DEVELOPMENT OF A MODIFIED TOKEN BASED CONGESTION CONTROL SCHEME WITH ADAPTIVE FORWARDING FOR OPPORTUNISTIC NETWORK

BY

Idris Salawu SHAIBU B.Eng. (FUT., MINNA) 2006

P13EGCP8003

salawuidriss@gmail.com

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES,

AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE WARD OF A

MASTER OF SCIENCE (M.Sc.) DEGREE IN COMPUTER ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

FACULTY OF ENGINEERING

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

NOVEMBER,2017

DECLARATION

I Idris Salawu SHAIBU, hereby declare that the work in this dissertation entitled "Development of a Modified Token Based Congestion Control Scheme with Adaptive Forwarding" for Opportunistic Network has been carried out by me in the Department of Computer Engineering. The information derived from literature has been duly acknowledged in the text and a list of references provided. No part of this dissertation was previously presented for another degree or diploma at this or any other institution.

Idris Salawu SHAIBU _____ _____

Signature Date

CERTIFICATION

This Dissertation entitled DEVELOPMENT OF A MODIFIED TOKEN BASED CONGESTION CONTROL SCHEME WITH ADAPTIVE FORWARDING FOR OPPORTUNISTIC NETWORK meets the regulations governing the award of degree of Master of Science (MSc) in Computer Engineering of the Ahmadu Bello University, and is approved for its contribution to knowledge and literary presentation.

Dr. E. A. Adedokun	_____	_____
Chairman, Supervisory Committee	(Signature)	Date
Dr. H. A. Adamu	_____	_____
Member, Supervisory Committee	(Signature)	Date
Prof. M.B. Muazu	_____	_____
Head of Department	(Signature)	Date
Prof. S.Z. Abubakar	_____	_____
Dean. School of Postgraduate Studies	(Signature)	Date

DEDICATION

This dissertation is dedicated to Almighty Allah; The Most Beneficient, The Most Gracious and The Most Merciful. Also, to my parents, Malam Shaibu Salawu and Malama Saratu Shaibu and my Beloved Hawawu Idris and Idris Salawudeen Masood.

ACKNOWLEDGEMENT

Indeed, I am grateful to Almighty Allah for His infinite blessings, protections and guidance towards the successful completion of this work. I wish to express my utmost gratitude to my supervisor, role model and the chairman of my supervisory committee, Dr. E. A. Adedokun for his time, immerse contributions and valuable guidance towards the success of this work. Indeed, the completion of this work could not have been possible without your consistent participation and assistance. I am proud to have you as my supervisor. You helped me to shape the research problem and provided valuable insight on the solution to the problem. Thank you very much Dr. My thanks also go to my co-supervisor Dr. H. A. Adamu for his valuable input and constant encouragement throughout the stages of the work. I really appreciate the training I received from you. My profound gratitude also goes to Prof. M.B Muazu, and Dr. Eric Coe, Ming Hsieh Department of Electrical Engineering, University of Southern California, Los Angeles, for their great support and enlightment throughout this research. May almighty Allah reward you all abundantly. My deep appreciation goes to the Computer & Control Research Group for their valuable contributions, suggestions and constructive criticisms during all those stages of my presentations.

I acknowledge and appreciate the contributions of all the lecturers of Electrical and Computer Engineering, Ahmadu Bello University, namely: Prof. B. G. Bajoga, Prof. U.O Aliyu, Prof. B. Jimoh, Dr. A. M. S. Tekanyi, Dr. S. M. Sani, Dr. S. Man-Yahaya, Dr. I. J. Umoh and Dr. Y. Jibril, Dr. K.A. Abu-Bilal, Dr. G.A Olarinoye, Engr. M. J. Mu'azu, Engr. A. I. Abdullahi, and most especially, those whose names could not be mentioned. My sincere appreciation also goes to Malam Abdullahi Tukur for his administrative support towards the completion of the work.

My special thanks go to Salawudeen A. Tijani, Mrs Basira Yahaya, Abubakar Umar, Abdulmumini Yesufu. Zaharudeen Haruna, Engr. S.M Aminu, Ajayi Ore-Ofe, Gideon Atuman, Joel Nwajiaku Ndubisi, Busayo Adebisi, Ovie Ese, Umar Musa, Olaniyan Abdulrahman, and Suleiman Hussein for their valuable contributions and support towards the success of this work. I am indeed grateful to you people, your handsome reward is with God. I am very much thankful to all my course mates, especially Shittu Danraka, Abdulazeez Abdulhafeez and Tukur Lawal for their encouragement and persistence. God will reward you all.

