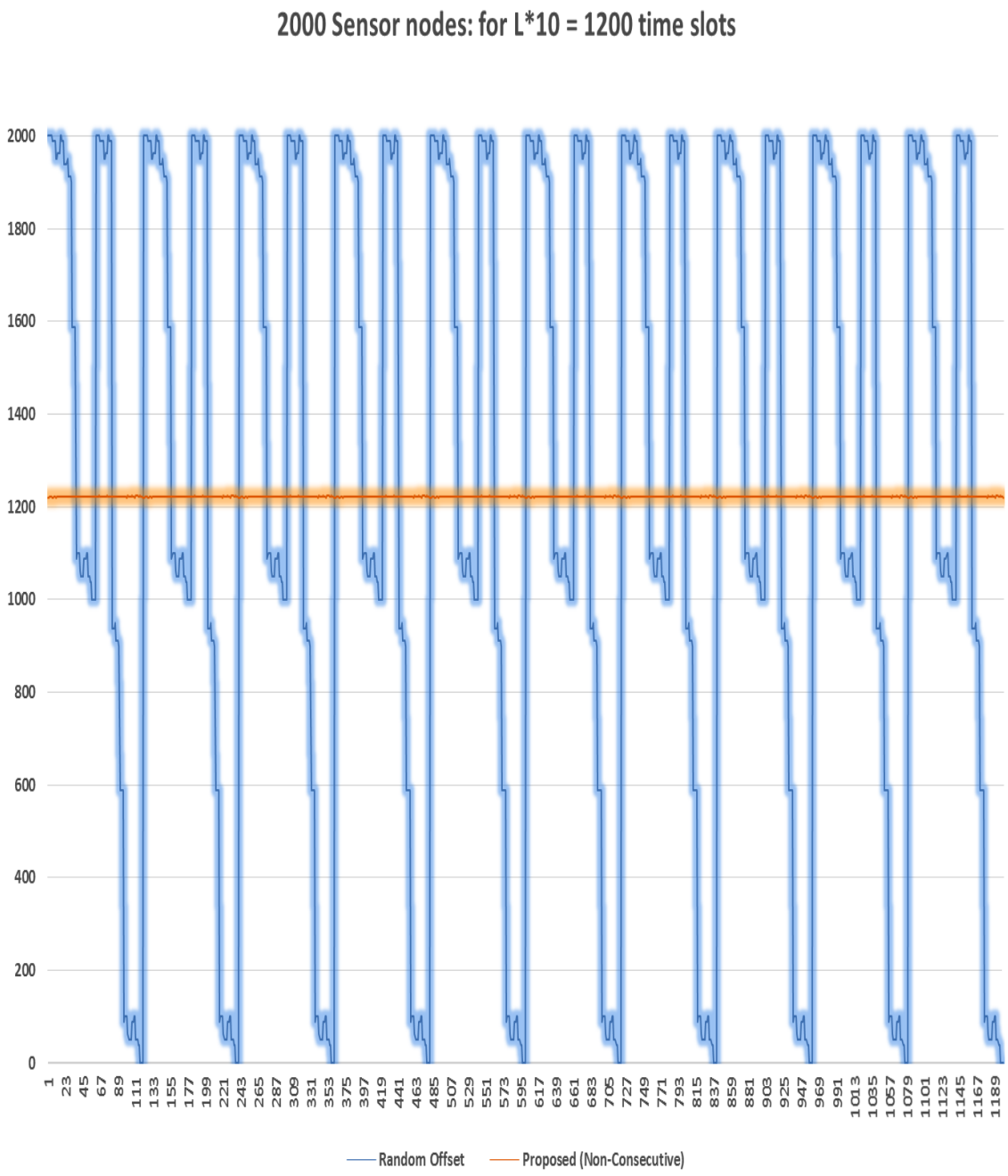


FIGURE 4.26: Aggregated traffic load per time slot in case of non-consecutive execution of transactions

The lowest and highest peak value in case of random offset is 0 and 2000. Whereas, in case of the proposed algorithm the lowest and the highest peak is 1219 and 1224 respectively. Thus collectively, from the comparisons found according to the difference in peak values (highest peak - lowest peak) and in figure 4.26, it is quite evident that the load distribution is more even in case of the proposed algorithm having a highest peak value of 1224, instead of 2000 as in case of random offsets.

