

CoAP BASED CONGESTION CONTROL MECHANISMS FOR INTERNET OF THINGS

Thesis

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

Rathod Vishal Jitendrakumar
(155014 CS15F11)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA,
SURATHKAL, MANGALORE - 575 025

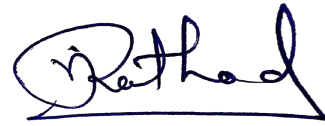
January, 2022

*Dedicated to
Lord Swaminarayan Bhagwan, the Almighty God
and to my ancestors*

DECLARATION

by the Ph.D. Research Scholar

I hereby **declare** that the Research Thesis entitled **CoAP based Congestion Control Mechanisms for Internet of Things** which is being submitted to the **National Institute of Technology Karnataka (NITK), Surathkal** in partial fulfilment of the requirements for the award of the Degree of **Doctor of Philosophy in Computer Science and Engineering** is a **bonafide report of the research work carried out by me**. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.



(155014 CS15F11, Rathod Vishal Jitendrakumar)

(Register Number, Name & Signature of Research Scholar)

Department of Computer Science and Engineering

Place: NITK, Surathkal.

Date: January 28, 2022

CERTIFICATE

This is to *certify* that the Research Thesis entitled **CoAP based Congestion Control Mechanisms for Internet of Things** submitted by **Rathod Vishal Jiten-drakumar**, (Register Number: **155014 CS15F11**) as the record of the research work carried out by him, is *accepted as the Research Thesis submission* in partial fulfillment of the requirements for the award of degree of **Doctor of Philosophy**.

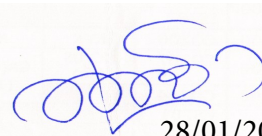


28/01/2022

Dr. Mohit P. Tahiliani

Research Supervisor

(Name and Signature with Date and Seal)



28/01/2022

Dr. Shashidhar G. Koolagudi

Chairman - DRPC

(Name and Signature with Date and Seal)

Acknowledgements

First and foremost, I would like to thank **Swaminarayan Bhagwan**, the Almighty, for his blessings, strength and a healthy life in bringing this research to a successful conclusion.

I would like to express my deep and sincere gratitude to my supervisor, **Dr. Mohit P. Tahiliani** (Assistant Professor, CSE Department) for his priceless advice and unwavering support. During my Ph.D. at NITK, Surathkal, he inspired me to achieve all of my goals and ambitions. I will always cherish all the useful discussions and brainstorming sessions that we did. His wealth of knowledge and experience has aided me at each stage of my research. He assisted me in overcoming many obstacles along the way, and without his assistance, I would not have been able to accomplish what I did.

I would like to thank my Research Progress Assessment Committee (RPAC) Members **Dr. Basavaraj Talawar** (Assistant Professor, CSE Department) and **Dr. Ashvini Chaturvedi** (Associate Dean and Professor, ECE Department) for their guidance, valuable comments and support throughout the period of my research.

I am extremely grateful to **Dr. Shashidhar G. Koolagudi** (HoD, Associate Prof. and DRPC Chairman, CSE Department) and **Dr. Alwyn Roshan Pais** (ex-HoD, Associate Prof., CSE Department) for their perseverance, inspiration, and encouragement throughout my Ph.D. journey.

My heartfelt gratitude goes to **Mrs. Manasa Tahiliani** for providing a family-like environment. I will always remember the precious memories of **Ved** and **Ridvi** playing together. I would also like to thank **Dr. Prakash U. Tahiliani** for his medical assistance.

I am grateful to the faculties of the CSE department, **Prof. P. Shanti Thilagam**, **Prof. K. Chandrasekaran**, **Prof. Annappa**, **Associate Prof. Vani M.**, **Dr. Manu Basavaraju** and **Assistant Prof. Dr. Jeny Rajan**, **Dr. Saumya Hegde**, **Dr. Biswajit R Bhowmik**, **Dr. M. Venkatesan**, **Dr. Mahendra Pratap Singh** and **Dr. Sourav Kanti Addya** for their support during my study.

I would like to thank the technical and support staff of our department, **Mr. Dinesh**

Kamath, Mrs. Harshitha Shetty, Mrs. Mohini, Mr. Rakesh, Mrs. Seema Shivararam, Mr. Sumedha Rao, Mrs. Yashavanthi, Mr. Pradeep D, Miss. Supriya, Ravi, Arun Kumar and Mrs. Vanitha, for assisting me in running my Proposal and Progress Seminars smoothly and for their ongoing support during my time at NITK.

I am grateful to **Dr. B. R. Chandavarkar** and **NITK WiNG** (Wireless Information Networking Group) for giving me the opportunity to be part of this group. I would like to thank the other members of WiNG, **Radhesh, Bhanu Priya, Vivek, Vyom, Sourabh, Shefali, Jendaipo, Shikha, Apoorva, Aditya, Susma** for the wonderful moments we spent on my Ph.D. journey. I would like to thank a few B.Tech students, **Natasha, Samanvita, Shruti, Leslie, Gautam, Ayush, Gauri** and **Sanjana** or their lengthy discussions and assistance in problem solving.

A special thanks to **Sachin Sir (Bhau)**, who is always willing to help me in any situation at any time, and I will never forget the conversations we had during tea break. It was a pleasure to share the research laboratory with my department colleagues, **Mr Pravin Ramteke, Dr. Manoj, Dr. Likewin, Dr. Vishnu, Dr. Sumith, Dr. Bheemappa, Dr. Girish, Dr. Anil, Dr. Khyamlinga, Dr. Prabhu, Dr. Alok, Dr. Apurva, Ambikesh, Dr. D. V. N. Shiva, Majunatha, Dr. Manjunath Mulimani, Dr. Nagaratna, Dr. Raghavan, Dr. Pramod, Dr. Nikhil, Kallinatha, Sneha, Shubham, Spoorthy, Arabhi** and other peers for their encouragement and made my stay happier at NITK. I would also like to thank research scholars from other departments, in particular **Arun Kumar (AMD), Uday Patil (Electrical), Deepak Kumar and Swamy Sir (Mechanical)** and **Sunil Metti and Khalifa sir (MME)**, for the wonderful moments we shared during the Research Premier League (RPL) tournament.

I would like to express my gratitude to **Mr. Pritam (Broker)** for assisting me in my search for a home near NITK. My special thanks to **Mr. Devraj Salian Kodical and his family members** for their support and for making my stay at 301, Nandshree Apartment a better place to stay during my Ph.D. I would like to thank everyone at Nandshree Apartment for making my stay so memorable. I would like to express my gratitude to **Mr. Namdev Suryavanshi** and **Mrs. Jayshree Suryavanshi** for treating us like family. His daughter **Neha** is Ridvi's best friend, and the moments she spent with my daughter were unforgettable. His son **Sujay** is always willing to assist at any time and freely shared his thoughts.

I would like to express my gratitude to **Dr. Biju R. Mohan** (HoD, IT Department)

and **Prof. Sai Dutta**, **Prof. Vidya Shetty** and **Prof. Dyanand Shetty** for giving me the opportunity to become a part of Direct Admission of Students Abroad (DASA) for two consecutive years 2019 and 2020. Working in a real-time application was a fantastic experience that allowed me to brush up on my programming skills.

I would like to thank the Director of NITK (**Prof. K. N. Lokesh** and **Prof. Uma Maheshwara Rao**) and all the staff members of the main building for their support and the provision of infrastructure to work in a wonderful environment.

A special thanks to my childhood friends, **Malav**, **Pranav**, **Pathik**, **Bhaumik**, **Palak** and **Jigar**, who have always inspired and encouraged me to pursue higher education.

I am eternally grateful to my B.Tech friends, **Shashin**, **Chintan**, **Manan**, **Nisarg**, **Kaustubh**, **Nirmal**, **Ravi**, **Harshil**, **Dipak**, **Ripal** and **Vibha** for their mental and financial support.

I would like to express my gratitude to my financial advisor, **Mr. Jignesh Shah** for managing my financial portfolio. I would also like to thank **Mr. Vipul Gajjar**, **Tarjni** and **Rekha madam** for their enduring assistance.

I would like to thank my colleagues from the CSE Department, CHARUSAT University, Changa, where I worked before coming to NITK, for always encouraging me to pursue Ph.D. at one of the NITs or IITs. A special thank you to **Mr. Nishidh Chavda** for teaching me a core subject and guiding me on how to solve problems quickly and efficiently. I would like to thank the other CHARUSAT colleagues who kept in touch with me throughout my Ph.D. journey, especially **Dr. Bimal Patel**, **Dr. Nirav Bhatt**, **Mr. Ashish Patel**, **Mr. Hardik Mandora**, **Dr. Nikita Bhatt**, **Dr. Chintan Bhatt**, **Mr. Divyesh Patel**, **Mr. Sagar Patel**, **Dr. Ashwin Makawana**, **Dr. Amit Ganatra**, **Mrs. Arpita Shah**, **Mr. Dhaval Bhoi**, **Mr. Mayur Patel**, **Dr. Ritesh Patel** and **Dr. Parth Shah**.

Special thanks to all the people who have always pleased me with their wishes with a smile on my daily routine to NITK.

I am extremely grateful to my parents, **Mr. Jitendrakumar Rathod** and **Mrs. Sadhanaben Rathod**, for their love, care and unwavering support in educating and preparing me for the future. Without them, I would not be the person I am today.

I express my gratitude to my sister **Tejal**, her husband **Shubham** and niece **Krishika** for their support and prayers. I would also like to thank **Kalpesh** and **Bhaves**, my

elder cousins, for always being there for me and inspiring me.

I would like to thanks to my in-laws **Pansiniya family** (**Ashwinbhai, Nayanaben, Manishbhai, Jigarbhai, Latabhabhi, Kavitabhabhi, Vishv, Manvi and Arsh**, who have always supported me throughout my Ph.D. journey.

And last but not least, aa heartfelt thanks to my better half, **Margi**. She was a constant source of inspiration and strength. She remained by my side to provide mental support and to keep me from making a number of mistakes. I would like to acknowledge my daughter, **Ridvi**, for being the most important person in my life. I feel rejuvenated after playing with her. The time I spent with her was memorable, and I will remember one thing she said every day, "Are you going to college? Please return early, and I will open a door for you."

Finally, I would like to express my heartfelt gratitude to everyone who has contributed to this thesis and supported me in some way during this incredible journey.

Place: Surathkal

Rathod Vishal Jitendrakumar

Date: January 28, 2022

Abstract

Internet of Things (IoT) is a network where physical objects with Internet connectivity can interact and exchange information with other connected objects. IoT devices are constrained in terms of power and memory, and have limited communication capabilities. A large number of IoT devices are characterized by small memory and low processing speeds that lead to congestion in the network when many such devices try to communicate with each other. Typically, TCP is responsible for end-to-end congestion control and reliability. However, in a constrained network, this can cause performance problems because most of the communication happens over wireless links that are known to be challenging for TCP. Consequently, the responsibility for controlling congestion is entrusted to the application layer protocols. Among several IoT application layer protocols, only the Constrained Application Protocol (CoAP) has a built-in congestion control mechanism and has been standardized by the IETF.

CoAP is a lightweight messaging protocol which is widely used by various IoT applications in low power and lossy wireless networks. CoAP provides reliability and *minimal* congestion control via a *fixed* Retransmission Timeout (RTO) and Binary Exponential Backoff (BEB). It does not maintain end-to-end connection information and therefore, cannot adapt RTO based on the network conditions. CoAP Simple Congestion Control/Advanced (CoCoA) is an *improved* congestion control mechanism for CoAP which adapts RTO based on the network conditions. Congestion control mechanisms for CoAP can be classified into non-RTT and RTT based mechanisms. Non-RTT based mechanisms (e.g., CoAP) are advantageous for applications that require a low memory footprint and do not maintain the state information. RTT based mechanisms (e.g., CoCoA) are useful for applications wherein memory footprint is not a major concern and maintaining the state information is desirable.

Recently, it has been shown that the congestion control techniques in CoAP and CoCoA require further optimizations. In this work, we thoroughly evaluate the congestion control mechanisms in CoAP and CoCoA, and subsequently propose one non-RTT and

three RTT based optimizations: (i) Geometric Sequence Technique for Effective RTO Estimation in CoAP (GST-CoAP), (ii) Effective RTO estimation using Eifel Retransmission Timer in CoAP (CoAP-Eifel), (iii) Geometric Series based effective RTO estimation Technique for CoCoA (GSRTC), and (iv) CoCoA++.

GST-CoAP is a non-RTT based mechanism to improve RTO estimation in CoAP by using geometric sequence. CoAP-Eifel, GSRTC and CoCoA++ are RTT based mechanisms. CoAP-Eifel integrates the Eifel Retransmission Timer with CoAP to improve RTO estimations and control congestion. GSRTC is a simple enhancement which *adapts* the weight used in the Strong RTO estimator instead of using the *fixed* weight (0.5). CoCoA++ uses delay gradients to get a better measure of network congestion, and implement a probabilistic backoff to deal with congestion. A common characteristic in the proposed mechanisms is that they are easy to deploy and do not add user configurable parameters. All the four proposed mechanisms are extensively evaluated in a wide variety of network environments using the Cooja simulator and a real testbed at FIT/IoT-LAB. The performance metrics used for evaluation are Flow Completion Times (FCT), number of retransmissions, throughput, RTO, delay and packet sending rate. We observe that the proposed mechanisms offer a notable improvement in network performance compared to the existing mechanisms.

Keywords: Internet of Things, Congestion Control, CoAP, CoCoA

Table of Contents

List of Figures	vii
List of Tables	ix
Abbreviations and Nomenclature	xi
1 Introduction	1
1.1 The Problem	3
1.1.1 Packet loss	3
1.1.2 Round Trip Time (RTT)	4
1.2 Contributions of this thesis	6
1.2.1 GST-CoAP	6
1.2.2 CoAP-Eifel	7
1.2.3 GSRTC	7
1.2.4 CoCoA++	8
1.3 Outline of the thesis	9
2 Background and Literature Review	11
2.1 Background	12
2.1.1 TCP RTO calculation	12
2.1.2 Congestion control in CoAP	13
2.1.3 CoAP Simple Congestion Control/Advanced (CoCoA)	15
2.2 Related Work	17
2.2.1 Application layer solutions	18
2.2.2 Routing layer solutions	27
3 Geometric Sequence Technique for Effective RTO Estimation in CoAP	29
3.1 Motivation	30
3.1.1 Fullbackoff1 Variant	31
3.1.2 Fullbackoff2 Variant	31

3.2	GST for RTO estimation in CoAP	32
3.2.1	Design	32
3.2.2	Implementation Challenges	33
3.3	Evaluation	35
3.3.1	Network Configuration	35
3.3.2	Performance Metrics	36
3.3.3	Results and Discussions	36
3.4	Inferences	38
4	Effective RTO estimation using Eifel Retransmission Timer in CoAP	41
4.1	Motivation	41
4.2	Overview of TCP-Eifel	42
4.2.1	TCP-Eifel	42
4.2.2	Eifel Retransmission Timer	43
4.3	CoAP-Eifel	45
4.3.1	Design	45
4.3.2	RTO estimation in CoAP using Eifel Retransmission Timer	45
4.3.3	Implementation Challenges	46
4.4	Evaluation	47
4.4.1	Network Configuration	47
4.4.2	Results and Discussions	48
4.5	Inferences	52
5	Geometric Series based effective RTO estimation Technique for CoCoA	53
5.1	Motivation	54
5.1.1	Fullbackoff1 Variant	54
5.1.2	Fullbackoff2 Variant	55
5.2	Geometric Series based RTO estimation Technique for CoCoA (GSRTC)	56
5.2.1	Design	56
5.2.2	Parameter Settings	58
5.2.3	Implementation Challenges	59
5.3	Evaluation	59
5.3.1	Experimental Setup	60
5.3.2	Performance Metrics	62
5.3.3	Results and Discussion	62

5.4	Inferences	68
6	CoCoA++: Delay Gradient based Congestion Control for Internet of Things	69
6.1	CoCoA++: Delay Gradient based Congestion Control for Internet of Things	70
6.1.1	Overview of CAIA Delay-Gradient (CDG)	70
6.1.2	Design	71
6.1.3	Implementation Challenges	72
6.2	Evaluation	74
6.2.1	Simulation Setup	74
6.2.2	Simulation Results	79
6.2.3	Testbed Setup	88
6.2.4	Testbed Results	90
6.3	Inferences	91
7	Conclusions and Future Work	93
7.1	Conclusions	93
7.2	Future work	95
	References	97
	List of Publications	105

List of Figures

1.1	Retransmission ambiguity	5
1.2	Our Contributions	6
2.1	IoT Protocol Stack	11
2.2	Working of CoAP	14
2.3	Working of CoCoA	18
3.1	Contribution (highlighting (GST-CoAP))	29
3.2	Working of <i>Fullbackoff1</i> variant of CoAP	30
3.3	Working of <i>Fullbackoff2</i> variant of CoAP	31
3.4	Working of GST for RTO estimation in CoAP	34
3.5	Grid Topology - GST-CoAP	35
3.6	Comparison of FCT in Cooja and FIT/IoT-LAB	37
3.7	Total number of retransmissions: Cooja and FIT/IoT-LAB	38
3.8	Throughput of CoAP and Variants in FIT/IoT-LAB	39
4.1	Contribution (highlighting (CoAP-Eifel))	41
4.2	Components of TCP-Eifel	42
4.3	Half Dumbbell Topology	47
4.4	CoAP-Eifel: RTT vs RTO	48
4.5	RTO: CoAP vs CoAP-Eifel	49
4.6	Throughput: CoAP vs CoAP-Eifel	50
4.7	RTT: CoAP vs CoAP-Eifel	51
5.1	Contribution (highlighting (GSRTC))	53
5.2	Working of <i>Fullbackoff1</i> variant of CoCoA	55
5.3	Working of <i>Fullbackoff2</i> variant of CoCoA	56
5.4	Working of GSRTC	57

5.5	Comparison of W_{strong} with different values of r	59
5.6	Grid Topology - GSRTC	60
5.7	Deployment of nodes in Saclay site	62
5.8	Comparison of FCT in Cooja and FIT/IoT-LAB	63
5.9	Total number of retransmissions: Cooja and FIT/IoT-LAB	64
5.10	Throughput of CoCoA and Variants in Cooja	65
5.11	Throughput of CoCoA and Variants in FIT/IoT-LAB	67
6.1	Contribution (highlighting (CoCoA++))	69
6.2	Working of CoCoA++ Algorithm	73
6.3	Grid Topology	75
6.4	Flower Topology	76
6.5	Dumbbell Topology	77
6.6	Chain Topology	77
6.7	Comparison of CoCoA and CoCoA++ in Static Topologies	82
6.8	Comparison of CoCoA and CoCoA++ in Mobile Topologies	83
6.9	Comparison of CoCoA and CoCoA++ in terms of CDF	84
6.10	Grid Topology - CoCoA vs CoCoA++	85
6.11	Flower Topology - CoCoA vs CoCoA++	85
6.12	Dumbbell Topology - CoCoA vs CoCoA++	86
6.13	Chain Topology - CoCoA vs CoCoA++	86
6.14	Random Way Point Mobility Model - CoCoA vs CoCoA++	87
6.15	Gauss-Markov Mobility Model - CoCoA vs CoCoA++	87
6.16	ManhattanGrid Mobility Model - CoCoA vs CoCoA++	87
6.17	Pursue Mobility Model - CoCoA vs CoCoA++	88
6.18	Deployment of nodes in Lille, Grenoble and Paris sites at FIT/IoT-LAB	88
6.19	Topology Selection from Lille, Grenoble and Paris sites of FIT/IoT-LAB	89
6.20	Comparison of Performance Metric in Grenoble, Paris and Lille sites at FIT/IoT-LAB	90

List of Tables

2.1	Classification of Congestion Control Mechanisms for CoAP	20
2.1	Classification of Congestion Control Mechanisms for CoAP - Contd... . . .	21
2.2	Summary of existing CoAP-based congestion control mechanisms	27
4.1	Throughput (in bytes/second)	50
4.2	Packet Transmission Time (in seconds)	51
4.3	Packet Delivery Ratio (PDR)	52
5.1	Experimental Parameters and Configuration	61
5.2	Average RTO	66
6.1	Sensor mote configuration	74
6.2	Simulation Parameters	79
6.3	Results with Static Topologies	80
6.4	Results with Mobile Topologies	81

Abbreviations and Nomenclature

Abbreviations

6LoWPAN	IPv6 over Low-power Wireless Personal Area Networks
ACK	Acknowledgement
AMQP	Advanced Message Queue Telemetry Transport
BEB	Binary Exponential Backoff
CDG	CAIA Delay-Gradient
CON	Confirmable
CoAP	Constrained Application Protocol
CoCoA	CoAP simple Congestion control/Advanced
CoRE	Constrained Restful Environments
DDS	Data Distribution Services
FCT	Flow Completion Time
GST	Geometric Sequence Technique
HTML	Hyper Text Transfer Language
IETF	Internet Engineering Task Force
IoT	Internet of Things
IoT-A	Internet of Things Architecture

LLN	Low-power and Lossy Networks
MQTT	Message Queue Telemetry Transport
MQTT-SN	Message Queue Telemetry Transport - Sensor Network
NON	Non-confirmable
PBF	Probabilistic Backoff Factor
RFC	Request For Comment
RPL	Routing Protocol for Low power and lossy network
RST	Reset
RTO	Retransmission TimeOut
RTT	Round Trip Time
RTTVAR	Round Trip Time VARiation
RoLL	Routing over Low-power and Lossy Networks
SRTT	Smoothed Round Trip Time
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VBF	Variable Backoff Factor
WSN	Wireless Sensor Network

Nomenclature

α	weight constant used in RFC 6298
β	weight constant used in RFC 6298
δ	absolute value
RTO_{init}	initial timeout value
RTO_{new}	new RTO value
RTO_{prev}	previous timeout value
RTT_{max}	maximum RTT
RTT_{min}	minimum RTT
a	sampling period
$cwnd$	congestion window
g_{max}	maximum gradient
g_{min}	minimum gradient
$ssthresh$	slow start threshold
w_n	congestion window for n_{th} iteration
$weight_{next}$	new weight value
$weight_{prev}$	previous weight value

Chapter 1

Introduction

Internet of Things (IoT) is a network where physical objects with Internet connectivity can interact and exchange information with other connected objects. IoT makes these physical objects smart by leveraging their underlying communication and computing technologies, Internet protocols and various applications (Al-Fuqaha et al., 2015; Atzori et al., 2010). According to Juniper and Martech Advisor, there will be 46 billion Internet-connected devices by the end of 2021, increasing to 125 billion by 2030 (Nick, 2021). When a huge number of devices attempt to interact with one another, large amounts of data are generated (i.e., International Data Corporation (IDC) forecasts 79.4 zettabytes (ZB) data will be generated by IoT devices in 2025 (Helpnet Security, 2019)). IoT devices have limited resources, such as memory, processing and power. A large number of IoT devices have been deployed in a constrained environment, leading to a number of challenges faced while developing network protocols for IoT. Due to several challenges posed by the constrained nature of IoT devices, conventional protocols are not suitable for IoT networks; so new network protocols are being developed to meet the requirements.

Congestion control mechanisms are vital to ensure efficient and fair use of network resources. Typically, TCP is responsible for end-to-end congestion control and reliability. However, in a constrained network, this can cause performance problems because most of the communication happens over wireless links that are known to be challenging for TCP (Patel et al., 2001; Fahmy et al., 2003; Tian et al., 2005; Gomez et al., 2018). Consequently, the responsibility for controlling congestion is entrusted to the application layer protocols. Message Queue Telemetry Transport (MQTT) (Banks and Gupta, 2014), Extensible Messaging and Presence Protocol (XMPP) (Saint-Andre et al., 2004), Data Distribution Services (DDS) (Pardo-Castellote, 2003), Advanced Message Queuing Protocol (AMQP) (OASIS Standard, 2012) and Constrained Application Protocol (CoAP)

(Shelby et al., 2014) are popular application layer protocols in IoT.

CoAP (RFC 7252) (Shelby et al., 2014) is a lightweight application layer protocol that is standardized and developed by the Constrained Restful Environments (CoRE) Working Group of the Internet Engineering Task Force (IETF). It is used as the default application protocol for the messaging model in IoTivity (Open Connectivity Foundation (OCF), 2015), an open-source IoT platform¹ that enables seamless machine-to-machine connectivity. CoAP is a simple low overhead protocol that supports end-to-end communication and has been specially designed for constrained devices and constrained networks. CoAP has similar features to HTTP, but cannot be viewed as a compressed HTTP protocol. Unlike HTTP which operates on top of TCP, CoAP operates on top of the UDP² and provides support for reliability with *minimum* congestion control. It adopts a simple stop-and-wait approach for data transmission (Shelby et al., 2014). CoAP provides basic congestion control technique based on Retransmission Time Out (RTO) and Binary Exponential Backoff (BEB). CoAP initializes the RTO to a fixed value, which is randomly selected between [2s, 3s] *for every new transmission* and it doubles the value of RTO when the packets get retransmitted. The default CoAP congestion control mechanism needs improvements because it does not estimate the RTO based on RTT, resulting in prolonged network delays and spurious retransmissions.

Several new solutions (Järvinen et al., 2018; Ancillotti and Bruno, 2017) have been proposed to address the concerns with the default CoAP congestion control mechanism, but CoAP Simple Congestion Control/Advanced (CoCoA) (Bormann et al., 2020) is the one that is actively discussed at IETF and is used by several researchers to build new congestion control mechanisms for CoAP (Jarvinen et al., 2018; Bolettieri et al., 2018). CoCoA estimates the RTO based on the RTT measurements by running two estimators in parallel, named Strong and Weak. CoCoA uses Variable Backoff Factor (VBF) instead of a fixed backoff factor (BEB) when the packets get retransmitted. In addition, CoCoA includes an aging mechanism which is used to estimate RTO when RTT values are not updated/measured frequently, thus ensuring that the value of RTO is not outdated. Nevertheless, recent studies have shown that both CoAP and CoCoA require further optimizations (Järvinen et al., 2018; Ancillotti and Bruno, 2017; Demir and Abut, 2018; Bolettieri et al., 2017).

¹IoTivity is hosted by the Linux Foundation and funded by Open Connectivity Foundation (OCF).

²Though guidelines have been provided to use CoAP on top of TCP, the original CoAP RFC 7252 recommends using CoAP on top of UDP.

1.1 The Problem

The advent of IoT and its inclusion in everyday scenarios brings with it a fair share of problems. A large number of IoT devices are characterized by small memory and low processing speeds that lead to congestion in the network when many such devices try to communicate with each other. The data packets being shared by the constituent devices across the network in IoT systems have small payloads and hence, packet loss due to congestion results in expensive retransmissions that lead to additional delays and large overheads. IoT networks have a mobility factor associated with them as they do not always deal with static nodes. The small packet size, costly re-transmissions and mobility together make the nature of these constrained networks very different from the conventional networks currently in place (Al-Fuqaha et al., 2015). Several mechanisms have been designed that use different aspects of the network to control congestion, and either actively or passively deal with it (Ghaffari, 2015). This section discusses the two most important aspects that can be used to detect congestion in an IoT network.

1.1.1 Packet loss

Packet loss is one of the most commonly used signs of congestion. Once a source sends a packet, it waits for a fixed amount of time for an acknowledgement from the destination by maintaining a timer at its end. If no acknowledgement is received within this duration, the packet is considered lost due to network congestion. Once a packet loss is detected, the congestion control algorithm in place takes charge to control the network congestion. Mechanisms that depend on packet loss to detect network congestion prove to be of no use in IoT networks because: (i) these mechanisms assume that a packet loss happens only due to congestion, which is not true for low power and lossy networks e.g., packet loss can happen due to link errors or poor signal strength (Gomez et al., 2018), and (ii) delaying congestion response until a packet is lost deteriorates the performance since IoT packets usually carry time sensitive payloads and their retransmissions is costly, causing additional delays (Gomez et al., 2018). Typically, the congestion control mechanisms that rely on packet loss are tightly coupled with TCP. The major concern is that TCP is not a de-facto transport protocol for IoT applications (Atzori et al., 2010), and hence, designing congestion control mechanisms for IoT applications is a non-trivial task.

1.1.2 Round Trip Time (RTT)

Besides using packet loss, some mechanisms use one-way delay or Round Trip Time (RTT) as an additional measure of network congestion (Brakmo and Peterson, 1995; Betzler et al., 2013; Mishra et al., 2018). An increase in the RTT values is considered as an indication of congestion building up in the network. These algorithms compare RTT with a threshold value and trigger the congestion control mechanism if the measured RTT exceeds the threshold. However, RTT is not a reliable metric to predict network congestion because multiple factors, such as the processing time at the endpoints or mobility of nodes, could lead to an increase/decrease in RTT. Finally, fairness issues arise when RTT based congestion control algorithms share bandwidth with loss based algorithms. In such shared systems, a backoff by RTT based algorithms may cause the loss based algorithms to take up the freed bandwidth and the RTT based algorithms may never get a chance to transmit messages (Mo et al., 1999; Ahn et al., 1995; Kurata et al., 2000). It is, therefore, necessary to make sure that the congestion control algorithm in place is able to detect the presence of such loss based algorithms sharing the network bandwidth.

CoAP uses packet loss as an indication of network congestion and runs on top of UDP. It tackles network congestion by using a simple congestion control mechanism based on RTO with BEB. RTO is used to identify packet losses, and BEB ensures that retransmitted packets do not worsen the state of network congestion (e.g., premature retransmissions can lead to duplicate packets and increase congestion). However, it does not follow TCP-like RTT measurements to update its RTO. CoCoA is an enhanced congestion control mechanism for CoAP that follows TCP-like RTT measurements to update the RTO. CoCoA runs two RTO estimators in parallel, named *Strong* and *Weak*. The Strong RTO estimator is maintained for all the transmissions that get cleared without retransmissions and the Weak RTO estimator is maintained for the transmissions that are cleared after retransmissions. The Weak estimator is used to address the RTT ambiguity problem which is described next.

RTT ambiguity

RTT is normally estimated as the difference between the time the packet was sent to the time of receipt of the ACK. But this causes ambiguity when packets are retransmitted, which is referred to as *retransmission ambiguity*, as shown in Figure 1.1. It can be seen in the figure that it is difficult to state that ACK received by the client belongs to the

original Confirmable (CON) message (Karn and Partridge, 1987) or a retransmitted CON message. If we assume that the ACK belongs to the retransmitted CON message, then the measured RTT will be small. If we assume that the ACK belongs to the original CON message, then the measured RTT will be large. A small value of RTT might reduce the overall RTO and lead to spurious retransmissions, and on the other hand, a large value of RTT might increase the overall RTO, thus causing long idle periods between transmissions.

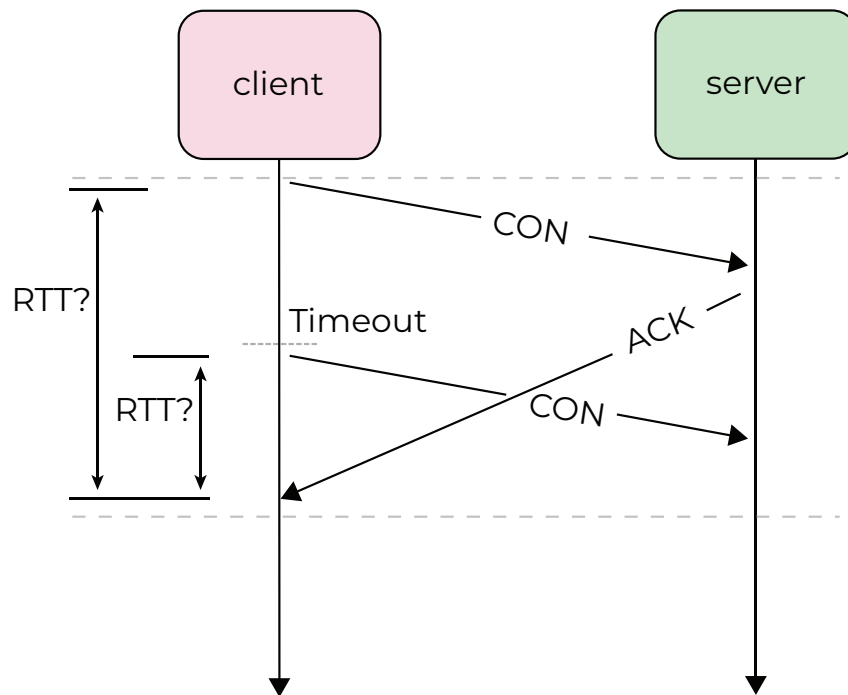


Figure 1.1: Retransmission ambiguity

This *retransmission ambiguity* was noticed by Karn in the context of calculating RTO for TCP (Karn and Partridge, 1987). He suggested the following rules to resolve the ambiguity:

1. do not consider the RTT of the retransmitted packets for calculating RTO.
2. double the RTO for the retransmitted packets (Binary Exponential Backoff).

If the second rule is not applied, the client will resort to using the previous RTO which was updated based on the RTT samples obtained from successful transmissions. This has a risk of RTO value ending up being too low (because RTT measurements of retransmitted packets are not accounted for), thus leading to spurious retransmissions.

CoCoA follows the second rule suggested by Karn, but not the first one because it does not ignore the RTT measurements of retransmitted packets while calculating RTO.

CoCoA calculates the RTT of the retransmitted packet by considering the first packet transmission time to the received ACK time (as shown in *Retransmission phase* of Figure 2.3). Such a RTT measurement is termed as Weak RTT measurement and the resulting RTO is called Weak RTO. In order to limit the effect of the Weak RTT measurements obtained from retransmitted packets, CoCoA uses a lower weight of 0.25 to estimate the Weak RTO in Eq. (2.5). The weight used in Strong RTO is 0.5.

1.2 Contributions of this thesis

The main goal of this work is to develop new CoAP-based congestion control mechanisms for IoT. We decided to base our work on CoAP because: (i) It has been standardized by the IETF (RFC 7252). (ii) It has a built-in congestion control, and (iii) CoAP is the default application protocol for communication in an open-source IoT platform called IoTivity, that enables seamless machine-to-machine connectivity. Figure 1.2 highlights the contributions made in this work (i.e., highlighted with green color in Figure 1.2).

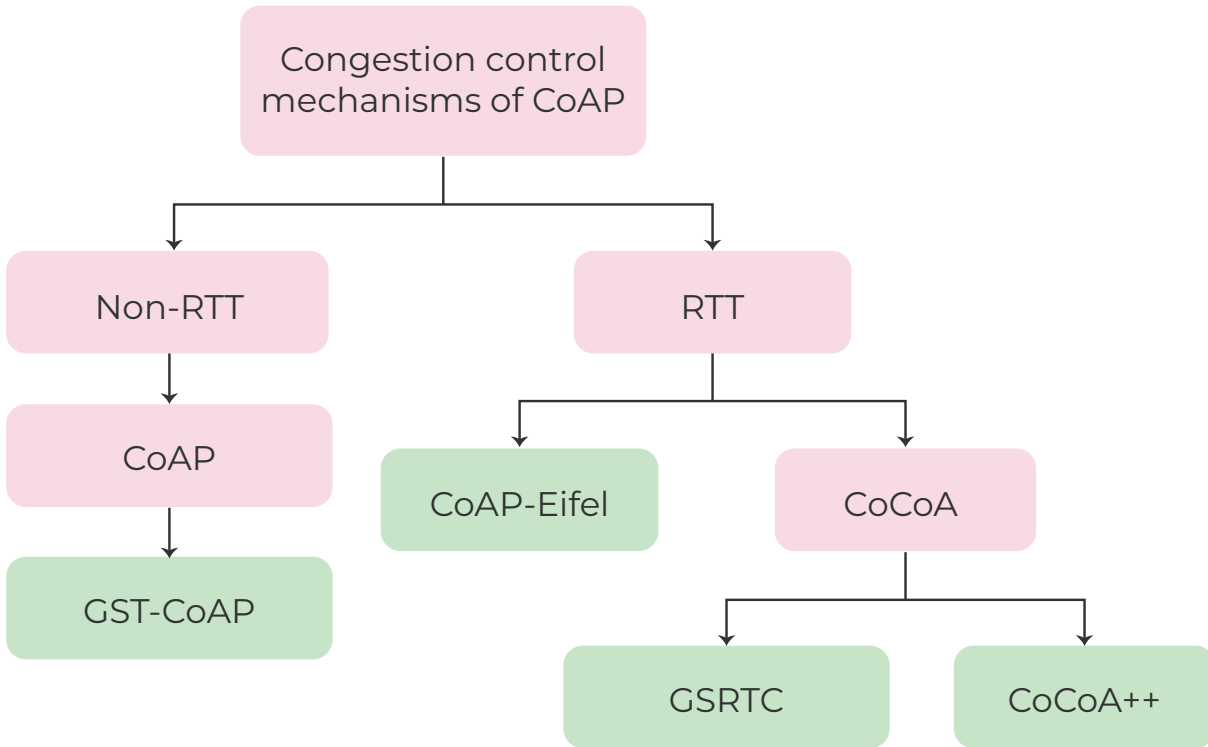


Figure 1.2: Our Contributions

1.2.1 GST-CoAP

CoAP sets the initial RTO to a value selected randomly from the *fixed interval* $[2s, 3s]$ for every new transmission. If the transmission fails, BEB doubles the RTO value in order to

prevent congestion. When the network is *not congested*, CoAP does not adapt the RTO and instead resets it to a low RTO value which could lead to spurious retransmissions when RTT is high.