Above all, I am indebted to my parents Malam Shaibu Salawu and Malama Saratu Shaibu for their continuous love and support. Your kind advice and understanding will always be appreciated. Thank you very much. My gratitude also goes to my siblings, Malam Yahaya Shaibu, Malam Rahman Shaibu, Malama Nana shaibu and malama Medinat Shaibu, whose love, caring and prayer have kept me strong and sound all through my life.

Shaibu Idris Salawu

November, 2017.

ABSTRACT

This research presents the development of a modified token-based congestion control scheme with adaptive forwarding mechanism (mTBCC) algorithm for addressing congestion problems in opportunistic networks (OppNets). The algorithm addresses the limitations associated with the standard token –based congestion control (TBCC) in terms of its ability to redirect the traffic from more congested nodes of OppNet to congestion-free nodes without necessarily compromising computational time. This is because the TBCC has the tendency to drop significant number of messages when node is full (overflow) in order to control congestion. OppNet was modeled using ONE simulator with Eclipse, which is a java based programming language. The node density was controlled by varying the greatest connected component (GCC) expressed in percentage and the corresponding results were used to evaluate the performance of the proposed approach using (dropped messages and network transit time) as performance metrics. The results showed reduction in dropped messages and network transit time across all scenarios considered. At queue size of 10(QS-10), TBCC had 38592 messages, and mTBCC has 36037 messages, yielding an average improvement of 13.91%, at queue size of 20 (QS-20), TBCC had 30330 messages and mTBCC had 27845 messages resulting in average improvement of 10.78%, at queue size of 30(QS-30) TBCC produced 28356 messages and mTBCC yielded 26767 messages resulting to an average improvement of 5.68% and at queue size of 40(QS-40) , TBCC had 23150 messages while mTBCC had 22197 messages, providing an average improvement of 4.22% respectively for dropped messages. In addition, at 0.5GCC, TBCC had 29401.70 time and mTBCC had 27151.41 time, producing an average improvement of 8.34%, at 0.6GCC, TBCC produced 16319.29 time and mTBCC had 15966.42 time, resulting in average improvement of 2.19%, at 0.7GCC, TBCC yielded 13178.21 time and mTBCC produced 12581.01 time, resulting in an average improvement of 4.61%, and at 0.8GCC, TBCC had 12333.55 time and mTBCC had 11453.23 time, yielding an average improvement of 7.63% for network transit time.

TABLE OF CONTENTS

TITLE PAGE		i
DECLARATION		ii
CERTIFICATION		iii
DEDICATION		iv
ACKNOWLEDGEMENTS		v
ABSTRACT		vii
TABLE OF CONTENTS		viii
LIST OF TABLES		xii
LIST OF FIGURES		xiv
LIST OF ABBREVIATIONS		xvi

CHAPTER ONE: INTRODUCTION

1.1	Background	1
1.2	Problem statement	3
1.3	Motivation	3
1.4	Aim and objectives	4
1.5	Significant of Research	4
1.6	Scope of the Research	5