4.7 Summary of the Chapter

In this chapter, the topic of sensor scheduling in sensor-cloud infrastructure has been discussed. The importance of sensor scheduling has been realized and then two problem scenarios in case of sensor-cloud has been introduced. Both the problems have been defined comprehensively. In order to solve the problems two algorithms have been discussed. For each of the two algorithms a pseudo code is also provided. For both the algorithms discussed, an example for each has been evaluated for comprehensive understanding of the proposed algorithms. The chapter ends with the results section in which for each problem few data sets have been considered and solved according to the proposed algorithms. The results achieved in case of the proposed algorithms have been been compared with the worst case and best case scenarios to realize their efficiency in performance.

Chapter 5

Concluding Remarks and Future Direction

5.1 Overview

In a Sensor-Cloud infrastructure, to acquire the sensor nodes deployed in a WSN that resides behind a NAT , the sensor nodes must be signalled with an message that must be able to reach the device after successfully traversing the NAT. In chapter 3 this problem was considered and solved with a unique communication protocol that establishes a connection between the sensor nodes and the cloud server. The proposed scheme establishes a communication channel to wake up a sensor node that is behind NAT from a remotely located publicly addressable server. This communication scheme can be used for IoT based solutions to enable cloud services to gain on demand unattended access to the sensing devices. The experiments carried out shows that communication cost of the sensing devices is significantly low, thereby leading to less energy consumption of resource constrained devices and more service lifetime. Instead of repeated polling from the client, our scheme gradually learns the maximum permitted interval at which the server can poll to keep the connection alive. This protects the small devices from energy drain and thus provides a more energy efficient solution.

With respect to the second problem, it can be concluded that the two proposed polynomial time algorithms for both the variations provide results which are significantly better than the worst case. Though they do not provide the optimal result every time being heuristic algorithms, the results are quite satisfactory with respect to the worst case scenario.

In case of the consecutive execution of transactions unlike the straight forward optimization which checked $\left(\frac{L}{p} \times \left((p - f \times T) + 1\right)\right)^n$ combinations, the proposed algorithm only checks $\left(n \times (p - f \times T)\right)$ combinations and works in polynomial time in the order of $\mathcal{O}(Ln)$.

And in case of non-consecutive execution Unlike the straight forward optimization of exponential time that checks $\left(\frac{L}{p} \times \left((p - f \times T) + 1\right) \times \mathrm{T}\right)^n$ combinations, the

proposed heuristic algorithm checks only $\Big(n \times (p - f \times T) \times \mathrm{T}\Big)$ combinations and works in polynomial time in the order of $\mathcal{O}(nLT)$.

The maximum required bandwidth at the wireless interface and the maximum required channel capacity at the access link for any given WSN can be determined in polynomial time with the help of the two proposed algorithms in their respective scenarios.

Thus, as the required bandwidth and channel capacity needed for a WSN deployment can be determined, the IoT WSN service applications provided by the Sensor-Cloud will not consume excess resources of the cloud. This also extends the lifetime of the sensor nodes. The power consumption gets minimized as the sensor nodes send data only when activated. Resource allocation also gets optimized in the Sensor-Cloud platform.

5.2 Future Work

The communication protocol discussed in chapter 3 can be used to communicate with various other devices residing behind NAT. For example, to directly communicate with an IoT application hosted at home, the proposed mechanism can be used by the owner via a mobile application. If the IoT devices at home are battery powered then the proposed scheme can help to establish a channel between the devices and the cloud which may be maintained by the cloud, avoiding energy drainage in those devices. An owner of a house can communicate with a IoT application running at home any time by sending a packet through the communication channel that can be established by means of the proposed algorithm. In the proposed work only a communication scheme is addressed, however for the security of both, the server and the client (sensor devices), other methodologies are needed to be incorporated. This may include detecting malicious nodes, preventing Denial of Service (DOS) attacks on the wake up server, and other malicious attempts to gain unsolicited access.

