Dynamic NSTART variation for better channel utilization

A project report submitted in fulfilment of the requirement for the award of

Degree of Bachelor of Technology in Computer Science and Engineering

by

Kunal Verma (2019UCP1358)
Pratha Jaiswal (2019UCP1360)
Sachin Raj (2019UCP1328)

Under the Supervision of
Dr. Arka Prokash Mazumdar

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY
MAY2023

Certificate

We,
Kunal Verma (2019UCP1358)
Pratha Jaiswal (2019UCP1360)
Sachin Raj (2019UCP1328)

Declare that this project titled, "Dynamic NSTART variation" and the work presented in it are our own. I confirm that:

1. This project work was done wholly or mainly while in candidature for a B.Tech. degree in the department of computer science and engineering at Malaviya National Institute of Technology Jaipur (MNIT).
2. Where any part of this report has previously been submitted for a degree or any other qualification at MNIT or any other institution, this has been clearly stated. Where we have consulted the published work of others, this is always clearly attributed. Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this project is entirely our own work.
3. We have acknowledged all main sources of help.

Signed:
Date:

Dr. Arka Prokash Mazumdar

Assistant Professor

Department of Computer Science and Engineering
Malaviya National Institute of Technology

Abstract

At the application layer of the Internet of Things (IoT), the Constrained Application Protocol (CoAP) is a communication protocol. To maintain outstanding parallel connections in the direction of different destinations for data transfer, it defines a parameter called NSTART. The NSTART value in the CoAP implementation is statically determined. While this parameter's higher values up to a threshold improve throughput in times of moderate traffic, its lower values regulate congestion when there is an overload. Higher values of NSTART have a negative impact on throughput when there is an overload because they increase overhead. Because of the parameter's static nature and associated performance concerns, CoAP must control NSTART dynamically. In this study, we provide an approach to dynamically change the value of NSTART based on network congestion. The suggested strategy aims to liberate the client from dynamically managing NSTART. We set up simulations on the Raspberry Pi v1.2 server and Californium CoAP client for the proof of concept. The results show that the suggested strategy improves throughput and client turnaround times in circumstances with severely congested networks when compared to the best case of CoCoA (NSTART = 4), all without placing an undue burden on the server.

ACKNOWLEDGEMENT

Working on this project of Dynamic NSTART variation for better channel utilization was very interesting. During the period of working on this project, we learnt about IoT communication protocols. I would like to thank Dr. Arka Prokash Mazumdar, Assistant Professor, Department of CSE for giving me this opportunity to learn and be a part of such a wonderful project.

Kunal Verma (2019ucp1358)

Pratha Jaiswal (2019ucp1360)

Sachin Raj (2019ucp1328)

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 IoT

The Internet of Things (IoT) is a new paradigm that makes it possible for electrical gadgets and sensors to communicate with one another over the internet to make our lives easier. IoT uses smart gadgets and the internet to offer creative answers to problems faced by businesses, governments, and both public and private sectors around the world.

1.1.1 What is IoT

IoT refers to a network of interconnected computing or electronic devices embedded in everyday objects that are capable of transmitting and receiving data.

The following components are used in the Internet of Things:

1)Low-power embedded systems: While designing an electronic system, two

things should be kept in mind: battery consumption and performance.

2)In a sensor, a physical quantity is measured and detected. Thea is converted into a signal that can be provided to a processing or control unit for analysis.

3)Control Units consist of a microprocessor or processing core, memory, and programmable input/output devices/peripherals on one integrated circuit. It performs all logical operations associated with IoT devices, and is responsible for performing the majority of the processing work.

4)The data collected by IoT devices is massive and must be stored on a reliable storage server in the cloud. Cloud computing serves as a solution to this problem. By processing and learning the data, we are able to discover where things like electrical faults/errors are located within the system.

5)It is well known that the Internet of Things is heavily dependent on sensors, particularly in real-time. Electronic devices are becoming increasingly prevalent in every field, resulting in a massive flood of big data.

6)Networking connection: For communication to take place, internet connectivity is a necessity, where each physical object is assigned an IP address. However, IP naming limits the number of addresses available. Due to the increasing number of devices, this naming system will no longer be feasible. In order to represent each physical object, researchers are seeking an alternative naming system.

1.1.2 Major Application of IoT

1)The internet of things has vast and diverse potential uses because it affects almost every area of daily life for people, organisations, and society as a whole.

2)IoT applications span a wide range of industries, including manufacturing and the Industrial, health, agricultural, and smart city sectors, as well as security and emergency situations [6] asserts that the IoT is essential for boosting general infrastructure and increasing the smartness of cities.

3)Intelligent transportation systems [7], smart buildings, traffic congestion [7, 8], waste management [9], smart lighting, smart parking, and urban maps are a few IoT application areas for developing smart cities. This could involve installing sound monitoring equipment in sensitive areas of cities, monitoring the levels of pedestrians and vehicles, as well as monitoring the vibrations and material conditions of bridges and buildings, as well as monitoring the availability of parking spaces within the city.

4)IoT with AI capabilities can be used in smart cities to monitor, manage, and lessen traffic congestion [6]. Additionally, by monitoring garbage collection schedules, IoT enables the installation of intelligent and weather-adaptive street lighting as well as waste detectors and waste containers. Intelligent roadways can deliver vital information and warnings, such as access to detours based on weather conditions or unanticipated occurrences like traffic jams and accidents.

1.2 CoAP

According to RFC 7252, the limited Application Protocol (CoAP) is a specific Internet application protocol for limited devices. It makes it possible for those limited "nodes" to interact with the larger Internet using similar protocols. CoAP is intended for use between devices connected to the same restricted network (such as low-power, lossy networks), between devices and Internet-connected general nodes, and between devices connected to separate confined networks but connected by an internet. Other techniques, such SMS on

mobile communication networks, are also used to implement CoAP.

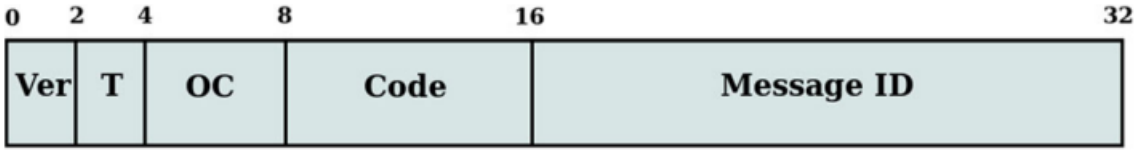

Figure 1.1: Coap Header

Field	Length	Data type	Description
Version	2 bits	Unsigned integer	CoAP version number
Type	2 bits	Unsigned integer	Confirmable; Acknowledgment
Token length	4 bits	Unsigned integer	0–8 bytes
Code	8 bits	Unsigned integer	Request–Response/Empty
Message ID	16 bits	Unsigned integer	Used to match the message of Acknowledgment

Table 1.1: Coap Headers

1.2.1 What is CoAP

1. Constrained Application Protocol (CoAP) is used in the Internet of things (IoT) networks for communication.

2. For use with confined nodes and constrained networks in the Internet of Things, the confined Application Protocol (CoAP) is described as a

specialised web transfer protocol (CoAP, 2014). This protocol is primarily designed for IoT and M2M applications, as may be inferred from the definition.