This work proposes a Geometric Sequence Technique for effective RTO estimation in CoAP (GST-CoAP). Unlike CoAP, which resets the RTO to its default value after receiving an ACK for the retransmitted packet, GST-CoAP gradually reduces the RTO to its default value based on the number of *consecutive* successful transmissions. The main idea is to minimize the Flow Completion Time (FCT), number of retransmissions and enhance per-flow throughput.

1.2.2 CoAP-Eifel

The default CoAP congestion control mechanism is not efficient because it does not estimate the RTO based on RTT. This makes CoAP insensitive to the network conditions. If the selected RTO value is less than the RTT, it will result in spurious retransmissions. If the RTO value is substantially higher than the RTT, the network will experience long idle delays and the resources will be wasted. Thus, to improve network performance, an adaptive mechanism that estimates RTO based on RTT measurements is required. Using RTT measurements to estimate RTO has benefits when compared to loss based mechanisms because it helps the end systems to become aware of network conditions and react accordingly.

This work proposes a new congestion control mechanism by integrating the Eifel Retransmission Timer (Ludwig and Sklower, 2000) with CoAP (CoAP-Eifel) in order to obtain better RTO estimates and control congestion. Eifel Retransmission Timer is a RTO estimation technique that was initially designed for BSD-Lite's (Berkeley Software Distribution) distribution of TCP (TCP-Lite). Integrating it requires modifications in CoAP because the design of this timer is tailored to work with TCP. The main idea is to use Eifel Retransmission Timer to obtain an accurate prediction of the upper bound of RTT and react faster to packet losses.

1.2.3 GSRTC

CoCoA uses an Exponential Weighted Moving Average (EWMA) to measure the RTO for the next packet transmission. The weights used in the EWMA equation are *fixed* (0.5 for Strong and 0.25 for Weak) and the weights are selected on the basis of the recent

transmission made by a strong or weak estimator. If the network conditions are *lossless* and *consecutive* packets are getting successfully cleared without retransmissions, these *fixed* weights affect the RTO adaptation process.

Hence, in order to improve the RTO estimation, this work proposes a Geometric Series based effective RTO estimation Technique for CoCoA (GSRTC). GSRTC improves the network performance by rapidly adapting the weight of Strong RTO when the network is *not congested* and packet transmissions are getting cleared *consecutively without retransmissions*. This helps CoCoA to adapt quickly and perform efficiently when the network conditions are lossless.

1.2.4 CoCoA++

CoCoA uses per packet RTT measurements to predict network congestion. Per packet RTT measurements are typically noisy and cannot be relied upon for predicting network congestion. Hence, this work proposes an efficient congestion control mechanism called CoCoA++ which uses CAIA Delay-Gradient (CDG) (Hayes and Armitage, 2011) to measure network congestion and the Probabilistic Backoff Factor (PBF) to control congestion.

CDG was designed to work with TCP (Jonassen, 2015), but is not bound to it; it can be used with any other end-to-end protocol, such as CoCoA. CDG provides a better estimate of network congestion as it mainly attempts to track the changes in the queuing delay component in the RTT, and ignores other delay components such as processing, propagation and transmission delays. Instead of considering per packet RTT samples, CDG obtains a gradient of RTT over time and PBF helps to adjust the RTO on the basis of the inferred congestion signal.

A key aspect of all contributions is that they are easy to deploy and do not require setting additional parameters to obtain the performance gain. The proposed mechanisms have been extensively evaluated using the Cooja network simulator (Osterlind et al., 2006) and a real testbed at FIT/IoT-LAB (Adjih et al., 2015). We have considered static topologies such as grid, flower, dumbbell and chain, and mobility models such as random waypoint, Manhattan grid, pursue, and Gauss-Markov for evaluation. Dumbbell is the most popular topology used to evaluate the performance of congestion control algorithms. Grid topologies are popular in applications requiring long range and broad area coverage such as smart grid, industrial automation and building automation (Betzler et al., 2013, 2015b; Ancillotti and Bruno, 2017). Flower topologies are used in applications such as

smart greenhouse management systems, cellular network, satellite network and wide area network applications (Chan, 2008). The mobility models used for evaluation relate to applications such as smart traffic management, self-driving cars, home automation systems with handheld devices and communication for security drones (Aschenbruck et al., 2010). The primary metrics used to validate the effectiveness of all contributions are FCT, number of retransmissions, throughput, RTO, delay and packet sending rate. The proposed mechanisms outperform the existing ones in terms of estimating network congestion and improving the network performance. Since the proposed mechanisms are targeted towards different set of use-cases, we do not evaluate their performance against each other.

1.3 Outline of the thesis

Chapter 2 presents the existing congestion control mechanisms for IoT. Furthermore, it provides an overview of different low-power operating systems for constrained devices, as well as simulators and open source testbeds used for evaluating the congestion control mechanisms in IoT. Lastly, the literature related to IoT congestion control mechanisms is presented, with a particular emphasis on the application and routing layers.

Chapter 3 presents the motivation to optimize the RTO estimation technique in CoAP. Subsequently, it provides a brief overview of the *Fullbackoff* variants (the existing work in the literature which is directly related to this work) and their issues. It presents the GST-CoAP technique and how it seamlessly aligns with the design of CoAP. Finally, the effectiveness of GST-CoAP is studied by comparing its performance with the default CoAP and *Fullbackoff* variants using a simulator and real testbed.

Chapter 4 presents the design and implementation of the Eifel Retransmission Timer in CoAP which calculates the RTO based on RTT measurements. Subsequently, it discusses the difficulty of incorporating the Eifel Retransmission Timer into CoAP. Lastly, it presents the results and analysis based on a comparative study of CoAP-Eifel and CoAP.

Chapter 5 highlights the impact of using *fixed* weight values for RTO estimation in CoCoA. Subsequently, the existing *Fullbackoff* mechanisms for CoCoA are discussed. Later, it presents how to integrate geometric series into CoCoA, resulting in GSRTC, to use an adaptive weight for Strong RTO estimation instead of using a fixed one (0.5). Lastly, the results of GSRTC are compared with CoCoA and the *Fullbackoff* mechanisms.

Chapter 6 describes the CDG technique which was originally designed for TCP, but is used in conjunction with CoCoA in this work. Subsequently, it presents the design and

development of CoCoA++. Lastly, the effectiveness of CoCoA++ is studied by comparing its performance with CoCoA in both static and mobile scenarios.

Chapter 7 summarizes the work done in this thesis and discusses the future work in this area.

Chapter 2

Background and Literature Review

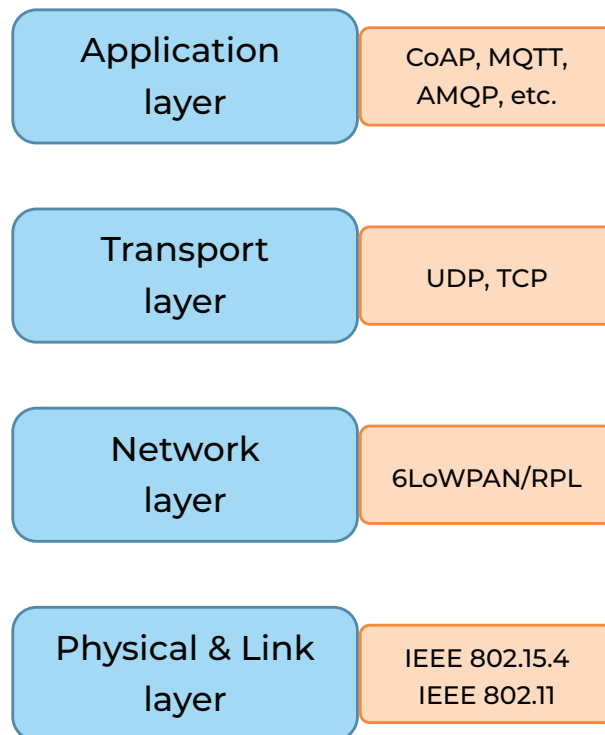


Figure 2.1: IoT Protocol Stack

One of the main components required for communication is a protocol stack. A desirable property of an IoT stack is that it should consume less processing power, should be lightweight, flexible and configurable. Figure 2.1 depicts the IoT stack proposed by IETF. The physical and link layer of IoT stack consists of IEEE 802.15.4 (Low-rate Wireless Personal Area Network) and IEEE 802.11 (Kovatsch et al., 2011). Routing Protocol for Low-power and lossy networks (RPL) has been proposed as a solution for routing in Low-power and Lossy Networks (LLNs) like IoT, and cater to unique routing challenges (Parasuram et al., 2016). RPL is designed to be highly adaptive to network conditions and provides alternate routes whenever default routes are inaccessible. As far as transport

layer is concerned, UDP is the popular choice for IoT owing to its simplicity of deployment and lightweight functionality rather than TCP. The application layer of the IoT stack consists of various lightweight application protocols such as CoAP, MQTT, AMQP and others.

2.1 Background

2.1.1 TCP RTO calculation

The basic algorithm for calculating the value of the retransmissions timer in TCP is described in RFC 6298 (Paxson et al., 2000). There are two state variables: smoothed RTT (SRTT) and RTT variation (RTTVAR). SRTT keeps running average of the current RTT sample and the previous RTT samples, and RTTVAR keeps the mean deviation of the RTT samples. Until the first RTT measurement is received, the RTO value is set to 1 second, and then the RTO is calculated using Eq. (2.1).

$$\begin{aligned}
 SRTT &= R \\
 RTTVAR &= \frac{R}{2} \\
 RTO &= SRTT + \max(G, K \times RTTVAR)
 \end{aligned} \tag{2.1}$$

where, R represents the current RTT measurement. The RTO value for the subsequent transmissions is calculated as shown in Eq. (2.2).

$$\begin{aligned}
 SRTT &= (1 - \alpha) \times SRTT + \alpha \times R' \\
 RTTVAR &= (1 - \beta) \times RTTVAR + \beta \times |SRTT - R'| \\
 RTO &= SRTT_x + \max(G, K \times RTTVAR)
 \end{aligned} \tag{2.2}$$

where, R' represents the new RTT measurements and the values of the constants α , β and K are $(\frac{1}{8})$, $(\frac{1}{4})$ and 4, respectively. The values of α and β control how quickly the SRTT and RTTVAR adapt to changes.

This basic mechanism is leveraging the Karn/Partridge algorithm; thus, the RTT of the retransmitted packet is ignored due to ambiguity. When the packet is retransmitted, this mechanism uses exponential backoff and doubles the RTO value instead of basing it on the previous RTT. The aim of using this backoff logic is to ensure that when network

congestion occurs, the sender does not react too quickly on every timeout, causing the sender to be more cautious. This affirms that the timeout mechanism has an influence on congestion. If the timeout occurred too soon, there may be unnecessary retransmissions, which will cause network overheads.

2.1.2 Congestion control in CoAP

Constrained Application Protocol (CoAP) is a lightweight alternative to HTTP that is specially designed for constrained devices to cope up with constrained network characteristics. The main aim of designing CoAP is not to blindly compress HTTP functionalities, but to follow the REpresentational State Transfer (REST) architecture by modifying or optimizing some of the functionalities. CoAP has two logical sub-layers: the message layer and the request/response layer. The message layer deals with UDP and addresses the asynchronous message interaction. The request/response layer manages exchanges between the client and the server using different request/response methods and codes. Like HTTP, CoAP uses the methods such as GET, POST, PUT and DELETE to access or manipulate resources on the server. CoAP has four types of messages: Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK) and Reset (RST).

$$RTO_{new} = 2 \times RTO_{prev} \quad (2.3)$$

CoAP is primarily designed to work on top of UDP, making it suitable for IoT applications. Additionally, it provides end-to-end reliability through CON messages that require an ACK from the destination endpoint. CoAP supports the built-in congestion control mechanism by using RTO and BEB techniques. Moreover, CoAP limits the number of outstanding CON messages to 1 to prevent congestion. CoAP sets the initial RTO to a *fixed* value, which is randomly selected from [2s, 3s] for each new CON message transmission. The CoAP client waits for a response after a CON message has been sent. If the reply is not received until the timeout, the message is retransmitted and RTO is doubled, as shown in Eq. (2.3).

The working of the default CoAP congestion control mechanism is illustrated in Figure 2.2. There are two main phases in Figure 2.2: *Normal* phase and *Congestion* phase. The *Normal* phase depicts CoAP's standard behaviour, initializing every new transmission with the default timeout value (i.e., randomly sets the RTO from the *fixed* interval [2s, 3s]). The *Congestion* phase depicts the CoAP's behaviour when packets get dropped due

to congestion. If a client does not receive an ACK for a transmitted packet, the packet is considered lost. When the timer expires (RTO), CoAP retransmits the same packet by increasing the RTO to twice of its previous value (See '(BEB)' label in Figure 2.2). When the algorithm enters *Normal* phase again, it resets the RTO to a randomly selected value between $[2s, 3s]$.

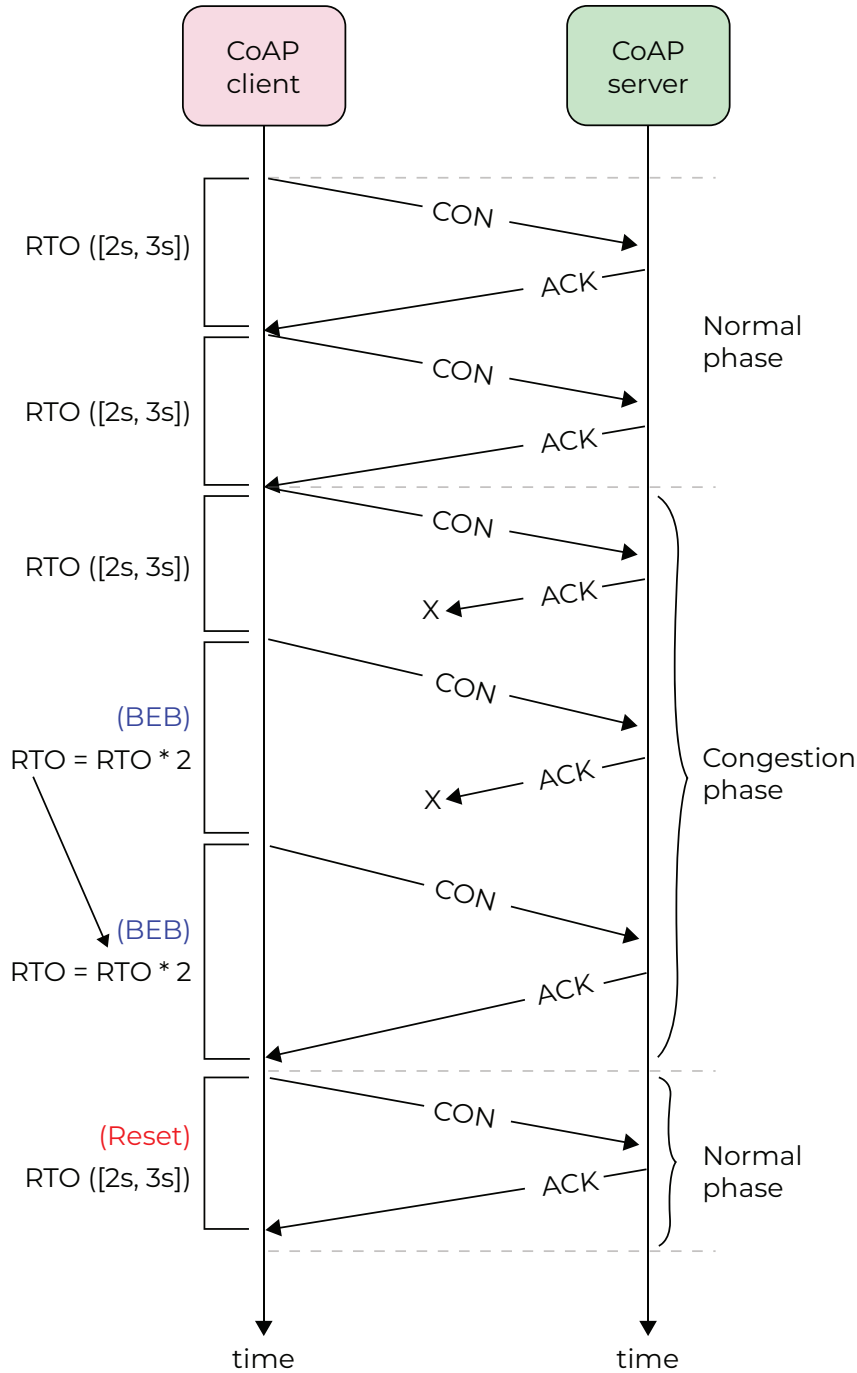


Figure 2.2: Working of CoAP

CoAP supports a non-adaptive congestion control mechanism because it selects RTO from a *fixed* interval without estimating it from the RTT. If the estimated RTO is less

than the RTT, spurious retransmissions can occur, and if it is greater, longer wait times lead to connections becoming idle, thus under utilizing the available network resources. Therefore, an adaptive RTO mechanism, which estimates RTO based on RTT, is needed to address the above mentioned issues and to improve the network performance.

2.1.3 CoAP Simple Congestion Control/Advanced (CoCoA)

CoAP Simple Congestion Control/Advanced (CoCoA) (Betzler et al., 2014) is an enhanced congestion control mechanism for CoAP. CoCoA estimates RTO based on RTT measurements where RTT samples are obtained according to the Karn/Partridge algorithm (Karn and Partridge, 1987). However, one potential problem with updating RTO by using RTT measurements is that it is difficult to calculate the RTT of retransmitted packets. To overcome this problem, CoCoA maintains two RTO estimators namely, *Strong* and *Weak*. *Strong* estimator considers RTT samples of transactions that are transmitted successfully in a single transmission, while *weak* estimator considers RTT samples for the transactions that are retransmitted (only the first two retransmissions are considered).

CoCoA leverages TCP's standard algorithm (RFC 6298) (Paxson et al., 2000) to estimate the RTT and RTO. CoCoA maintains two copies of Eq. (2.4) for *Strong* and *Weak* estimator, respectively.

$$\begin{aligned}
 RTT_x &= (1 - \alpha) \times RTT_x + \alpha \times RTT_{x_new} \\
 RTTVAR_x &= (1 - \beta) \times RTTVAR_x + \beta \times (RTT_x - RTT_{x_new}) \\
 RTO_x &= RTT_x + K_x \times RTTVAR_x
 \end{aligned} \tag{2.4}$$

where, x represents either strong or weak, the values of the constants α and β are $(\frac{1}{8})$ and $(\frac{1}{4})$ respectively, and the RTO is computed using the constant K , where $K = 4$ for both strong and weak estimator.

The overall RTO for CoCoA is then calculated as a weighted average of the strong or weak RTO estimate and the previous overall RTO as shown in Eq. (2.5). CoCoA uses an equal weight (0.5) for the current RTO estimate (strong or weak) and the previous RTO. CoCoA uses the BEB approach to estimate the RTO for a retransmitted packet.

$$RTO_{new} = W_x \times RTO_x + (1 - W_x) \times RTO_{prev} \tag{2.5}$$

CoCoA+ is an extension of CoCoA with three major enhancements (Betzler et al.,

2015b): (i) it modifies the Eq. (2.5) in case of RTO_{weak} by assigning less weight to RTO_{weak} and more weight to previous RTO. This is to ensure that inaccurate RTT measurements done in RTO_{weak} do not largely affect the overall RTO. No changes are made to Eq. (2.5) in case of RTO_{strong} . (ii) it adds a RTO aging mechanism to CoCoA. (iii) it replaces the Binary Exponential Backoff (BEB) by a Variable Backoff Factor (VBF)¹.

After estimating RTO_x based on Eq. (2.4), a cumulative RTO_{new} is determined for CoCoA as a weighted moving average as shown in Eq. (2.5) where the values of weight for strong and weak estimators are $W_{strong} = 0.5$ and $W_{weak} = 0.25$, respectively. The RTO_{new} estimated using Eq. (2.5) is used as the initial RTO for the next transmission to the same destination. Furthermore, in Eq. (2.4), the value of K is reduced from 4 to 1 when estimating the RTO_x for weak estimator.

To estimate the RTO for the retransmitted packet, CoCoA uses VBF instead of BEB as shown in Eq. (2.6). The RTO for retransmitted packets is estimated by applying the multiplicative factor. The RTO value is multiplied by 3 for each retransmission, as long as the RTO value is less than 1s, to ensure that the maximum retransmission limit is not exhausted in a short time span. Similarly, the RTO is multiplied by 1.5 for each retransmission to ensure that transactions with large RTOs wait for some time to receive an acknowledgement from the client before giving up. In other circumstances, RTO, like BEB, is multiplied by 2.

$$VBF = \begin{cases} 3, & RTO < 1s \\ 2, & 1s \leq RTO \leq 3s \\ 1.5, & RTO > 3s \end{cases} \quad (2.6)$$

Finally, the RTO for the subsequent packet is estimated by using the Eq. (2.7).

$$RTO_{new} = RTO_{previous} \times VBF \quad (2.7)$$

The existing RTO values may become outdated if there are no RTT measurements over a long period of time. Hence, CoCoA applies an RTO aging mechanism when the RTO has not been updated for a long time. In IoT, network behaviour changes rapidly,

¹Although the CoCoA paper (Betzler et al., 2014) contains discussions on using VBF instead of BEB, the latest paper on CoCoA+ from same authors states VBF as one of the features added in CoCoA+ (Betzler et al., 2015b)

thus affecting RTT values. As a consequence, in order to prevent having incorrect RTO values in such a network, aging mechanisms is applied to small and large RTO estimations. As per Internet draft of the CoCoA (Bormann et al., 2020), if the RTO value is small ($RTO < 1s$) and no new RTT measurements are made for upto 16 times to its current value, then the RTO value is doubled. However, if the RTO value is large ($RTO > 3s$) and has not been updated for 4 times to its current value, the RTO should be estimated using Eq. (2.8).

$$RTO = 1s + (0.5 \times RTO) \quad (2.8)$$

Note: Since CoCoA+ is a minor extension of CoCoA, the congestion control mechanism in Contiki OS (Dunkels et al., 2004) is referred to as CoCoA although it actually implements CoCoA+. Hence, for the sake of simplicity, we have also used the same terminology throughout this thesis i.e., we have based our work on CoCoA+, but we use the term CoCoA.

Figure 2.3 depicts the working of CoCoA, which is divided into two main phases: *Normal* phase and *Retransmission* phase. The *Normal* phase illustrates the working of the strong estimator that measures RTT_{strong} and updates the RTO_{strong} only if a client receives an ACK for the transmitted packet without any retransmissions. The *Retransmission* phase exhibits the working of a weak estimator that calculates RTT_{weak} and updates the RTO_{weak} for packets that encounter retransmissions. A packet is considered lost if a client does not receive an ACK for it. When the timer expires (RTO), CoCoA retransmits the same packet and estimates the RTO for the retransmitted packet using the VBF. When the algorithm returns to the *Normal* phase, it estimates the RTT and updates RTO according to the strong estimator.

2.2 Related Work

The literature on congestion control in IoT spans across different layers of the network stack. In this section, we focus on the recent developments directly related to our work i.e., application layer solutions to control network congestion in IoT. We also briefly discuss a few routing layer solutions for network congestion control. Table 2.1 summarizes the mechanisms proposed to improve the performance of CoAP and CoCoA.

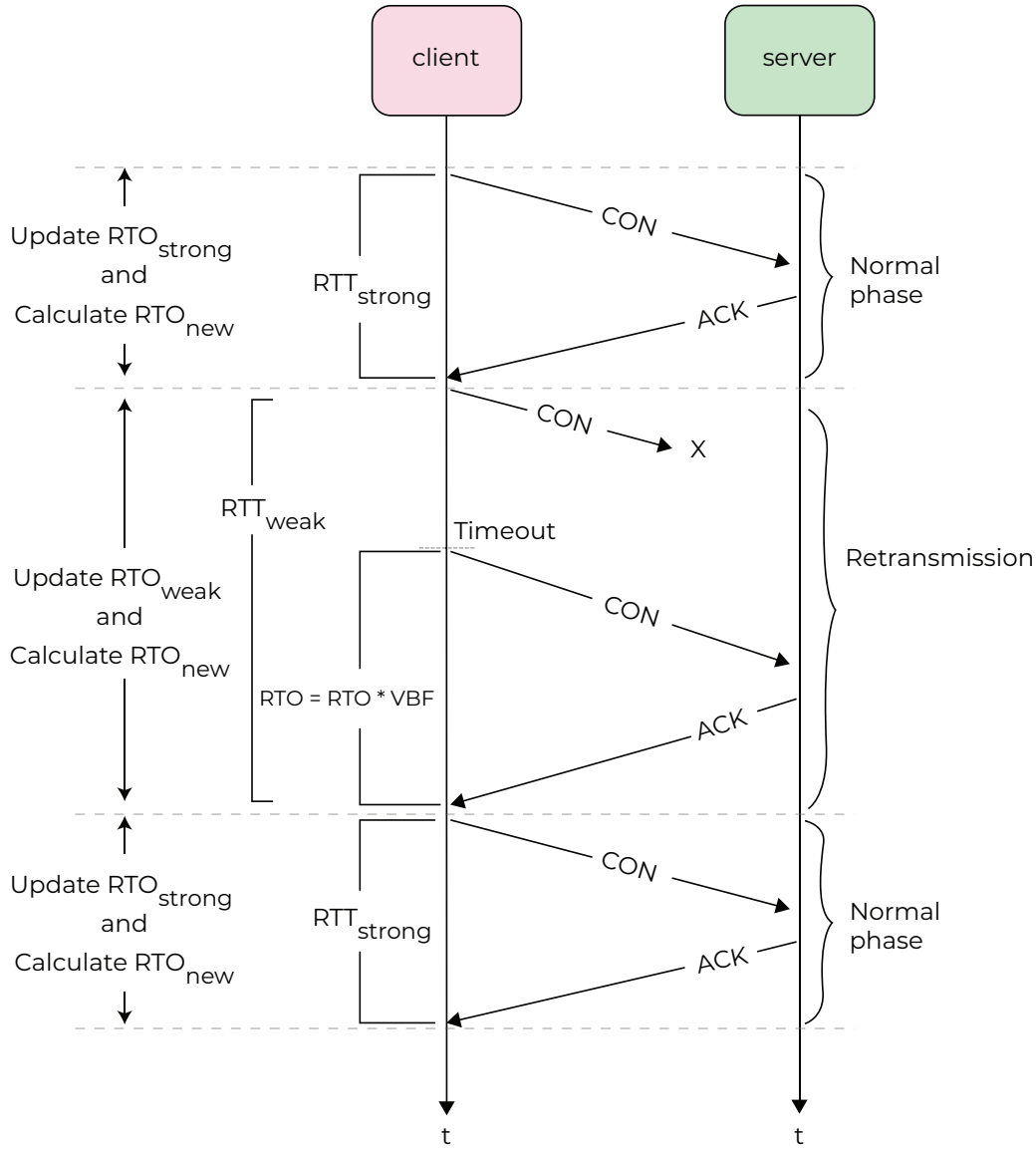


Figure 2.3: Working of CoCoA

2.2.1 Application layer solutions

There has been a lot of interest in designing congestion control algorithms for CoAP. A comprehensive survey of the congestion control mechanisms designed for CoAP is presented in (Pramanik et al., 2017; Tariq et al., 2020). Majority of the solutions include modifying the default RTO estimation technique used by CoAP. These solutions improve the performance of CoAP/CoCoA, but do not extract a clear congestion signal from noisy RTT samples.

Several studies have focused on evaluating the performance of CoCoA in a wide variety of scenarios ranging from emulated Zigbee networks to large scale IoT deployments (Betzler et al., 2016; Järvinen et al., 2015). Some of these studies have shown that CoCoA provides better performance than CoAP (Betzler et al., 2015a; Ancillotti and Bruno, 2017;

Hasan and Ahmed, 2018), whereas others have discussed the shortcomings of CoCoA e.g., (Bolettieri et al., 2017) shows that CoCoA has more retransmissions than CoAP when the number of requests increase.

A new method is proposed in (Balandina et al., 2013) to calculate RTO in CoAP based on the experimental study of Eifel retransmission timer (Ludwig and Sklower, 2000), which is originally proposed for calculating TCP timeouts. During this study, the authors observed that the default values of α ($\frac{1}{8}$), β ($\frac{1}{4}$) and K (4) suggested in RFC 6298 might not be suitable for large senders load. Subsequently, they propose a new RTO estimation algorithm that uses a single coefficient γ instead of α and β . γ is defined as a ratio between the current sample of RTT and RTO. Depending on the value of γ , the authors suggest to carefully adapting RTO.

Four modifications of CoCoA are discussed in (Bhalerao et al., 2016): CoCoA-Fast (CoCoA-F), CoCoA-Strong (CoCoA-S), CoCoA-4-State and CoCoA-4-State-Strong. This work mainly highlights that the performance of CoCoA is conservative in the presence of lossy wireless links, owing to the side effect of its weak estimator. Initially, the authors try to increase the aggressiveness of CoCoA by reducing the values of VBF, backoff thresholds, initial and maximum RTO, and name the resulting variant as CoCoA-F. However, it is observed that the performance of CoCoA-F remains conservative like CoCoA. Subsequently, the authors propose a 4-state higher granularity estimator that helps to distinguish between wireless link losses and congestion losses. Finally, three schemes: CoCoA-Strong, CoCoA-4-State and CoCoA-4-State-Strong are discussed to overcome the performance problems of CoCoA in lossy wireless links. The results show that CoCoA-4-State-Strong adapts to packet losses and achieves 35-60% more throughput than CoCoA. However, it also leads to 20% more retransmissions than CoCoA.

An adaptive mechanism for handling congestion in CoAP is proposed in (Hassan et al., 2016) that depends on traffic priority and considers the packet loss rate. This mechanism has three main components: assigning traffic priority, adaptive RTO and adaptive backoff timer. Priorities are assigned depending on whether the traffic emerges from a critical device (e.g., medical sensors or fire monitoring systems) or a non-critical device (e.g., smart home devices). Depending on traffic priorities and the packet loss rate, RTO and backoff timer values are calculated. The major limitation of this approach is that it is not implemented and evaluated in the paper which raises concerns about the feasibility of its deployment.

Table 2.1: Classification of Congestion Control Mechanisms for CoAP

Paper Name	Overview	Applications	Performance Metrics	Simulation Environments
Congestion Control in Reliable CoAP Communication	Mechanism to handle congestion in CoAP	Client - Server	Throughput and Packet Delivery Ratio (PDR)	Contiki OS, Cooja Experimental
Congestion Control for CoAP Cloud Services	Analysis of congestion in CoAP with cloud services	Publisher - Subscriber	Throughput	Testbed, FlockLab
Modeling and Analysis on Congestion Control in Internet of Things	Develop new protocol IRED	Accessing the IoT services through AQM	PDR and Latency	OMNeT++
Back pressure Congestion Control for CoAP/6LoWPAN Networks	Comparison of congestion control scheme with back pressure and UDP based protocol stack	client transmits packet to server through border router	Latency	network simulator-3
CoCoA+: An Advanced Congestion Control Mechanism for CoAP	Enhancement in CoCoA	Client - Server	Throughput and Latency	Contiki OS, Cooja
Evaluation of Advanced Congestion Control Mechanism for Unreliable CoAP Communication	Analysis of congestion for unreliable communication with CoAP Observe	Publisher - Subscriber	Throughput, PDR and Latency	Experimental Testbed
Experimental Evaluation of alternative Congestion Control algorithms for Constrained Application Protocol (CoAP)	Comparison of CoAP, CoCoA and other two TCP based congestion control algorithms	Client - Server	Number of retransmissions, Flow Completion Time (FCT) and RTO	Experimental Testbed
CoAP Congestion Control for the Internet of Things	Explain how we can handle congestion using CoAP in IoT	Client - Server	Throughput	Experimental Testbed, FlockLab
An Analysis and Improvement of Congestion Control in the CoAP Internet of Things Protocol	Improvement in the mechanism to estimate Variable Backoff Factor	Client - Server	Throughput	Contiki OS, Cooja
Experiment Evaluation of Congestion Control for CoAP Communications without End-to-End Reliability	Handle the congestion that caused by NON Confirmable messages	Publisher - Subscriber	PDR and Latency	Experimental Testbed, FIT/IoT-LAB
Adaptive Congestion Control mechanism in CoAP Application Protocol for Internet of Things	Mechanism to handle congestion in CoAP based on traffic priority	Client - Server	-	-

Table 2.1: Classification of Congestion Control Mechanisms for CoAP - Contd...

Paper Name	Overview	Applications	Performance Metrics	Simulation Environments
Round Trip Time based Adaptive Congestion Control with CoAP for Sensor Network	Mechanism to handle congestion in CoAP through retransmissions count	Client - Server	RTO and Throughput	Experimental Testbed
Design and Evaluation of a Rate based Congestion Control mechanism in CoAP for IoT Applications	Traffic based congestion control mechanism for CoAP	Client - Server	Data collection delay, Inter-packet delivery delay and Packet Loss Ratio (PLR)	Contiki OS, Cooja
pCoCoA: A precise Congestion Control algorithm for CoAP	An adaptive congestion control mechanism for CoCoA	Client - Server	RTO, Number of retransmissions, Throughput and Delay	Contiki OS, Cooja
FASOR Retransmission TimeOut and Congestion Control mechanism for CoAP	A new adaptive Retransmission Timeout based congestion control mechanism for IoT	Client - Server	FCT, Number of retransmissions and RTO	Experimental Testbed
Is CoAP Congestion Safe?	Proposed new backoff techniques for CoAP based congestion control mechanism	Client - Server	FCT and Number of retransmissions	Experimental Testbed
BDP-CoAP: Leveraging Bandwidth-Delay Product for Congestion Control in CoAP	A new rate based congestion control mechanism for CoAP (BBR + CoAP)	Client - Server	Goodput, Fairness and Number of retransmission	Contiki OS, Cooja
CoCo-RED: Congestion Control in CoAP Observe group Communication	Modified the RED AQM mechanism and used Fibonacci based backoff to develop a new congestion control mechanism for CoAP	IoT services through AQM	Response time and Packet loss	Contiki OS, Cooja
CACC - Context Aware Congestion Control algorithm for lightweight CoAP/UDP based Internet of Things traffic	A novel congestion control mechanism for CoAP	Client - Server	Throughput, Energy consumption, Delay and Packet loss	Contiki OS, Cooja
Enhancement of CoAP based Congestion Control IoT Network - a novel approach	A new mechanism to handle the congestion control in CoAP	Client - Server	Latency, Energy consumption, and Packet loss	Contiki OS, Cooja
mlCoCoA - a machine learning based Congestion Control for CoAP	A machine learning based adaptive congestion control mechanism for CoAP	Client - Server	Throughput	Contiki OS, Cooja
Distance based Congestion Control mechanism for CoAP in IoT	A distance and RTT based technique to predict the network congestion	Client - Server	PDR and Delay	Contiki OS, Cooja
EnCoCo-RED: Enhanced Congestion Control mechanism for CoAP Observe group Communication	Enhancement to the CoCo-RED congestion control mechanism	IoT services through AQM	Packet loss and Response time	Contiki OS, Cooja

Yet another adaptive congestion control scheme for CoAP is proposed in (Lee et al., 2016) which relies on accurately estimating the RTT *of retransmitted packets*, precisely when a single packet needs *multiple retransmissions* (i.e., when retransmissions fail). The authors show that this approach offers improvements in terms of throughput and number of successful transactions when the number of clients is high (i.e., when the network is congested), but otherwise performs similarly to CoAP. However, this approach is not effective when a packet does not need *multiple retransmissions* (i.e., when a single retransmission suffices).