1.7 Dissertation outline 5

CHAPTER TWO: LITERATURE REVIEW

2.1 Introduction 6

2.2 Review of fundamental concepts 6

2.2.1 Delay tolerant networks 6

2.2.1.1 Delay tolerant network architecture 7

2.2.1.2 Bundle protocol 8

2.2.2 Delay tolerant networks routing protocols 9

2.2.2.1 Flooding strategies 10

2.2.2.2 Forwarding strategies 11

2.2.3 Delay tolerant networks congestion control schemes 13

2.2.3.1 Congestion detection 14

2.2.3.2 Congestion control 14

2.2.4 Opportunistic networks 15

2.2.5 Token based congestion control 18

2.2.6 Adaptive forwarding strategy 19

2.2.7 Opportunistic network environment (ONE) simulator 20

2.2.8 Research performance metrics 21

2.3 Review of similar works 22

CHAPTER THREE: METHODS

3.1 Introduction 32

3.2 Methodology 32

3.3 Replication of the token-based congestion control 33

3.3.1 Initializing token based parameter 33

3.3.2 Token based congestion control scenario settings 34

3.4 Development of a modified token-based congestion control 35

3.4.1 Modified token based congestion control scenario settings 37

3.5 Installation and configuration 38

3.6 Opportunistic network modelling 38

3.7 Simulation model 39

3.8 PRoPHET routing protocol 39

3.9 Visualization 39

3.10 Validation of the congestion control strategies 40

3.11 Performance evaluation 41

3.11.1 Percentage improvement 42

3.11.1 Rrelevant equations 42

CHAPTER FOUR: RESULTS AND DISCUSSION

4.1 Introduction 43

4.2 Results of the token-based congestion control strategy 43

4.3 Results of the modified token-based congestion strategy 49

4.4 Comparison of the results 55

4.4.1 Comparison of mTBCC and TBCC performance for dropped message 55

4.4.1.1 Dropped message percentage improvement 58

4.4.2 Comparison of mTBCC and TBCC performance for NTT 60

4.4.2.1 NTT percentage improvement 63

CHAPTER FIVE: CONCLUSION AND RECOMMENDATIONS

5.1 Summary 65

5.2 Conclusion 65

5.3 Limitation 65

5.4 Significant contribution 66

5.5 Recommendations for further work 66

REFERENCES 67

APPENDICES 70

LIST OF TABLES

Table 4.1: TBCC dropped messages 44

Table 4.2: TBCC network transit time at 50% GCC 45

Table 4.3: TBCC network transit time at 60% GCC 45

Table 4.4: TBCC network transit time at 70% GCC 45

Table 4.5: TBCC network transit time at 80% GCC 46

Table 4.6: mTBCC dropped message 49

Table 4.7: mTBCC network transit time at 50% GCC 50

Table 4.8: mTBCC network transit time at 60% GCC 51

Table 4.9: mTBCC network transit time at 70% GCC 51

Table 4.10: mTBCC network transit time at 80% GCC 51

Table 4.11: Comparison of dropped messages between mTBCC and TBCC 55

Table 4.12: Dropped message improvement of mTBBC over the TBCC at QS-10 58

Table 4.13: Dropped message improvement of mTBCC over the TBCC at QS-20 58

Table 4.14: Dropped message improvement of mTBCC over the TBCC at QS-30 59

Table 4.15: Dropped message improvement of mTBCC over the TBCC at QS-40 59

Table 4.16: Comparison of NTT between mTBCC and TBCC at GCC connected 60

Table 4.17: NTT percentage improvement of mTBCC over the TBCC at 50% GCC 63

Table 4.18: NTT percentage improvement of mTBCC over the TBCC at 60% GCC 63

Table 4.19: NTT percentage improvement of mTBCC over the TBCC at 70% GCC 64

Table 4.20: NTT percentage improvement of mTBCC over the TBCC at 80% GCC 64

LIST OF FIGURES

Figure 2.1: Typical DTN architecture 8

Figure 2.2: Comparison between TCP/IP Stack and DTN Protocol Stack 9

Figure 2.3: DTN congestion control schematic diagram 14

Figure 2.4: Opportunistic network scenario 17

Figure 2.5: Flowchart of the TBCC algorithm 19

Figure 2.6: ONE simulator overview 21

Figure 3.1: TBCC Simulation Environment 35

Figure 3.2: Flowchart of the mTBCC algorithm 36

Figure 3.3: mTBCC simulation environment 37

Figure 3.4: Helsinki Simulation Area Overview 41

Figure 4.1: Dropped Messages for TBCC 44

Figure 4.2: Network Transit Time for TBCC at 50% GCC 46

Figure 4.3: Network Transit Time for TBCC at 60% GCC 47

Figure 4.4: Network Transit Time for TBCC at 70% GCC 47

Figure 4.5: Network Transit Time for TBCC at 80% GCC 48

Figure 4.6: Dropped Messages for mTBCC 50

Figure 4.7: Network Transit Time for mTBCC at 50% GCC 52

Figure 4.8: Network Transit Time for mTBCC at 60% GCC 53

Figure 4.9: Network Transit Time for mTBCC at 70% GCC 53

Figure 4.10: Network Transit Time for mTBCC at 80% GCC 54

Figure 4.11: Comparison of dropped message between mTBCC and TBCC at QS-10 56