3. In order to perform an activity on a server-side resource, the client sends a CoAP request. The server sends a response with a response code. An asynchronous datagram-oriented transport, such as UDP, is used for CoAP communication. Confirmable, Non-confirmable, Acknowledgement, and Reset are the four different categories of CoAP specified messages. Confirmable and non-confirmable requests are made using CoAP, and confirmable and non-confirmable responses are also possible.

4. The CoAP option NSTART determines the number of concurrent connections between the client and server. Modern versions of CoAP all have NSTART set statically on the client side. In instances with moderate traffic, a lower value of NSTART keeps the channel underutilised and reduces throughput, whereas a greater value of NSTART places a stress on the channel and unnecessarily uses resources in situations with low traffic. Therefore, based on changes in the offered load, it is necessary to dynamically adapt the value of NSTART.

1.3 CoCoA

CoCoA stand for Congestion Control Advance. It is basically advanced version of CoAP protocol.

1.3.1 What is CoCoA

This protocol is same as CoAP but only difference between CoAP and this protocol is in calculation of Retranmission Time Out time. Whenever there is packet drop occurs in network then there should be some particular time for that sender need to wait for the acknowledgement and if in that period of time acknowledgement for that packet not sent by the receiver then sender need to retransmit that packet. Coap protocol is using binary exponential backoff factor for the calculation of RTO time where as this protocol is using variable backoff factor which basically makes this protocol more efficient than CoAP protocol.

1.4 Congestion

Congestion is basically the things that is experiened in network when there is too much traffic in the system. It causes packet drop and decrease in throughput of channel.

1.4.1 How to identify congestion in the network

Bandwidth

Bandwidth is the most frequent reason for network congestion. The term bandwidth describes the network's ideal ability to move a given volume of data from a source to a destination in a specific length of time. Network interruptions might result from a lack of bandwidth.

Latency

The amount of time it takes to transmit, capture, and process data from source to destination is known as latency. It speaks of the millisecond-based speed of your network traffic. A network can become slower due to high latency. Depending on the programme and network connection being used, latency figures may change.

Jitter

When transferring data packets from a source via a network to a destination, there is a time delay known as jitter. Network congestion and jitter are caused by unpredictable traffic. Jitter can also affect how well your network transmits audio and video. As network hardware and software attempt to adapt to changes in traffic patterns, jitter and congestion are produced as a cascade effect.

1.4.2 Packet Retransmissions

When the transmission of data packets is interrupted by packet loss, packet damage, or another factor, packet retransmission is necessary. Such circumstances result in the resending of data packets from source to destination, causing network congestion.

1.4.3 Collisions

When two or more network nodes attempt to deliver data at once, packet collision happens. Network performance may suffer as a result of packet loss and the need to resend packets. All packets must wait to be processed after a collision in order to relieve network congestion. It could be caused by a bad connection, inadequate wiring, and more.

1.4.4 What are the reasons for the congestion in network

Over-Used Devices

Overused and too many devices: Overused devices can also contribute to network congestion. Pushing devices to their maximum capacity can often result in over-utilization. Excessive volumes of device usage can also cause network congestion as they can provide a surplus of requests for data.

Unneeded Traffic

Another typical reason for network congestion is unnecessary traffic, like movie streaming on a work computer. Junk VoIP calls and unwanted traffic like advertising are two further instances of unnecessary traffic that consumes bandwidth. Unneeded traffic can be found using the network administration console.

Antiquated Hardware and Faulty Devices

As the demands of the organisation change, it may be necessary to update or replace hardware, devices, and the ethernet cables and wire connections that connect them. As part of a network performance assessment for network congestion, it is important to examine the data speed and other metrics for each component of the network.

Deficient Design or Misconfiguration

A more major cause of network congestion is subpar design or improper device setup. Each network must be set up to accommodate the necessary

loads and the requirements of the particular organisation. All segments are connected by a network that has been optimised to maximise performance in each segment.

Security Attack

Worms, malware, and DoS attacks are just a few examples of security threats that can clog up networks.

1.5 What is NSTART

NSTART is basically a parameter that determines the number of outstanding connections between client and server. This is a parameter that we adjust in in our proposed approach to reduce the congestion in network and maximize the throughput of the channel. In our proposed approach we are changing the value of this parameter so that no of connections between client and server can be changes according to congestion in the network.

Chapter 2

MOTIVATION

When studying about the CoAP, we found that the congestion control mechanism mentioned in its rfc was exponential backoff.This method was good to avoid congestion but prevented good channel utilization. Then we came to know about the CoCoA implementation which modifies the calculation of RTO using a Variable BackoffFactor . Then we found that the NSTART value was mentioned as a constant in the rfc of CoAP .So we thought if we could change the value of this variable so as to improve channel utilization. There we came across this paper which proposed making the NSTART value dynamic. It proposes an algorithm for doing the same .Upon studying the algorithm we found that there is no condition on equality of current and previous congestion ratio if the desired threshold was high. So we tried to find the drawback of the same and found a case where the non existence of the equality condition will cause problems. When we found the case next we proceeded to find a new algorithm that can solve this issue.This is how we came up with the idea of this project.

Chapter 3

RELATED WORK

According to Carles Gomez [8], the elements of the linked health vision include rising healthcare expenditures and the growing accessibility of innovative health devices for personal care (PHDs). In addition, growing demand of consumer computerized devices are capable of acting as a doorway for well-being. So based on PHDs, it offers a solution based on novel protocol for IoT. With the help of CoAP, PHDs are able to communicate particulars and details with Internet. They discuss about the combination of health care systems with suggested system. In addition, they show an actual personal health devices' illustration and the results of its analysis. These results confirmed the practicality of the suggested approach. In this work, they describe a system architecture in which PHDs and mobile devices use CoAP and IEEE 11073 to communicate health sensor data with local and Internet services. They assessed and outlined how the IEEE 11073 communication model should be changed for CoAP, as well as the restrictions that are introduced. In addition, they integrate this new architecture with UPnP-based home networks and demonstrate how this new design can be used with legacy connected health systems.

According to Vinesh Kumar Jain [**?**], Constrained Application Protocol (CoAP) is an application layer communication protocol used in the Internet of Things (IoT). It makes use of the NSTART parameter to ensure excellent parallel connections to particular destinations for data transfer. Static value of NSTART is defined in the existing implementation of CoAP.In case of moderate traffic larger values of this parameter up to threshold will improve the performance, lower values of it helps in controlling congestion in conditions of overloaded situations . Higher levels of NSTART result in greater overhead and so perform badly in terms of throughput in overload conditions. Because of the parameter's static nature and the performance difficulties it raises, NSTART must be managed dynamically in CoAP. NSTART is basically CoAP parameter that determines the number of outstanding connections between client and server ,. In all contemporary implementations of CoAP,on client side NSTART is set static. If NSTART having lower value then keeps the channel underutilised and reduces throughput in moderate traffic scenarios, whereas in case of higher value of NSTART stresses the channel and utilised resources unnecessarilly. As a result, the value of NSTART must be modified on a regular basis in response to fluctuations in the offered load. The experimental system is divided into two parts: (i) a PC-based Californium (Cf) CoAP client and (ii) an Erbium (Er) CoAP server operating on M3 motes with Contiki OS at the FIT/IoT-Lab testbed. Erbium (Er) CoAP servers offer resources that allow for GET requests. Contiki 6LoWPAN communication stack is installed on M3 motes. For routing, an RPL border router is utilised. In the FIT/IoT-Lab testbed, one M3 mote is set as a border router. This border router is linked to the LLN via the Internet. The FIT/IoT-Lab testbed's border router has a public IP address. Californium (Cf) CoAP runs on an i5 PC on the client side. The NSTART parameter is kept by the CoAP client. CoAP's performance is examined for five different NSTART (NSTART =1-5) and NSTARTd values. For each network traffic scenario, all tests are run 20 times for a total of 20 minutes. NSTARTd performance is measured and compared to CoCoA using three direct parameters:

throughput, average exchange duration, and average number of retries. For the calculation of retries and throughput in between client and server , they utilise the i per f tool. - Throughput is defined as the number of requests processed per second. - The average exchange duration is the time it takes to obtain an ACK in response to a CON request. - The average number of retries is the how many number of attmepts it would take to get the ACK.