Real-time IoT traffic is diverse in nature. Hence, a thorough study is presented (Ancillotti and Bruno, 2017) to evaluate the performance of CoCoA by considering typical IoT scenarios with different traffic patterns. CoAP and CoCoA have been assessed by means of simulations with different loads that have been offered as per the increasing number of clients. The performance of CoCoA is worse than that of CoAP when the number of clients are less and with bursty traffic; it worsens further. It was observed that the performance of CoCoA was poor for bursty traffic patterns due to inappropriate RTO estimation.

CoAP with a new rate based congestion control (CoAP-R) (Ancillotti et al., 2018) is yet another algorithm targeted to improve the performance of CoAP in bursty traffic environments. CoAP-R controls the sending rate of CoAP sources and endorses a rate-based mechanism (instead of window-based mechanisms) to control the traffic. CoAP-R is designed to achieve maximum bandwidth from the bottleneck link capacity and allocate the network resources based on the max-min fairness. The simulation results in the paper show that CoAP-R distributes the network resources equally amongst the senders and reduces the delay compared to CoAP and CoCoA.

Depending on extensive evaluation, (Bolettieri et al., 2018) highlights the shortcomings of CoCoA: RTO too close to RTT, lack of weak estimator update, insufficient weak estimator weight, RTO peaks in response to RTT decrease and Excessive RTO growth. This work also makes an observation about the poor performance of CoCoA in bursty traffic patterns. To address these limitations, a new extension called precise CoCoA (pCoCoA) is proposed and evaluated. pCoCoA eliminates the use of weak estimator, introduces a new way to initialize RTT measurement parameters and reduces spurious retransmissions. The results show that pCoCoA reduces the number of retransmissions without affecting the throughput and delay. However, some of the fixed values introduced in the paper

might not be suitable for a wide range of IoT scenarios.

IoT plays a crucial role in health-related applications. A comparative study of CoAP and CoCoA in e-health scenarios is presented in (Hasan and Ahmed, 2018). The authors evaluate the performance of CoAP and CoCoA in different topologies such as single client to single server, multiple clients to single server, single client to multiple servers and multiple clients to multiple servers. The parameters considered for comparison are goodput, packet delivery ratio, average delay and total number of packets dropped. It is observed that the total number of packets dropped in CoCoA is much lesser than CoAP. Moreover, CoCoA outperforms CoAP in terms of goodput in all topologies. The authors also show that CoCoA consumes less bandwidth, is more adaptive and scalable than CoAP.

Bufferbloat refers to the problem of having excessive queue delays in the network. In (Järvinen et al., 2018), the authors state that different congestion control mechanisms designed for CoAP fail to handle the problem of Bufferbloat and perform spurious retransmissions. Through extensive evaluations, the authors show that CoAP and CoCoA fail to handle the problem of Bufferbloat, and due to this, there is significant wastage of the network bandwidth. Moreover, authors also find the main cause of inaccuracy in the backoff logic of RTO. Subsequently, the RTO backoff is modified to overcome this concern.

Along similar lines, (Jarvinen et al., 2018), attempts to overcome the problem of bufferbloat in scenarios that are prone to heavy congestion. A new mechanism called Fast-Slow RTO (FASOR) is proposed which tries to infer whether the packet loss is due to the wireless link disruption or due to congestion. It implements the following three fundamental functionalities: Fast RTO computation, Slow RTO computation, and Self-adaptive retransmission timer backoff mechanism. Fast RTO computation is analogous to the TCP RTO mechanism except the minimum RTO bound, and it is evaluated for explicit RTO samples. Slow RTO's functionalities are similar to those of Karn's algorithm. Its usage is to keep minimal time for backoff. The results show that FASOR has shorter Flow Completion Times (FCT) compared to CoAP and CoCoA, and it controls the RTO in heavily congested scenarios.

Evaluating a new algorithm in large-scale IoT deployment is a non-trivial task. In (Vallati et al., 2018), the authors present an evaluation of the default congestion control mechanism of CoAP in real time experiments with the WiSHFUL platform (Wireless Software and Hardware platforms for Flexible and Unified radio and network Control)

(Ruckebusch et al., 2017). WiSHFUL is a large-scale experimental platform which offers a unique interface to design and control the experiment at run-time. The main goal of this work was to verify the working of WiSHFUL platform, and as part of a case study, the default CoAP congestion control is compared with a *simple algorithm* based on the measured RTT value over time. The results show that the default congestion control mechanism is more scalable and robust than the *simple algorithm*.

In (Mišić et al., 2018) a new feature called *observe* is proposed which provides information about the state of a resource at CoAP server to a CoAP client. The authors conduct experiments and highlight the need to have congestion control at both CoAP clients and servers, particularly when features like *observe* are deployed. It is mentioned that different RTT estimation algorithms can be used at clients and servers but there are no recommendations about the choice of congestion control algorithms that are most suitable. The authors in (Huang et al., 2014) use TCP-AQM mechanisms to handle congestion in IoT.

Similarly, the authors in (Suwannapong and Khunboa, 2019) attempt to solve the problem of group communication and *observe* resources in CoAP that cause buffer overflow and packet loss. To overcome this, a new congestion control mechanism for CoAP Observe Group Communication is proposed called Congestion Control Random Early Detection (CoCo-RED). It incorporates the following features: adaptive RTO calculation, a modification and integration of the TCP AQM algorithm (Random Early Detection), and the Fibonacci Pre-Increment Backoff (FPB) mechanism. EnCoCo-RED (Suwannapong and Khunboa, 2021) is an enhancement over CoCo-RED that replaces the FPB with static backoff and improves the drop probability in the revised RED algorithm to better control the network congestion in different traffic scenarios. The authors show that EnCoCo-RED effectively prevents the buffer overflow that leads to congestion and outperforms the CoAP and the CoCo-RED in terms of the settling time, throughput, packet loss, and response time.

A new rate based congestion control algorithm is proposed in (Ancillotti and Bruno, 2019) for CoAP, called BDP-CoAP, which is derived from the TCP BBR (Bottleneck Bandwidth and Round-trip propagation time) protocol. BBR’s bottleneck bandwidth estimator has been redesigned to deal with lossy links and channel unfairness, which are common in constrained IoT networks. A node in an IoT network might acquire a channel for a short time and achieve high instantaneous delivery rates. To avoid overestimation of bandwidth in the event of short-term unfairness of channel access, BDP-CoAP employs

an estimator that combines both maximum and minimum delivery rate measurements to derive bottleneck bandwidth estimates. CoAP sends only one packet at a time, hence the typical BBR estimation technique does not fit in BDP-CoAP. Thus, BDP-CoAP monitors the number of missed bandwidth samples over the observation period and uses this information to make the bottleneck bandwidth estimator more or less aggressive in order to adjust the transmission rate accordingly. The results show that BDP-CoAP improves throughput and fairness while obtaining similar goodput as CoAP and CoCoA.

According to the Karn and Partridge algorithm (Karn and Partridge, 1987), RTT calculation of retransmitted packets causes ambiguity which we described in Chapter 1. CoCoA calculates the RTT of retransmitted packets, but it has been shown to be inaccurate when the traffic is bursty (Ancillotti and Bruno, 2017; Hasan and Ahmed, 2018), and hence, results in spurious retransmissions. In (Akpakwu et al., 2020), the author proposed a new lightweight Context-Aware Congestion Control (CACC) mechanism for CoAP that leverages CoCoA. CACC provides an adaptive congestion control mechanism that employs strong, weak, and failed RTT estimators to determine the exact network characteristics, as well as Retransmission Count (RC) for smoothed RTT observation and lower bound restriction to address the issue of fluctuating IoT traffic. The results show that CACC reduces the number of retransmissions while maintaining higher throughput, and minimizes packet losses in various network scenarios when compared to CoAP and CoCoA.

Several studies show that CoCoA does not adapt to the network conditions because it uses constant coefficients and weight values to estimate the RTO (Ancillotti and Bruno, 2017; Hasan and Ahmed, 2018; Demir and Abut, 2020). Hence, the authors in (Demir and Abut, 2020) proposed mlCoCoA, a new congestion control mechanism that uses machine learning techniques to dynamically determine RTO-related coefficients based on IoT network characteristics. The results show that mlCoCoA outperforms existing mechanisms in terms of throughput. However, mlCoCoA makes use of a predefined dataset that contains the constant coefficient values. It also takes into account the *fixed* input variables such as number of clients, packet size, and physical layer PDR and applies a machine learning method on top of that. Furthermore, the authors obtained the training set offline during the learning phase and manually passed the predicted coefficients to mlCoCoA for RTO estimation. These features of mlCoCoA make it unsuitable for real IoT deployments because parameters such as the number of clients, packet size and PDR

are highly varying in nature, thus requiring more training.

A new CoAP based congestion control mechanism called FLCoCoA is proposed in (Aimtongkham et al., 2021) that optimally adjusts the initial RTO using relative weights and an adaptive RTO backoff mechanism by using fuzzy logic for retransmissions. It estimates RTT based on the relative signal strength, RTT Interval, and RTT Jitter to better respond to early congestion scenarios. The results show that FLCoCoA reduces the number of retransmissions and consumes less power while achieving higher throughput and less delay compared to other congestion control mechanisms.

The authors in (Gheisari and Tahavori, 2019) propose a new congestion control mechanism for CoAP that uses a learning automata to tune the group of parameters used to control the congestion. An adaptive congestion control mechanism for CoAP is proposed in (Deshmukh and Raisinghani, 2020) that employs a dynamic scaling factor rather than a fixed scaling factor for RTO estimation.

To improve the CoAP congestion control mechanism, a new Distance-based Congestion Control mechanism called DCC-CoAP is proposed in (Bansal and Kumar, 2020). DCC-CoAP uses node distance and RTT measurements to limit the number of retransmissions. The primary goal of measuring distance is that it is inversely proportional to signal strength and directly proportional to RTO. DCC-CoAP considers the distance between the two communicating nodes in the network. If the distance is large, the probability of link failures is high due to low signal strength. The authors evaluate the performance of the DCC-CoAP in homogeneous and mixed traffic scenarios with varying data rate. The results show that DCC-CoAP has lesser retransmissions and a better packet delivery ratio and delay than CoAP. However, DCC-CoAP does not take the samples for the retransmitted packet into account. Moreover, DCC-CoAP calculates the distance between the client and the server using Euclidean distance, which does not provide an accurate distance when the nodes are dispersed in a multidimensional space. It also employs *fixed* RTT values for the first two transmissions. For constants such as k and δ , DCC-CoAP uses *fixed* values. DCC-CoAP is built on top of CoAP and estimates RTT using the timestamp option, which adds overhead to the IoT network. Furthermore, DCC-CoAP employs a maximum cap limit (i.e., 15 meters) for the distance between two nodes.

Table 2.2 lists the various CoAP-based application layer congestion control mechanisms, along with their characteristics and the underlying mechanisms from which they were derived.

Table 2.2: Summary of existing CoAP-based congestion control mechanisms

Congestion control scheme	Backoff mechanism	RTT/RTO estimators	Derived from
CoAP	BEB	None	None
CoCoA	BEB	Strong and Weak	LinuxRTO
CoCoA+	VBF	Strong and Weak	CoCoA
CoCoA-S	VBF	Strong and Weak	CoCoA
CoCoA-E	VBF	Strong and Weak	CoCoA and Eifel
4-State-Strong	4-state VBF	Four estimators	CoCoA
Enhanced CoCoA	VBF	Strong	CoCoA
pCoCoA	VBF	Strong	CoCoA
CoAP-R	Rate control	Monitoring RTT variations	CoAP and RTT monitoring
BDP-CoAP	Rate control	Strong	TCP BBR
CCCLA	Rate control	Strong	Learning automata
mlCoCoA	VBF	Strong and Weak	Machine learning
CACC	Dynamic RTO	Strong, Weak and Failed	CoCoA
CoCo-RED	FPB	None	CoAP

2.2.2 Routing layer solutions

Another approach of controlling network congestion is by designing load-aware routing protocols. It is done by making all communications follow a hop-by-hop approach, but this significantly increases the control overhead in the network as each node has to share the measured congestion level with other nodes. Several insightful RPL surveys have been proposed, which reviewed the current state of RPL and the new enhancements required to meet recent trends (Lim, 2019; Iova et al., 2016; Kim et al., 2017a; Kamgueu et al., 2018; Ghaleb et al., 2018).

An analysis of congestion in 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) networks shows that a large number of packet drops at the routing layer are due to buffer overflow rather than channel loss (Al-Kashoash et al., 2016a). So, the authors in (Al-Kashoash et al., 2016b) proposed a new routing metric for RPL which prohibited the number of packet losses due to buffer overflow when congestion occurred. The authors proposed a new routing metric called Buffer Occupancy (BO) and a new objective function called Congestion-Aware Objective Function (CA-OF). The proposed

technique shows improvement in the throughput and packet delivery ratio. Along the same lines, the authors in (Al-Kashoash et al., 2016a) proposed that buffer occupancy should be used as a measure of congestion, and accordingly proposed a new version of RPL. Such an approach requires extra control messages to exchange the buffer occupancy information among neighbor nodes, and hence incurs more overhead.

Most of the packet losses in heavy traffic scenarios are due to congestion and load balancing problem that emerge during the parent selection in routing (Kim et al., 2016, 2017a). In order to tackle this issue, a new mechanism called simple Queue Utilization based RPL (QU-RPL) has been proposed in (Kim et al., 2016). In QU-RPL, parent selection is done based on queue utilization of the neighbor nodes as well as hop distances from the RPL border router. This mechanism is very adequate in heavy queue losses and increases the packet delivery ratio. In (Kim et al., 2017b), the authors note that when transmission power and routing topology are controlled collectively and attentively, a multihop wireless network gains better bandwidth and stability. There is a significant loss in bandwidth due to the construction of routing topology (based on link quality and hop distance) with fluctuation in transmission power. So, a robust and dispersed mechanism called Power Controlled RPL (PC-RPL) is proposed. It's a control mechanism which handles transmission power and routing topology efficiently. In (Kim et al., 2015), to understand the trade-offs in choosing TCP over RPL, an empirical study has been done on it. Their results show how embedded TCP interoperates with Linux TCP and underlying RPL (and vice versa).

This work mainly focuses on optimizing the congestion control mechanisms used in CoAP and CoCoA.

Chapter 3

Geometric Sequence Technique for Effective RTO Estimation in CoAP

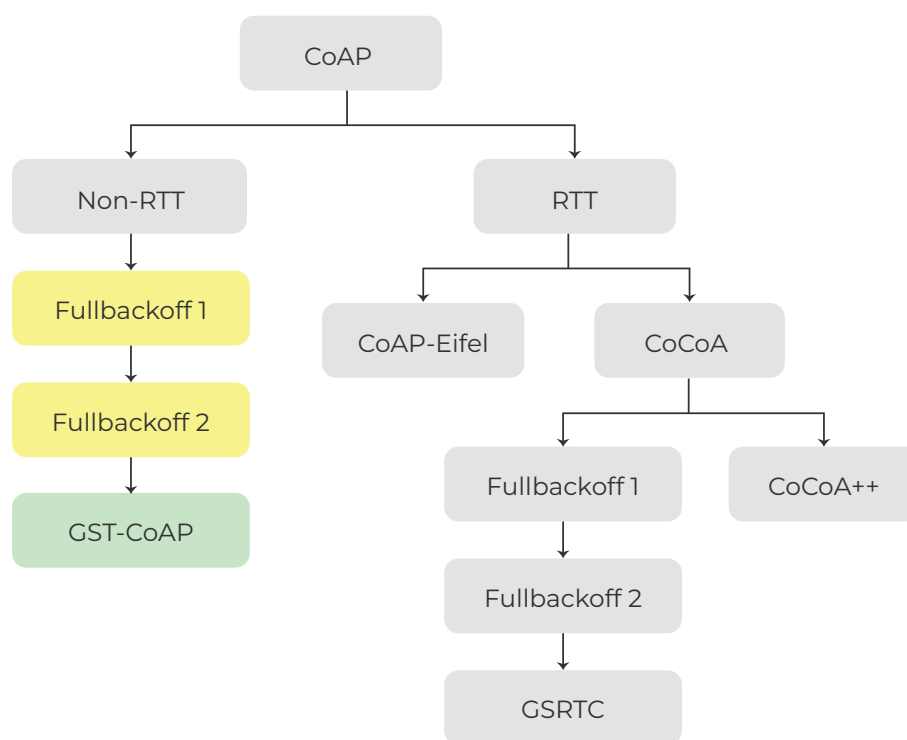


Figure 3.1: Contribution (highlighting (GST-CoAP))

This chapter discusses a simple non-RTT based congestion control mechanism proposed in this work, called GST-CoAP, to improve RTO estimation in CoAP (as shown in Figure 3.1). Unlike CoAP, which resets the RTO to its default value after receiving an ACK for the retransmitted packet, GST-CoAP gradually reduces the RTO to its default value based on the number of *consecutive* successful transmissions. This avoids a *single* successful transmission causing a large value of RTO (due to several failed transmissions) to fallback to the default value and unlearn the network behaviour.

3.1 Motivation

This section emphasizes on the need for a better RTO estimation technique for CoAP. For every new transmission, CoAP sets the initial RTO to a value that is selected randomly between $[2s, 3s]$. This means that when the network is *not congested*, CoAP does not adapt the RTO and instead resets it to a low RTO value which could lead to spurious retransmissions when RTT is high.

Two new techniques, namely *Fullbackoff1* variant and *Fullbackoff2* variant are proposed in (Järvinen et al., 2018) to improve the RTO estimation in CoAP. These techniques do not use RTT measurements; hence, the working of the original CoAP algorithm remains largely unaffected. Unlike CoAP, both variants avoid resetting the RTO immediately after receiving an ACK for the *retransmitted* packet.

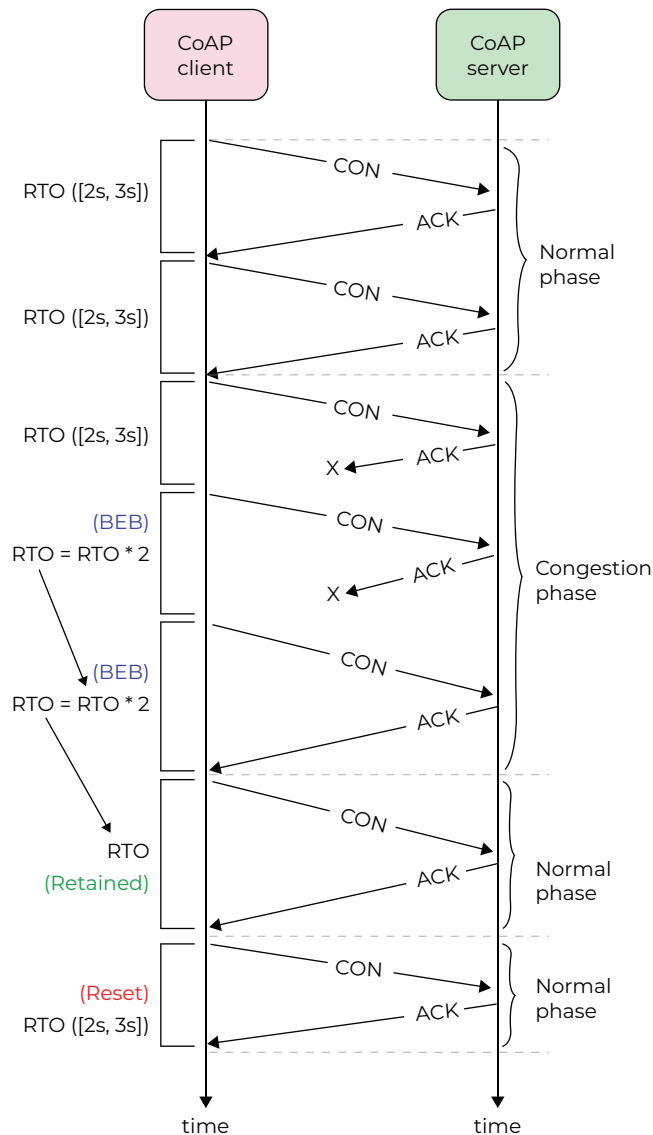


Figure 3.2: Working of *Fullbackoff1* variant of CoAP

This technique overcomes the limitation of *Fullbackoff1* variant. Instead of resetting the RTO when a single packet transmission succeeds without *retransmissions*, this technique gradually reduces the RTO value if *consecutive* packets are successfully cleared without *retransmissions*. For every packet that is cleared without *retransmissions*, the RTO is reduced by half as shown in Figure 3.3. Eventually, the RTO is reset when its value falls below the recommended default value i.e., when $RTO < [2s, 3s]$. Thus, the RTO value is increased in a binary exponential manner (i.e., doubled) when the packet transmission fails and it is decreased in a binary exponential manner (i.e., halved) when a packet is cleared without *retransmissions*. This approach of decreasing the RTO value gradually ensures that a few successful transmissions in between several failed transmissions will not cause the RTO to be reset, like it was done in the *Fullbackoff1* variant. The geometric sequence technique proposed in this work builds upon the *Fullbackoff2* variant.

3.2 GST for RTO estimation in CoAP

3.2.1 Design

GST-CoAP is a simple enhancement to the *Fullbackoff2* variant. Unlike *Fullbackoff2* variant which decreases RTO in a binary exponential manner (i.e., reduces RTO by half) when packets are cleared without *retransmissions*, GST-CoAP decreases RTO in a pattern that follows a geometric sequence, as shown in Eq. (3.1).

$$RTO_{next} = RTO_{prev} \times r^{count}, \quad count \neq 0 \quad (3.1)$$

where RTO_{next} represents the RTO value to be used for the next transmission, RTO_{prev} represents the RTO value that was used for the previous transmission, r is a constant ($\frac{1}{2}$), and $count$ indicates the number of *consecutive* ACKs received.

The main goal of GST-CoAP is to increase the network throughput and minimize the FCT by quickly reducing the RTO value to its default when the network is *not congested* and packets are getting cleared *consecutively without retransmissions*. For example, assume that the *Fullbackoff2* variant technique is used and the current value of RTO is 128s ($= 2^7$). It would take 7 packets to get cleared *consecutively without retransmissions* for RTO to get reduced to its initial value of $2s^1$ because with every ACK arrival, the RTO is reduced to half. In GST-CoAP, it would take only 3 packets to get cleared *consecutively*

¹Typically the initial value of RTO is between 2s and 3s, but we avoid this and assume it to be exactly 2s to simplify the explanation.

Algorithm 1: Geometric Sequence Technique

```
1 Initialize the RTO from the interval  $[2s, 3s]$ 
  // count represents the number of consecutive ACKs received so far
2 count = 0
  // RTO calculation
3 if this is a confirmable message then
4   if maximum number of retries are not exhausted then
5     if this is the first transmission then
6       |  $RTO = RTO + (\text{random}() \% \text{response timeout backoff mask})$ 
7     else
8       | // Double the RTO
9       |  $RTO \ll= 1$ 
10    end
11  end
12 if there are more packet transmissions to be done then
13   if this is not a first transmission then
14     | // Retain the previous RTO
15     |  $RTO = RTO_{\text{previous}}$ 
16   else
17     | // Increment the count variable
18     |  $count++$ 
19     |  $RTO = RTO \times (\frac{1}{2})^{\text{count}}$ 
20     if the current RTO is lesser than the default RTO then
21       | // reset the RTO to the default
22       |  $RTO = \text{from the interval } [2s, 3s]$ 
23     end
24   end
25 end
```

without retransmissions for RTO to get reduced to its initial value of 2s because every *consecutive* ACK arrival increases the *count* by 1 in Eq. (3.1). Thus, GST-CoAP avoids unnecessarily keeping RTO value higher when the network is not congested and packets are getting cleared *consecutively without retransmissions*. This approach improves the network throughput and minimizes the FCT for IoT devices. GST-CoAP is described in Algorithm 1 and its basic working is demonstrated in Figure 3.4.

3.2.2 Implementation Challenges

CoAP's implementation in Contiki OS (Dunkels et al., 2004) has been used to implement *Fullbackoff* variants of CoAP and GST-CoAP. It requires 52 lines of code change in the CoAP implementation. *er-coap-transactions.c* and *er-coap-transaction.h* files in *contiki/apps/er-coap* directory have been modified to implement *Fullbackoff* variants of CoAP and GST-CoAP.

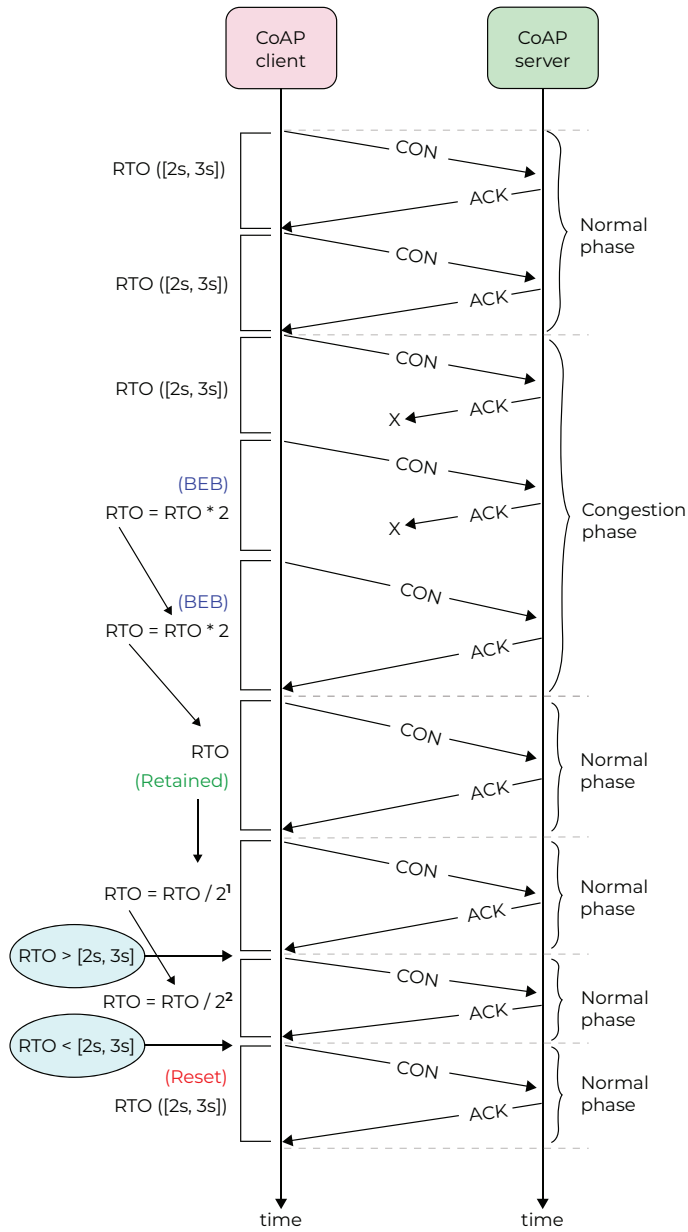


Figure 3.4: Working of GST for RTO estimation in CoAP

GST-CoAP uses a geometric sequence to reduce RTO as shown in Eq. (3.1). One of the most difficult aspects of the GST-CoAP's implementation was to integrate the $exp()$ function in Contiki to reduce the RTO value in an exponential pattern. Although Contiki contains some built-in functions, such as $expf()$, the implementation complexity causes memory overflows. Hence, we have created a simple C function in *er-coap-transaction.c* that uses the shift operator (integer arithmetic).

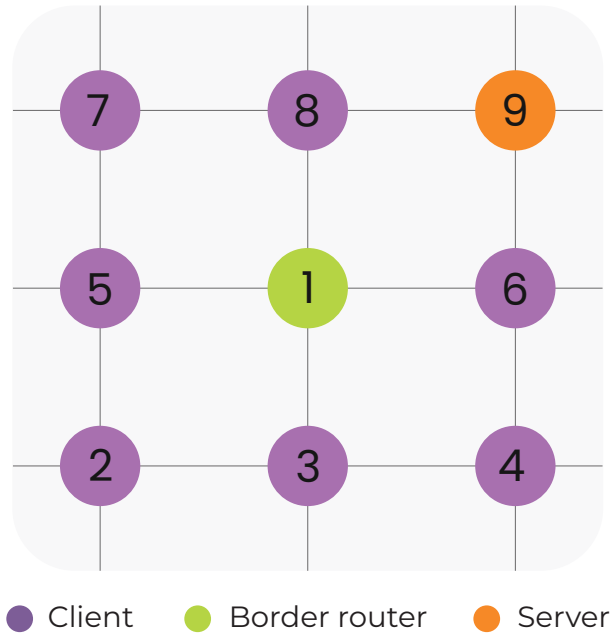


Figure 3.5: Grid Topology - GST-CoAP

3.3 Evaluation

3.3.1 Network Configuration

There are several implementations of CoAP that are openly available, but the *Fullbackoff1* and *Fullbackoff2* variants in (Järvinen et al., 2018) were implemented in libcoap (Bergmann, 2012). However, for our experiments, we have used the Cooja simulator (Osterlind et al., 2006) in Contiki-3.1 and a real testbed at FIT/IoT-LAB (Adjih et al., 2015). Thus, besides GST-CoAP, we implemented the *Fullbackoff1* and *Fullbackoff2* variants in Contiki-3.1. FIT/IoT-LAB is a state-of-the-art facility which provides a large-scale platform for the research community to test their IoT protocols and applications. To perform experiments in Cooja and FIT/IoT-LAB, we have considered a grid topology of (3×3) as shown in Figure 3.5. Node 1 at the center is configured to be a Border router, Node 9 at the top-right corner is configured as a Server and all the other nodes are configured as Clients. This is a popular topology used in the literature to evaluate the congestion control mechanisms of CoAP (Betzler et al., 2014, 2015b). The most congested nodes in the grid topology are internal nodes as they are within the transmission range of four adjacent nodes. In the Cooja simulator, T-mote Sky (AdvantivSystemsServices, 2014) is used for the Border router and Zolerita z1 (Zolertia, 2014) is used for Clients and Server, whereas ARM Cortex microcontroller based M3 mote is used for all nodes in FIT/IoT-LAB. Contiki’s erbium CoAP (er-coap) application has been used for the Clients to send a request via the Border router, and wait for the Server to respond. The packet sending

interval has been set to 100ms. All experiments are repeated five times and the duration of each experiment is 600s. We have used the average of these five runs to compare the results obtained by using GST-CoAP to those obtained with original CoAP, CoAP with *Fullbackoff1* and CoAP with *Fullbackoff2*. The source code of GST-CoAP is available at (SourceCode, 2020a).

3.3.2 Performance Metrics

The following performance metrics have been considered for evaluating the effectiveness of the proposed technique.

- **Flow Completion Time (FCT):** It indicates the total amount of time needed to complete the transmission of all the packets in one flow. In this work, FCT is measured as the time taken to exchange 50 messages successfully.
- **Total number of retransmissions:** Retransmissions are very expensive in constrained networks because they lead to wastage of resources. Hence, this parameter has been considered for our work.
- **Throughput:** The throughput is defined as the total number of bytes sent per second. It has been measured over a 50ms interval for every client in this work.

3.3.3 Results and Discussions

A Flow Completion Time (FCT)

Figure 3.6 shows the comparison of different techniques in terms of FCT. We have measured the FCT for 50 request-response pairs successfully completed per client. Figure 3.6a and 3.6b show the results obtained from Cooja simulator and FIT-IoT/LAB, respectively. In the experiments performed on Cooja simulator, the FCT with GST-CoAP is 31.25% lesser than that with CoAP, and 20.5% and 14.97% lesser than that with *Fullbackoff1* and *Fullbackoff2*, respectively. In the experiments performed on FIT/IoT-LAB, the FCT with GST-CoAP is 34.86% lesser than that with CoAP, and 23.36% and 16.84% lesser than that with *Fullbackoff1* and *Fullbackoff2*, respectively.

The default CoAP mechanism has the maximum FCT among all the techniques because it resets RTO for every new transmission, which leads to a large number of spurious retransmissions. Hence, the transmission of 50 messages takes a lot more time due to

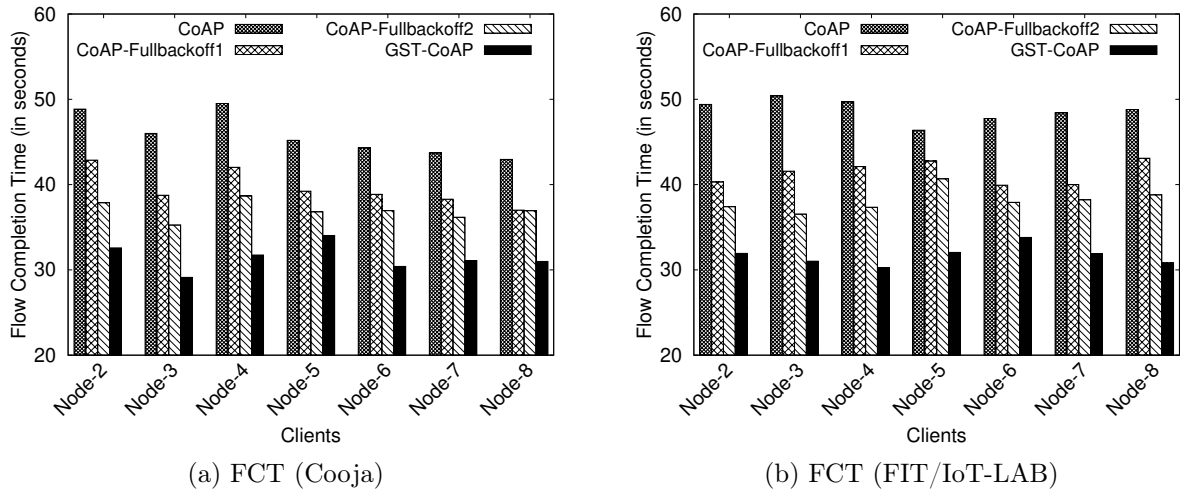


Figure 3.6: Comparison of FCT in Cooja and FIT/IoT-LAB

frequent and unnecessary retransmissions. The *Fullbackoff* variants clearly offer an improvement over the default CoAP. When an ACK for a *retransmitted* packet is received, *Fullbackoff1* retains the previous RTO value rather than resetting it to the default value, and hence, it has lesser FCT compared to CoAP. *Fullbackoff2* has lesser FCT than *Fullbackoff1* because *Fullbackoff2* gradually reduces the RTO and hence, avoids spurious retransmissions.