In case of Sensor-Cloud there exist many issues that needs to be taken care of other than bandwidth optimization while designing a Sensor-Cloud application. Few issues that must also be focussed on are reliability, Service Level Agreement (SLA) violation, fault tolerance, security and privacy. A critical patient must be monitored continuously without network failure which can not be guaranteed everytime. A patient can be in motion and can go out of network coverage. In such a scenario, if the patient gets into a critical condition then there shall be no means of medical assistance available to the patient until the patient gets network coverage to get the server notified about the patient's illness. For any particular type of service it is very hard to choose the correct combination of cloud providers to support the IoT application flawlessly. Security and privacy of sensed sensitive data like heath records need to be kept secured. As the cloud is providing the computational resources for various Sensor-Cloud applications, it is expected that the SLA is preserved and for this the cloud providers are charging the owner of the application

for hosting. Now, due to huge processing of sensor data in any critical environment with harsh climatic conditions, if the services fail to preserve the SLA then it will have an adverse effect on the faith and trust that the user had on the cloud providers.

Thus, there are many research areas to focus on in order to bring smartness and innovation to the Sensor-Cloud infrastructure.

Bibliography

[1] Alamri, A., Ansari, W.S., Hassan, M.M., Hossain, M.S., Alelaiwi, A. and Hossain, M.A., 2013. A survey on sensor-cloud: architecture, applications, and approaches. *International Journal of Distributed Sensor Networks*, 9(2), p.917923.

[2] Lozano, J., Apetrei, C., Ghasemi-Varnamkhasti, M., Matatagui, D. and Santos, J.P., 2017. Sensors and Systems for Environmental Monitoring and Control. *Journal of Sensors*, 2017.

[3] Puiu, D., Barnaghi, P., Tönjes, R., Kümper, D., Ali, M.I., Mileo, A., Parreira, J.X., Fischer, M., Kolozali, S., Farajidavar, N. and Gao, F., 2016. Citypulse: Large scale data analytics framework for smart cities. *IEEE Access*, 4, pp.1086-1108.

[4] Liao, Y., Mollineaux, M., Hsu, R., Bartlett, R., Singla, A., Raja, A., Bajwa, R. and Rajagopal, R., 2014. Snowfort: An open source https://www.overleaf.com/project/5ce406992f38b8239fc51c3bwireless sensor network for data analytics in infrastructure and environmental monitoring. *IEEE Sensors Journal*, 14(12), pp.4253-4263.

[5] K. Romer and F. Mattern, "Te design space of wireless sensor networks," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54–61, 2004.

[6] T. Haenselmann, "Sensor networks," GFDL. In, *Wireless Sensor Network textbook* 2006.

[7] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114, 2002.

[8] Alemdar, H. and Ersoy, C., 2010. Wireless sensor networks for healthcare: A survey. *Computer networks*, 54(15), pp.2688-2710.

[9] G. Simon, G. Balogh, G. Pap et al., "Sensor network-based countersniper system," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pp. 1–12, Baltimore, Md, USA, November 2004.

[10] S. K. Dash, J. P. Sahoo, S. Mohapatra, and S. P. Pati, "Sensorcloud: assimilation of wireless sensor network and the cloud," in *Advances in Computer Science and Information Technology. Networks and Communications*, vol. 84, pp. 455–464, SpringerLink, 2012

[11] Stanczyk, B. and Buss, M., 2004, September. Development of a telerobotic system for exploration of hazardous environments. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)* (Vol. 3, pp. 2532-2537). IEEE.

[12] M. Castillo-Een, D. H. Quintela, R. Jordan, W. Westho, and W. Moreno, Wireless sensor networks for ash-ood alerting, in *Proceedings of the 5th IEEE International Caracas Conference on Devices, Circuits and Systems (ICCDCS '04)*, pp. 142–146,November 2004

[13] G. Werner-Allen, K. Lorincz, M. Welsh et al., "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, vol. 10, no. 2, pp. 18–25, 2006.

[14] Chen, D. and Varshney, P.K., 2004, June. QoS Support in Wireless Sensor Networks: A Survey. In *International conference on wireless networks* (Vol. 233, pp. 1-7).