Figure 4.12: Comparison of dropped message between mTBCC and TBCC at QS-20 56

Figure 4.13: Comparison of dropped message between mTBCC and TBCC at QS-30 57

Figure 4.14: Comparison of dropped message between mTBCC and TBCC at QS-40 57

Figure 4.15: Comparison of NTT between mTBCC and TBCC at 50% GCC 61

Figure 4.16: Comparison of NTT between mTBCC and TBCC at 60% GCC 61

Figure 4.17: Comparison of NTT between mTBCC and TBCC at 70% GCC 62

Figure 4.18: Comparison of NTT between mTBCC and TBCC at 80% GCC 62

LIST OF ABBREVIATIONS

Acronyms	Definition
DTN	Delay Tolerant Network
OppNets	Opportunistic Networks
FIFO	First-in-First-out
GUI	Graphical User Interface
LEO	Low Earth Orbit
ONE	Opportunistic Network Environment
TCP	Transmission Control Protocol
TTL	Time –to- Live
GCC	Greatest Connected Component
RAM	Random Access Memory
RTT	Round Triple Time
IDE	Integrated Development Environment
GPS	Global Positioning System
URL	Universal Resource Locator
EMN	Exotic Media Networks
PRoPHET	Probabilistic Routing Protocol using History of Encounter and Transitivity

CHAPTER ONE

INTRODUCTION

1.1 Background

In the last decade, the deployment of wireless-enabled devices have achieved tremendous growth as the number of mobile wireless device increases giving rise to spontaneous communication between devices. However in many cases the spontaneous network formed between these devices are characterized by partitioning and long periods of disconnectivity (Piórkowski *et al.*, 2008; Zhang *et al.*, 2007). To address these spontaneous network constraints such as intermittent connectivity, high latency and support, delay tolerant applications have emerged using store-carry-and- forward approach to forward messages in Delay Tolerant Networks (Thompson *et al.*, 2010).

Delay Tolerant Networks (DTNs) are decentralized networks designed to function effectively in extreme environment such as intermittent connectivity, long delays and typically are wireless mobile networks (Coe & Raghavendra, 2010). The DTN applications represent a broad class of networks such as interplanetary internets (Burleigh *et al.*, 2003) , underwater sensor networks (Partan *et al.*, 2007), vehicular ad-hoc networks (Lin *et al.*, 2008; Lu *et al.*, 2008) and opportunistic networks (OppNets) (Zhang *et al.*, 2014), etc. DTN has several attributes contrary to the traditional network. In traditional networks, some assumptions are made such as (Yang *et al.*, 2012):

1. The packet loss ratio is minimal (low error rate)
2. The link between two connecting nodes is always stable

3. The data transfer time between two nodes should not be more than two times round triple time.

The goal of DTN architecture and protocols is to provide delivery of data messages between nodes in extreme environments where traditional network protocols, such as transmission control protocol and internet protocol (TCP/IP) networks and MANET protocols fail. Such environments violate many underlying assumptions of TCP/IP networks, which comprises the existence of end-to-end path between nodes, short delays on communication links and stable connectivity (Coe & Raghavendra, 2010).

The DTN architecture described in (Fall, 2003), uses the so called store-carry-and-forward approach, as contrary to the concept of Internet's store-and-forward, to deliver messages before forwarding them, however the time elapses while waiting to be forwarded are much smaller when compared to DTNs. The DTN architecture extends the reach of network and facilitates communication among nodes in a challenged environment. It possesses the following characteristics (Yang *et al.*, 2012):

1. High latency
2. Low data speed
3. Intermittent connectivity
4. High error rate
5. Long Queueing time
6. Bandwidth is asymmetry in nature.

Opportunistic networks (OppNets) are characterized by sparse multi-hop variant of ad-hoc networks where nodes utilize any available contact opportunities to exchange and forward messages. It is one types of challenged networks developed from delay tolerant networks, which

is the focus of this research work. The OppNets implement store-carry-and-forward operation to transmit messages, which is distinct from the store and forward mechanism in MANET. The current intermediate node has the capability to store messages during blackout and forward it when connectivity resumes as nodes move (Zhang *et al.*, 2014).