According to August Betzler [7], the development of extendible along with dependable IoT protocols contexts remains a challenge. Only CoAP delivers systematized RESTful services for tools of IoT, implements a basic strategy for controlling congestion named stop-and-wait strategy and a superficial dependability procedure.
When broadcasting a new packet, CoAP considers the received ACKs. Furthermore, loss of packets is already considered as a mark of congestion in the network, and an exponential back-off between retransmissions is provided to minimise rates of sending. CoAP extensions have recently been proposed to offer dynamic retransmission timeout adaption.

According to Carles Gomez [4], in the IoT, the number and variety of networked devices are continuously rising, resulting in wide range of novel. It was assumed that more than 30 billion things will be linked with internet by the end of year 2020. CoAP is intended to be the primary application-layer protocol for IoT devices to use for IP-based HTTP-like interactions. Limited devices are used for IoT communications. These gadgets have extremely low processor and memory capacity. Furthermore, these devices' communication methods have severe disadvantages, like limited data speeds. The main advantage of CoAP is to accomodate these restrictions. Network congestion

is one of the most important issues to address while developing a new communication model. This situation mainly occurs when network capacity is reached by the network. When the traffic load given to , this situation occurs. TCP enables end-to-end congestion control in many classic Internet applications. The main problem which should be kept in mind when developing a new model is congestion in network. TCP allows for end-to-end congestion control in many traditional Internet applications. CoAP, on the other hand, uses UDP to enable lightweight programmes and must deal with congestion on its own.

According to Josep Paradells [11], IPv6 enabled networks of limited devices are critical in the endevour to merge the IoT in daily life. The Internet of Things has created new obstacles for the development of protocols and standards used by devices with limited hardware and communication capabilities. This study focuses on the application layer protocol CoAP, which was intended for IoT networks of restricted devices. Such networks have restricted cache. Congestion, however, is a serious issue for communications in these networks. The main reason for the occurrence of network congestion is when the network capacity and traffic lost seem to be equal or when node queuing and storing limits are surpassed. When messages are sent and received between many devices, the network congestion may occur. To solve this critical issue, there is a congetion management method which is defined by CoAP. In prior work, they differentiated between the default CoAP CC mechanisms to the alternative CC mechanisms given in the CoCoA draught version 0. The results demonstrated that CoCoA can improve the default CoAP's inadequate CC capacities. Modifications and enhancements to the default CoAP and CoCoA are described in this work to solve these inadequacies, leading to an enhanced CC procedure for CoAP, CoCoA+.

According to Shruti Singala [9], The introduction of the Internet of Things (IoT) and if we incorporate in to everyday life then it will bring a number of issues. When a significant number of IoT devices is trying to connect with each other at same time , the network becomes congested due to they tiny memory and slow processing rates. Because of data packets exchanged by connected devices in IOT network having significant amount of payload , packet loss due to congestion leads in increase in re-transmissions time with extra delays and huge overheads. Mobile component is can also connected to IoT along with static nodes. Because of the small packet size, costly re-transmissions, and mobility, the nature of these limited networks is very different from the traditional networks that are now in place. The first issue to be addressed in the Internet of Things is network congestion. Existing congestion control methods in the Internet of Things rely on immediate Round Trip Time (RTT) data. The main problem statement is to develop a more efficient parameter for measuring network congestion. In this paper, a new congestion control algorithm is proposed called as CoCoA++ to handle the issue of congestion that occurs in network of Internet of Things (IoT). As like congestion control mechanisms that operate by measurement of instantaneous Round Trip Time (RTT) in IoT, they consider different parameter that is delay gradients for the better measurement of congestion in the network , and implement also a new probabilistic backoff factor so that it can deal with congestion effectively. they integrate the delay gradients and the probability backoff factor with Constrained Application Protocol (CoAP). On the Cooja network simulator proposed algorithm is implemented and evaluated provided by Contiki OS. After that it is also deployed and evaluated in a real testbed by using the FIT/IoT-LAB. They find that gradients of the delay to provide a more accurate estimate of congestion and that the Retransmission Time Out (RTO) is dramatically lowered, resulting in less delays and higher packet sending rates. CoCoA++, a small enhancement over the present technique, is simple to implement.

According to Carlo Vallati b [5], the INTERNET OF THINGS (IoT) is gaining traction in academia as a hot research area, as well as in industry, with smart products that are designed to have a profound impact on our lives. IoT systems are created on the backs of IoT devices that collect data and interact with the actual world. IoT devices are often low-cost, battery-powered devices with little compute capability. These devices are outfitted with low-power transceivers, allowing the formation of Low Power and Lossy Networks (LLN), which are characterised by potentially high packet error rates and limited throughput. These characteristics necessitate the development of new communication protocols designed for limited devices working in lossy networks. To that end, the IETF established a communication protocol stack for IoT devices. Congestion control is critical to enable proper and timely data delivery in LLNs due to the limited capacities of nodes and the limited amount of bandwidth available. Traditional networks typically carry application data using the Transmission Control Protocol (TCP), which contains a congestion control mechanism. Instead, in LLNs, CoAP uses the User Datagram Protocol (UDP), which lacks a congestion control mechanism. As a result, CoAP enables optional dependable data delivery via retransmissions and regulates data transfer with a simple congestion avoidance method. The latter, on the other hand, is not so simple to define: the shared nature of the wireless channel causes transmission collisions, and the restricted buffer size of IoT devices frequently causes buffer overflows. Both behaviours can result in frequent packet losses, which leads to additional message retransmissions and, eventually, congestion. They give an in-depth performance evaluation of the CoCoA congestion control algorithm in this paper.1 First, they show an evaluation of the method in two scenarios: one with solely CoAP traffic and one with CoAP traffic competing with interfering greedy traffic that is broadcast without rate control. An in-depth examination of these simulation data enables us to identify some novel key flaws with the technique. As a result, they suggest a set of improvements to the original method to address these

shortcomings. The resulting pCoCoA algorithm is compared to CoAP, CoCoA+, and two more modern algorithms: 4-state-strong and CoCoA-E. The results showed that pCoCoA effectively mitigates the original algorithm's shortcomings, reducing the overall number of retransmissions while maintaining throughputs and latency equivalent to CoAP, CoCoA, CoCoA-E, and 4-state-strong.