[15] W. Kim, "Cloud computing: today and tomorrow," *Journal of Object Technology*, vol. 8, pp. 65–72, 2009.

[16] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure physical sensor management with virtualized sensors on cloud computing," in *Proceedings of the IEEE 13th International Conference on Network-Based Information Systems (NBiS '10)*, pp. 1–8, September 2010.

[17] L. P. D. Kumar, S. S. Grace, A. Krishnan, V. M. Manikandan, R. Chinraj, and M. R. Sumalatha, "Data fltering in wireless sensor networks using neural networks for storage in cloud," in *Proceedings of the IEEE International Conference on Recent Trends in Information Technology (ICRTIT '11)*, 2012.

[18] M. O'Brien, Remote Telemonitoring—A Preliminary Review of Current Evidence, *European Center for Connected Health*, 2008.

[19] http://www.ntu.edu.sg/intellisys.

[20] K. T. Lan, "What's Next? Sensor+Cloud?" in *Proceeding of the 7th International Workshop on Data Management for Sensor Networks*, pp. 978–971, ACM Digital Library, 2010.

[21] Sensor-Cloud, http://sensorcloud.com/system-overview.

[22] C. O. Rolim, F. L. Koch, C. B. Westphall, J. Werner, A. Fracalossi, and G. S. Salvador, "A cloud computing solution for patient's data collection in health care institutions," in *Proceedings of the 2nd International Conference on eHealth, Telemedicine, and Social Medicine (eTELEMED '10)*, pp. 95–99, February 2010.

[23] U. Varshney, "Pervasive healthcare and wireless health monitoring," *Mobile Networks and Applications*, vol. 12, no. 2-3, pp. 113–127, 2007.

[24] U. Varshney, "Managing wireless health monitoring for people with disabilities," *IEEE IT-Professional*, pp. 12–16, 2006.

[25] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys '09)*, pp. 1–14, November 2009.

[26] S. Madden and M. J. Franklin, "Fjording the stream: an architecture for queries over streaming sensor data," in *Proceedings of the 18th International Conference on Data Engineering*, pp. 555– 566, March 2002.

[27] A. Rowe, V. Gupta, and R. Rajkumar, "Low-power clock synchronization using electromagnetic energy radiating from AC power lines," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys '09)*, pp. 211–224, November 2009

[28] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *Proceedings of the 9th International Conference Architectural Support for Programming Languages and Operating Systems*, pp. 93–104, November 2000.

[29] Zhu, C., Zheng, C., Shu, L. and Han, G., 2012. A survey on coverage and connectivity issues in wireless sensor networks. *Journal of Network and Computer Applications*, 35(2), pp.619-632.

[30] R. Katsuma, Y. Murata, N. Shibata, K. Yasumoto, and M. Ito, "Extending k-coverage lifetime of wireless sensor networks using mobile sensor nodes," in *Proceedings of the 5th IEEE International Conference on Wireless and Mobile Computing Networking and Communication (WiMob '09)*, pp. 48–54, October 2009

[31] K.Matsumoto,R.Katsuma,N.Shibata,K.Yasumoto,andM.Ito, "Extended abstract: minimizing localization cost with mobile anchor in underwater sensor networks," in *Proceedings of the 4th ACM International Workshop on UnderWater Networks (WUWNet '09)*, November 2009.

[32] T. Sookoor, T. Hnat, P. Hooimeijer, W. Weimer, and K. Whitehouse, "Macrodebugging: global views of distributed program execution," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys '09)*, pp. 141–154, November 2009

[33] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, and M. Seltzerand Welsh, "Hourglass: an infrastructure for connecting sensor networks and applications," *Harvard Technical Report* TR-21-04, 2004.

[34] M. Gaynor, S. L. Moulton, M. Welsh, E. LaCombe, A. Rowan, and J. Wynne, "Integrating wireless sensor networks with the grid," *IEEE Internet Computing*, vol. 8, no. 4, pp. 32–39, 2004.

[35] R. S. Ponmagal and J. Raja, "An extensible cloud architecture model for heterogeneous sensor services," *International Journal of Computer Science and Information Security*, vol. 9, no. 1, 2011.