B Total number of retransmissions

If an ACK for a *retransmitted* packet is received, the default CoAP mechanism resets the RTO to a low value between [2s, 3s]. This low value of RTO could have an impact on the next packet transmission because if the network is congested and RTT is higher than the RTO, spurious retransmissions would occur and overload the buffer, resulting in packet drops. In *Fullbackoff* variants, the RTO value from the previous transmission is retained to allow sufficient time for the client to wait for the ACK and avoid unnecessary retransmissions. As shown in Figure 3.7, GST-CoAP reduces the number of retransmissions significantly when compared to other techniques. GST-CoAP uses a geometric sequence to reduce the number of *consecutive* successful packet transmissions needed to reach the default RTO value. Hence, if the network is not congested and packets get cleared without *retransmissions*, GST-CoAP senses this as an opportunity to quickly reduce the RTO and maximize the throughput. But if the network is congested and packet transmissions require retransmissions, it keeps the RTO to a higher value to avoid spurious timeouts. *Fullbackoff2* has lesser retransmissions than *Fullbackoff1* as expected, and *Fullbackoff1* has lesser retransmissions than the default CoAP technique as anticipated.

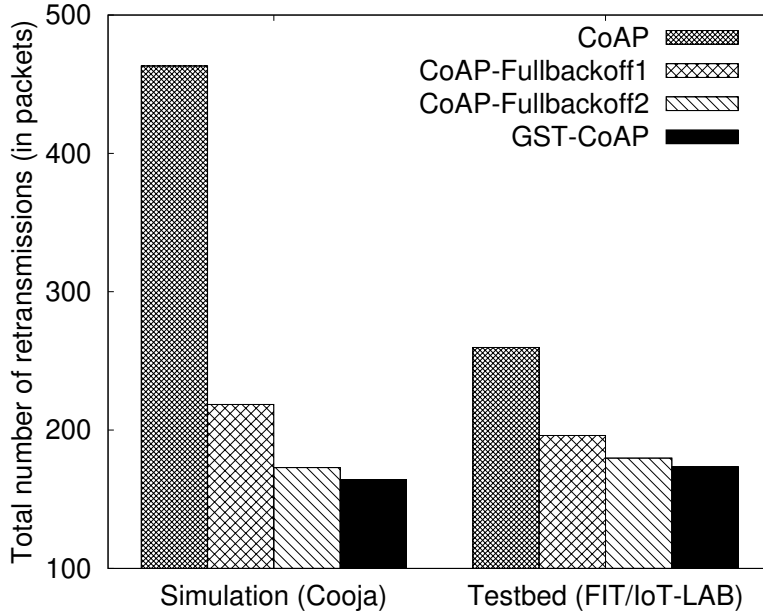


Figure 3.7: Total number of retransmissions: Cooja and FIT/IoT-LAB

C Throughput

A Cumulative Distribution Function (CDF) is used in Figure 3.8 to compare the throughput obtained with GST-CoAP and other techniques. The throughput of every client is calculated in terms of the number of bytes sent per second. It is observed from the plots that GST-CoAP offers a better overall throughput because it maintains RTO values such that there are fewer timeouts. CoAP has the lowest throughput due to unnecessary retransmissions. It means that the client has to wait longer before sending new packets over the network, resulting in lower network throughput. Additionally, GST-CoAP has lesser FCT and retransmission rates than other techniques, which increases the chances of the packet being transmitted successfully. As a result, the throughput of GST-CoAP is higher when compared to other techniques. The default CoAP mechanism achieves the least throughput, and *Fullbackoff2* variant has a marginally better throughput than *Fullbackoff1*.

3.4 Inferences

GST-CoAP uses a geometric sequence technique to gradually reduce RTO to its default value based on the number of consecutive successful transmissions. The results obtained from simulation studies and real time experiments validate that GST-CoAP minimizes the FCT, reduces the total number of retransmissions in the network and enhances the per-node throughput compared to existing *Fullbackoff* mechanisms and CoAP.

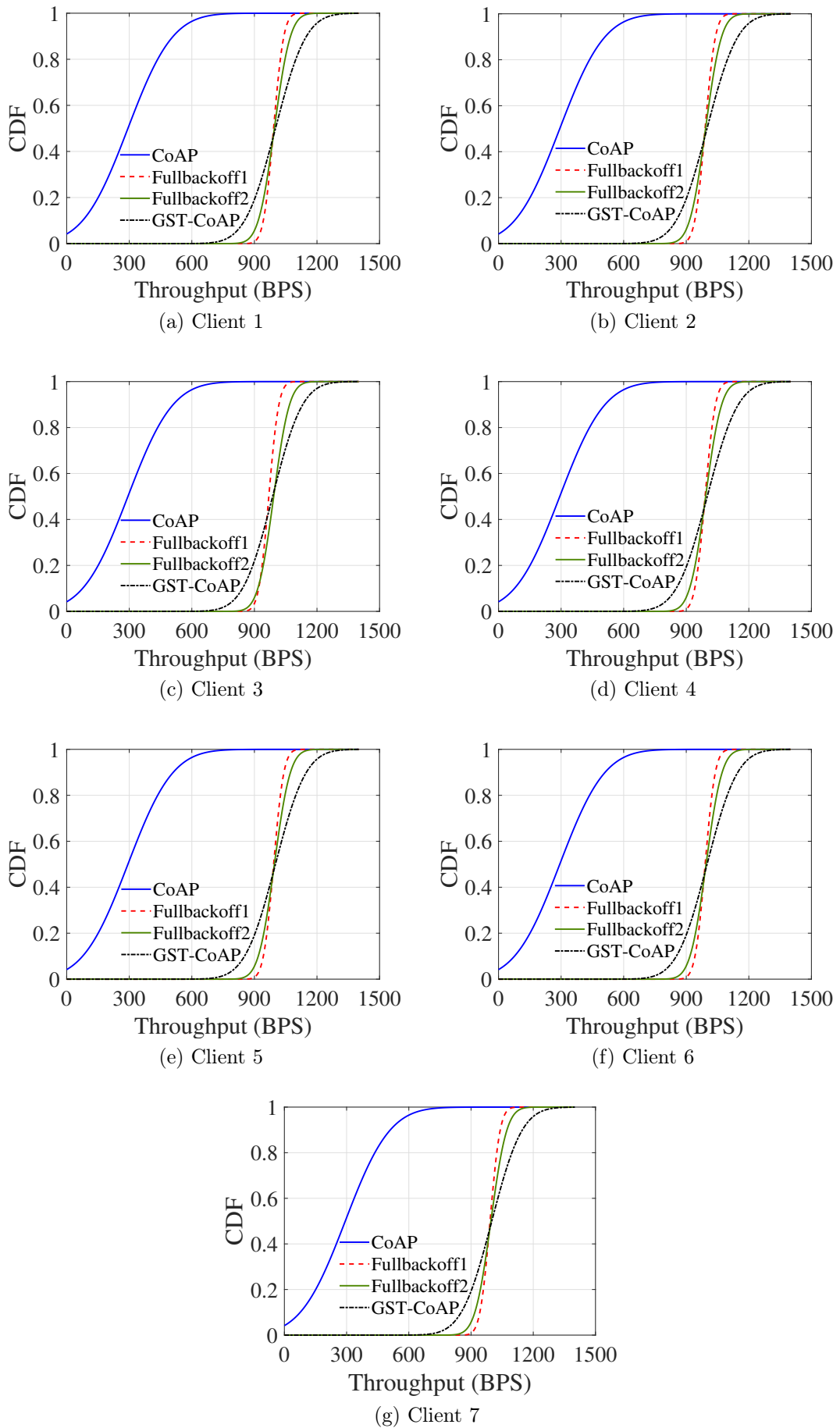


Figure 3.8: Throughput of CoAP and Variants in FIT/IoT-LAB

GST-CoAP is easy to deploy since it requires minor modifications to the working of original CoAP mechanism, and it is easy to integrate with other CoAP-based congestion control mechanisms developed so far. CoAP and GST-CoAP are non-RTT based, hence the overhead involved in RTT calculations are avoided, thereby minimizing the memory requirements. Thus, GST-CoAP is most suitable for devices that require memory footprint to be extremely low. Example use cases involve industry automation (smart grids) and healthcare, where sensor devices with extremely low memory footprints are deployed.

Chapter 4

Effective RTO estimation using Eifel Retransmission Timer in CoAP

4.1 Motivation

The default RTO estimation in CoAP is insensitive to network conditions because the RTO is randomly selected from a *fixed* interval, relies on packet loss to update it, and does not adapt according to the RTT. If the selected RTO value is lower than the RTT, network resources will be wasted due to spurious retransmissions, whereas if the RTO is significantly higher than the RTT, network resources will be wasted due to the large amount of time taken to detect packet losses. Thus, an adaptive RTO which is based on RTT is necessary to alleviate the aforementioned issues and improve network performance.

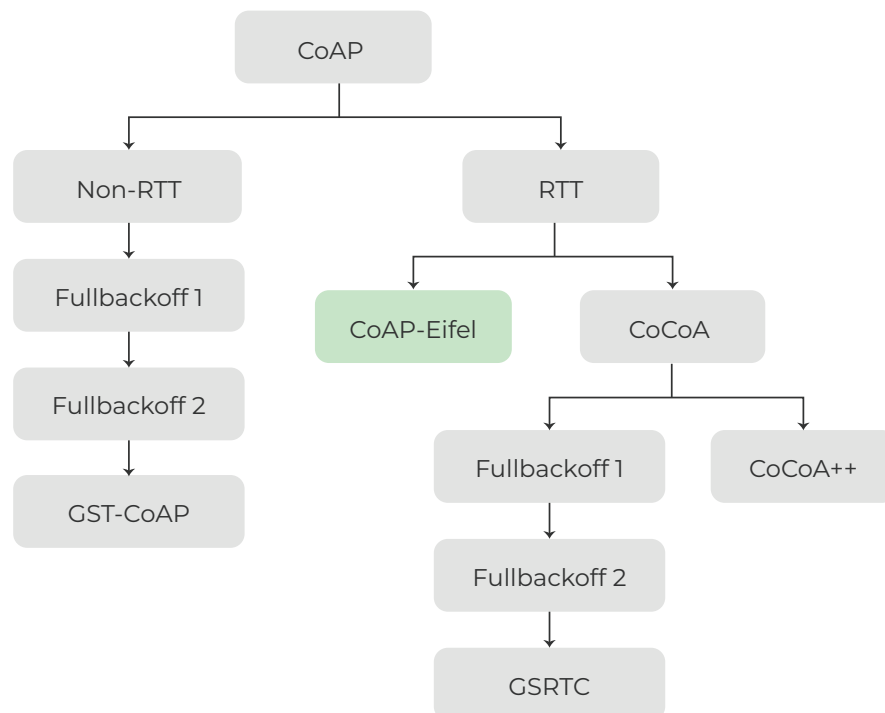


Figure 4.1: Contribution (highlighting (CoAP-Eifel))

This chapter discusses an adaptive congestion control mechanism for CoAP which estimates the RTO based on RTT measurements, as shown in Figure 4.1. The Eifel Retransmission Timer is a popular algorithm originally proposed for TCP. It assesses prevalent network conditions by measuring RTT and accordingly sets the TCP RTO. This work proposes to leverage the benefits of Eifel Retransmission Timer by integrating it with CoAP (when used with UDP) to obtain better RTO estimates and control congestion.

4.2 Overview of TCP-Eifel

4.2.1 TCP-Eifel

TCP-Eifel consists of three components: Eifel Algorithm (Ludwig and Sklower, 2000), Eifel Retransmission Timer and Retransmit forever, as shown in Figure 4.2.

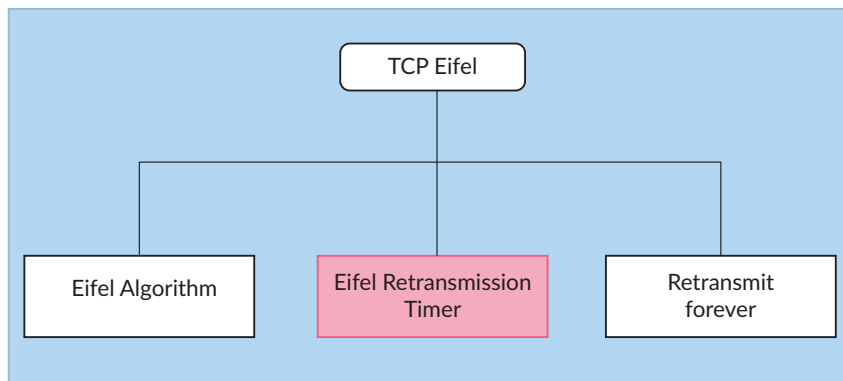


Figure 4.2: Components of TCP-Eifel

In the event of a packet loss followed by a retransmission, TCP RTO estimation process faces an ambiguity due to the fact that an ACK received could belong to an original transmission or the retransmission of the packet. The Eifel Algorithm eliminates this ambiguity by using the *timestamp* information available in the TCP header to distinguish ACK of an original packet and retransmitted packet.

Standard TCP decides to close down a connection after a fixed number of unsuccessful retransmissions. However, this is not suitable for time-insensitive applications as they can tolerate longer disconnections. Thus, the Retransmit Forever feature of TCP-Eifel proposes that the decision to close down a connection must be made by the application layer rather than the transport layer.

Eifel Retransmission Timer was introduced to eliminate the problems identified in BSD-Lite's (Berkeley Software Distribution) distribution of TCP (TCP-Lite). It was

observed that Eifel Retransmission Timer is a more accurate predictor of the upper bound of RTT and reacts faster to packet losses. Hence, it was shown to be an effective alternative to TCP-Lite’s retransmission timer.

4.2.2 Eifel Retransmission Timer

This timer was proposed to overcome four major problems identified with the RTO estimation technique in TCP-Lite, which is the name given to the TCP implementation in 4.4 BSD-Lite distribution of Berkeley Socket Distribution (BSD). The RTO estimation technique of TCP-Lite is governed by Eq. (4.1) to (4.4), where RTT is the last measured RTT or current RTT, $SRTT$ is the smoothed RTT estimator, δ is the difference between RTT and $SRTT$, $RTTVAR$ is the SRTT deviation estimator and $ticks$ represent the clock ticks.

$$\delta = RTT - SRTT \tag{4.1}$$

$$SRTT = SRTT + 0.125 \times \delta \tag{4.2}$$

$$RTTVAR = RTTVAR + 0.25 \times (|\delta| - RTTVAR) \tag{4.3}$$

$$RTO = MAX(SRTT + 4 \times RTTVAR, 2 \times ticks) \tag{4.4}$$

The four problems identified by the authors of Eifel Retransmission Timer with this technique are: (i) using an absolute value of δ in Eq. (4.3) leads to an increase in RTO even when $RTT < SRTT$ i.e., δ is negative. (ii) the constants 0.125, 0.25 and 4 do not work when RTT sampling rate is more than 1 packet per congestion window ($cwnd$) of TCP sender. (iii) a bug in the re-initialization of RTO which makes the overall technique conservative, and (iv) high timer granularity used in BSD operating system. A detailed discussion of these problems is provided in (Ludwig and Sklower, 2000).

Accordingly, the authors (Ludwig and Sklower, 2000) propose Eifel Retransmission Timer technique for RTO estimation, which is shown by Eq. (4.5) to (4.11), where $flight$, $ssthresh$ and $cwnd$ indicate the amount of data sent by a TCP sender, slow start threshold and the congestion window of a TCP flow, respectively. The constants 0.125, 0.25 and 4 used in TCP-Lite are replaced by a parameter $GAIN$. Thus, Eq. (4.2), (4.3) and (4.4) are

replaced by Eq. (4.9), (4.10) and (4.11), respectively to solve the problem (ii) described earlier. Problem (i) of using an absolute value of δ is addressed by Eq. (4.8) and (4.10).

$$\delta = RTT - SRTT \quad (4.5)$$

$$flight = MAX(ssthresh, \frac{cwnd}{2}) \quad (4.6)$$

$$GAIN = \begin{cases} \frac{1}{flight}, & \text{if } RTT \text{ sampling rate} = 1 \\ \frac{2}{flight}, & \text{if } RTT \text{ sampling rate} = 0.5 \\ \frac{1}{3}, & \text{if 1 sample is obtained per } RTT \end{cases} \quad (4.7)$$

$$\overline{GAIN} = \begin{cases} GAIN, & \text{if } (\delta - RTTVAR) \geq 0 \\ GAIN^2, & \text{if } (\delta - RTTVAR) < 0 \end{cases} \quad (4.8)$$

$$SRTT = SRTT + (GAIN \times \delta) \quad (4.9)$$

$$RTTVAR = \begin{cases} RTTVAR + \overline{GAIN} \times (\delta - RTTVAR), & \text{if } \delta \geq 0 \\ RTTVAR, & \text{if } \delta < 0 \end{cases} \quad (4.10)$$

$$RTO = MAX((SRTT + \frac{RTTVAR}{GAIN}), RTT + (2 \times ticks)) \quad (4.11)$$

Among the three optimizations proposed in TCP-Eifel, this work proposes to leverage only one of the features in CoAP: the Eifel Retransmission Timer. Integrating the Eifel Algorithm in CoAP is not feasible due to the lack of support of timestamps in CoAP. Additionally, CoAP supports the functionality which is partially similar to the one suggested in Retransmit forever i.e., the application layer decides when to stop attempts to retransmit. Nonetheless, this is not the first proposal towards leveraging the benefits of TCP-Eifel in CoAP. In (Balandina et al., 2013), the authors propose a method to calculate RTO for CoAP Simple Congestion Control/Advanced (CoCoA) (Bormann et al., 2020) by combining ideas from the Eifel Retransmission Timer, Eifel Response Algorithm (Ludwig and Gurtov, 2005) and wireless rate control techniques. However, the approach proposed in this work significantly differs from the one proposed in (Balandina et al., 2013) and follows a simplistic implementation in CoAP.

4.3 CoAP-Eifel

4.3.1 Design

Initially, a feasibility study to integrate all the three components of TCP-Eifel with CoAP was carried out. The Eifel Algorithm could be used with the Eifel Retransmission Timer to estimate an increasingly optimistic RTO that adapts to the measured fraction of spurious timeouts. This approach relies on the TCP timestamp option field to resolve retransmission ambiguity. To replicate this mechanism in CoAP, a timestamp option is required, but this introduces additional protocol overheads as CoAP does not have a timestamp option field like TCP. The Retransmit Forever feature aims to keep the connection alive by continuously retransmitting the last outstanding segment until the application layer decides to close down the connection. However, this is an overhead for devices that are constrained by memory and power. Moreover, CoAP partially supports this functionality by attempting to retransmit atleast 4 times before marking the transmission as a failure. Hence, only the Eifel Retransmission Timer is chosen to be integrated with CoAP.

4.3.2 RTO estimation in CoAP using Eifel Retransmission Timer

Eifel Retransmission Timer requires modifications before it could be integrated with CoAP because it was designed to work alongside TCP. Two major changes are required: (i) CoAP does not measure RTT, so a new RTT measurement technique is required for CoAP (ii) CoAP does not maintain *ssthresh* and *cwnd* parameters, and as a consequence, the *flight* parameter used in Eq. (4.6) and (4.7). Depending on the number of packets in flight and the RTT sampling rate, (Ludwig and Sklower, 2000) suggests different values for GAIN. CoAP limits the number of outstanding requests per destination to 1 for CON. It follows a simple Stop-and-Wait mechanism, hence the number of unacknowledged messages is always restricted to 1, and 1 RTT sample is obtained per RTT. Thus, the third case shown in Eq. (4.7) holds true for CoAP because it keeps only one outstanding packet in the network. The following sections describe the details of RTT measurement and modified RTO estimation technique for CoAP.

A RTT Calculation

For RTT measurements in CoAP, the Karn/Patridge algorithm (Karn and Partridge, 1987) has been used i.e., RTT is not measured for retransmitted packets. Since the timestamp option is not supported in the CoAP header, the sender cannot distinguish

whether the ACK received is for the original packet or the retransmitted packet. The sender stores the transmission time of a CON message. Upon receiving an ACK, RTT is measured using the current time and the stored time of transmission of the corresponding CON message. This latest RTT sample is used to calculate RTO for subsequent packets exchanged between a sender and receiver. Initially, the RTO is set to $2 \times \text{ticks}$.

B Estimating RTO

The proposed RTO estimation technique in CoAP by using the modified Eifel Retransmission Timer is described by Eq. (4.12) to (4.17). The main difference is that Eq. (4.7) has been reduced to Eq. (4.13) for CoAP.

$$\delta = RTT - SRTT \quad (4.12)$$

$$GAIN = \frac{1}{3}, \text{ because 1 sample is obtained per RTT} \quad (4.13)$$

$$\overline{GAIN} = \begin{cases} GAIN, & \text{if } (\delta - RTTVAR) \geq 0 \\ GAIN^2, & \text{if } (\delta - RTTVAR) < 0 \end{cases} \quad (4.14)$$

$$SRTT = SRTT + (GAIN \times \delta) \quad (4.15)$$

$$RTTVAR = \begin{cases} RTTVAR + \overline{GAIN} \times (\delta - RTTVAR), & \text{if } \delta \geq 0 \\ RTTVAR, & \text{if } \delta < 0 \end{cases} \quad (4.16)$$

$$RTO = MAX\left(\left(SRTT + \frac{RTTVAR}{GAIN}\right), RTT + (2 \times \text{ticks})\right) \quad (4.17)$$

During the lifetime of a connection, $SRTT$ and $RTTVAR$ are updated for every sampled RTT. Until an RTT sample has been obtained between the sender and receiver, RTO is set to $(2 \times \text{ticks})$ (Betzler et al., 2015b). Once an RTT sample is obtained, RTO is computed using Eq. (4.17). In the event of a retransmission, the default BEB mechanism of CoAP is used.

4.3.3 Implementation Challenges

The implementation of CoAP in Contiki OS has been extended to implement CoAP-Eifel. We have added two new files to the `contiki/apps/er-coap` directory: `er-coap-eifel.c` and `er-`

coap-eifel.h. It also necessitates a 62 lines of code change in the *er-coap-transactions.c* and *er-coap-transactions.h* files. In addition, to ensure that CoAP-Eifel compiles successfully, we have included the *er-coap-eifel.c* file in *Makefile*.

Eifel Retransmission Timer uses a RTT calculation to calculate the RTO. Hence, the most challenging aspect of CoAP-Eifel implementation was to integrate RTT calculation to CoAP.

4.4 Evaluation

4.4.1 Network Configuration

The Eifel Retransmission Timer has been integrated in the CoAP implementation of Contiki OS. To test the proposed approach, experiments have been carried out in a testbed using in FIT/IoT-LAB. FIT/IoT-LAB is a real testbed that provides complete control over a large number of heterogeneous network nodes and allows to monitor network-related metrics such as throughput, delay, and others. For the purpose of validation, specially developed M3 node (M3Node, 2012) on the Saclay site has been used for experiments, which has STM32 (ARM Cortex M3) microcontroller and an Atmel AT86RF231 2.4 GHz transceiver. The M3 node has 64 kB of RAM and 512 kB of ROM. The basic erbium CoAP (*er-coap*) Contiki application has been used as the firmware for the nodes.

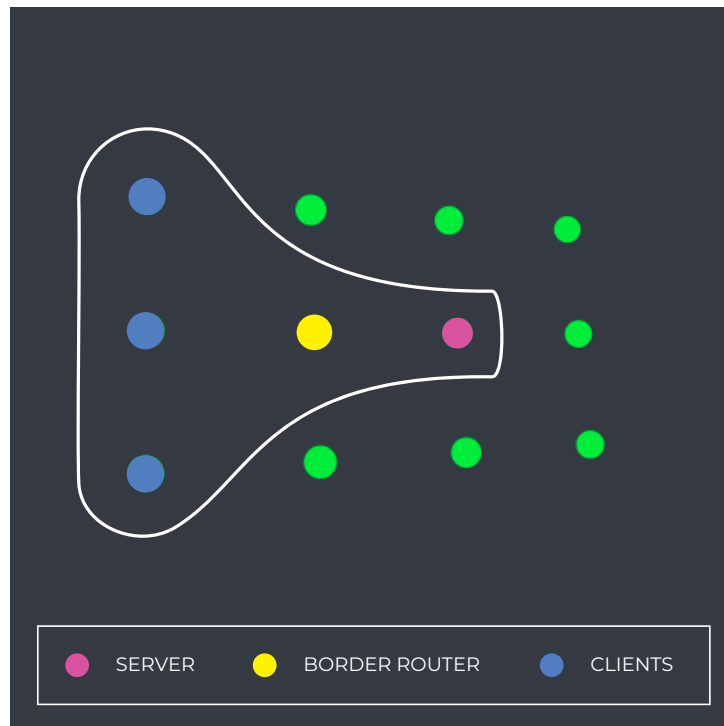


Figure 4.3: Half Dumbbell Topology (Saclay, 2012)

Dumbbell topology is the most commonly used topology for studying network congestion control. In this work, a half-dumbbell topology has been considered for validation that consists of three clients on one side connected to a server on the opposite side through a router, as shown in Figure 4.3. The link between the router and the server creates a bottleneck which provides an ideal condition to test the performance of the CoAP-Eifel. Every experiment has been repeated five times for a duration of 300s each, and the average of these runs has been considered for the analysis. Furthermore, the performance improvement at each client in the topology has been studied to understand the robustness of the proposed approach.

4.4.2 Results and Discussions

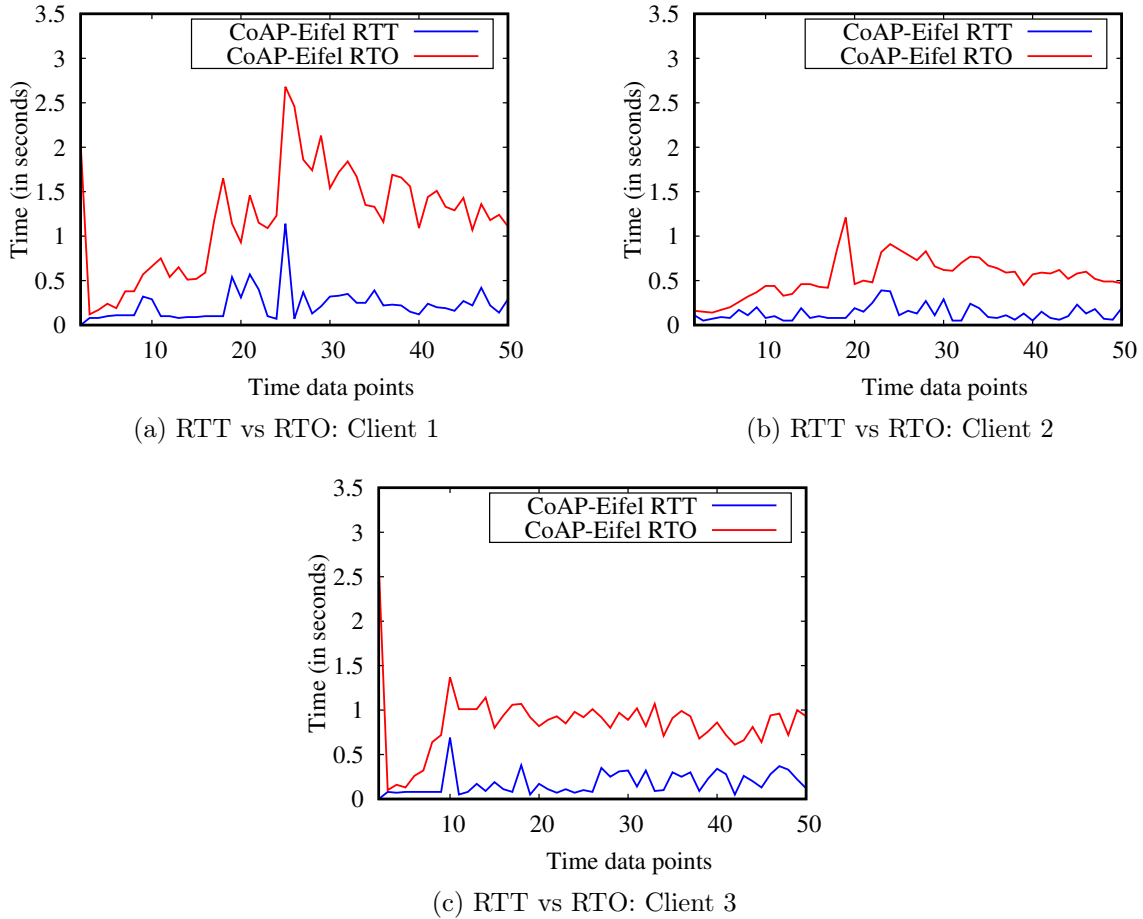


Figure 4.4: CoAP-Eifel: RTT vs RTO

CoAP-Eifel is compared with the default RTO mechanism in CoAP. We have used a testbed as described in Section 4.4.1, with all clients using either CoAP or CoAP-Eifel. For the ease of understanding, we refer to C (CoAP) and E (CoAP with Eifel Retransmission Timer) as qualifiers for a metric when referring to its definition or implementation. The

key metrics used to evaluate the performance are throughput, delay, and Packet Delivery Ratio (PDR). Most IoT applications are sensitive to large delays even though a smaller payload is being transmitted. Hence, a reduction in delay can improve the performance of most of the IoT applications.

CoAP-Eifel enables RTO in CoAP to respond to changes in RTT. Figure 4.4 shows the RTT and RTO obtained for each client in the network. For the purpose of validation, we have compared the RTO of *retransmitted packets* (obtained after BEB) with the last sampled RTT in CoAP-Eifel, because RTT of *retransmitted packets* is not considered in CoAP-Eifel. As described in Section B of 4.3.2, the initial value of RTO in CoAP-Eifel is set to $(2 \times \text{ticks})$. Hence, we see an initial peak in RTO in Figure 4.4. It can be confirmed from Figure 4.4a that RTO converges to RTT and backs off to a conservative level when a retransmission occurs. The peaks in the graphs can be attributed to retransmissions as the client falls back to BEB upon packet failure.

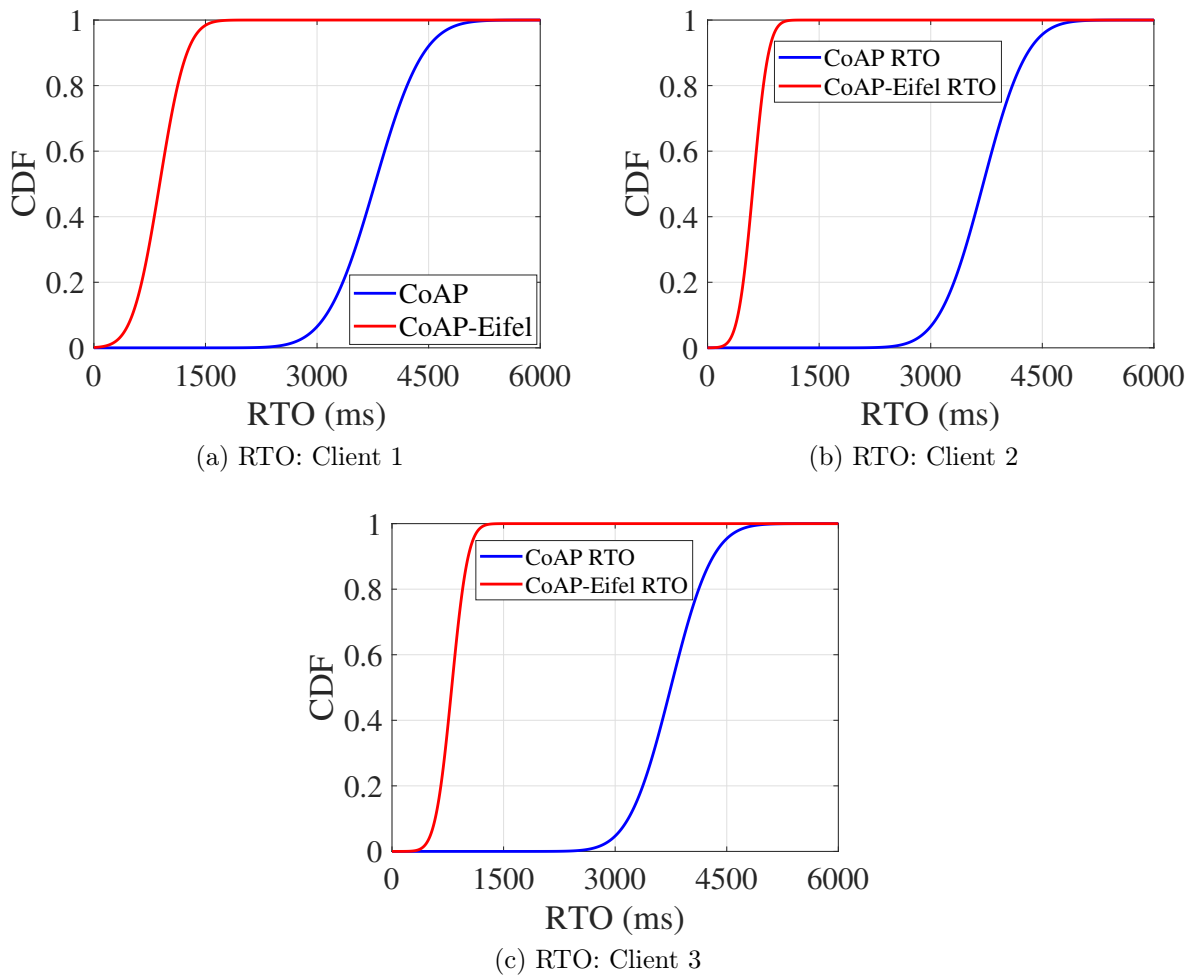


Figure 4.5: RTO: CoAP vs CoAP-Eifel

A Cumulative Distribution Function (CDF) is used to compare RTO_E with RTO_C and the results are shown in Figure 4.5. It is observed that the range of maximum and minimum values of RTO_C are greater than that of RTO_E . This is due to the conservative approach of RTO_C because CoAP does not maintain the network state information in terms of RTT, whereas RTO_E adapts to RTT_E .

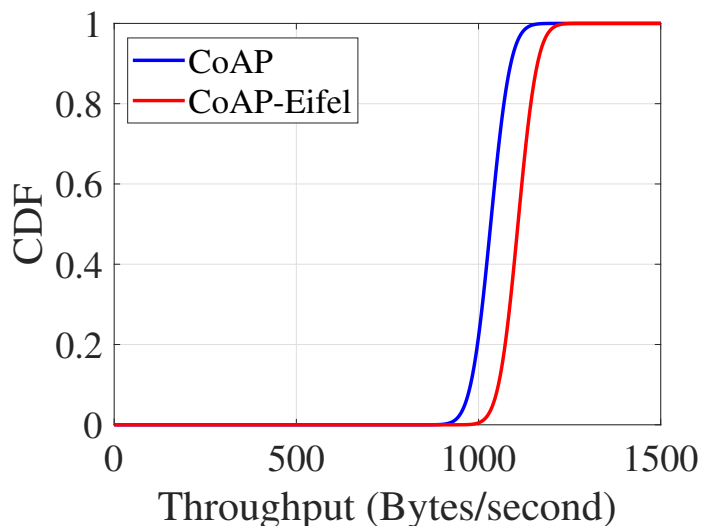


Figure 4.6: Throughput: CoAP vs CoAP-Eifel

$Throughput_E$ is larger than $Throughput_C$ as shown in Table 4.1. A Simple Moving Average (SMA) with an interval size of 50 has been used to compute the CDF of $Throughput_E$ and $Throughput_C$ as shown in Figure 4.6. Note that the distribution range of $Throughput_E$ contains larger values as compared to $Throughput_C$. This behaviour is due to the lower RTO_E values, as observed in Figure 4.5. Hence, with the use of the

Table 4.1: Throughput (in bytes/second)

Time	CoAP	CoAP-Eifel
50	1141.0	1271.4
100	846.7	1043.2
150	945.4	978.0
200	880.2	945.4
250	1075.8	1238.8
300	358.6	1238.8

Eifel Retransmission Timer, CoAP can realize a packet drop earlier. This reduces the idle time between retransmissions which results in higher throughput.