According to Simone Bolettieri [**?**], the development of scalable and dependable transport protocols for IoT contexts remains a challenge. Only CoAP, an application protocol that delivers standardised RESTful services for IoT devices, implements a basic stop-and-wait congestion control strategy and a lightweight dependability mechanism. When broadcasting a new packet, CoAP considers the received ACKs. Furthermore, packet loss is implicitly assumed as a network congestion indication, and an exponential back-off between retransmissions is provided to minimise sending rates. CoAP extensions have recently been proposed to offer dynamic retransmission timeout adaption. Recent research, however, have looked into the drawbacks of these techniques, such as unfairness and bogus retransmissions when given loads are close to congestion. One of their objectives is to show that valid RTT measurements may be obtained in constrained environments with minimal extra overhead. Unlike earlier similar systems, they do not use specified RTT parameters to predict the beginning of congestion. Instead, they exploit RTT variability to build a rate control technique that seeks to minimise data transaction losses. RTT-CoAP is a modified version of the CoAP web transfer protocol that includes RTT-based congestion control technique for 6LoWPAN networks. RTT-CoAP, as opposed to traditional TCP, employs a rate-based algorithm that directly controls the pacing rate of packet transmissions to prevent traffic burstiness and data losses.

Chapter 4

APPROACH

4.1 Introduction

One of the key factor in a network that affects the throughput of the network is congestion in the network. Large number of clients leads to large traffic in the network due to more packets being pushed in the network this leads to congestion in the network. As the congestion in the netowrk increases packets take longer to reach the destination server and some of the packets may get dropped or timedout.Mostly protocols have mechanism for tackling this congestion issue .The Coap Protocol [2] uses exponential back off technique, in this RTO(Retransmission Timeout) increases exponentially every time a packet is timedout. RTO is the time period for which the client stops sending traffic to the server.Method of RTO calculation was changed in the CoCoA [7] Protocol which did dynamic RTO estimations using something known as a VBF(Variable Backoff Factor).But the basic idea is common in all the approaches that is if congestion increases which is measured as the number of packets that timed out RTO will increase.

Large value of RTO lead to lesser channel utilization due to the fact that

clients will wait longer before sending a request.This is good in the case when there is too much congestion and sending in more packets will lead to more traffic . But in the scenario when the traffic has decresed and congestion has gone down ,higher RTO will lead to wastage of channel resources.Less channel utilization leads to less throughput. So the main aim of our approach is to increase the NSTART just so as to increase throughput and prevent timeout condition. So in our approach have varied NSTART with the goal of preventing RTO value from getting too large. We have ensured this by avoiding RTT values from getting too large. As larger RTT value means more congestion and RTT is also used to calculate the dynamic RTO estimate in CoCoA protocol [7].

A Problem with existing approach[10] (dynamic NSTART) is that in the algorithm proposed by it to vary NSTART dynamically there is no constraint on the equality condition when $PC_r = CC_r$ that when previous congestion ratio is equal current congestion ratio.In that case the algorithm dependent on the Th_d value which if set high can cause RTT values get too big and affect overall throughput by increasing the RTO value.For example,
Assume,
$RTT_0 = 1$
$RTT_1 = 2$
$cf > 2$
Now next $RTT_2 = 4$
As per dynamic NSTART this doesnot cause any change in the NSTART despite the increasing RTT.

4.2 Approach

When we discussing about the approach we used to modify the NSTART value there are 3 main questions.

1. When do we increase the value of NSTART?
2. When do we decrease the value of NSTART?
3. When do we not change the NSTART?

We have proposed answers to these questions after discussing the flow chart and algorithm in the next section.

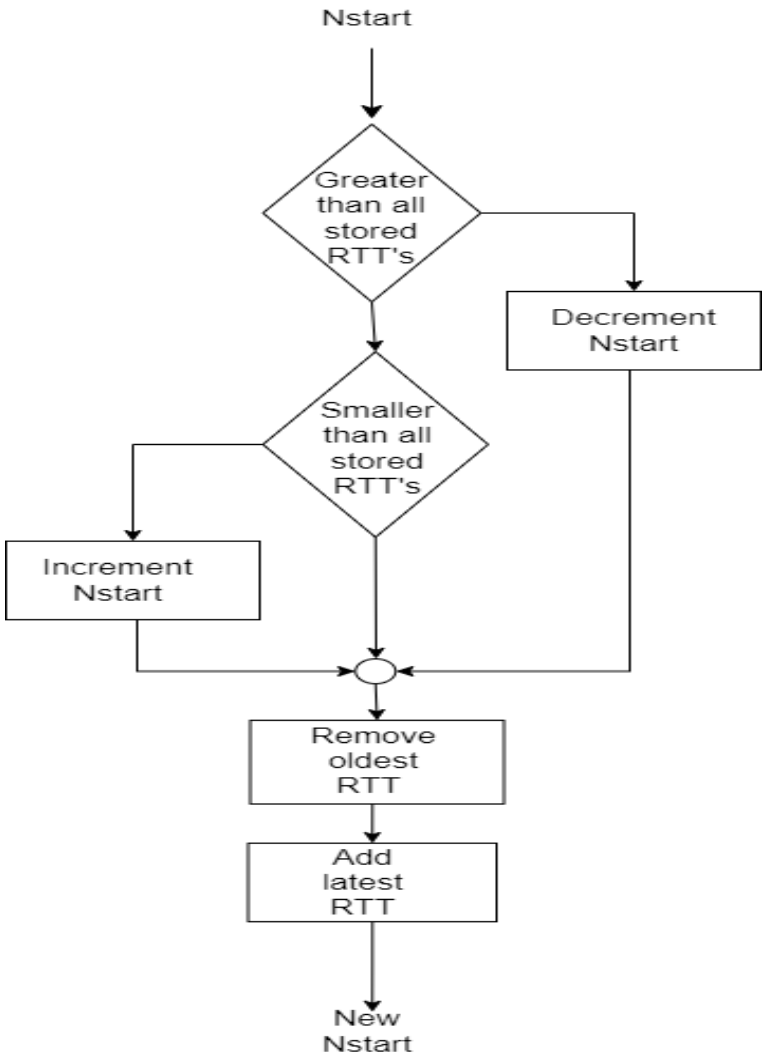

Figure 4.1: Flow chart of the proposed algorithm

4.2.1 Algorithm

Algorithm 1 An algorithm for dynamic NSTART value

Require: $NSTART$, RTT_Q , Q_LEN , MAX_NSTART , RTT
Ensure: $NSTART = 4$ ▷ Intial NSTART value is default 4
  if len(RTT_Q) $\neq Q\_LEN$ **then**
    push $RTT \rightarrow RTT\_Q$
    return
  end if
  if $RTT > max(RTT\_Q)$ & $NSTART > MIN\_NSTART$ **then**
    $NSTART \leftarrow NSTART - Df$
  end if
  if $RTT < min(RTT\_Q)$ & $NSTART < MAX\_NSTART$ **then**
    $NSTART \leftarrow NSTART + If$
  end if
  pop $\rightarrow RTT\_Q$
  push $RTT \rightarrow RTT\_Q$

In the algorithm we use a queue RTT_Q which is used to maintain the last Q_LEN RTT's We use these RTT's to decide the NSTART values. We ensure that the NSTART values are always in the range $[MIN\_NSTART, MAX\_NSTART]$. The value of Q_LEN can be set according to the network.If and Df are the steps of increment and decrement that are taken whilst changing the NSTART value.