[36] A. Alexe and R. Exhilarasie, "Cloud computing based vehicle tracking information systems," *International Journal of Computer Science and Telecommunications*, vol. 2, no. 1, 2011.

[37] C. Doukas and I. Maglogiannis, "Managing wearable sensor data through cloud computing," in *Proceedings of the IEEE 3rd International Conference on Cloud Computing*, 2011

[38] X. H. Le, "Secured WSN-integrated cloud computing for ulife care," in *Proceedings of the Consumer Communications and networking Conference (CCNC '10)*, pp. 1–2, IEEE, 2010.

[39] T.-D. Nguyen and E.-N. Huh, "An efcient key management for secure multicast in Sensor-Cloud"," in *Proceedings of the IEEE 1st ACIS/JNU International Conference on Computers, Networks, Systems, and Industrial Engineering*, 2011.

[40] Bose S, Gupta A, Adhikary S, Mukherjee N. Towards a sensor-cloud infrastructure with sensor virtualization. In *Proceedings of the Second Workshop on Mobile Sensing, Computing and Communication*, 2015 Jun 22 (pp. 25-30). ACM.

[41] Nimbits Data Logging Cloud Sever, http://www.nimbits.com.

[42] Pachube Feed Cloud Service, http://www.pachube.com.

[43] iDigi—Device Cloud, http://www.idigi.com.

[44] IoT—TingSpeak, http://www.thingspeak.com.

[45] G. Demiris, B. K. Hensel, M. Skubic, and M. Rantz, "Senior residents' perceived need of and preferences for "smart home" sensor technologies," *International Journal of Technology Assessment in Health Care*, vol. 24, no. 1, pp. 120–124, 2008.

[46] Tunnel Monitoring System: http://www.advantech.com/intelligent-automation/IndustryD2-499E-9E16-6C1F41D1CD6/.

[47] Tovar, A., Friesen, T., Ferens, K. and McLeod, B., A DTN wireless sensor network for wildlife habitat monitoring. In *CCECE, 2010*.

[48] N. Kurata, M. Suzuki, S. Saruwatari, and H. Morikawa, "Actual application of ubiquitous structural monitoring system using wireless sensor networks," in Proceedings of the 14th World Conference on Earthquake Engineering (WCEE '08), 2008.

[49] Masayuki Hirafuji,*Agriculture Working Group*. http://www.apan.net/meetings/HongKong2011/Session/Agriculture.php/.

[50] H. H. Tran and K. J. Wong, "Mesh networking for seismic monitoring—the sumatran cGPS array case study," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC '09)*, April 2009.

[51] Gil Jiménez, V.P. and Fernández-Getino García, M.J., 2015. Simple design of wireless sensor networks for traffic jams avoidance. *Journal of Sensors*, 2015.

[52] Port forwarding : Provides information for using port forwarding with Burk products. in *Technical Bulletin, BURK TECHNOLOGY*,

[53] Er. M. kaur and Dr. K.S. Kahlon. "Study and Comparison of Network Security in IPv4 and IPv6" in *International Journal of Science and Research (IJSR)*,(2017)

[54] P. Srisuresh, M. Holdrege. "RFC 2663: IP Network Address Translator (NAT) terminology and considerations", *IETF*, (August 1999)

[55] P. Srisuresh, K. Egevang. "RFC 3022: Traditional IP network address translator (Traditional NAT)",*IETF*, (2000)

[56] P. Srisuresh, B. Ford, D. Kegel. "RFC 5128: State of peer-to-peer (P2P) communication across network address translators (NATs)", *IETF*, (2008)

[57] Y. Wei , D. Yamada, S. Yoshida, S. Goto. "A new method for symmetric NAT traversal in UDP and TCP", *Network Research Workshop*, (Aug 2008)

[58] A. Müller, G. Carle, A. Klenk. "Behavior and classification of NAT devices and implications for NAT traversal". *IEEE network*, 22(5),14-9, (2008)

[59] M. Boucadair, R. Penno, D. Wing. " RFC 6970: Universal plug and play (UPnP) internet gateway device-port control protocol interworking function (IGD-PCP IWF)", *IETF*, (2013).