Table 4.2: Packet Transmission Time (in seconds)

Number of Packets	CoAP	CoAP-Eifel
500	64.58	65.73
1000	133.75	128.76
1500	204.56	193.65
2000	266.82	255.98

The packet Transmission Time (TT) for various number of packets is shown in Table 4.2. For the first 500 packets, TT_C is lesser than TT_E . This is due to different initial RTOs used in CoAP and CoAP-Eifel. Contiki’s implementation of CoAP sets the initial RTO value to $(3 \times \text{ticks})$ whereas, CoAP-Eifel sets it to $(2 \times \text{ticks})$. Hence, CoAP-Eifel is initially more aggressive in retransmitting packets, but over time as congestion builds up, the difference between TT_E and TT_C increases and stabilises to an average value of 11 seconds, with TT_E being lesser than TT_C . For convenient comparison of the distribution of delays across both the experiments, the CDF of Delay_E and Delay_C is also compared in Figure 4.7.

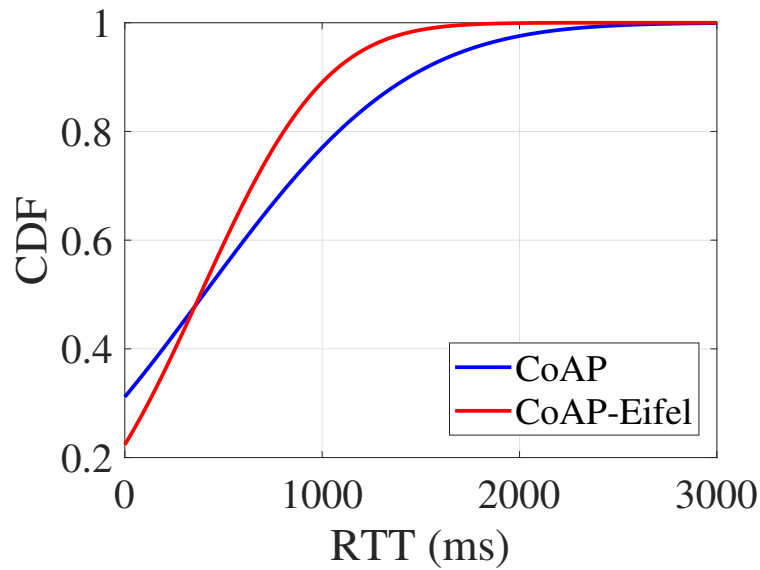


Figure 4.7: RTT: CoAP vs CoAP-Eifel

PDR is the ratio of the total number of packets received by the receiver to the total number of packets sent by the sender. Table 4.3 presents the PDR obtained with both algorithms, and it can be observed that both provide the same PDR.

Table 4.3: Packet Delivery Ratio (PDR)

CoAP	CoAP-Eifel
0.9961	0.9987

The source code of CoAP-Eifel along with the instructions to reproduce the results presented in this work is available at (SourceCode, 2020b).

4.5 Inferences

CoAP defines a conventional congestion control mechanism that is insensitive to network conditions. CoAP-Eifel is a simple enhancement to the default algorithm, and it includes an adaptive RTO calculation based on RTT measurements. The performance of CoAP-Eifel has been validated and evaluated by performing experiments in a real testbed using FIT/IoT-LAB. When compared to CoAP in a half dumbbell topology, CoAP-Eifel provides a better trade-off between delay and throughput, while providing a similar packet delivery ratio.

The implementation of CoAP-Eifel is different from that of GST-CoAP because it involves RTT measurements. However, RTT measurements in Eifel are lightweight as compared to those used in TCP. This ensures that CoAP-Eifel can be easily deployed in devices with low to moderate memory requirements. Example use cases involve IoT deployments for home automation (smart homes) and smart agriculture.

Chapter 5

Geometric Series based effective RTO estimation Technique for CoCoA

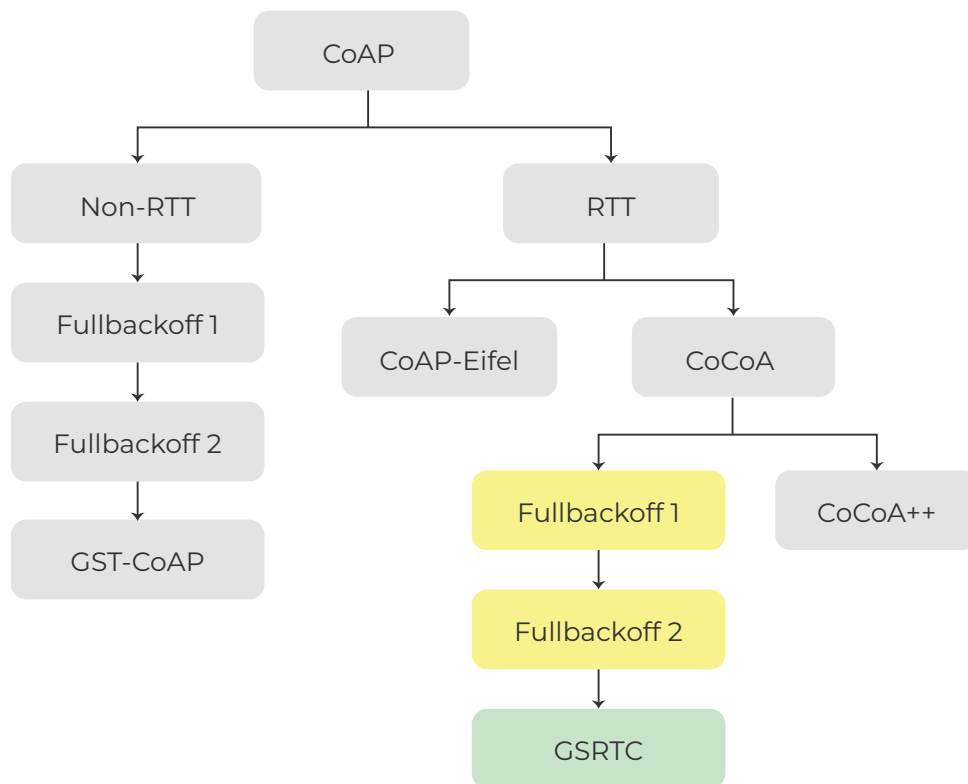


Figure 5.1: Contribution (highlighting (GSRTC))

This chapter discusses a Geometric Series based effective RTO estimation Technique for CoCoA (GSRTC), as shown in Figure 5.1. CoCoA uses *fixed* weights (0.5 for strong RTO and 0.25 for weak RTO) to estimate the RTO for the subsequent transmissions. GSRTC adapts the weight of *Strong* RTO by using a geometric series based on the number of *consecutive* successful transmissions. This helps CoCoA to adapt quickly when the network conditions are lossless.

5.1 Motivation

CoCoA is an enhanced congestion control mechanism over CoAP that adapts RTO based on RTT. CoCoA uses an EWMA to estimate the RTO for the next transmission. However, the weights used to estimate the RTO are *fixed*. These fixed weights lead to slow adaptation of RTO and affect the performance of the IoT applications. When the network conditions are lossless, CoCoA fails to adapt the network conditions due to a *fixed* value for *Strong* RTO estimation. Hence, an adaptive RTO estimation technique is needed when the network is *not congested*.

Two approaches proposed in (Järvinen et al., 2018), namely *Fullbackoff1* and *Fullbackoff2* aim to improve the RTO estimation technique in CoCoA. These techniques are simple improvements over CoCoA in order to make the RTO estimation technique more efficient and prudent in terms of congestion.

5.1.1 Fullbackoff1 Variant

The working of *Fullbackoff1* variant, as shown in Figure 5.2, is similar to the CoCoA algorithm. The key difference in the *Fullbackoff1* variant is that after the *retransmitted* packets are successfully acknowledged, the client retains the backed off RTO value for the subsequent transmissions, unlike CoCoA where Eq. (2.5) is used to calculate the RTO for the next transmission.

The following example demonstrates the difference between original CoCoA design and *Fullbackoff1* variant design. Assume that the RTO is 32s when the packet is *retransmitted*. Hence, the client would wait for 32s for an ACK, and if it does not receive an ACK within this time period, the packet will be *retransmitted again*. The RTO for this *second retransmission* would be 48s after applying the VBF (i.e., based on Eq. (2.6), 32s is multiplied by 1.5). Let's further assume that the client receives the ACK for the *second retransmitted* packet. In case of CoCoA, RTT_{weak} and RTO_{weak} would be calculated based on Eq. (2.4) and the RTO_{new} will be calculated for the next packet transmission based on Eq. (2.5). On the other hand, *Fullbackoff1* variant avoids using Eq. (2.5) and instead chooses 48s (the backed off RTO) directly as RTO_{new} (highlighted with the Green color in Figure 5.2) for the next packet transmission.

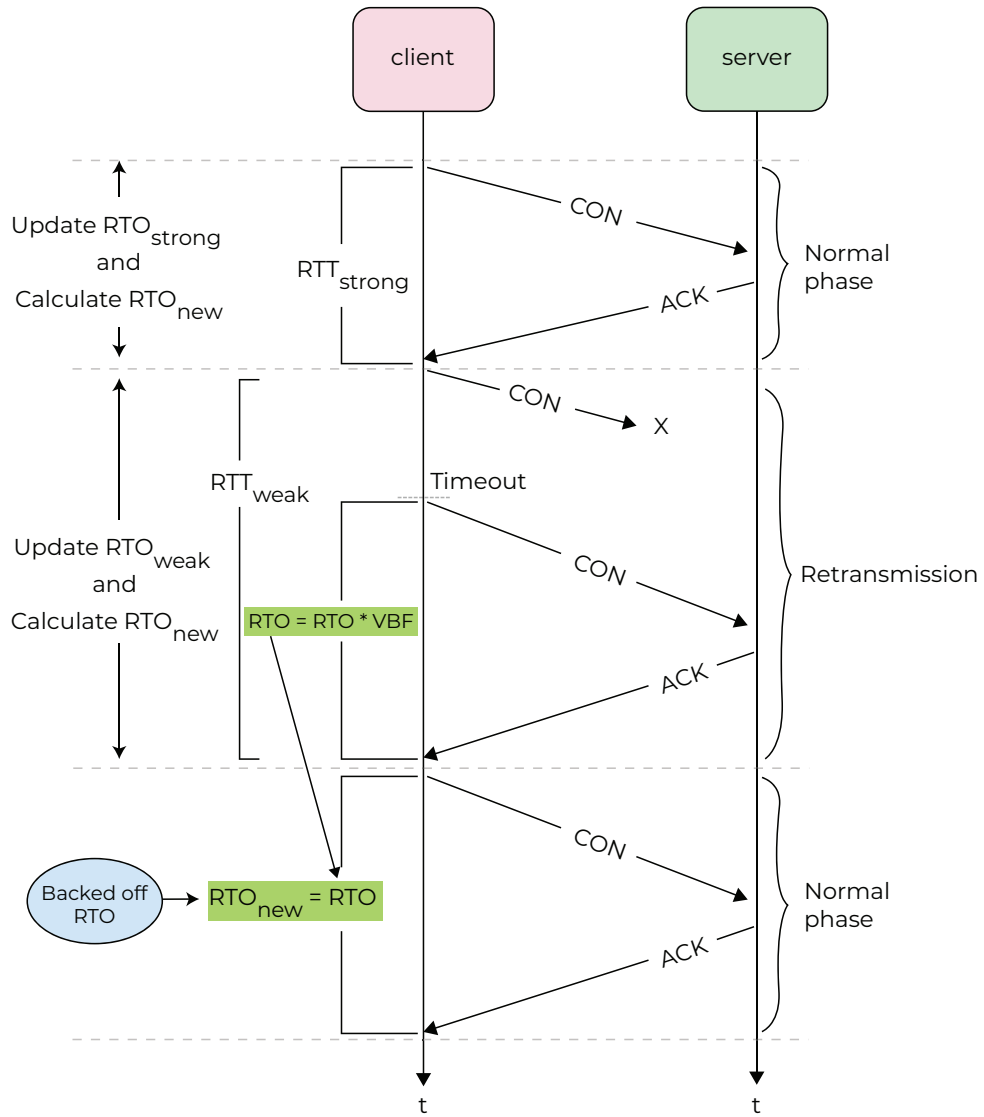


Figure 5.2: Working of *Fullbackoff1* variant of CoCoA

5.1.2 Fullbackoff2 Variant

The *Fullbackoff2* variant is a minor improvement over *Fullbackoff1* variant. Unlike CoCoA which uses Eq. (2.5) to calculate the RTO_{new} for the next transmission and *Fullbackoff1* variant which considers the backed off RTO directly as the RTO_{new} for the next transmission, *Fullbackoff2* variant takes a maximum of both these cases as shown in Figure 5.3.

It can be noted from *Fullbackoff1* and *Fullbackoff2* variants that both focus on improving the performance of CoCoA when the network is lossy and there are frequent *retransmissions* (i.e., RTO_{new} is mostly updated by RTO_{weak}). This work, instead, focuses on improving the performance of CoCoA when the network conditions are lossless i.e., the client observes *consecutive* successful transmissions. RTO_{new} in such cases would be mostly updated by RTO_{strong} .

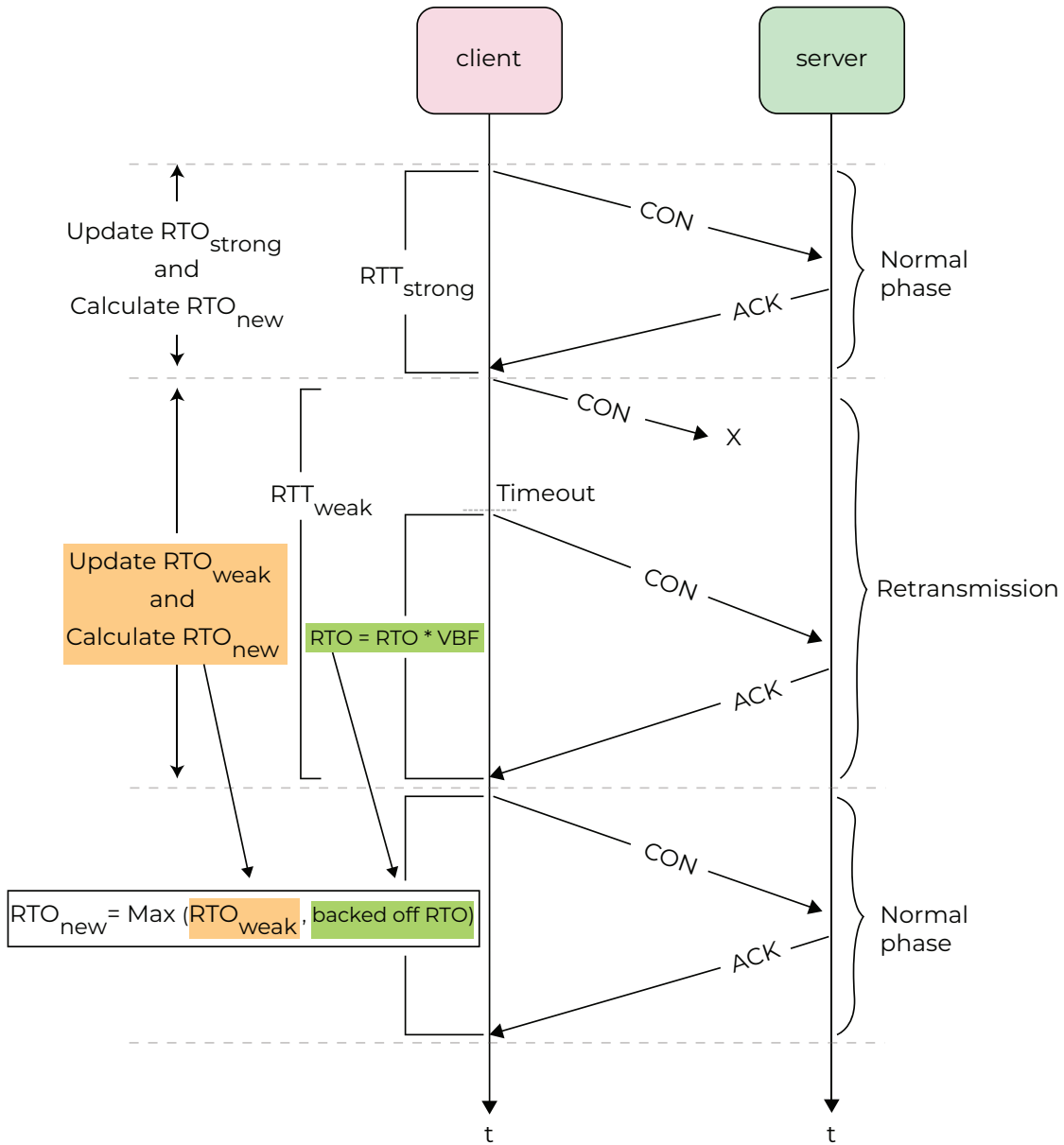


Figure 5.3: Working of *Fullbackoff2* variant of CoCoA

5.2 Geometric Series based RTO estimation Technique for CoCoA (GSRTC)

5.2.1 Design

GSRTC is an enhancement over the *Fullbackoff2* variant. It is most effective when the network conditions are lossless. It improves the network performance by rapidly adapting the weight (W_{strong}) of RTO_{strong} when the network is *not congested* and packet transmissions are getting cleared *consecutively without retransmissions*. GSRTC uses a geometric series to adapt the weight (W_{strong}) applied to RTO_{strong} while calculating RTO_{new} in Eq. (2.5). It keeps a *count* of *consecutive successful* transmissions. Depending on this *count*, W_{strong} is adapted as shown in Eq. (5.1).

$$weight_{next} = weight_{prev} + r^{count}, \quad count \neq 0 \quad (5.1)$$

where $weight_{next}$ is used as W_{strong} to estimate RTO_{new} in Eq. (2.5), $weight_{prev}$ denotes the weight obtained from the previous sample, r is the constant ratio between terms in a geometric series and $count$ indicates the number of *consecutive* successful transmissions. The main goal of GSRTC is to improve the network efficiency in terms of FCT, number of retransmissions and throughput.

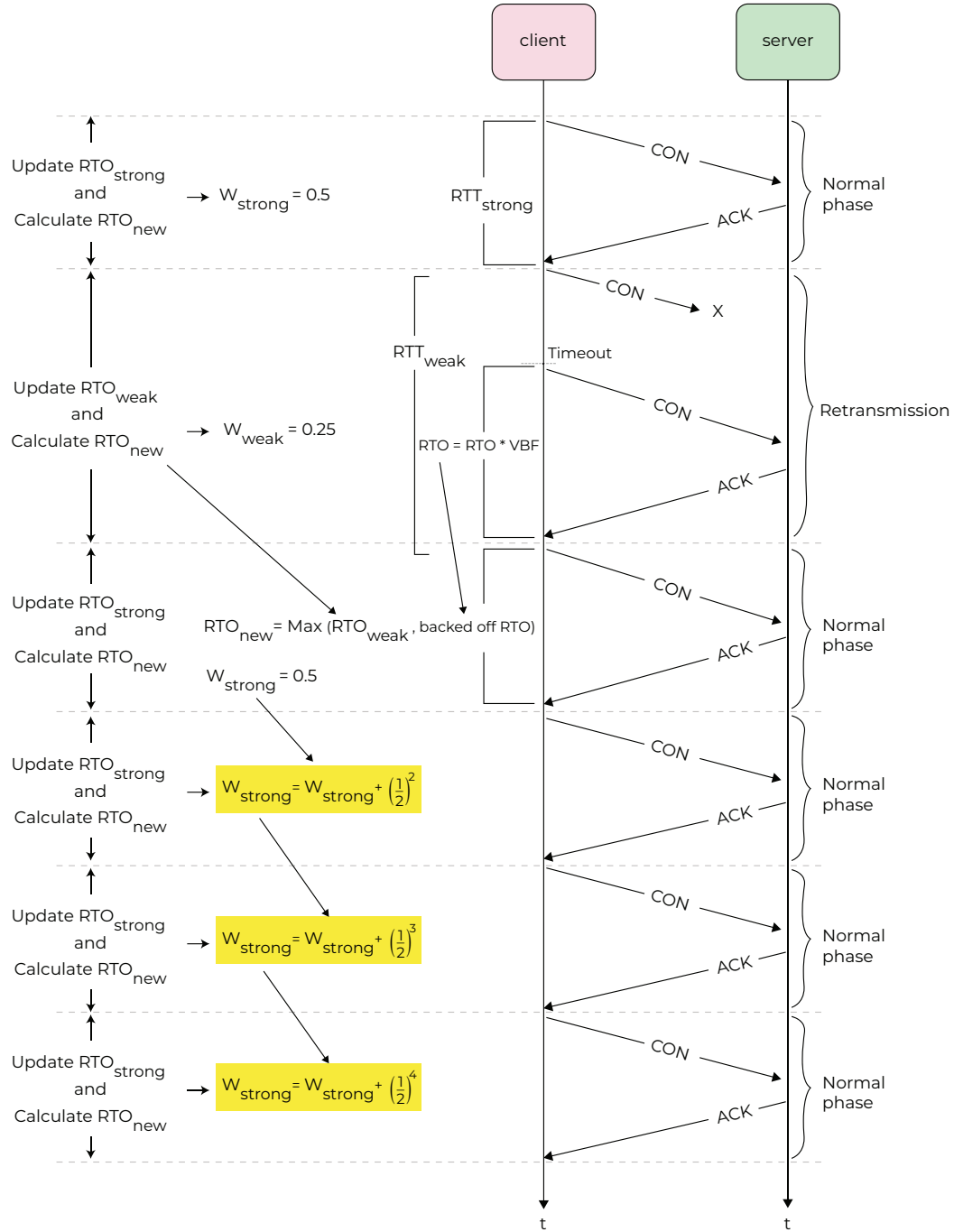


Figure 5.4: Working of GSRTC

Algorithm 2 shows the working and Figure 5.4 depicts an example of GSRTC. After every *consecutive* successful transmission, the weight associated with RTO_{Strong} is updated by adding a r^{count} to the previous value of the weight (Line 4-6 in Algorithm 2 and the part highlighted in yellow in Figure 5.4). GSRTC falls back to the default CoCoA algorithm when the network is lossy and RTO_{Weak} is to be calculated (Line 10-11 in Algorithm 2).

Algorithm 2: GSRTC

```

Initialization: count = 0,  $r = \frac{1}{2}$ ,  $weight_{prev} = 0$ 
1 On arrival of every ACK
2 Check whether the ACK belongs to original packet transmission or retransmission
3 if ACK belongs to an original packet transmission then
4   Calculate  $RTT_{Strong}$ ,  $RTTVAR_{Strong}$ ,  $RTO_{Strong}$  as shown in Eq. (2.4)
   // count represents the number of consecutive ACKs received
5   count ++
   // Calculate weight based on GSRTC
6    $weight_{new} = weight_{prev} + r^{count}$ 
7    $weight_{prev} = weight_{new}$ 
8 else
   // Reset count and  $weight_{prev}$ 
9   count =  $weight_{prev} = 0$ 
10  Calculate  $RTT_{Weak}$ ,  $RTTVAR_{Weak}$  and  $RTO_{Weak}$  as shown in Eq. (2.4)
   // Use the weight of Weak estimator
11   $weight_{new} = 0.25$ 
12 end
   // Calculate  $RTO_{new}$  based on Eq. (2.5)
13  $RTO_{new} = weight_{new} \times RTO_{current} + (1 - weight_{new}) \times RTO_{previous}$ 

```

5.2.2 Parameter Settings

A Choice of r

IoT devices are typically connected to lossy wireless channels and do not experience a large number of *consecutive successful* packet transmissions often. Hence, to ensure that GSRTC rapidly increases W_{strong} , r is chosen as $\frac{1}{2}$ (i.e., it follows a binary increase pattern). Figure 5.5 shows the rise of W_{strong} when the number of *consecutive successful* transmissions increases. For values lower than $\frac{1}{2}$ (e.g., $\frac{1}{3}$, $\frac{1}{4}$), we see that despite a large

number of *consecutive successful* transmissions, the rise of W_{strong} is not sufficient to enhance the performance of CoCoA when the network conditions are lossless.

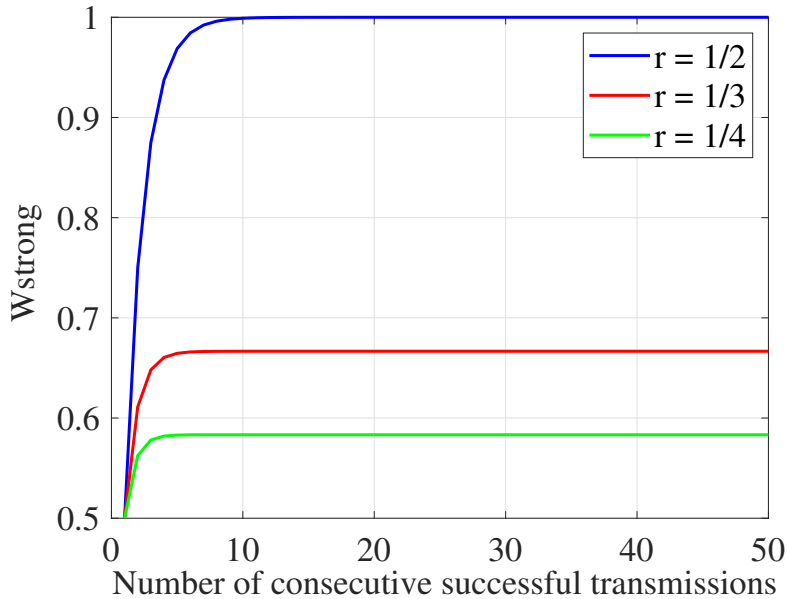


Figure 5.5: Comparison of W_{strong} with different values of r

B Lower and Upper Bound of W_{strong}

GSRTC uses 0.5 and 0.99 as the lower and upper bound for the W_{strong} , respectively. We chose 0.5 as the lower bound because it is the recommended default value defined in the Internet draft of CoCoA (Bormann et al., 2020). The upper bound is chosen to 0.99 so that the lossless network conditions can be leveraged to a maximum, while ensuring that the previous values of RTO are not totally discarded.

5.2.3 Implementation Challenges

CoCoA’s implementation in Contiki OS¹ has been used to implement *Fullbackoff* variants of CoCoA and GSRTC. It requires 34 lines of code change in the CoCoA implementation. *er-coap-transaction.c* and *cocoa.c* files in *contiki/apps/er-coap* directory have been modified to implement *Fullbackoff* variants of CoCoA and GSRTC.

5.3 Evaluation

This section describes the experimental setup and the performance metrics that have been used to compare the performance of the GSRTC with CoCoA and its existing *Fullbackoff*

¹Thanks to the authors of CoCoA for sharing their code (Betzler et al., 2015b).

variants.

5.3.1 Experimental Setup

The experiments have been performed on the Cooja simulator in Contiki OS (version 3.0) and using a real testbed at FIT/IoT-LAB. Cooja is a network simulator and one of its most notable features is that it supports emulation of off-the-shelf real sensor node hardware. A compiled binary image file is to be uploaded to the simulated nodes, wherein the compiled code is executed during simulation with the emulated model of the selected node type. FIT/IoT-LAB is an open source platform that provides a comprehensive scalable framework for users to test their applications by allowing them to perform experiments with a variety of nodes.

The authors of *Fullbackoff* variants (Järvinen et al., 2018) integrated CoCoA in *libcoap* (Bergmann, 2012), an open source C-based implementation of CoAP, and subsequently implemented the *Fullbackoff* variants on top of it. However, we first integrated CoCoA in Contiki based Cooja simulator, subsequently implemented the *Fullbackoff* variants in the Contiki OS and lastly implemented GSRTC.

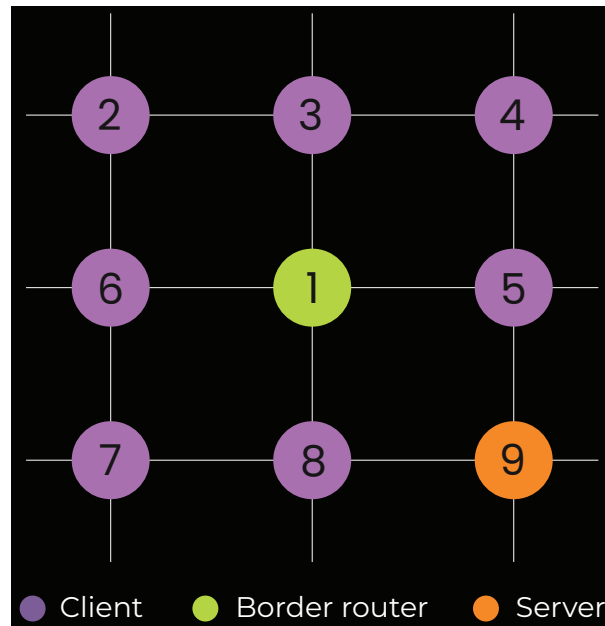


Figure 5.6: Grid Topology - GSRTC

The parameters defined in Table 5.1, along with their respective values, have been used to perform the experiments. T-Mote Sky (AdvantivSystemsServices, 2014) mote is used for the Border Router, and Zolerita z1 (Zolertia, 2014) mote is used for Clients and Server for experiments performed using the Cooja simulator. In Cooja, the transmission

Table 5.1: Experimental Parameters and Configuration

Parameters	Value
Operating System	Contiki-3.0
Simulator	Cooja
Testbed	FIT/IoT-LAB
Radio Medium	Unit Disk Graph Medium - Distance Loss (UDGM)
Mote Type	T-Mote Sky, Zolerita Z1 (Cooja) and M3 open node (FIT/IoT-LAB)
Topology	Grid (3 x 3)
Node Transmission Range	10 m
Node Interference Range	20 m
Application	Erbium-CoAP (er-coap)
Packet Sending Interval	100 ms
Duration	600 seconds

and interference range are set to 10m and 20m, respectively. We have used M3 motes (M3Node, 2012) for performing experiments in FIT/IoT-LAB. It includes an STM32 microcontroller that is based on ARM Cortex M3 and has a built-in radio with four different sensors. We conducted all our experiments at the Saclay site of FIT/IoT-LAB. Figure 5.7 shows the deployment of nodes on the Saclay site.

We have considered a grid network topology of 9 nodes (3×3) for the experiments in Cooja and FIT/IoT-LAB, as shown in Figure 5.6. Grid is a standard network topology used to evaluate and validate the congestion control mechanisms in CoAP (Ancillotti and Bruno, 2017; Betzler et al., 2015b, 2014). Node 1 is configured to be a border router, Node 9 to be a server and Node 2-8 to be the clients. The Unit Disk Graph Medium-Distance Loss (UDGM) model in Cooja is used for radio transmissions containing circular transmission and interference areas.



Figure 5.7: Deployment of nodes in Saclay site (Saclay, 2012)

We have used the Erbium rest (er-rest) application for both client and server. Every client generates CoAP messages and sends them to the server periodically at an interval of 100ms. The CoAP client sends a request through the border router to retrieve data from the server. Each experiment is run for 600 seconds and has been repeated five times. An average of five runs has been considered for evaluation.

5.3.2 Performance Metrics

In order to determine the efficacy of GSRTC, we have considered the following performance metrics:

1. **Flow Completion Time (FCT):** We have measured the FCT as the time taken by a client to successfully exchange 50 messages.
2. **Total number of retransmissions:** In this metric, we have considered the total number of packets retransmitted by all clients.
3. **Throughput:** Throughput has been measured as the total number of bytes delivered per second in a 50ms time interval for every client.

5.3.3 Results and Discussion

The results obtained from the experiments performed on Cooja and FIT/IoT-LAB are discussed in this section. The results obtained for GSRTC are compared with those obtained for CoCoA and the *Fullbackoff* variants.

A Flow Completion Time (FCT)

A wide range of IoT applications, such as healthcare, industrial and power systems are critical and require rapid response from IoT devices. It is therefore desirable to complete the transmissions as soon as possible. This makes FCT as one of the most critical parameters for evaluating the effectiveness of GSRTC. We have measured the FCT for every client as the difference between the time the first packet sent to the ACK received for the fiftieth packet. The performance of GSRTC with CoCoA and *Fullbackoff* variants in terms of FCT is shown in Figure 5.8; the results obtained from the Cooja Simulator and FIT/IoT-LAB are shown in Figures 5.8a and 5.8b, respectively.

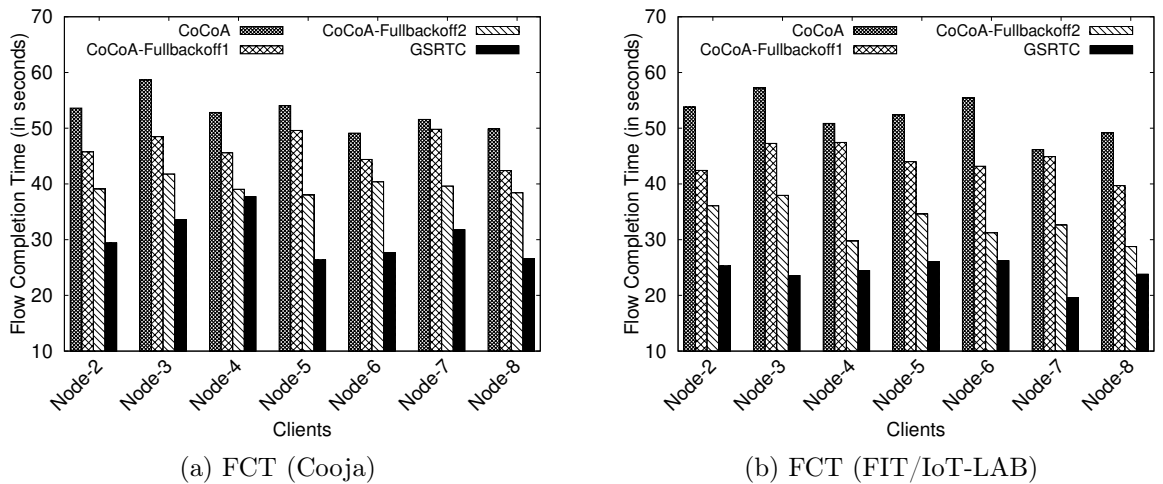


Figure 5.8: Comparison of FCT in Cooja and FIT/IoT-LAB

The average FCT of CoCoA, *Fullbackoff1* and *Fullbackoff2* is 53, 46 and 39 seconds in Cooja, respectively. The average FCT of GSRTC is 30 seconds in Cooja, which is 42.34%, 33.93% and 22.90% lower than CoCoA, *Fullbackoff1* and *Fullbackoff2*, respectively. Similarly, the average FCT of CoCoA, *Fullbackoff1* and *Fullbackoff2* is 52, 44 and 33 seconds in FIT/IoT-LAB, respectively. The average FCT of GSRTC is 24 seconds in FIT/IoT-LAB, which is 53.71%, 45.09% and 26.24% lower than CoCoA, *Fullbackoff1* and *Fullbackoff2*, respectively. To summarize, CoCoA has the highest FCT and GSRTC has the least FCT among all the techniques.

The FCT of CoCoA is high because there are many packet retransmissions, as shown in Figure 5.9. Retransmissions significantly increase the FCT and affect the overall throughput as well (See Figures 5.10 and 5.11). If the number of retransmissions is high, RTO_{weak} (Eq. (2.4)) governs the value of RTO_{new} (Eq. (2.5)). Even when the network conditions

improve (i.e., packet retransmissions reduce), it takes quite some time for the RTO_{new} to adjust according to the actual network conditions because the value of W_{strong} is fixed. This affects the throughput of CoCoA application. *Fullbackoff* variants clearly show an improvement over CoCoA because the RTO estimation techniques adopted in both closely reflect the actual network conditions, specially when the network is lossy. GSRTC further improves the RTO estimation and brings it closer to actual network conditions in cases when the network is lossless. Since GSRTC is built on top of *Fullback* variants, it has the most effective RTO estimation for both network conditions.