Going back to example we took to explain the flaw in Dynamic NSTART [10] .For our approach we consider Q_LEN to be 2 for this example then after time $t = 1$:
$RTT\_Q \rightarrow [1, 2]$
when RTT_2 comes since it is greater than max(RTT_Q) we decrease NSTART hence staying in the safe zone and preventing the congestion condition.

4.2.2 When do we increase the value of NSTART

The major factor affecting if we need to increase the NSTART value is the congestion of the network. If the network is already congested we should not push more packets in the network as it will increase the congestion which will in turn increase the value of RTO *Retransmissiontimeout* , which eventually lead to poor throughput.

But since we managing the congestion from the end devices. We are using RTT as a measure of the congestion in the network. We only increase the NSTART value if the current RTT value is smaller than the minimum value present in the RTT_Q. This ensures that the network is in the least congested state of the last Q_LEN packets transactions. So it is safe to increase the NSTART value.

Then the next step is how much do we increase the value of NSTART . This is denoted by the If or the increment factor. It is the step size by which we increase the NSTART value after we hit the increment condition. This can also be set according to network depending in how prone it is to congestion or it fast it recovers from congestion.

4.2.3 When do we decrease the value of NSTART

We only decrease the NSTART value if the current RTT value is larger than the maximum value present in the RTT_Q. This ensures that the network is in the most congested state of the last Q_LEN packets transactions. So NSTART must decrease to prevent packet drops and increasing RTO value.

Then the next step is how much do we decrease the value of NSTART . This is denoted by the Df or the decrement factor. It is the step size by which we decrease the NSTART value after we hit the decrement condition. This can also be set according to network depending in how prone it is to congestion or it fast it recovers from congestion.

4.2.4 When do we not change the NSTART

We don't vary the value of NSTART if there a value greater than and a value smaller than the current RTT value present in the RTT_Q . This ensures safe congestion state.This condition means that if the congestion was high it is decreasing and if it was low we are trying to improve throughput.

Chapter 5

RESULT & TESTING

5.1 Test setup

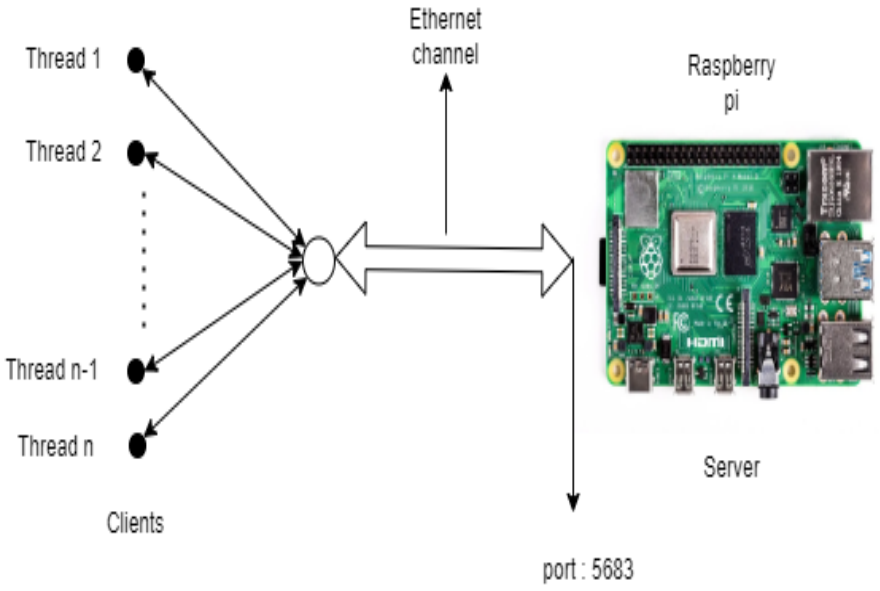

Figure 5.1: Testing Prototype

For our testing our approach we have used the single server multiple client network model similar to the one used in [6] for testing their approach.We have used Raspberry Pi Model B+ V1.2 as the server.The specification of the server is 700 MHz CPU, 512 MB SDRAM, $frac$10100 BaseT Ethernet, and 8 GB Micro SD card with Raspbian Linux OS. The server is setup using Californium [1] which is an open source java based implementation of CoAP. For clients multiclients are implemented based on threads which are spawned on a laptop. The specification of client PC is 8 GB DDR4-3200 MHz RAM (1 x 8 GB) 512 GB PCIe® NVMe™ M.2 SSD ,64. Between client and server UDP (User Datagram Protocol) is used for Ethernet communication.

Our algorithm was tested on different number of clients each sending 50 requests each.We ran all the clients with CoCoA basic which was provided by Californium and with our congestion control proposal and compared the results with standard CocoA implementation and with the algorithm proposed in .Since there is great variation in results ,We ran the same scenario 10 times and have taken the average results.

As parameters of performance measure for this proof of work we have used the throughput every client gets and % CPU utilization at the server end. Throughput has been measured by the californium clients.

$$\text{Throughtput} = \frac{Sizeofdatatransfered(d)}{timetaken(t)}$$

Since size of data transfered is constant for all clients we take it to be constant.So we can see Throughput is inversely propotional to the time a client takes to transfer its data also known as the turn around time.So from now on we will evalutaing the performance in terms of this turnaround time

For testing we have used 5 scenarios each with the same setup but with a different of requesting clients. Probablity of congestion in the network increases with number of clients.The number of clients in the 5 scenarios are

10,20,30,40,50 respectively.

5.2 Performance Evaluation

Average turnaround time for all of the testing scenarios are given in the table. In this also our algorithm out performs the others.There is not much difference in the case where number of clients in less that is 10,20. But when number of client are 30,40,50 or algorithm gives way better results.We have also plotted the results for better understanding . The X axis shows the 5 testing scenarios 0 represents case 0 when number of clients equal 10 and 4 represents case 4 when number of clients is 50.We can see the trend that our proposed algorithm gives better results than the other 2 algorithms.

Congestion Control techniques	n=10	n=20	n=30	n=40	n=50
CoCoA	159.2ms	398.7ms	452.5ms	609.5ms	811.1ms
Proposed	126.3ms	261.1ms	360.3ms	479.2ms	624.0ms
Dynamic NSTARTd	124.3ms	296.8ms	401.4ms	534.5 ms	706.0ms