[60] S. Cheshire, M. Krochmal. "RFC 6886: Nat port mapping protocol (NAT-PMP)", *IETF*, (2013)

[61] D. Wing, S. Cheshire, M. Boucadair, R. Penno, P. Selkirk "RFC 6887: Port control protocol (PCP)", *IETF*, (2013)

[62] O. Maennel, R. Bush, L. Cittadini, SM. Bellovin. "A Better Approach than Carrier-Grade-NAT", *Columbia University Computer Science Technical Reports, CUCS-041-08*, Department of Computer Science, Columbia University, (2011)

[63] A. Durand, R. Droms, J. Woodyatt, Y. Lee, "RFC 6333: Dual-stack lite broadband deployments following IPv4 exhaustion", *IETF*, (2011)

[64] J. Rosenberg, R. Mahy, P. Matthews, D. Wing, "RFC 5389: Session traversal utilities for NAT (STUN)", *IETF*, (2008).

[65] H. Khlifi, J. C. Gregoire, J. Phillips. "VoIP and NAT/firewalls: issues, traversal techniques, and a real-world solution", *IEEE Communications Magazine*,44(7), (2006)

[66] M. Petit-Huguenin, S. Nandakumar, G. Salgueiro, P. Jones. "RFC 7065: Traversal Using Relays around NAT (TURN) Uniform Resource Identifiers", *IETF*, (2013)

[67] J. Rosenberg. " RFC 5245: Interactive connectivity establishment (ICE): A protocol for network address translator (NAT) traversal for offer/answer protocols", *IETF*, 2010.

[68] C. C. Tseng, C. L. Lin, L. H. Yen, J. Y. Liu, C. Y. Ho. "Can: A context-aware NAT traversal scheme", *Journal of network and computer applications*, 36(4), (2013)

[69] B. Ford, P. Srisuresh, D. Kegel. " Peer-to-Peer Communication Across Network Address Translators". In *USENIX Annual Technical Conference, General Track*, (2005)

[70] M. Handley. "RFC 3261: Session Initiation Protocol (SIP)", *IETF*, (2002)

[71] R. Cuevas, A´. Cuevas, A. Cabellos-Aparicio, L. Jakab, C. Guerrero. "A collaborative P2P scheme for NAT Traversal Server discovery based on topological information", *Computer Networks*,54(12), (2010)

[72] A Muller, A. Klenk, G. Carle. "ANTS-a framework for knowledge based NAT traversal". In *IEEE Global Telecommunications Conference 2009, GLOBECOMM*, (2009)

[73] S. Guha, Y. Takeda, P. Francis. "NUTSS: A SIP-based approach to UDP and TCP network connectivity". In *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, ACM, 2004

[74] R. P. Swale, P. A. Mart, P. Sijben, S. Brim, M. Shore. "RFC 3304: Middlebox communications (MIDCOM) protocol requirements", (2002)

[75] J. L. Eppinger. "TCP connections for P2P apps: A software approach to solving the NAT problem", *Technical Report CMUISRI-05-104*, Carnegie Mellon University, (2005)

[76] Y. C. Chen, W. K. Jia. "Challenge and solutions of NAT traversal for ubiquitous and pervasive applications on the Internet", *Journal of Systems and Software* 82(10), (2009).

[77] A. Biggadike, D. Ferullo, G. Wilson, A. Perrig. "NATBLASTER: Establishing TCP connections between hosts behind NATs", In *ACM SIGCOMM Asia Workshop 2005* (Vol. 5), (2005).

[78] Oh, S. and Jang, J., 2017. A scheme to smooth aggregated traffic from sensors with periodic reports. Sensors, 17(3), p.503.

[79] Gharbieh, M., ElSawy, H., Bader, A., Alouini, M.S. Tractable stochastic geometry model for IoT access in LTE networks. In *Proceedings of the IEEE Globecom 2016*, Washington, DC, USA, 4–8 December 2016.