B Total number of retransmissions

Figure 5.9 shows the performance of different congestion control techniques in terms of the total number of retransmissions. CoCoA has the highest number of retransmissions. Due to the retransmission ambiguity, CoCoA measures the RTT of the retransmitted packet from its first transmission to the received ACK, which renders a high RTT sample (RTT_{weak} in Eq. (2.4)) and accordingly influences the RTO_{weak} and RTO_{new} samples.

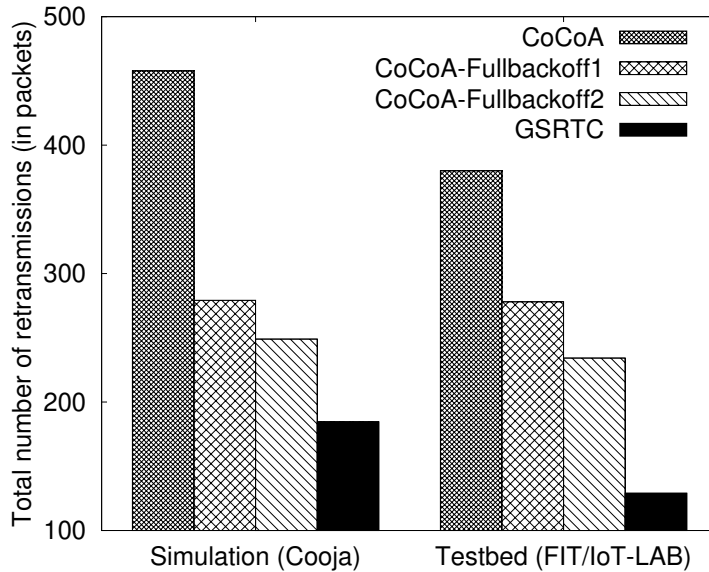


Figure 5.9: Total number of retransmissions: Cooja and FIT/IoT-LAB

The *Fullbackoff* variants have lesser retransmissions than CoCoA because they retain the RTO value of the previous transmission, thus giving the client ample time to wait for the ACK and reduce unnecessary retransmissions. GSRTC further improves the performance of *Fullbackoff* variants by adapting W_{strong} . GSRTC has a total 184 retransmissions in the Cooja simulator, an average of 26 retransmissions per client and 129 retransmissions in FIT/IoT-LAB, an average of 18 retransmissions per client. The total

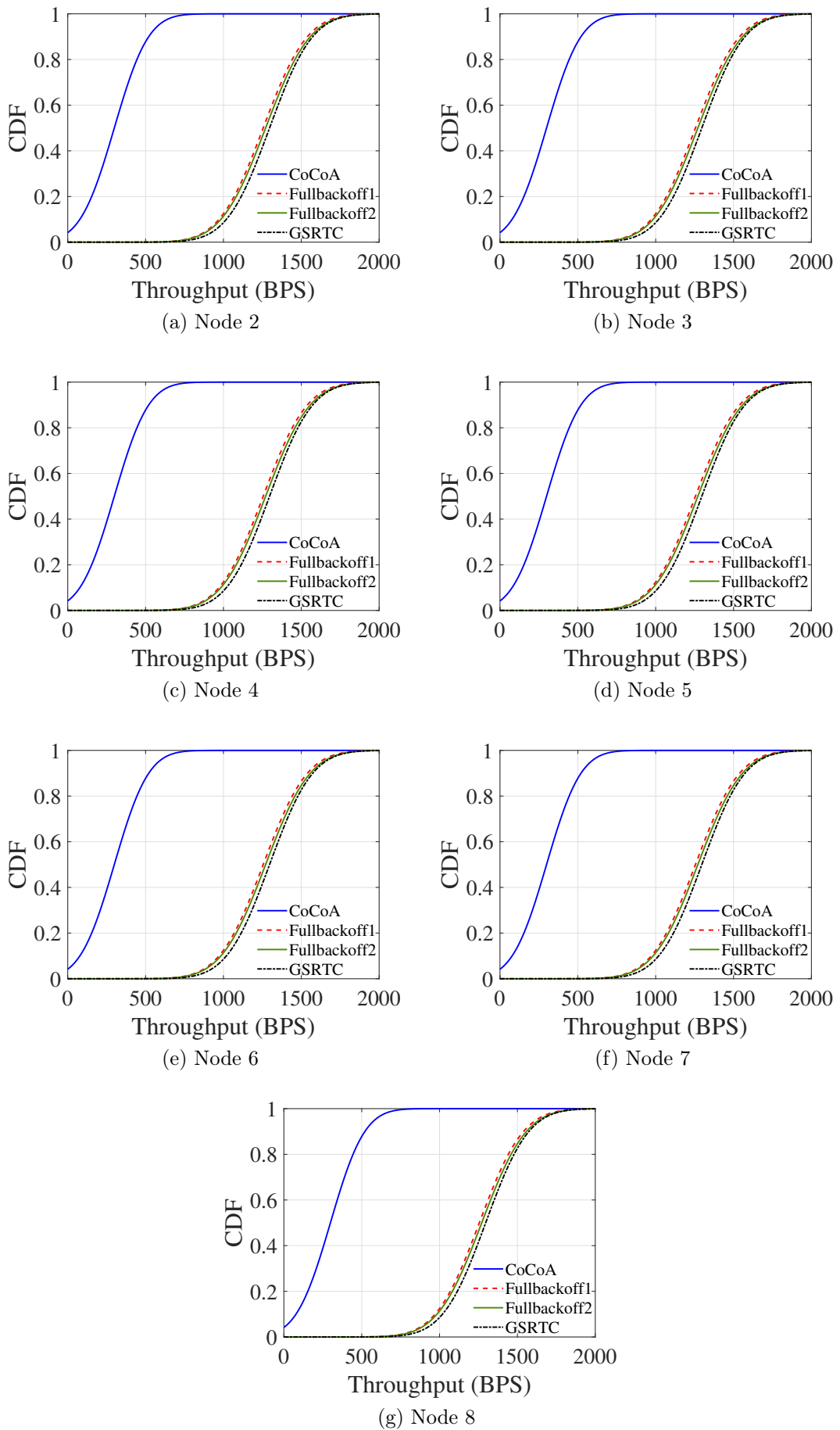


Figure 5.10: Throughput of CoCoA and Variants in Cooja

number of retransmissions in GSRTC is 59.68%, 33.88%, 25.86% and 66.05%, 53.60%, 44.92% lower than CoCoA, *Fullbackoff1* and *Fullbackoff2* respectively.

C Throughput

For every client, the throughput is measured as the total number of bytes sent per second over an interval of 50ms. Cumulative Distribution Function (CDF) has been used to compare the throughput of GSRTC with other techniques. We have measured the throughput by conducting experiments in the Cooja simulator and the FIT/IoT-LAB testbed as shown in Figures 5.10 and 5.11, respectively. This representation style is selected because the CDF plots provide a better distribution of the throughput samples among different values.

CoCoA has the least throughput in simulations and real testbed as shown in Figures 5.10 and 5.11, respectively. Due to a large number of retransmissions, the average RTO in case of CoCoA remains high (as shown in Table 5.2). It implies that the sender waits for a longer duration before transmitting new packets in the network, thus reducing the network throughput and increasing the FCT.

Table 5.2: Average RTO

	Simulation (Seconds)	Testbed (Seconds)
CoCoA	79.97	81.92
Fullbackoff1	61.40	73.53
Fullbackoff2	61.24	60.85
GSRTC	55.96	47.21

There is an interesting observation with respect to the results obtained from *Fullbackoff* variants. The throughput of *Fullbackoff2* variant is slightly lesser than that of *Fullbackoff1* variant in the plots obtained from FIT/IoT-LAB testbed. This behaviour can be attributed to the fact that, unlike CoCoA which uses Eq. (2.5) to calculate the RTO_{new} for the next transmission and *Fullbackoff1* variant which considers the backed off RTO directly as the RTO_{new} for the next transmission, *Fullbackoff2* variant considers RTO value which is a

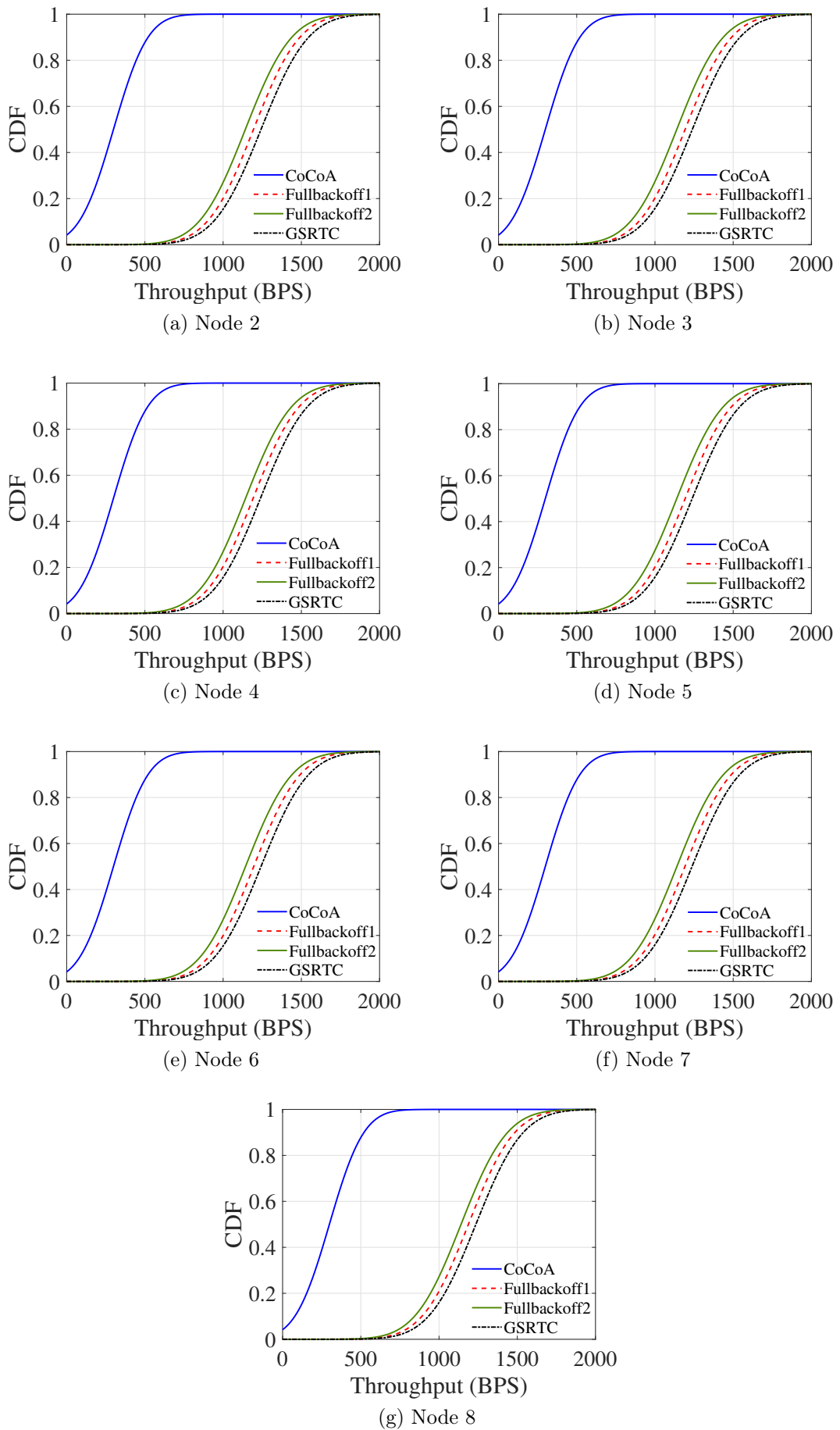


Figure 5.11: Throughput of CoCoA and Variants in FIT/IoT-LAB

maximum of both these cases. This is done with an aim to balance the trade-off between throughput and other parameters, such as FCT and number of retransmissions.

Nevertheless, the *Fullbackoff* variants significantly improve the network throughput as compared to CoCoA, and the performance of GSRTC is similar to those of *Fullbackoff* variants in terms of throughput.

5.4 Inferences

GSRTC uses geometric series to adapt the weight used in the *Strong* RTO estimator instead of using the *fixed* weight (0.5). GSRTC's performance is evaluated using the Cooja simulator and the FIT/IoT-LAB, and it is compared to the CoCoA and *Fullbackoff* variants. The results indicate that GSRTC reduces the Flow Completion Times, minimizes the number of retransmissions and provides higher network throughput.

GSRTC is a minor modification of the original CoCoA. Since CoCoA involves TCP-like RTO calculations, and hence RTT measurements, such algorithms are most suitable for devices with relatively more memory and computation capability. TCP-like RTO calculations consume several CPU cycles and require a bit more memory to store RTT variables. The same is true for all CoCoA based algorithms, including GSRTC. Nevertheless, the resources available in modern day IoT devices are sufficiently high to implement CoCoA based algorithms. Example use cases involve IoT deployments in smart traffic management.

Chapter 6

CoCoA++: Delay Gradient based Congestion Control for Internet of Things

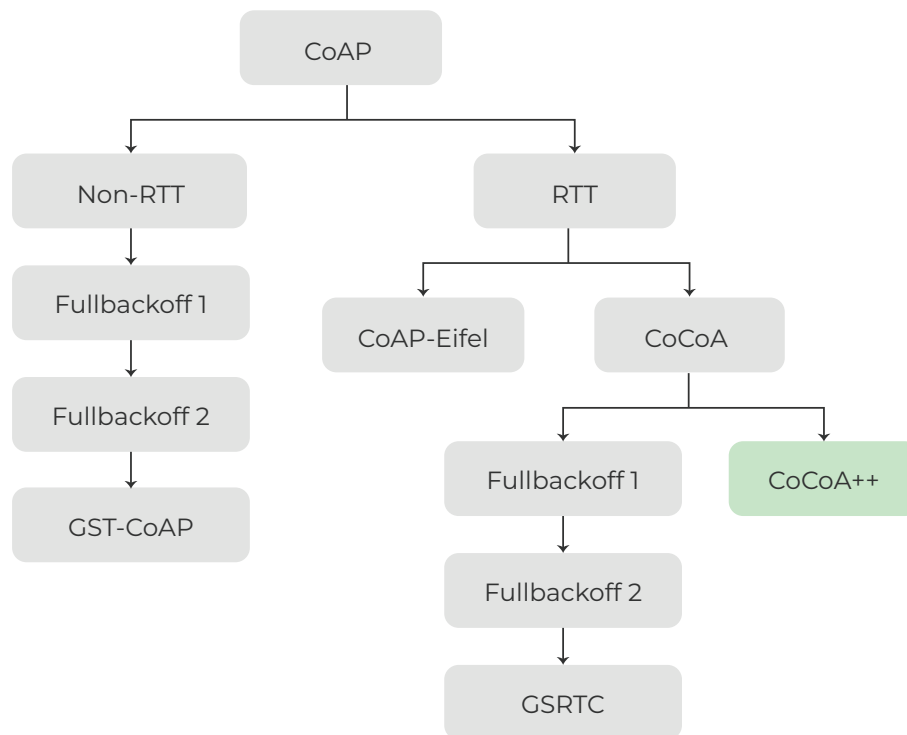


Figure 6.1: Contribution (highlighting (CoCoA++))

This chapter discusses a congestion control mechanism for CoAP, called CoCoA++ (as shown in Figure 6.1), that relies on CAIA Delay-Gradient (CDG) (Hayes and Armitage, 2011) to *predict* network congestion and a Probabilistic Backoff Factor (PBF) to *control network congestion*. CDG provides a better measure of network congestion by obtaining a gradient of RTT over time (Hayes and Armitage, 2011) and PBF helps to adjust the RTO based on the inferred congestion (Hayes and Armitage, 2011).

6.1 CoCoA++: Delay Gradient based Congestion Control for Internet of Things

6.1.1 Overview of CAIA Delay-Gradient (CDG)

CDG was proposed to work alongside TCP to provide better inferences of network congestion. However, CDG can be integrated with any other end to end protocol (e.g., CoAP in our case) since it is not tightly coupled with TCP. This section explains the working of CDG with TCP to control network congestion. In the subsequent section, we provide the details of integrating CDG with CoCoA to control network congestion in IoT.

TCP congestion control algorithms that use CDG try to predict congestion by tracking minimum and maximum RTT for a fixed period of time (5 seconds as recommended in (Hayes and Armitage, 2011)) for several intervals (denoted by n). The gradients (g) track the rate of change of RTT_{max} and RTT_{min} values to determine if the network is congested. Eq. (6.1) shows the calculation of delay gradients:

$$\begin{aligned}g_{min,n} &= RTT_{min,n} - RTT_{min,n-1} \\g_{max,n} &= RTT_{max,n} - RTT_{max,n-1} \\ \bar{g}_n &= \sum_{i=n-a}^n \frac{g_i}{a}\end{aligned}\tag{6.1}$$

where, g is the delay gradient, $g_i = g_{min,i}$ for calculating $\bar{g}_{min,n}$ or $g_i = g_{max,i}$ when calculating $\bar{g}_{max,n}$, \bar{g}_n represents the average of gradients and a is the number of samples in the moving average window.

The rate of change of RTT is used to calculate a probability of backoff. The probability approaches 1 as the rate increases. The probability is then compared with a random value to determine whether to reduce the congestion window of TCP (by a factor of 0.3 as suggested in (Hayes and Armitage, 2011)). When it decides not to reduce congestion window, it is increased by 1 as per Eq. (6.2).

$$P[backoff] = 1 - \exp^{-\left(\frac{\bar{g}_n}{G}\right)}\tag{6.2}$$

where, $G > 0$ is a scaling parameter used to control the probability of backoff due to delay gradient indications (Jonassen, 2015).

The decision to reduce the TCP congestion window is taken not only based on the

probability but also by differentiating between packet loss caused due to congestion and packet loss caused because of other factors. In general, congestion occurs when the queue becomes full. CDG estimates this based on the RTT_{min} and RTT_{max} values. If RTT_{max} reaches its largest possible value and stops increasing and if RTT_{min} is still increasing, it means that the queue is full and any packet loss during this time is because of congestion. Hence, CDG updates the congestion window (W), as shown in Eq. (6.3) (Hayes and Armitage, 2011):

$$W_n = \begin{cases} W_{n-1} * 0.7, & P[\text{backoff}] > X \text{ and } g_{min} > 0 \\ W_{n-1} + 1, & \text{otherwise} \end{cases} \quad (6.3)$$

where, W_n indicates the congestion window during the n^{th} interval and X is a uniformly distributed random number, $X = [0, 1]$.

The advantage of using CDG over other congestion indicators is that it is able to handle shared systems that use loss-based congestion control. When a CDG connection shares bandwidth with a loss based system, in the event of congestion building in the network, CDG will back off. This gives the loss based system an opportunity to continuously pump packets and take up the entire bandwidth. To deal with this, the CDG algorithm keeps track of how the RTT values change after the back off. The correct functioning of CDG demands that a decision to slow down must result in decreasing RTT values (g_{min} and g_{max} become negative). If this is not observed, it is understood that CDG is competing with a loss based system and the algorithm stops backoff. CDG also stores the previous value of maximum congestion window which can be easily restored in case of a packet loss.

6.1.2 Design

CoCoA++ is a congestion control mechanism for CoAP that integrates CDG with CoCoA. The main goal of the proposed algorithm is to provide a better estimate of network congestion to end points by leveraging the advantages of CDG. However, integrating CDG with CoCoA is a non-trivial task because the design of CDG is tailored to work alongside TCP. CDG provides information about network congestion which is then used by TCP to update the congestion window. The major concern is that the concept of congestion window does not exist in the CoCoA protocol, which makes it difficult to seamlessly integrate CDG with CoCoA.

This issue is tackled in CoCoA++ by using the information obtained from CDG to derive a better RTO estimate. CoCoA++ eliminates the need to maintain two RTO estimates i.e., strong and weak RTO estimates. Moreover, CoCoA++ does not update the RTO based on per packet RTT samples like CoCoA. Instead, it updates RTO when periodic information about the delay gradients g_{min} and g_{max} is received from CDG¹. As a consequence, CoCoA++ does not rely on VBF for backoff since VBF assumes that RTT is calculated for every packet. CoCoA++ replaces VBF by a Probabilistic Backoff Factor (PBF) which is calculated as shown in Eq. (6.4).

$$PBF = \begin{cases} 1.42, & P[\text{backoff}] > X \text{ and } g > 0 \\ 0.7, & \text{otherwise} \end{cases} \quad (6.4)$$

The probability of backoff obtained from CDG (Eq. (6.2)) is compared with a uniform random value X to ensure that the flows with small RTT and those with large RTT have a similar probability of backoff. The condition $g > 0$ ensures that the backoff is applied to RTO only when the rate of change of RTT is positive.

When congestion is detected in the network ($P[\text{backoff}] > X$ and $g > 0$), the RTO is increased by a factor of 1.42. RTO is reduced by a factor of 0.7 when no congestion is detected. The value 0.7 to reduce RTO is obtained by performing simulations with a range of values from $[0.5, 1.0]$ ². This range is considered with a goal to ensure that RTO value reduces gradually, and that this reduction does not lead to congestion in the network. It was observed that the best performance is obtained with 0.7. Accordingly, its reciprocal (i.e., 1.42) was selected to increase the value of RTO when the network is congested.

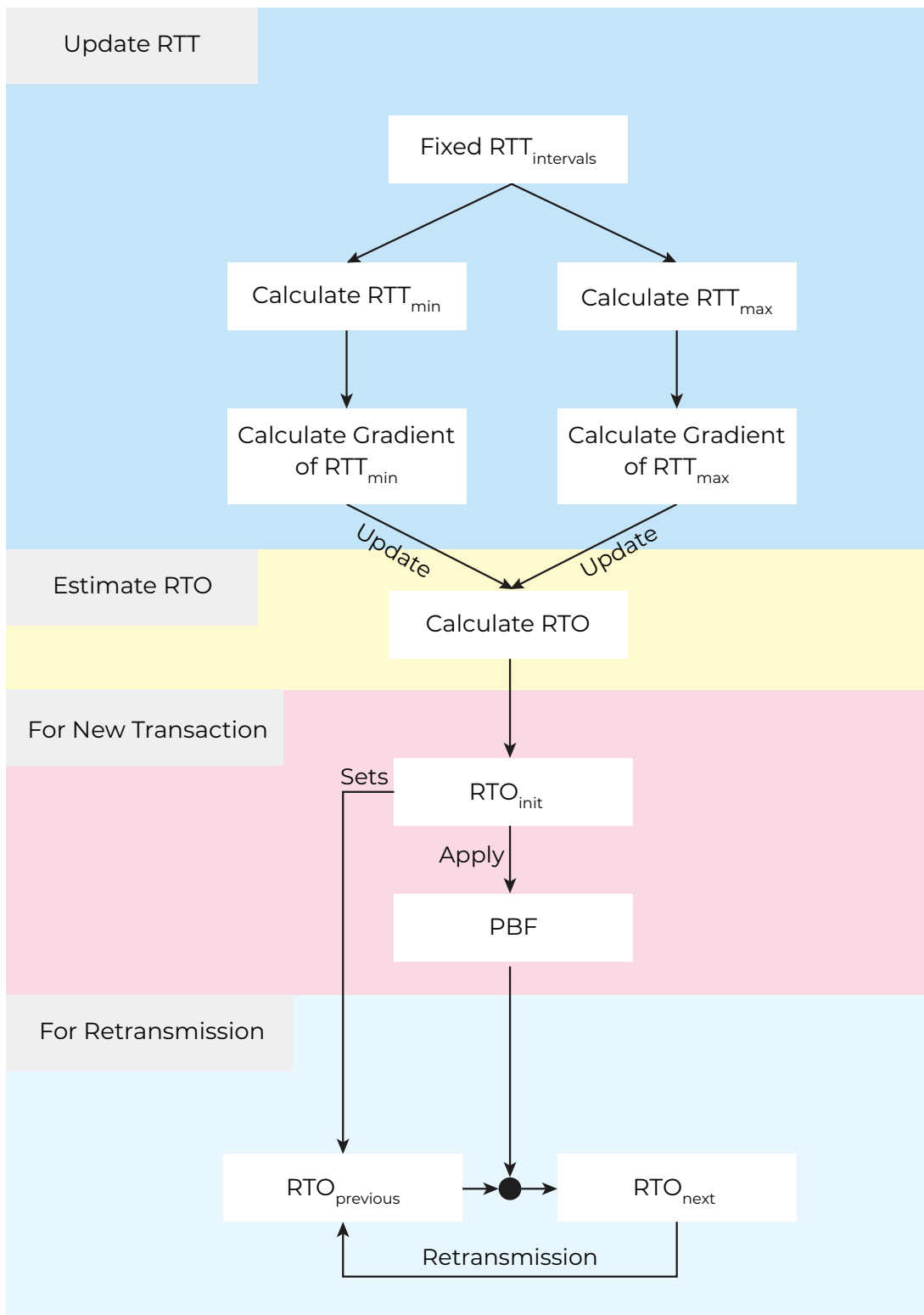
The overall working of CoCoA++ algorithm is summarized in Figure 6.2.

6.1.3 Implementation Challenges

We implemented CoCoA++ in Contiki-3.0, simulated it using the Cooja network simulator and tested its performance in a real testbed at FIT/IoT-LAB. Modifications were made to the existing CoCoA implementation to integrate CDG and PBF. All experiments use the default Contiki OS stack for implementation and evaluation. Our implementation

¹At most once every two RTT intervals as mentioned in (Hayes and Armitage, 2011). Although designed to work with TCP, this period is also suitable for CoCoA++ because RTO calculations in CoCoA++ are not significantly different than in TCP. Our results provided in (Results, 2019) confirm that this period works well with CoCoA++.

²Results obtained from simulations with these values are provided in (Results, 2019).



RTT - **R**ound **T**rip **T**ime, *RTO* - **R**etransmission **T**ime**O**ut,
RTT_{min} - Minimum RTT, *RTT_{max}* - Maximum RTT,
PBF - **P**robability **B**ackoff **F**actor

Figure 6.2: Working of CoCoA++ Algorithm

of CoCoA++ uses a lookup table for e^x .

6.2 Evaluation

In this section, we discuss the methodologies used to evaluate the performance of CoCoA++ and compare it with CoCoA. The evaluation is carried out in two ways: (i) by configuring static and mobile topologies in Cooja simulations, and (ii) by configuring a real testbed at FIT/IoT-LAB. In addition, we extensively evaluate both the algorithms with different traffic rates in Cooja and testbed. We track how the RTO values change with time, and measure the average number of transactions per second.

6.2.1 Simulation Setup

From the available mote types in Cooja, Zolertia Z1 motes (Zolertia, 2014) are used for clients and servers, with transmission and interference ranges set to 10m and 20m, respectively. Z1 motes have large amounts of ROM that enables us to code applications and implement congestion control mechanisms. T-mote (AdvanticSystemsServices, 2014) Sky motes are used to configure the border router because we use IPv6 Routing Protocol for Low-Power and lossy networks (RPL) in storing mode, and this requires larger RAM capacity which is provided by Sky motes. Table 6.1 shows the configuration of T-mote Sky and Zolertia Z1 motes in terms of RAM, ROM, Microcontroller Unit (MCU) and Radio transceiver. After the RPL setup, the clients periodically send messages targeted at the servers. Every scenario is run 30 times with a different random seed. The simulation results for all runs are averaged and plotted against the simulation time.

Table 6.1: Sensor mote configuration

	T-mote Sky	Z1
RAM	10 kB	8 kB
ROM	48 kB	92 kB
MCU	MSP430F1611	MSP430F2617
Radio	CC2420	-

Initially, we evaluate the performance of CoCoA++ in static and mobile topologies,

which is followed by evaluation with different traffic rates.

A Static and mobile topologies

1. **Static topologies:** The nodes have fixed positions and zero mobility throughout the simulation. Four different topologies are considered: grid, flower, dumbbell and chain. Grid, dumbbell and chain topologies are analogous to the ones used for evaluating CoCoA (Betzler et al., 2015b).

Grid topology: Grid topologies are popular in applications requiring long range and broad area coverage such as smart grid, industrial automation and building automation (Betzler et al., 2013, 2015b; Ancillotti and Bruno, 2017). They are designed in such a way that every node (a sensor or router node) has at least one other node within its range of transmission. Data packets are relayed across several nodes before they reach the server, and hence the network range is independent of the transmission range of the individual nodes. In this topology, the internal nodes of the grid, or in other words, nodes that are adjacent to four neighbours, act as the potential points of congestion. The grid topology we use is a 6 x 6 grid consisting of 2 servers, a border router and the clients, arranged in a grid where the nodes are spaced 10m from each other (See Figure 6.3).

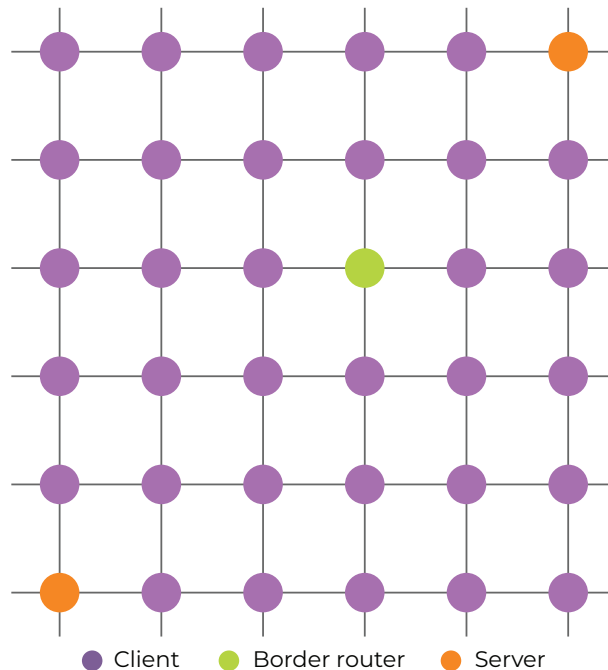


Figure 6.3: Grid Topology

Flower topology: Flower topologies like the one illustrated in Figure 6.4 are

used in applications such as smart greenhouse management systems, cellular network, satellite network and wide area network applications (Chan, 2008). These consist of a central processing system that controls environment parameters like water flow, temperature etc for the confined space based on the information it receives from sensors placed in the individual pots. All sensor information is relayed across the array of potted plants that are represented by nodes in the branches of the flower topology. The central server acts as a bottleneck, and hence is a potential point of congestion. The flower topology we use consists of 36 nodes with a single server, one border router and 34 clients as shown Figure 6.4. The communication from client to server (when not in range for direct communication) happens through the nodes arranged as petals.

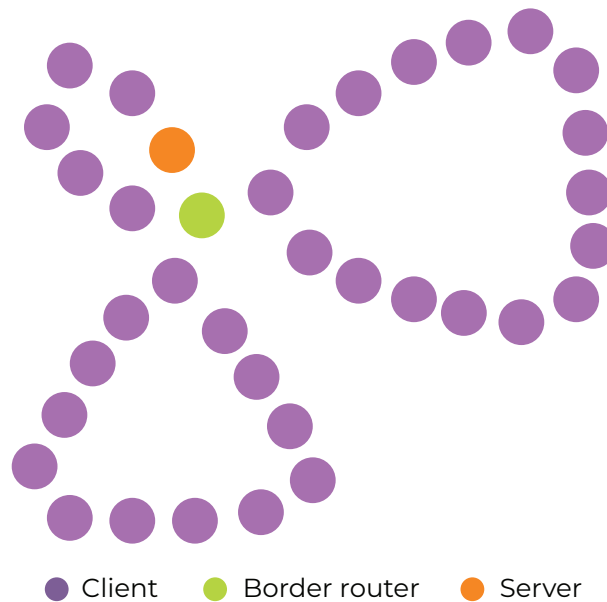


Figure 6.4: Flower Topology

Dumbbell topology: It is the most widely used topology for evaluation of network congestion algorithms (Betzler et al., 2013, 2015b). The link between the two halves of the dumbbell serves as the bottleneck. When the bandwidth offered by this link cannot accommodate all client transactions, the link needs to be shared in a fair manner which is ensured by the congestion control algorithm in place. The dumbbell topology we use consists of 19 client nodes, one server and a border router arranged in the shape of a dumbbell with the border router at the center as shown in Figure 6.5. The set of clients forming one half of the dumbbell on the right side are within the range of the server, whereas any communication between the clients in

the left half of the dumbbell and the server takes places through the border router.

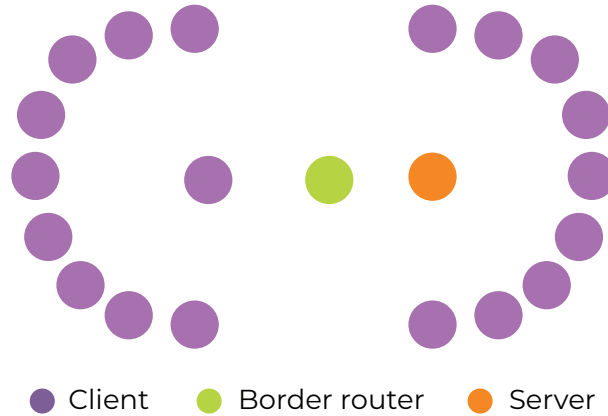


Figure 6.5: Dumbbell Topology

Chain topology: Networks using chain topology experience the least amount of congestion. The communication from one end to the other takes place across the chain as messages are relayed from a node to its immediate neighbour. The chain topology we use consists of 18 clients, a server and border router arranged in a line as shown in Figure 6.6.



Figure 6.6: Chain Topology

2. **Mobile topologies:** The nodes move during the simulation based on four different mobility models that exhibit the behaviour of mobile nodes in some IoT applications such as smart traffic management, self driving cars, home automation system with handheld devices and communication for security drones (Aschenbruck et al., 2010). Note that in all our mobility simulations we keep the border router and servers stationary and only the clients are given some non-zero velocity. We have used BonnMotion: a mobility scenario generation and analysis tool for generating a position file of a particular mobility model (Aschenbruck et al., 2010).

Random Waypoint Mobility Model (RWMM) (Camp et al., 2002): Random Waypoint is a mobility model where the nodes travel in a zig-zag manner. Initially the nodes remain stationary for some fixed amount of time during which they decide on a destination and a random speed within the feasible range. Once they reach their destinations, they again pause for some interval before repeating

the process. The model is incorporated into the simulation using the mobility plugin for Cooja where it is possible to specify a `position.dat` file that contains the position of every node at different points in time. These positions are generated using the Random Waypoint Mobility model (Ariyakhajorn et al., 2006).

Manhattan Grid Mobility Model (Martinez et al., 2008): Manhattan mobility model is used widely in Vehicular Ad-hoc Networks. It is typically used to simulate the behaviour of vehicles on a grid road topology and guide them to take the right path. The nodes in this model move along the horizontal and vertical grid lines. At each intersection the nodes turn right, left or continue straight with some probability.

Pursue Mobility Model (Camp et al., 2002): Pursue mobility model is designed to exhibit the behaviour of one node (pursuer) following another node (target). The target node moves across the simulation region based on Random Waypoint and velocity of the pursuer node is directed towards the target.

Gauss-Markov Mobility Model: Gauss-Markov mobility model correlates the velocity of mobile node with time (Ariyakhajorn et al., 2006). It is modelled as a Gauss-Markov stochastic process (Bai and Helmy, 2004) where the velocity of the node at some time slot t depends on its velocity in the previous time slot $t - 1$. The model is a temporally dependent model with a parameter α that determines the degree of dependence. For zero dependence, α is set to 0 (no memory or memoryless model) and for very strong dependence the parameter α is set to 1 (strong memory model).

B Varying traffic rates

The scenarios used for evaluation in this section are identical to the ones mentioned in the previous section for static and mobile topologies, except that we repeat the experiments with different traffic rates. In these scenarios, each sensor periodically sends measurement reports to the sink node. Different traffic loads are generated by varying the frequency of reporting the sensor measurements. The main goal of this scenario is to evaluate the robustness of CoCoA++ against different traffic rates.