Table 5.1: Average turnaround time of Approaches

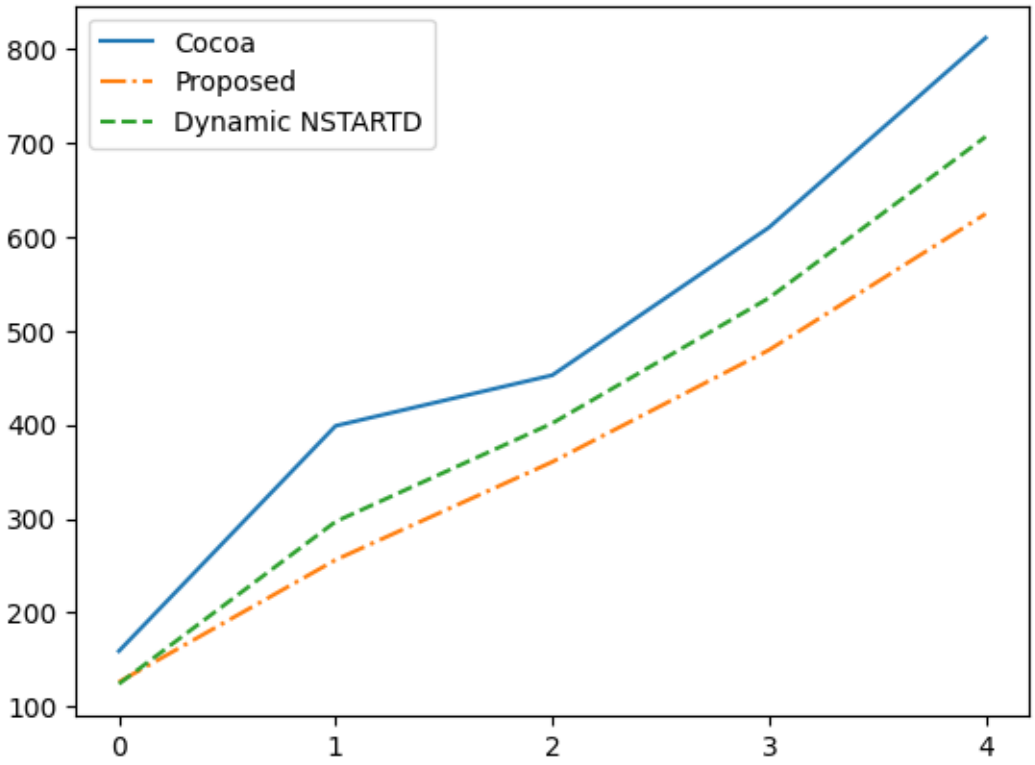

Figure 5.2: Average turnaround time vs Number of parallel clients

For better understanding these results we also plotted the turnaround for each individual client.The x-axis of these plots represent each client in increasing order of their turnaround time. The y-axis shows the corresponding turnaround time for the client
The throughput of every client is inversely proportional to the time taken by each client to finish their job since there message size are the same.So we have plotted a time versus number of clients graph. Each point on the x-axis is a client and the corresponding y-axis value is the time taken by client to finish its task or its turn around time.

As clearly visible in graphs the proposed algorithm does not offer much

benefits for the early number of clients as these are the ones who didnot get affected by congestion and hence their RTO was not affect much but as far as the later clients are concerned these are the ones whose packets got timedout due to congestion.These are one which we targeted in the algorithm and prevented their RTO from increasing too much and hence improved their throughput.For such clients the proposed algorithm performs best as it varies NSTART such as to prevent more congestion and at the same time maximizing channel utilization.

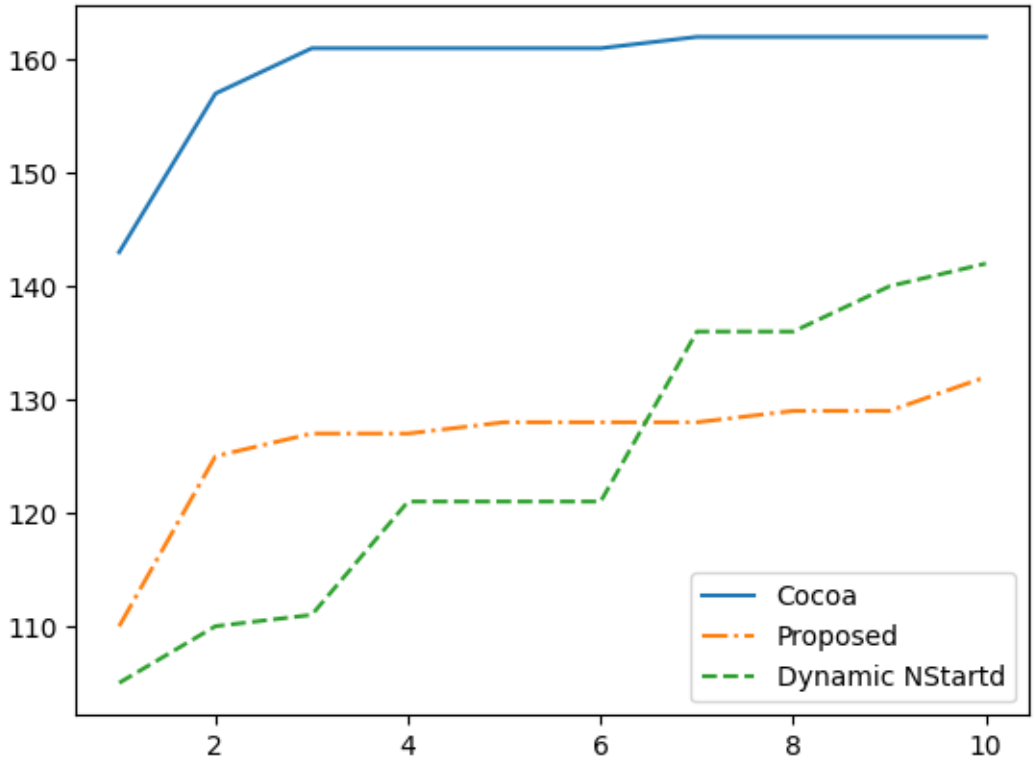

Figure 5.3: 10 clients

As explained above for less number of clients our approach does not provide much benefit as there is less congestion.But even then the clients that

are most affected by congestion that is the clients 6 and onwards show better trend on the turn around time because of prevention of congestion condition.CoCoA performs the worst because of static NSTART condition.