[80] Theodoridis, E., Mylonas, G., Chatzigiannakis, I. Developing an IoT Smart City framework. In *Proceedings of the International Conference on Information, Intelligence, Systems and Applications,* Piraeus, Greece, 10–12 July 2013; pp. 1–6.

[81] Sundmaeker, H., Guillemin, P., Friess, P., Woelfflé, S. Vision and Challenges for Realising the Internet of Things. *European Commision: Luxembourg*, 2010.

[82] Zaslavsky, A., Perera, C., Georgakopoulos, D. Sensing as a service and big data. In *Proceedings of the International Conference on Advances in Cloud Computing*, Bangalore, India, 4–6 July 2012; pp. 21–29.

[83] Zanella, A., Bui, N., Castellani, A., Vangelista, L., Zorzi, M. Internet of Things for Smart Cities. *IEEE Internet Of Things Journal.* 2014, 1, 22–32.

[84] Jin, J., Gubbi, J., Marusic, S., Palaniswami, M. An Information Framework for Creating a Smart City through Internet of Things. *IEEE Internet Of Things Journal.* 2014, 1, 112–121.

[85] Duffield, N., Grossglauser, M. Trajectory sampling for direct traffic observation. *IEEE/ACM Trans. Netw*. 2001, 9, 280–292.

[86] Georgiadis, L., Guerin, R., Peris, V., Sivarajan, K. Efficient network QoS provisioning based on per node traffic shaping. In *Proceedings of the IEEE INFOCOM '96 Conference on Computer Communications 1996,* San Francisco, CA, USA, 24–28 March 1996; pp. 481–501.

[87] Comparing Traffic Policing and Traffic Shaping for Bandwidth Limiting. Available online: http://www.cisco. com/c/en/us/support/docs/quality-of-service-qos/qos-policing/19645-policevsshape.html.

[88] Piri, E., Pinola, J. Performance of LTE uplink for IoT backhaul. In *Proceedings of the 13th IEEE Annual Consumer Communications Networking Conference, Las Vegas*, NV, USA, 9–12 January 2016; pp. 6–11

[89] Francois, J., Cholez, T., Engel, T. CCN traffic optimization for IoT. In *Proceedings of the 2013 Fourth International Conference on the Network of the Future*, Pohang, Korea, 23–25 October 2013; pp. 1–5.

[90] Traffic Shaping. Available online: http://www.computerhope.com/jargon/t/traffic-shaping.htm

[91] Marcon, M.; Dischinger, M.; Gummadi, K.P.; Vahdat, A. The local and global effects of traffic shaping in the internet. In *Proceedings of the 2011 3rd International Conference on Communication Systems and Networks*, Jammu, India, 3–5 June 2011; pp. 1–10.

[92] Comcast: Description of Planned Network Management Practices. Available online: `http://downloads.comcast.net/docs/Attachment_B_Future_Practices.pdf`

[93] Traffic Policing. Available online:`http://www.cisco.com/c/en/us/td/docs/ios/qos/configuration/guide/15_1/qos_15_1_book/traffic_policing.pdf`

[94] Gu, Z., Shin, K. Algorithms for effective variable bit rate traffic smoothing. In *Proceedings of the 2003 IEEE International Conference on Performance, Computing, and Communications*, Phoenix, Arizona, 9–11 April 2003; pp. 387–394.

[95] Ziermann, T.; Teich, J.; Salcic, Z. DynOAA—Dynamic offset adaptation algorithm for improving response times of CAN systems. In *Proceedings of the 2011 Design, Automation & Test.* in Europe, Grenoble, France, 14–18 March 2011; pp. 1–4.

[96] Grenier, M., Havet, L., Navet, N. Pushing the limits of CAN-scheduling frames with offsets provides a major performance boost. In *Proceedings of the 4th European Congress on Embedded Real Time Software*, Toulouse, France, 29 January–1 February 2008.

[97] R. Rosen. "Linux kernel networking: Implementation and theory", *Apress*, (2014).

[98] Bogdanoski, Mitko Shuminoski, Tomislav & Risteski, Aleksandar. (2013). Analysis of the SYN flood DoS attack. *International Journal of Computer Network and Information Security*, 5. 1-11. 10.5815/ijcnis.2013.08.01.