The parameters used for simulations and their respective values are mentioned in Table 6.2. The values shown in Table 6.2 are same as the ones used for evaluating CoCoA in (Betzler et al., 2015b), except that we use mobility models like Random Way Point,

Gauss Markov, Manhattan Grid and Pursue to evaluate the performance of CoCoA++ in mobile IoT applications such as self-driving cars and drones. The parameter Radio medium indicates the propagation loss model used in the simulation. Radio duty cycling handles the sleep and wakes up cycle depending on the type of the RDC method used by the motes. Motes are configured without Radio Duty Cycling (NullRDC) (Contiki Tutorial, 2013), which means that the motes are always in listening mode and do not sleep.

Table 6.2: Simulation Parameters

Operating System	Contiki-3.0
Simulator	Cooja
Radio Medium	Unit Disk Graph Medium - Distance Loss (UDGM)
Radio Duty Cycling	NullRDC
Mote Type	T-mote Sky and Zolerita Z1
Number of Nodes	Flower - 36, Grid - 36 , Dumbbell - 21 and Chain - 20
Node transmission range	10 m
Node Interference range	20 m
Transmission/Reception ratio	50 %
Traffic Type	Periodic traffic from Client to Server, packet interval 0.4 sec
Simulation Duration	18 minutes for each simulation
Retransmission limit	4
Mobility Model	RandomWayPoint, Gauss-Markov, Pursue and ManhattanGrid

6.2.2 Simulation Results

We evaluate and compare the performance of CoCoA and CoCoA++ under the same simulation setup by looking at the variation in RTO values with simulation time. RTO measurements give us an idea of how much delay is encountered in transmission, which indirectly gives us an estimate of packet sending rates.

Each scenario is run 30 times with different seed values and the average RTO values are plotted against the simulation time (Figures 6.7 and 6.8). The plots represent the

variations in RTO values with simulation time for both CoCoA and CoCoA++. Each point on the graph represents the RTO value for a transaction that is initiated at the time represented by the corresponding time on the simulation time axis. This gives us an idea of the number of transactions that take place in the given simulation interval and the distribution of RTO values. As shown in Table 6.3, the maximum and average RTO values are calculated over all simulations for a particular scenario by averaging the values obtained in the all 30 simulations.

Table 6.3: Results with Static Topologies

Scenario (Static topologies)	CoCoA			CoCoA++		
	Max	Avg	Average	Max	Avg	Average
	RTO (sec)	RTO (sec)	Transactions per second	RTO (sec)	RTO (sec)	Transactions per second
Grid	24.268	8.5789	8.7388	11.466	4.6999	21.7555
Flower	24.345	6.9081	10.2268	11.499	4.6174	19.9148
Dumbbell	37.717	6.2547	5.9268	12.968	3.5933	10.8953
Chain	24.288	8.4768	3.7666	11.480	4.6419	9.3231

From the plots and Tables 6.3 and 6.4; it is observed that in static scenarios, the RTO values obtained for CoCoA++ do not exceed 13s, whereas the RTO values for CoCoA go up to 38s, which is almost three times the maximum value obtained for CoCoA++. Similarly, with all the mobility models discussed above, the RTO values obtained for CoCoA++ do not exceed 20s, whereas the RTO values for CoCoA go up to 35s. On average, the RTO measurements obtained for CoCoA++ are significantly lower than CoCoA, thereby reducing the transmission delays. In IoT networks where messages comprise mostly of small payloads, a reduction in transmission delays can enhance performance by a large margin. Lower transmission delays also increase the number of transactions per second. As shown in Table 6.3, CoCoA++ records more than double the transaction rate as compared to CoCoA in static scenarios. Similarly, with all the mobility models discussed above, the transaction rates of CoCoA++ increase by nearly threefolds (Table 6.4). One of the many aspects that contribute to low RTO values and high packet trans-

mission rates in CoCoA++ is the use of PBF as shown in Eq. (6.4). The highest backoff factor in PBF (1.42) is smaller than the lowest backoff factor in VBF (1.5).

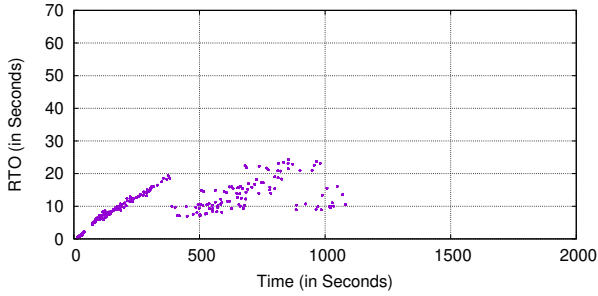
Table 6.4: Results with Mobile Topologies

Scenario (Mobile topologies)	CoCoA			CoCoA++		
	Max	Avg	Average	Max	Avg	Average
	RTO	RTO	Transactions	RTO	RTO	Transactions
	(sec)	(sec)	per second	(sec)	(sec)	per second
Random Waypoint	35.498	8.5824	8.9685	19.209	4.4809	22.4953
Gauss-Markov	27.992	8.4312	8.8675	11.506	4.5830	22.0685
Manhattan-Grid	26.084	8.4062	8.9722	18.438	4.4525	22.9666
Pursue	30.803	8.5731	7.8296	13.584	4.5615	22.0712

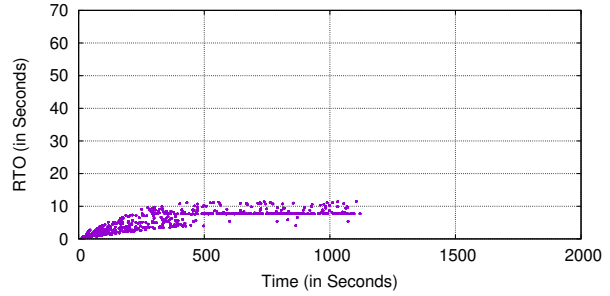
The above observations can be justified based on the fact that delay gradients give a better estimate of congestion that is persistent in a network. By taking the rate of change of RTT rather than instantaneous values, we focus only on change in the queuing delay components and ignore other delay components like processing time at the endpoints that otherwise contribute to the RTT as well. Any congestion in the network will result in an increase in queuing delay which is captured by the delay gradient and hence, acts as a sign of true congestion.

Figure 6.7 presents the variations in RTO values with both CoCoA and CoCoA++ for static topologies. A point in the graph represents the RTO values with respective to the simulation time. RTO values with CoCoA exhibit a linear incremental pattern followed by a sharp decrement, and this cycle repeats throughout the simulation. The noisy RTT signals in CoCoA lead to large variations in RTO values, making it difficult for the algorithm to converge to a stable operating point. On the other hand, CDG filters out the undesired delay signals and provides clear information about network congestion. Hence, the RTO values with CoCoA++, after the initial ramp up period, remain stable throughout the simulation. In summary, the RTO values with CoCoA++ are more stable,

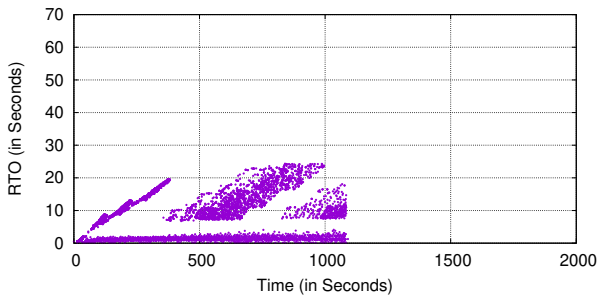
uniformly distributed and lower than those with CoCoA because the effect of the rate of change of RTT gives good control over the RTO values in CoCoA++. Due to these characteristics, CoCoA++ successfully completes more number of transactions in a unit time than CoCoA.



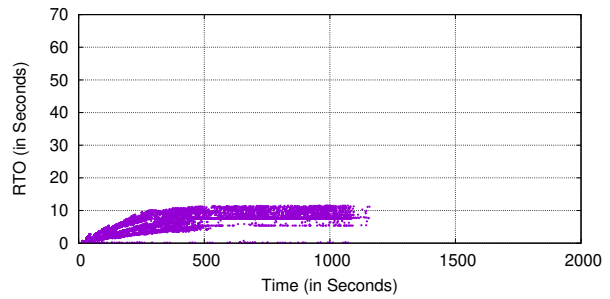
(a) Grid topology: CoCoA



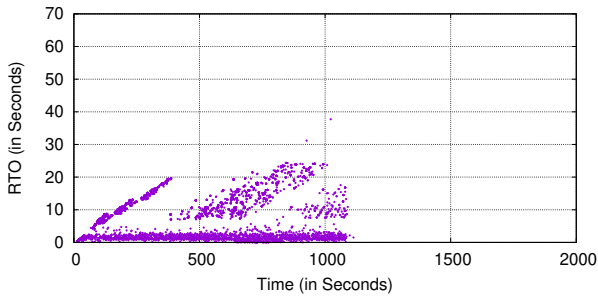
(b) Grid topology: CoCoA++



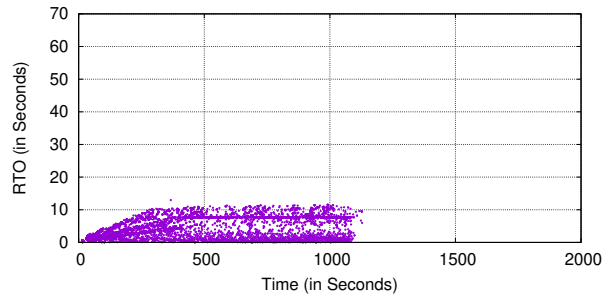
(c) Flower topology: CoCoA



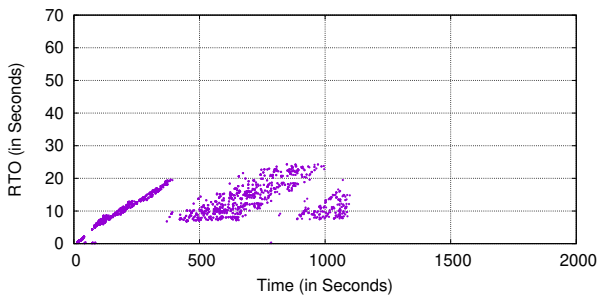
(d) Flower topology: CoCoA++



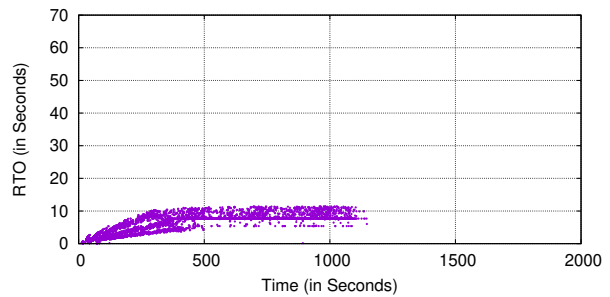
(e) Dumbbell topology: CoCoA



(f) Dumbbell topology: CoCoA++



(g) Chain topology: CoCoA



(h) Chain topology: CoCoA++

Figure 6.7: Comparison of CoCoA and CoCoA++ in Static Topologies

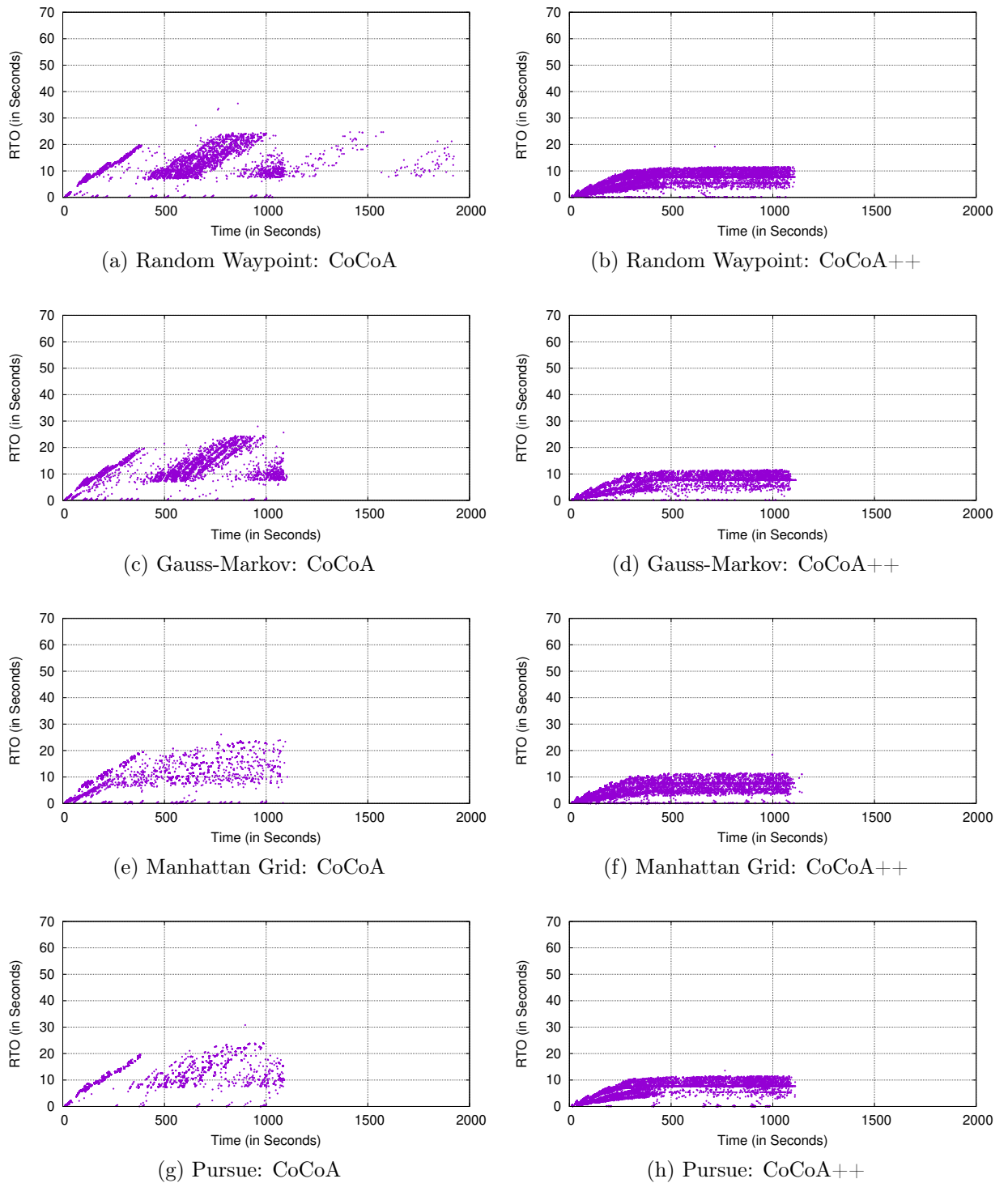
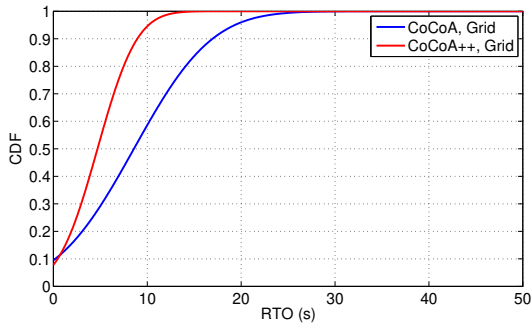
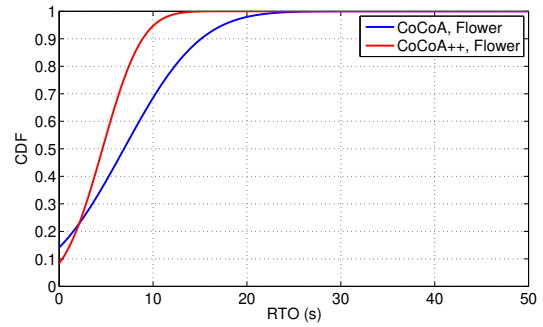


Figure 6.8: Comparison of CoCoA and CoCoA++ in Mobile Topologies

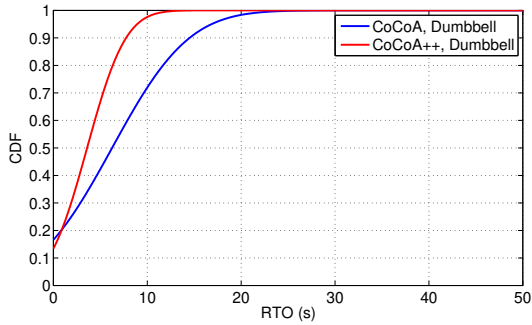
Figure 6.8 presents the variations in RTO values with both CoCoA and CoCoA++ for mobile topologies. Node mobility makes it further difficult to infer congestion from RTT measurements because mobility leads to variations in the received signal strength. Due to this, the overall RTT varies, and subsequently affects the RTO estimation. This is evident when we compare the RTO values with CoCoA in static topologies and mobile



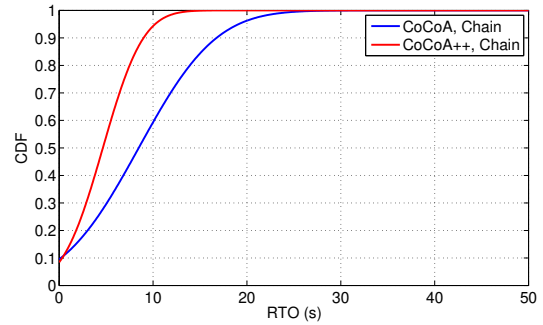
(a) Grid topology: CDF



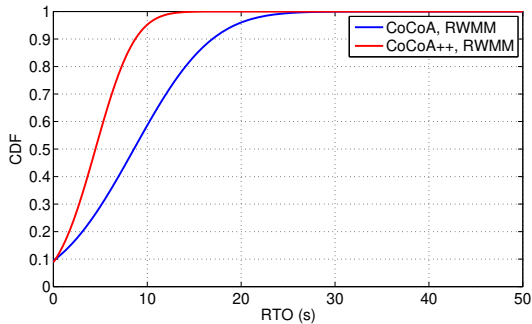
(b) Flower topology: CDF



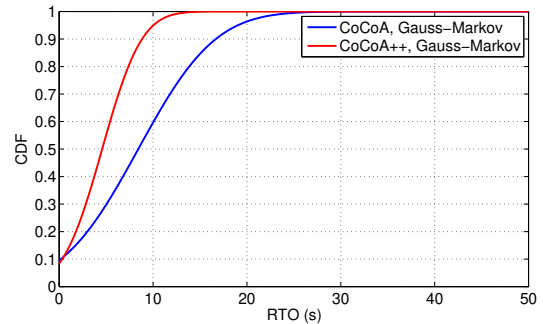
(c) Dumbbell topology: CDF



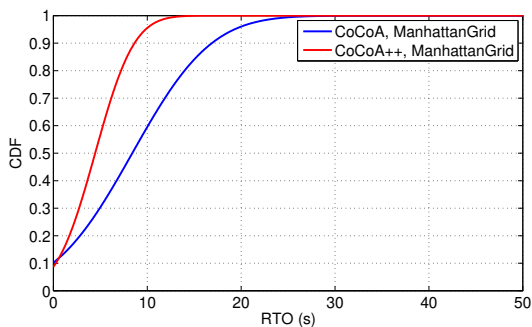
(d) Chain topology: CDF



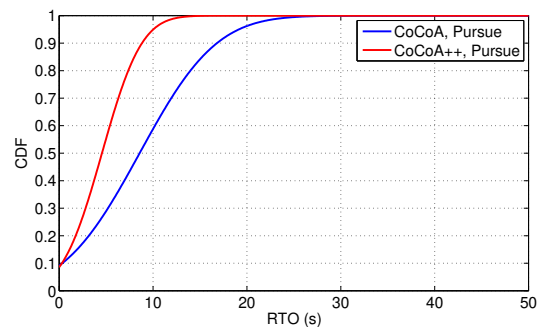
(e) Random Waypoint: CDF



(f) Gauss-Markov: CDF



(g) Manhattan Grid: CDF



(h) Pursue: CDF

Figure 6.9: Comparison of CoCoA and CoCoA++ in terms of CDF

topologies, and observe that the RTO values with CoCoA are largely scattered in mobile environments. However, this is not the case with CoCoA++ because it does not directly

depend on RTT measurements for RTO estimation, but instead uses CDG. Since CDG helps to detect network congestion in a reliable manner, the RTO values with CoCoA++ are not affected in mobile environments and continue to exhibit a stable behaviour.

We reproduce all the plots of CoCoA and CoCoA++ shown in Figure 6.7 and Figure 6.8 in terms of Cumulative Distribution Function (CDF) of the RTO values in Figure 6.9. We choose this style of representing the results because the CDF plots provide a clear understanding about the effectiveness of CoCoA++ in terms of keeping the RTO values lower than CoCoA.

A Varying traffic rates

This section discusses the simulation results obtained with different traffic loads. The traffic load in the network is varied from 0.5 Kbps to 2.5 Kbps, with an increment of 0.5 Kbps. We evaluate the performance of CoCoA and CoCoA++ in terms of the average RTO, the total number of packets transmitted and average packet sending rate.

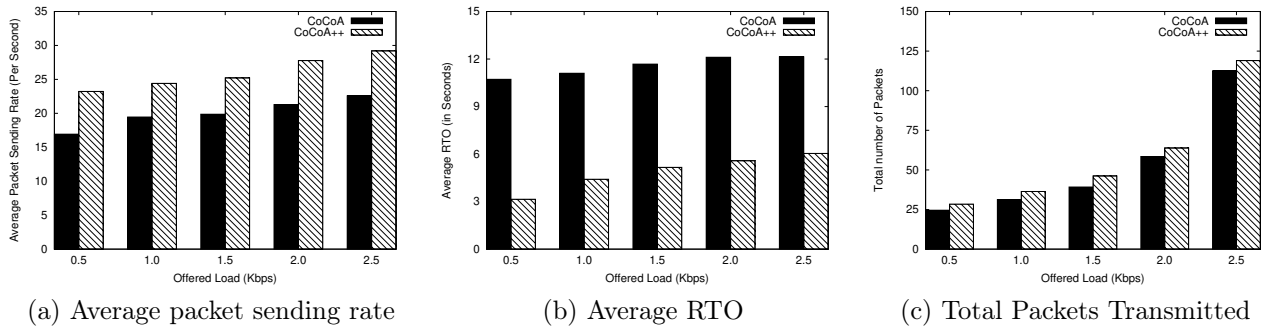


Figure 6.10: Grid Topology - CoCoA vs CoCoA++

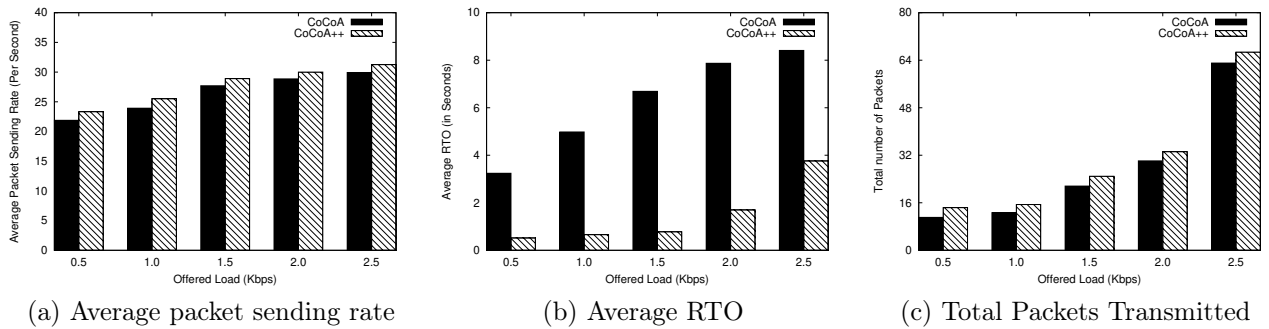


Figure 6.11: Flower Topology - CoCoA vs CoCoA++

1. **Static topologies:** Figures (6.10 - 6.13) show the results obtained in static topologies for CoCoA and CoCoA++ with different traffic rates. Increasing the traffic

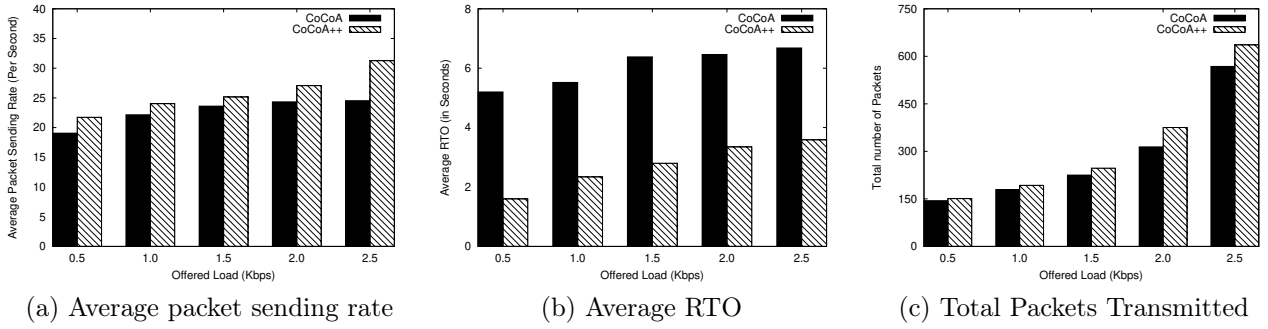


Figure 6.12: Dumbbell Topology - CoCoA vs CoCoA++

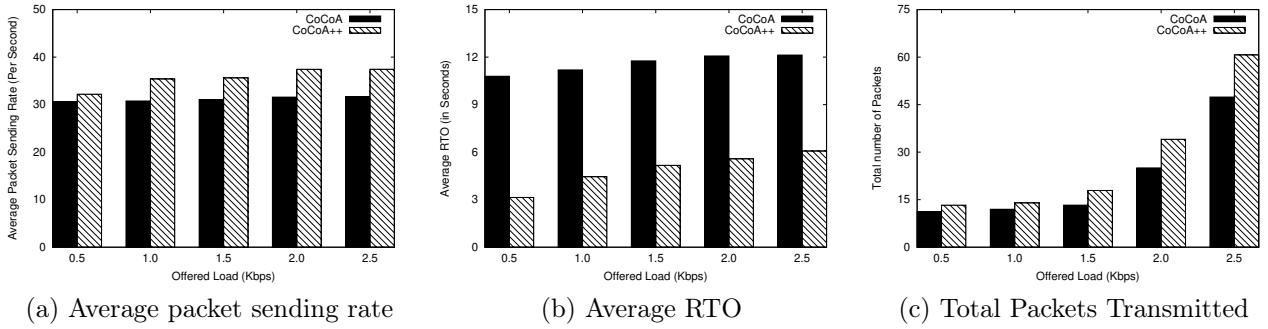


Figure 6.13: Chain Topology - CoCoA vs CoCoA++

rate increases congestion in the network and accordingly, the RTO values for both algorithms increase. RTO values with CoCoA remain consistently higher than CoCoA++ in all the topologies even when the offered load is low. This is expected because CoCoA relies on RTT and does not estimate queue delay. Similarly, the total number of packets transmitted and the average packet sending rate of CoCoA++ are always higher than CoCoA. Although the margin of improvement differs depending on the respective topology, the performance of CoCoA++ is significantly better in all topologies. The probability backoff factor of CoCoA++ plays a vital role in these scenarios by appropriately estimating the RTO values.

- Mobile topologies:** Figures (6.14 - 6.17) show the results obtained in mobile topologies for CoCoA and CoCoA++ with different traffic rates. In mobile topologies, the average RTO values are almost halved in CoCoA++ as compared to CoCoA. The reason for this is that CoCoA falsely assumes that the increment in RTT is an implication of congestion, and accordingly increases its RTO values. CoCoA++ measures queuing delay precisely, and it does not increase its RTO inadequately. The average packet sending rate of CoCoA++ in mobile scenarios is also higher than CoCoA. CDG and PBF both play an important role in these scenarios. The

retransmission timeout mechanism of CoCoA++ is highly dependent on the relative movement of RTT; thus the total number of packets sent by CoCoA++ are much higher than CoCoA.

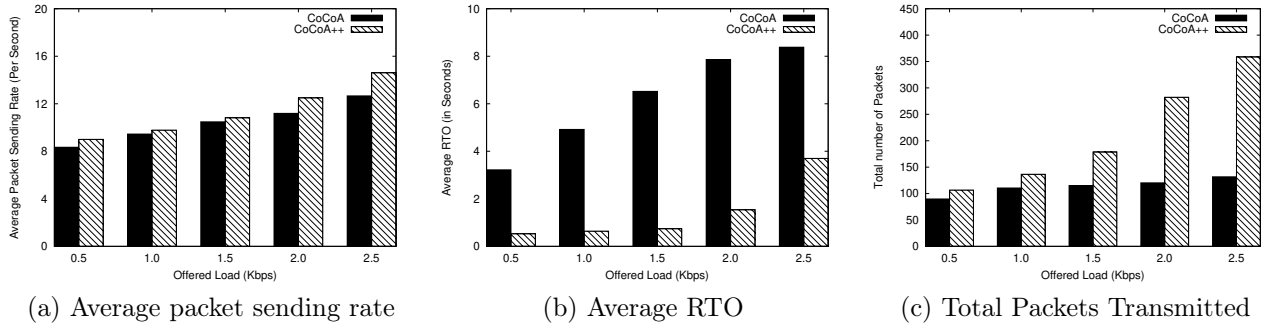


Figure 6.14: Random Way Point Mobility Model - CoCoA vs CoCoA++

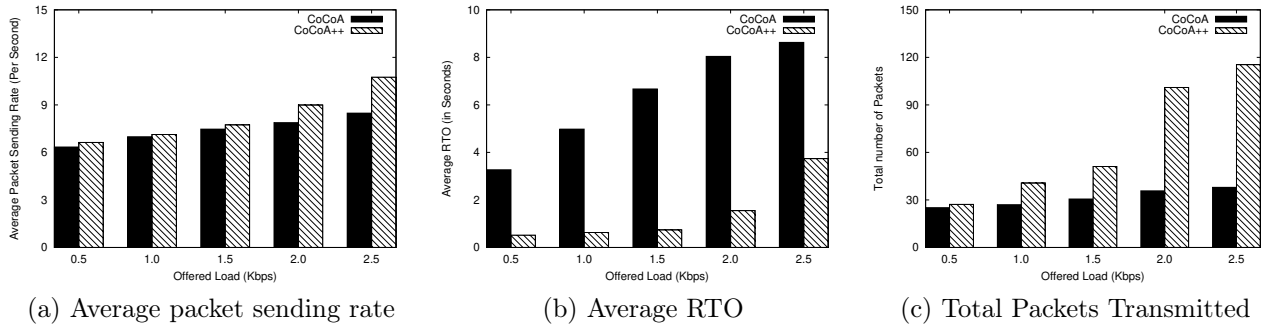


Figure 6.15: Gauss-Markov Mobility Model - CoCoA vs CoCoA++

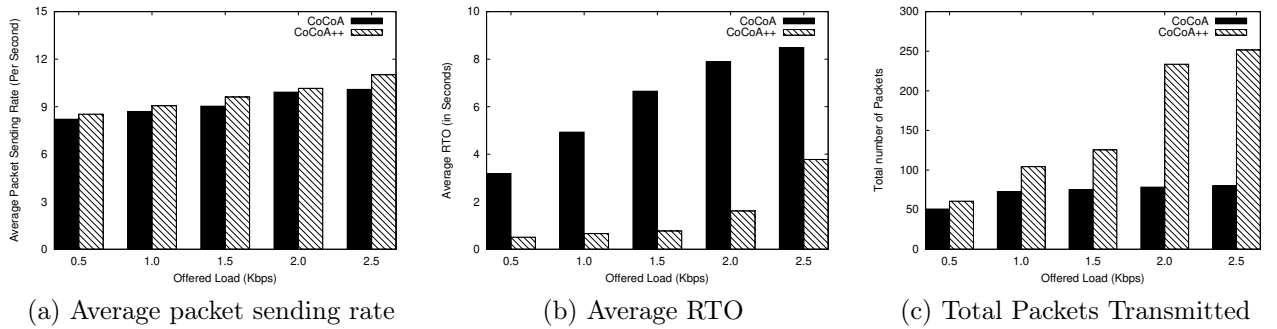


Figure 6.16: ManhattanGrid Mobility Model - CoCoA vs CoCoA++

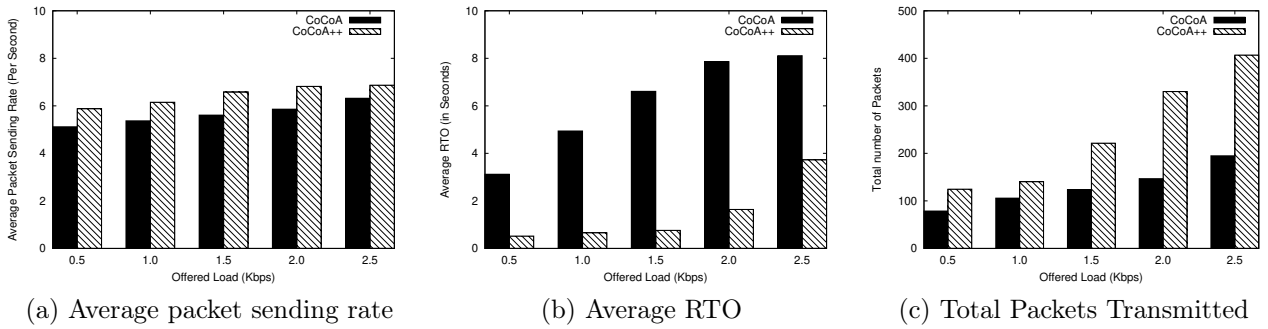


Figure 6.17: Pursue Mobility Model - CoCoA vs CoCoA++

6.2.3 Testbed Setup

There are several testbeds available for IoT experimentation, like FlockLab (Lim et al., 2013), FIT/IoT-LAB and others. For comparing the performance of CoCoA++ with CoCoA we choose FIT/IoT-LAB because of its scalability, functionality, and variety of nodes. FIT/IoT-LAB³ provides an extensive scalable framework to perform experimentation with tiny sensor devices and with complex communicating objects. This lab is a part of the Future Internet of Things (FIT) platform. It gives full control to the researchers for accessing the network nodes as well as the gateways to which the different nodes are connected. FIT/IoT-LAB allows monitoring of the various network related performance metrics like throughput, network overhead, power consumption of nodes and others. The lab offers quick deployment of experiments, along with the ease of evaluation, results collection and analysis. The only major limitation of this lab is that it does not provide adequate support to perform mobile IoT experiments.

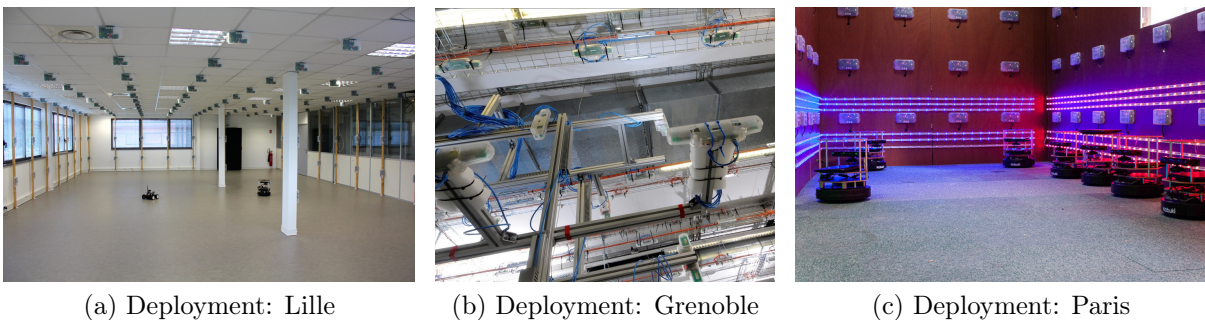


Figure 6.18: Deployment of nodes in Lille, Grenoble and Paris sites at FIT/IoT-LAB (Deployment, 2012)

The testbed contains various architectures which are distributed at seven different locations across France: Grenoble, Lille, Saclay, Strasbourg, Paris, Nantes and Lyon.