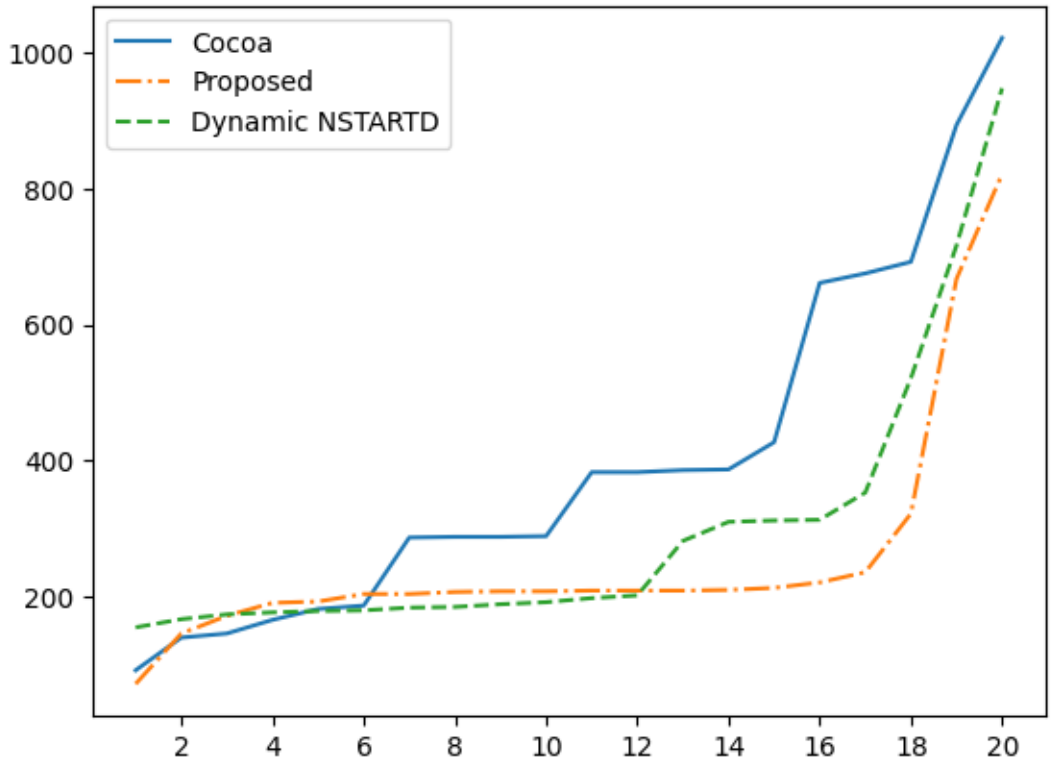

Figure 5.4: 20 clients

At 20 clients the distinction between Proposed and NSTARTd start to become clear.They both perform almost same for clients with low turnaround time that is client number ¡ 12, but after that proposed algorithm graph stays below the former.Although there is a steep increase in the turnaround value from the 18^{th} client onwards.This can be explained by the inevitable increase in congestion in the network despite both the algorithm.

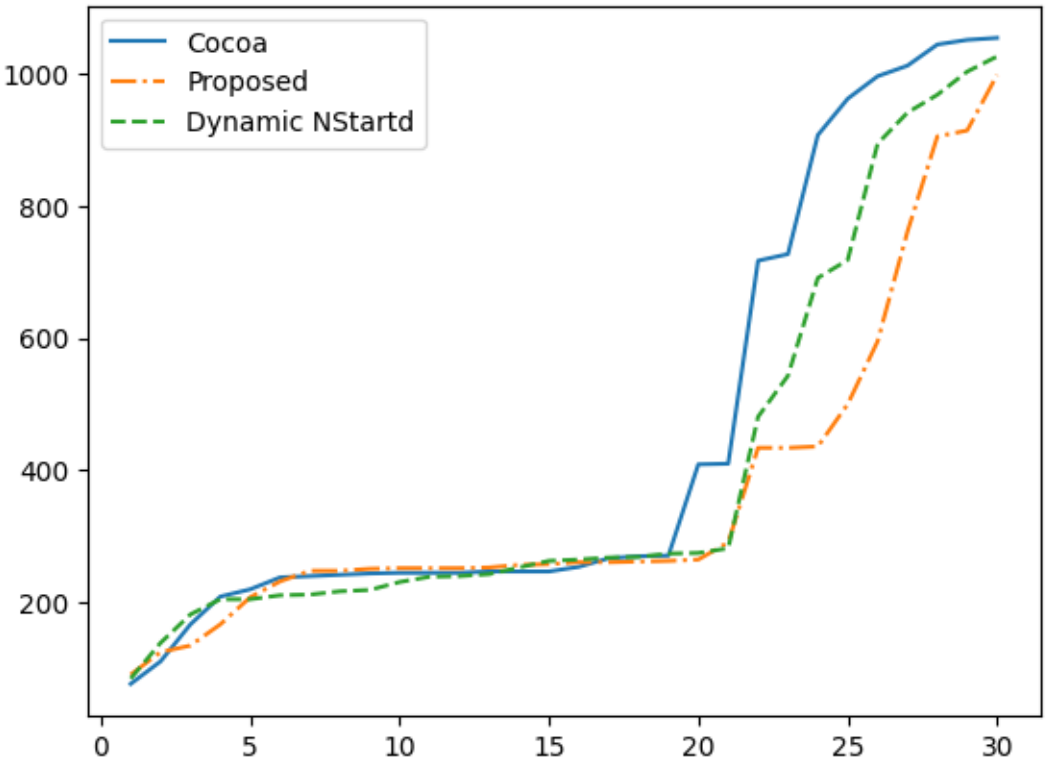

Figure 5.5: 30 clients

When number of clients is 30 , for less congested client (i.e. those least affected by congestion) around till 17 the 3 algorithm give almost identical performances.After that CoCoA shows a steep increase in turnaround time due to possible increase in RTO as discussed in earlier sections.Both Nstartd and Proposed algorithms also show this increase around the 21^{st} client but then Proposed shows better results for congested clients.

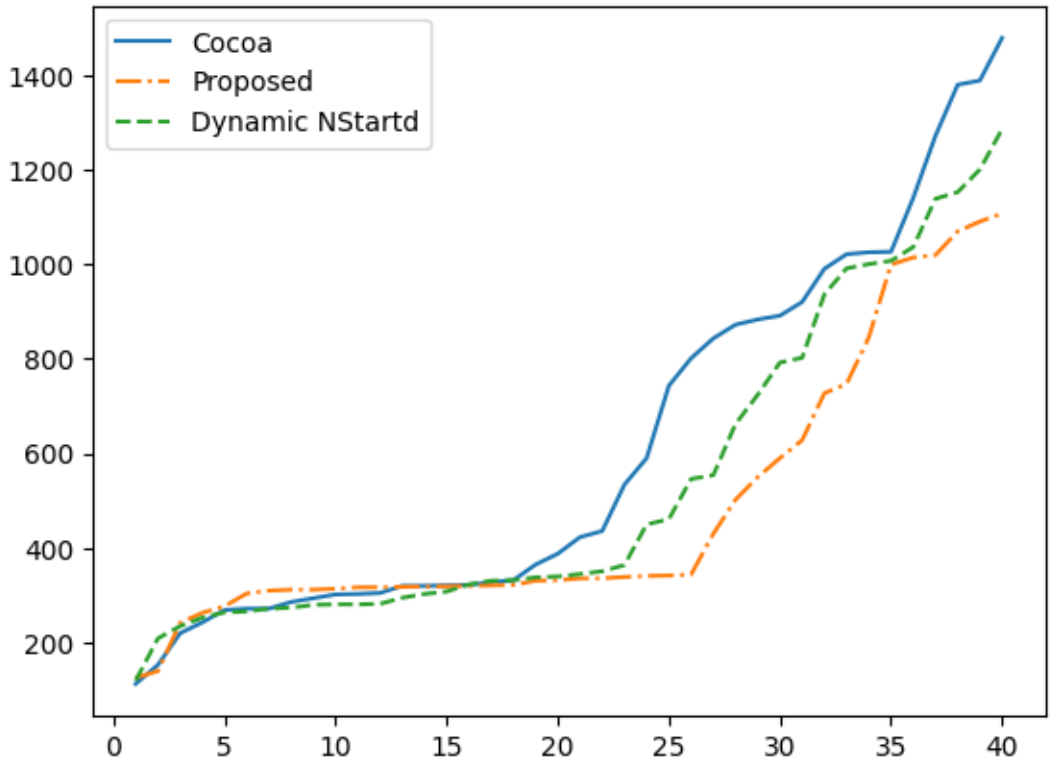

Figure 5.6: 40 clients

When number of clients is 40 the CoCoa is the first to show a increase in the turn around time followed by Nstartd and lastly proposed algorithm.There is a point where the graphs appear to be meeting this is due to the scale of the graph.