³<https://www.iot-lab.info/>

FIT/IoT-LAB uses specially designed boards for the testbed, e.g. WSN430 node, M3 node, A8 node. In our experiments, we use the M3 open node which is based on an STM32 (ARM Cortex M3) micro-controller. M3 open node supports FreeRTOS, Contiki and RIOT operating systems. Out of the seven different locations of FIT/IoT-LAB, we choose three locations for our experiments: Lille, Grenoble and Paris (See Figure 6.18). Lille, Grenoble and Paris are the most active locations of FIT/IoT-LAB that support 640, 332 and 160 wireless sensor nodes, respectively. They also support a large number of M3 open nodes. Figure 6.18 shows the real setup of the IoT devices in every site.

We configure a grid topology in our testbed because the internal nodes in the grid witness more congestion than any other nodes, and such a configuration would help to evaluate the effectiveness of CoCoA++ in heavily congested networks. Figure 6.19 shows the deployment of M3 open nodes and the selection of grid topology in each location. We have selected 9 (3*3) M3 open nodes at Lille and 6 (3*2) at Paris and Grenoble for our experiments.

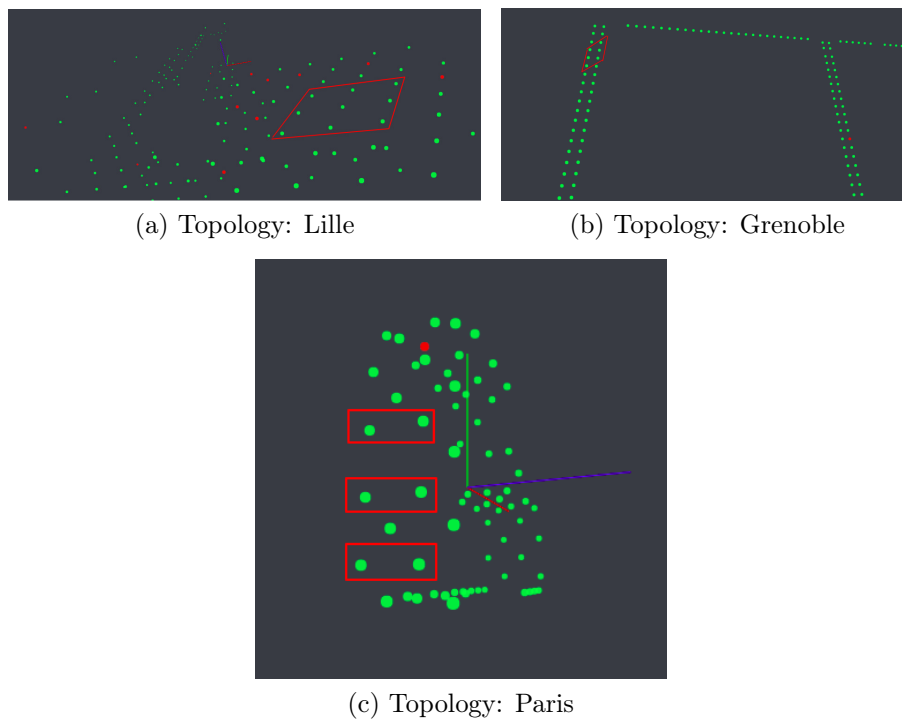


Figure 6.19: Topology Selection from Lille, Grenoble and Paris sites of FIT/IoT-LAB (TopologySelection, 2012)

6.2.4 Testbed Results

We configure five different traffic loads for evaluating the performance of CoCoA and CoCoA++ in FIT/IoT-LAB. The traffic load has been varied from 0.5 Kbps to 2.5 Kbps with an increment of 0.5 Kbps. We have used two performance metrics for evaluation: total number of packets transmitted and average packet sending rate. As shown in Figure 6.20, the number of packets transmitted by CoCoA++ is much higher than CoCoA in all three sites of the FIT/IoT-LAB. CoCoA++ uses CDG, and its RTO estimation depends

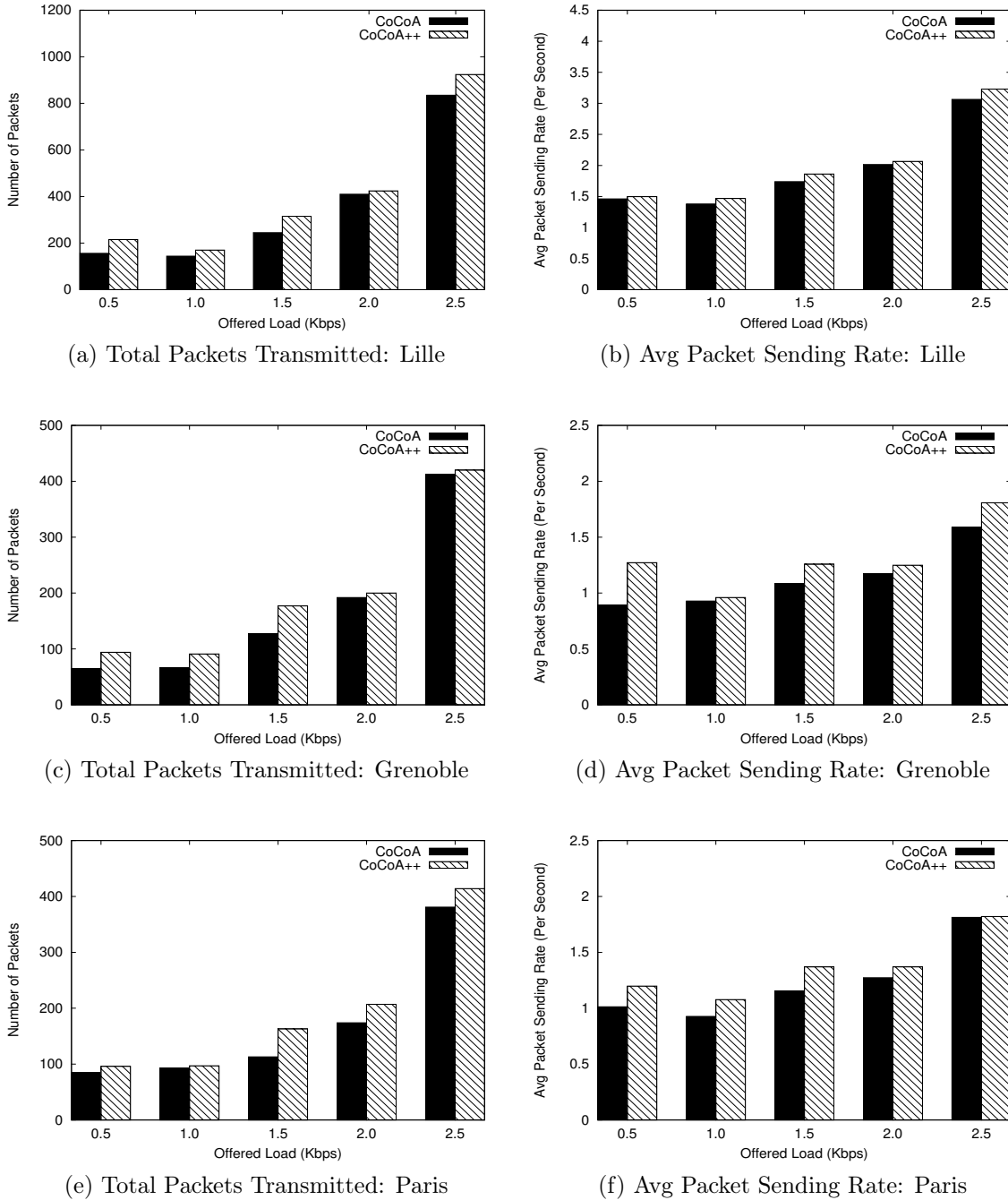


Figure 6.20: Comparison of Performance Metric in Grenoble, Paris and Lille sites at FIT/IoT-LAB

on the relative movement of RTT; hence it can transmit more packets than CoCoA. The average packet sending rate of CoCoA++ is also better than CoCoA.

6.3 Inferences

CoCoA does not perform well due to the inability to infer network congestion from noisy RTT samples. Since delay gradients provide a fairly accurate estimate of network congestion, CAIA Delay-Gradients and a Probability Backoff Factor have been integrated with CoCoA as an enhancement over the existing algorithm. The performance of CoCoA++ has been evaluated in static and mobile topologies to obtain a better understanding of how the algorithm fares in various IoT scenarios. The results show that CoCoA++ has low RTO values, thereby reducing transmission delays and increasing the transmission rate. Additionally, we evaluated the functionality of CoCoA++ with different traffic rates and noted that it provides the desired performance.

Depending on the inferences made from the simulations results, we tested the effectiveness of CoCoA++ by deploying it in a real testbed at FIT/IoT-LAB. We evaluated CoCoA++ with different workloads of a real-time IoT environment at three different locations of FIT/IoT-LAB: Lille, Paris and Grenoble. The results show that the total number of packets transmitted and average packet sending rate with CoCoA++ is higher than CoCoA.

The deployment complexity of CoCoA++ is highest when compared to GST-CoAP, CoAP-Eifel and GSRTC. The primary reason for this is the usage of exponential function in CDG. Nevertheless, we have used a table lookup to simplify the implementation of this function. CoCoA++ is most suitable for mobile IoT environments since the RTT samples in such scenarios can be extremely noisy. Example use cases of CoCoA++ include smart logistics and transportation, Internet of Vehicles (IoVs), communication among drones and others.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Congestion control remains a critical issue in communication networks. It has recently received a lot of attention in the context of IoT for efficient use of resources and optimum network performance. CoAP is a standardized application protocol for IoT deployments and has an in-built congestion control mechanism. However, the congestion control mechanism in CoAP is minimalistic and hence, a lot of improvements have been proposed recently to strengthen it. This work focused on the design and development of CoAP based congestion control mechanisms for IoT, named GST-CoAP, CoAP-Eifel, GSRTC and CoCoA++.

After receiving an acknowledgment for the retransmitted packet, CoAP resets the RTO to a default low value between $[2s, 3s]$. In this work, we demonstrated the shortcomings of this approach by performing experiments on the Cooja simulator and a real testbed in FIT/IoT-LAB. We showed that the existing *Fullbackoff* mechanisms outperform the default CoAP mechanism in terms of FCT, number of retransmissions and per-node throughput. GST-CoAP proposed in this work is a non-RTT based congestion control mechanism that requires a minor modification to the working of the original CoAP mechanism to improve the RTO estimation in CoAP. It is designed to overcome the limitations of CoAP and the existing *Fullbackoff* mechanisms. When the network is *not congested* and packets are getting cleared *consecutively without retransmissions*, GST-CoAP uses a geometric series to gradually reduce RTO. We carried out a performance evaluation of GST-CoAP with the original CoAP mechanism and the *Fullbackoff* mechanisms by using the Cooja simulator and the FIT/IoT-LAB. Our results show that GST-CoAP minimizes the FCT, reduces the total number of retransmissions in the network and enhances the per-node throughput.

CoAP defines a conventional congestion control mechanism that is insensitive to network conditions. Hence, we proposed CoAP-Eifel, a RTT based simple enhancement to CoAP that incorporates an adaptive RTO calculation based on RTT measurements. The Eifel Retransmission Timer which was originally designed for TCP has been integrated with CoAP (using UDP). The performance of CoAP-Eifel has been validated and evaluated by performing experiments in a real testbed using FIT/IoT-LAB. Our results show that CoAP-Eifel provides a better trade-off between delay and throughput, while providing a similar packet delivery ratio compared to CoAP.

CoCoA is an enhanced congestion control mechanism over CoAP and it estimates the RTO based on RTT measurements. It uses *fixed* weights to estimate RTO in addition to a strong and weak RTO estimator. The *Fullbackoff* mechanisms are an improvement to CoCoA. In this work, we demonstrated that the *Fullbackoff* mechanisms indeed outperform CoCoA in terms of FCT, number of retransmissions and throughput by conducting experiments on Cooja simulator and a testbed in FIT/IoT-LAB. Subsequently, we proposed GSRTC, a simple RTT based enhancement for predicting RTO in CoCoA. GSRTC uses geometric series to adapt the weight used in the Strong RTO estimator instead of using the *fixed* weight (0.5). The results obtained from simulation and real testbed show that GSRTC reduces the flow completion time, minimizes the number of retransmissions and provides higher network throughput compared to CoCoA and *Fullbackoff* mechanisms.

In this work, we show that congestion control algorithms that solely depend on per packet RTT measurements, such as CoCoA, do not perform well due to the inability to infer network congestion from noisy RTT samples. Hence, we proposed CoCoA++, a novel congestion control algorithm built on top of CoCoA that estimates the network congestion using delay gradients and a Probability Backoff Factor to control it. We evaluated CoCoA++ in static and mobile topologies to obtain a better understanding of how the algorithm fares in various IoT scenarios. Simulation results show that CoCoA++ has low RTO values, thereby reducing transmission delays and increasing the transmission rate. The results from the testbed show that the total number of packets transmitted and average packet sending rate with CoCoA++ is higher than CoCoA.

The newly proposed mechanisms are easy to deploy in real environments and can be adopted in IoT frameworks, such as IoTivity.

7.2 Future work

CoAP is actively used by a number of IoT applications, it would be interesting to investigate how these proposed algorithms share the resources fairly. This work can be expanded to assess fairness implications when CoAP-based and CoCoA-based congestion control mechanisms coexist. Moreover, this work has covered one non-RTT and three RTT-based optimizations to congestion control mechanisms for CoAP and CoCoA. This work can be extended to include a comparison of these proposed algorithms with various CoAP-based congestion control mechanisms such as FASOR, pCoCoA, BDP-CoAP, and others. It would be interesting to investigate the impact when the various CoAP-based congestion control mechanisms coexist with the algorithms proposed in this thesis.

The performance metrics covered in this work are: Flow Completion Times (FCT), number of retransmissions, throughput, RTO, delay, and packet sending rate. IoT devices are powered by batteries, hence it is important for the algorithms to be energy efficient. It would be interesting to investigate the proposed algorithm's effectiveness in various performance metrics such as energy consumption.

This work considered common traffic scenarios for evaluation, in which sensors send data to the gateway on a periodic or continuous basis. This work can be expanded by performing extensive evaluations of the proposed algorithms in different real-time traffic scenarios, such as bursty traffic. It would be interesting to see how the proposed algorithms perform in different traffic scenarios.

CoCoA uses *fixed* weights to estimate RTO in addition to a strong and weak RTO estimator (0.5 for strong and 0.25 for weak). However, in this thesis, GSRTC adapts the weight used in the Strong RTO estimator instead of using the *fixed* weight (0.5). This work can be extended by developing a congestion control mechanism that can adapt the weight used in the Weak RTO estimator rather than using *fixed* weight (0.25).

GSRTC adapts the weight of Strong RTO (0.5) using a geometric series. Stochastic Gradient Descent (SGD) can help in obtaining the optimal weight value over time between the lower (0.5) and upper (0.99) bounds for improving the performance of GSRTC. Besides, CAIA Delay Gradient in CoCoA++ continuously tracks the rate of change of RTT_{max} and RTT_{min} to determine whether the network is congested or not. SGD would aid in determining the optimal value of gradients of RTT_{max} and RTT_{min} , allowing for a more accurate estimate of network congestion.

IoTivity is an open source software framework that enables seamless device-to-device connectivity to address emerging requirements of the IoT network. It adheres to the Open Connectivity Foundation (OCF) standards which allows simple and secure communication between endpoints. CoAP is the default application protocol in IoTivity, which enables communication between two heterogeneous devices. Therefore, it would be interesting to evaluate the proposed algorithms' effectiveness by integrating them with IoTivity. For instance, one device communicates via CoCoA, while another communicates via CoCoA++, and both devices run distinct applications. It would be worth exploring how two algorithms work collaboratively.

References

- Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noel, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J., et al. (2015). FIT/IoT-LAB: A Large Scale Open Experimental IoT Testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464. IEEE.
- AdvanticSystemsServices (2014). Telosb (Sky) mote. <https://telosbsensors.wordpress.com/>. Accessed: 13-09-2017.
- Ahn, J. S., Danzig, P. B., Liu, Z., and Yan, L. (1995). Evaluation of TCP Vegas: Emulation and Experiment. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 185–195.
- Aimtongkham, P., Horkaew, P., and So-In, C. (2021). An Enhanced CoAP Scheme Using Fuzzy Logic With Adaptive Timeout for IoT Congestion Control. *IEEE Access*, 9:58967–58981.
- Akpakwu, G. A., Hancke, G. P., and Abu-Mahfouz, A. M. (2020). CACC: Context-Aware Congestion Control approach for lightweight CoAP/UDP-based Internet of Things traffic. *Transactions on Emerging Telecommunications Technologies*, 31(2):e3822.
- Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376.
- Al-Kashoash, H. A., Al-Nidawi, Y., and Kemp, A. H. (2016a). Congestion Analysis for Low power and Lossy Networks (LLNs). In *2016 Wireless Telecommunications Symposium (WTS)*, pages 1–6. IEEE.
- Al-Kashoash, H. A., Al-Nidawi, Y., and Kemp, A. H. (2016b). Congestion-aware RPL for 6LOWPAN Networks. In *2016 Wireless Telecommunications Symposium (WTS)*, pages 1–6. IEEE.

- Ancillotti, E. and Bruno, R. (2017). Comparison of CoAP and CoCoA+ Congestion Control Mechanisms for different IoT Application Scenarios. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 1186–1192. IEEE.
- Ancillotti, E. and Bruno, R. (2019). BDP-CoAP: Leveraging Bandwidth-Delay Product for Congestion Control in CoAP. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 656–661. IEEE.
- Ancillotti, E., Bruno, R., Vallati, C., and Mingozzi, E. (2018). Design and Evaluation of a rate-based Congestion Control Mechanism in CoAP for IoT applications. In *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 14–15. IEEE.
- Ariyakhajorn, J., Wannawilai, P., and Sathitwiriawong, C. (2006). A Comparative Study of Random Waypoint and Gauss-Markov Mobility Models in the Performance Evaluation of MANET. In *2006 International Symposium on Communications and Information Technologies*, pages 894–899. IEEE.
- Aschenbruck, N., Ernst, R., Gerhards-Padilla, E., and Schwamborn, M. (2010). BonnMotion: a Mobility Scenario Generation and Analysis Tool. In *Proceedings of the 3rd International Conference on Simulation Tools and Techniques*, page 51. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things: A Survey. *Computer Networks*, 54(15):2787–2805.
- Bai, F. and Helmy, A. (2004). A Survey of Mobility Models. *Wireless Adhoc Networks. University of Southern California, USA*, 206:147.
- Balandina, E., Koucheryavy, Y., and Gurtov, A. (2013). Computing the Retransmission Timeout in CoAP. In *Internet of Things, Smart spaces, and Next Generation Networking*, pages 352–362. Springer.
- Banks, A. and Gupta, R. (2014). MQTT version 3.1.1. OASIS Standard.
- Bansal, S. and Kumar, D. (2020). Distance-based Congestion Control mechanism for CoAP in IoT. *IET Communications*, 14(19):3512–3520.

- Bergmann, O. (2012). libcoap: C-Implementation of CoAP. URL: <https://libcoap.net/>. Accessed: 13-09-2020.
- Betzler, A., Gomez, C., and Demirkol, I. (2015a). Evaluation of Advanced Congestion Control Mechanisms for Unreliable CoAP Communications. In *Proceedings of the 12th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks*, pages 63–70.
- Betzler, A., Gomez, C., Demirkol, I., and Kovatsch, M. (2014). Congestion Control for CoAP Cloud Services. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–6. IEEE.
- Betzler, A., Gomez, C., Demirkol, I., and Paradells, J. (2013). Congestion Control in Reliable CoAP Communication. In *Proceedings of the 16th ACM International Conference on Modeling, Analysis & Simulation of Wireless and Mobile Systems*, pages 365–372.
- Betzler, A., Gomez, C., Demirkol, I., and Paradells, J. (2015b). CoCoA+: An Advanced Congestion Control Mechanism for CoAP. *Ad Hoc Networks*, 33:126–139.
- Betzler, A., Gomez, C., Demirkol, I., and Paradells, J. (2016). CoAP Congestion Control for the Internet of Things. *IEEE Communications Magazine*, 54(7):154–160.
- Bhalerao, R., Subramanian, S. S., and Pasquale, J. (2016). An Analysis and Improvement of Congestion Control in the CoAP Internet of Things Protocol. In *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 889–894. IEEE.
- Bolettieri, S., Tanganelli, G., Vallati, C., and Mingozi, E. (2018). pCoCoA: A precise Congestion Control Algorithm for CoAP. *Ad Hoc Networks*, 80:116–129.
- Bolettieri, S., Vallati, C., Tanganelli, G., and Mingozi, E. (2017). Highlighting some Shortcomings of the CoCoA+ Congestion Control Algorithm. In *International Conference on Ad-Hoc Networks and Wireless*, pages 213–220. Springer.
- Bormann, C., Betzler, A., Gomez, C., and Demirkol, I. (2020). CoAP Simple Congestion Control/Advanced (Work in Progress)(Latest). URL <https://core-wg.github.io/cocoa/draft-ietf-core-cocoa.html>.

- Brakmo, L. S. and Peterson, L. L. (1995). TCP Vegas: End-to-End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480.
- Camp, T., Boleng, J., and Davies, V. (2002). A Survey of Mobility Models for Ad Hoc Network Research. *Wireless Communications and Mobile Computing*, 2(5):483–502.
- Chan, M. T. Y. (2008). Wide-area Wireless Network Topology. US Patent App. 11/782,524.
- ContikiTutorial (2013). MAC protocols in ContikiOS. http://anrg.usc.edu/contiki/index.php/MAC_protocols_in_ContikiOS. Accessed: 07-02-2017.
- Demir, A. K. and Abut, F. (2018). Comparison of CoAP and CoCoA Congestion Control Mechanisms in Grid Network Topologies. *Gümüşhane Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, 8(ÖZEL SAYI):53–60.
- Demir, A. K. and Abut, F. (2020). mlCoCoA: a machine learning-based Congestion Control for CoAP. *Turkish Journal of Electrical Engineering & Computer Sciences*, 28(5).
- Deployment (2012). Wireless Sensor Nodes Deployment at FIT/IoT-LAB Site(s). <https://www.iot-lab.info/deployment/>. Accessed: 16-06-2021.
- Deshmukh, S. and Raisinghani, V. T. (2020). AdCoCoA-Adaptive Congestion Control Algorithm for CoAP. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–7. IEEE.
- Dunkels, A., Gronvall, B., and Voigt, T. (2004). Contiki-a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *29th annual IEEE International Conference on Local Computer Networks (LCN)*, pages 455–462. IEEE.
- Fahmy, S., Prabhakar, V., Avasarafa, S. R., and Younis, O. M. (2003). TCP over wireless links: Mechanisms and Implications. *Computer Science Technical Reports*, pages 1–6.
- Ghaffari, A. (2015). Congestion Control Mechanisms in Wireless Sensor Networks: A Survey. *Journal of Network and Computer Applications*, 52:101–115.

- Ghaleb, B., Al-Dubai, A. Y., Ekonomou, E., Alsarhan, A., Nasser, Y., Mackenzie, L. M., and Boukerche, A. (2018). A Survey of Limitations and Enhancements of the IPv6 Routing Protocol for Low-power and lossy networks (RPL): A Focus on Core Operations. *IEEE Communications Surveys & Tutorials*, 21(2):1607–1635.
- Gheisari, S. and Tahavori, E. (2019). CCCLA: A Cognitive approach for Congestion Control in Internet of Things using a Game of Learning Automata. *Computer Communications*, 147:40–49.
- Gomez, C., Arcia-Moret, A., and Crowcroft, J. (2018). TCP in the Internet of Things: from Ostracism to Prominence. *IEEE Internet Computing*, 22(1):29–41.
- Hasan, H. M. and Ahmed, A. I. (2018). A Comparative Analysis for Congestion Mechanism in CoAP and CoCoA. *Engineering and Technology Journal*, 36(8 Part A).
- Hassan, R., Jubair, A. M., Azmi, K., and Bakar, A. (2016). Adaptive Congestion Control Mechanism in CoAP Application Protocol for Internet of Things (IoT). In *2016 International Conference on Signal Processing and Communication (ICSC)*, pages 121–125. IEEE.
- Hayes, D. A. and Armitage, G. (2011). Revisiting TCP Congestion Control using Delay Gradients. In *International Conference on Research in Networking*, pages 328–341. Springer.
- Helpnet Security (2019). Connected IoT devices forecast. <https://www.helpnetsecurity.com/2019/06/21/connected-iot-devices-forecast/>. Accessed: 10-07-2021.
- Huang, J., Du, D., Duan, Q., Sun, Y., Yin, Y., Zhou, T., and Zhang, Y. (2014). Modeling and Analysis on Congestion Control in the Internet of Things. In *2014 IEEE International Conference on Communications (ICC)*, pages 434–439. IEEE.
- Iova, O., Picco, P., Istomin, T., and Kiraly, C. (2016). RPL: The Routing Standard for the Internet of Things... Or Is It? *IEEE Communications Magazine*, 54(12):16–22.
- Järvinen, I., Daniel, L., and Kojo, M. (2015). Experimental Evaluation of Alternative Congestion Control Algorithms for Constrained Application Protocol (CoAP). In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 453–458. IEEE.

- Jarvinen, I., Raitahila, I., Cao, Z., and Kojo, M. (2018). FASOR Retransmission Timeout and Congestion Control Mechanism for CoAP. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE.
- Järvinen, I., Raitahila, I., Cao, Z., and Kojo, M. (2018). Is CoAP Congestion Safe? In *Proceedings of the Applied Networking Research Workshop (ANRG)*, pages 43–49.
- Jonassen, K. K. (2015). Implementing CAIA Delay-Gradient in Linux. Master’s thesis, Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo.
- Kamgueu, P. O., Nataf, E., and Ndie, T. D. (2018). Survey on RPL enhancements: a focus on Topology, Security and Mobility. *Computer Communications*, 120:10–21.
- Karn, P. and Partridge, C. (1987). Improving Round-Trip Time estimates in Reliable Transport Protocols. *ACM SIGCOMM Computer Communication Review*, 17(5):2–7.
- Kim, H.-S., Im, H., Lee, M.-S., Paek, J., and Bahk, S. (2015). A Measurement Study of TCP over RPL in Low-power and Lossy Networks (LLNs). *Journal of Communications and Networks*, 17(6):647–655.
- Kim, H.-S., Kim, H., Paek, J., and Bahk, S. (2016). Load balancing under heavy traffic in Routing Protocol for Low power and lossy networks (RPL). *IEEE Transactions on Mobile Computing*, 16(4):964–979.
- Kim, H.-S., Ko, J., Culler, D. E., and Paek, J. (2017a). Challenging the IPv6 Routing Protocol for Low-power and lossy networks (RPL): A Survey. *IEEE Communications Surveys & Tutorials*, 19(4):2502–2525.
- Kim, H.-S., Paek, J., Culler, D. E., and Bahk, S. (2017b). Do not lose bandwidth: Adaptive Transmission Power and Multihop Topology Control. In *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 99–108. IEEE.
- Kovatsch, M., Duquennoy, S., and Dunkels, A. (2011). A low-power CoAP for Contiki. In *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, pages 855–860. IEEE.

- Kurata, K., Hasegawa, G., and Murata, M. (2000). Fairness Comparisons between TCP Reno and TCP Vegas for Future Deployment of TCP Vegas. In *Proceedings of INET*, volume 2000, page 2.
- Lee, J. J., Chung, S. M., Lee, B., Kim, K. T., and Youn, H. Y. (2016). Round Trip Time based Adaptive Congestion Control with CoAP for Sensor Network. In *2016 International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 113–115. IEEE.
- Lim, C. (2019). A Survey on Congestion Control for RPL-based Wireless Sensor Networks. *Sensors*, 19(11):2567.
- Lim, R., Ferrari, F., Zimmerling, M., Walser, C., Sommer, P., and Beutel, J. (2013). Flocklab: A testbed for Distributed, Synchronized Tracing and Profiling of Wireless Embedded Systems. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 153–166.
- Ludwig, R. and Gurtov, A. (2005). The Eifel Response Algorithm for TCP. Technical report, RFC 4015, February.
- Ludwig, R. and Sklower, K. (2000). The Eifel Retransmission Timer. *ACM SIGCOMM Computer Communication Review*, 30(3):17–27.
- M3Node (2012). IoT-LAB - M3 Node. Available at <https://www.iot-lab.info/docs/boards/iot-lab-m3/>. Accessed: 10-03-2020.
- Martinez, F. J., Cano, J.-C., Calafate, C. T., and Manzoni, P. (2008). Citymob: a Mobility Model Pattern Generator for VANETs. In *ICC Workshops-2008 IEEE International Conference on Communications Workshops*, pages 370–374. IEEE.
- Mishra, N., Verma, L. P., Srivastava, P. K., and Gupta, A. (2018). An Analysis of IoT Congestion Control Policies. *Procedia Computer Science*, 132:444–450.
- Mišić, J., Ali, M. Z., and Mišić, V. B. (2018). Architecture for IoT domain with CoAP Observe Feature. *IEEE Internet of Things Journal*, 5(2):1196–1205.
- Mo, J., La, R. J., Anantharam, V., and Walrand, J. (1999). Analysis and Comparison of TCP Reno and Vegas. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and*

- Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 3, pages 1556–1563. IEEE.
- Nick (2021). How Many IoT Devices Are There in 2021? [All You Need To Know]. <https://techjury.net/blog/how-many-iot-devices-are-there/>. Accessed: 10-07-2021.
- OASIS Standard (2012). Advanced Message Queuing Protocol (AMQP). <https://www.amqp.org/>. Accessed: 10-07-2021.
- Open Connectivity Foundation (OCF) (2015). IoTivity: an open-source IoT platform. <https://iotivity.org/>. Accessed: 25-10-2019.
- Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., and Voigt, T. (2006). Cross-level Sensor Network Simulation with Cooja. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks (LCN)*, pages 641–648. IEEE.
- Parasuram, A., Culler, D., and Katz, R. (2016). An Analysis of the RPL Routing Standard for Low power and Lossy Networks (LLNs). *Electrical Engineering and Computer Sciences University of California at Berkeley*.
- Pardo-Castellote, G. (2003). Object Management Group (OMG) Data-Distribution Service (DDS): Architectural overview. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pages 200–206. IEEE.
- Patel, M., Tanna, N., Patel, P., and Banerjee, R. (2001). TCP over wireless networks: Issues, Challenges and Survey of Solutions. *University of Texas, Dallas*, pages 1–6.
- Paxson, V., Allman, M., Chu, J., and Sargent, M. (2000). Computing TCP’s Retransmission Timer. Technical report, RFC 6298, November.
- Pramanik, A., Luhach, A. K., Batra, I., and Singh, U. (2017). A Systematic Survey on Congestion Mechanisms of CoAP based Internet of Things. In *International Conference on Advanced Informatics for Computing Research*, pages 306–317. Springer.
- Results (2019). CoCoA++ Supporting Files. <https://github.com/steps-to-reproduce/cocoapp-supporting-files.git>. Accessed: 07-04-2017.
- Ruckebusch, P., Giannoulis, S., Garlisi, D., Gallo, P., Gawlowicz, P., Zubow, A., Chwalisz, M., De Poorter, E., Moerman, I., Tinnirello, I., et al. (2017). WiSHFUL: Enabling

- coordination solutions for managing heterogeneous Wireless Networks. *IEEE Communications Magazine*, 55(9):118–125.
- Saclay (2012). Wireless Sensor Nodes Deployment at FIT/IoT-LAB Site(s). <https://www.iot-lab.info/deployment/>. Accessed: 16-06-2021.
- Saint-Andre, P. et al. (2004). RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core. *Internet Engineering Task Force (IETF)*.
- Shelby, Z., Hartke, K., and Bormann, C. (2014). The Constrained Application Protocol (CoAP) - RFC 7252. *Internet Engineering Task Force (IETF)*.
- SourceCode (2020a). Geometric Sequence Technique for Effective RTO Estimation in CoAP. <https://gst-coap.github.io/>.
- SourceCode (2020b). Implementation of Eifel Retransmission Timer in CoAP. <https://eifel-retransmission-timer-nitk.github.io>.
- Suwannapong, C. and Khunboa, C. (2019). Congestion Control in CoAP Observe Group Communication. *Sensors*, 19(15):3433.
- Suwannapong, C. and Khunboa, C. (2021). EnCoCo-RED: Enhanced Congestion Control mechanism for CoAP Observe Group Communication. *Ad Hoc Networks*, 112:102377.
- Tariq, M. A., Khan, M., Raza Khan, M. T., and Kim, D. (2020). Enhancements and Challenges in CoAP - A Survey. *Sensors*, 20(21):6391.
- Tian, Y., Xu, K., and Ansari, N. (2005). TCP in wireless environments: Problems and Solutions. *IEEE Communications Magazine*, 43(3):S27–S32.
- TopologySelection (2012). Topology Selection at FIT/IoT-LAB Site(s). <https://www.iot-lab.info/deployment/>. Accessed: 16-06-2021.
- Vallati, C., Righetti, F., Tanganelli, G., Mingozzi, E., and Anastasi, G. (2018). ECOAP: Experimental Assessment of Congestion Control Strategies for CoAP using the WiSHFUL Platform. In *2018 IEEE International Conference on Smart Computing (SMART-COMP)*, pages 423–428. IEEE.
- Zolertia (2014). Z1 mote. <https://github.com/Zolertia/Resources/wiki/The-Z1-mote>. Accessed: 13-09-2017.

List of Publications

Journal Publications

1. Rathod, V., Jeppu, N., Sastry, S., Singala, S., and Tahiliani, M. P. (2019). CoCoA++: Delay Gradient based Congestion Control for Internet of Things. *Future Generation Computer Systems*, 100: 1053-1072
2. Rathod, V. and Tahiliani, M. P. (2021). Geometric Series based effective RTO estimation Technique for CoCoA, *Ad Hoc Networks*, Elsevier journal. [**Under Minor Revision**]

Conferences

1. Rathod, V. J., Krishnam, S., Kumar, A., Baraskar, G., and Tahiliani, M. P. (2020). Effective RTO estimation using Eifel Retransmission Timer in CoAP. In *International Conference on Electronics, Computing and Communication Technologies (CONECCT)* (pp. 1–6). IEEE. [**Best Paper Award - Academia**]
2. Rathod, V. J. and Tahiliani, M. P. (2020). Geometric Sequence Technique for Effective RTO Estimation in CoAP. In *International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, (pp. 1–6). IEEE.

Brief Bio-Data

Personal Details

Rathod Vishal Jitendrakumar

Date of Birth :- 24th February 1988

Research Scholar

Department of Computer Science and Engineering

National Institute of Technology Karnataka, Surathkal

P.O. Srinivasnagar

Mangalore - 575025

Karnataka, India.

Phone: +91 9879957770

Email: rathodvishal78@gmail.com

Permanent Address

Rathod Vishal Jitendrakumar

S/o Jitendrakumar Chimanlal Rathod

15, Vishalnagar, Opp. Amit Park

Isanpur, Ahmedabad - 382 443

Gujarat, India.

Qualification M. Tech. in Computer Engineering, Dharmsinh Desai Institute of Technology (DDIT), Dharmsinh Desai University (DDU), Nadiad - 387 001, Gujarat, India, 2011.

B. E. in Computer Engineering, Charotar Institute of Technology Changa (CITC), Gujarat University (GU), Changa 388 421, Anand, Gujarat, INDIA, 2009.

Work Experience

Assitant Professor, Department of Computer Enigneering, Charotar University of Science and Technology (CHARUSAT), Anand, Gujarat. (2011-2015)