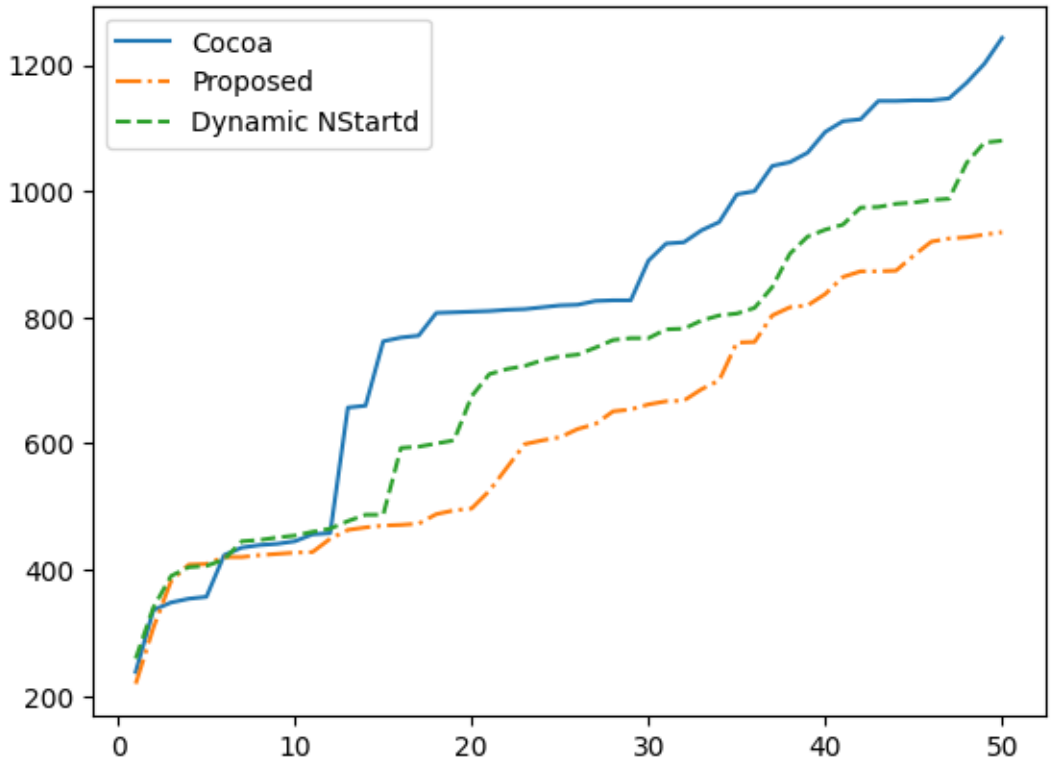

Figure 5.7: 50 clients

When number of clients is 50 the CoCoa is the first to show a increase in the turn around time followed by Nstartd and lastly proposed algorithm this is similar to the pattern visible in the n=30 and n=40 case.

Chapter 6

CONCLUSION

The main aim of this project was to find a way to change NSTART dynamically so as to improve the throughput of the network and our algorithm was able to do so on the network we tested. There were best overall throughput especially for congested clients.

Chapter 7

FUTURE SCOPE

The proposed algorithm has been tested in one kind of network scenario only. So future aspect of this project are to evaluate the proposed algorithm on different network environment and adding in new parameters for evaluation of the algorithm. The algorithm can also be tested on Arduino and other constrained devices and results shall be verified.

Bibliography

[1] Californium (cf) - coap for java. Technical report. URL: https://github.com/eclipse-californium/californium.

[2] The constrained application protocol (coap). Rfc, RFC Editor, 2004. URL: https://www.rfc-editor.org/rfc/rfc7252.

[3] Ilker Demirkol Matthias Kovatsh August Bet-zler, Carles Gomez. Congestion control for coap cloud services. pages 14–17. URL: https://ieeexplore.ieee.org/abstract/document/7005340?casa_token=5gudgXjsnnoAAAAA:JUzuBYnOtfphOAEkyo6lR134vaFabLb81ChHstTlMOhgmmUO1mHlq9OWQu4QlpMCWq6-T54Xngn5.

[4] Ilker Demirkol Josep Paradells August Betzler, Carles Gomez. Cocoa+: An advanced congestion control mechanism for coap. pages 22–24. URL: https://www.sciencedirect.com/science/article/abs/pii/S1570870515000888?casa_token=0JjGiFgh-vsAAAAA:5mVUvLORhOB6pFRd-0-Rog1EglRbMwLgwxgIMx9d4ttsxFbewKGfP31zBTmsUx5RFfZRp2SNhztW.

[5] Simone Bolettieri Raffaele Brun Emilio Ancillotti. Rtt-based congestion control for the internet of things. pages 29–30. URL: https://link.springer.com/chapter/10.1007/978-3-030-02931-9_1.

[6] Jung June Lee Kyung Tae Kim and Hee Yong Youn. Enhancement of congestion control of constrained application protocol/ congestion control/advanced for internet of things environment. pages

9–10. URL: https://journals.sagepub.com/doi/full/10.1177/1550147716676274.

[7] August Betzler; Carles Gomez; Ilker Demirkol; Josep Paradells. Coap congestion control for the internet of things. pages 18–21. URL: https://ieeexplore.ieee.org/abstract/document/7509394?casa_token=P5VgVrV6Hd4AAAAA:QP43J4EgugzMKVGAotc-bgUwCYH-5OXTRKInCbf9Pzftn5EgayxW92N08c5CkFrpnSNoTyP6n-eo.

[8] H.O.; Perkusich Santos, D.F.; Almeida. A personal connected health system for the internet of things based on the constrained application protocol. pages 10–11. URL: https://www.sciencedirect.com/science/article/abs/pii/S0045790615000683?casa_token=4w2qIf9SoAsAAAAA:B-yUmzT-0Gg2l5WUq1795_eiF-_qn4dhpwK0KTRWS-KjLKqSqK4pn8cfQr13AERunyki3mFJI0z.

[9] Carlo Vallati b Enzo Mingozzi Simone Bolettieri a, Giacomo Tanganelli b. pcocoa: A precise congestion control algorithm for coap. pages 26–28. URL: https://www.sciencedirect.com/science/article/abs/pii/S1570870518303834?casa_token=PCWcnkjPRkcAAAAA:kwlKrJ4PENi_D9KgzSYcRkwvqjZX4ozqlOEITrbCopf-3hdH-paqu2Qd-gsuQynAwdeYwBtCnBVf.

[10] Mahesh Chandra Govil Vinesh Kumar Jain, Arka Prokash Mazumdar1. Toward adaptive range for parallel connections in coap. pages 12–13. URL: https://link.springer.com/article/10.1007/s13369-020-05215-w.

[11] Samanvita Sastry Shruti Singala Mohit P. Tahiliani Vishal Rathod, Natasha Jeppu. Delay gradient based congestion control for internet of things. pages 25–26. URL: https://www.sciencedirect.com/science/article/abs/pii/S0167739X18308677?casa_token=_rXXXRzJsjEAAAAA:7cKxzAl5ju7SFCGlfKS1axmDMwIx_wXWrpZlhGerlkfoaz7hzDLQD0696s87KOySJtu2t1yeUlFK.