1.2 Problem Statement

Detecting and handling congestion in OppNets is vital and challenging task because of its representative properties. The existing OppNet forwarding algorithms mostly send packets towards specific nodes to maximize delivery ratio and minimize latency. However, as traffic continue to build up in the network, these nodes become congested and drop any incoming traffic. This research proposes development of a modified token-based congestion control scheme with adaptive forwarding to monitor the amount of traffic entering the network to the network capacity and reroute packet from congested node to congestion- free node of OppNet, taken cognizance of least migration cost and the largest available free buffer size from the reference node and drop the message if buffer of all nodes is full.

1.3 Motivation

Traditional Opportunistic Network congestion control mechanisms are mostly directed towards particular node on certain criteria, which eventually overload the node and consequently become congested with time under different applications. They are not adaptable to different opportunistic network scenarios, mainly because they were designed to a specific scenario with special features. Adaptive forwarding techniques often provide results with fairness and efficient resource use. Thus, this approach is especially relevant to opportunistic networks, where connected paths does not always exist between the source node and the destination node. To

sustain the network operating at an acceptable efficiency, nodes in opportunistic networks would need to evaluate the likelihood of congestion, by dissecting the local data. In addition, adaptive forwarding mechanisms provide the power to redirect the traffic from more congested node to congestion free node in the network. In other words, since the opportunistic network environment is dynamic, the opportunistic network node has to be able to adjust in order to cope adequately with the environment. Congestion avoidance is a basic and fundamental task of autonomous opportunistic network nodes. It is especially vital in challenging environments where an end-to-end communication is not always possible to reach a destination.

1.4 Aim and Objectives

The aim of this research is to develop a modified token-based congestion control scheme mTBCC approach to reduce the effects of congestion in opportunistic networks.

The objectives of the research are enumerated as follow:

1. To replicate the TBCC and develop the mTBCC with adaptive forwarding mechanism via opportunistic network environment (ONE) simulator
2. To develop the simulation setup of the mTBCC with adaptive forwarding using ONE simulator
3. To compare the performance of the TBCC-based model and that of the mTBCC-based model on the benchmark of Helsinki area test-bed using dropped message and network transit time as performance metrics.

1.5 Significance of Research

Communication in a challenged network environment occurs intermittently and characterized by high variable latency, making TCP/IP not suitable to be applied. In this situation, OppNets concept provides alternative measure necessary for data transfer. The major significant difference between Internet communication paradigm and OppNets communication is that end-

to-end communication path does not always exist leading to disconnection, high latency, and high error rate in communication. To address this challenge, OppNet uses store-carry-and-forward approach to forward packet from source to destination. OppNet has different routing protocols based on historical knowledge or replication schemes for message successful delivery.

1.6 Scope of the Research

The research presents the development of a modified token-based congestion control scheme with adaptive forwarding mechanism for OppNets. This approach is based on using token based techniques where nodes holding valid token are allowed to inject message into the network and at congestion point, reroute the message to the nearest neighboring node taking cognizance of least migration cost and largest available free buffer space. The algorithm is simulated using ONE simulator and validated using Helsinki area test bed.

1.7 dissertation outline

Chapter One presents the general introduction of the dissertation while Chapter Two presents the comprehensive review of related literature and relevant fundamental concepts about token based, delay tolerant network, delay tolerant network routing, opportunistic network, congestion control, adaptive forwarding strategy and opportunistic network environment simulator and Helsinki simulation area. Chapter Three presents an elaborate procedure for the actualization of modified token- based congestion control scheme with adaptive forwarding algorithm by using the cost function: largest available free buffer size and the least migration cost. The analysis, performance and discussion of the result are presented in chapter four. Chapter Five covers the conclusion and recommendation for areas of further research. Finally, relevant references and Appendixes are presented at the end of this dissertation.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This section consists of two subsections. The first section addresses the fundamental concepts critical to the research. The other part of the review focus on similar works of different researchers who worked in this domain. The works are critically studied to establish the ground to bring in the contribution of this work, which is aimed to achieve.

2.2 Review of Fundamental Concepts

This section introduces the fundamental concepts pertinent to the research, which includes delay tolerant networks, delay tolerant network routing protocols, opportunistic networks, congestion control scheme and opportunistic network environment (ONE) simulator.

2.2.1 Delay Tolerant Networks

Delay tolerant networks (DTNs) represent a full division of wireless networks which requires minimum to none infrastructures and has the potential to support network functionality experiencing frequent and long-lasting partition. DTNs are opted to tackle scenario such as heterogeneity of standards, intermittent connectivity between adjacent nodes, lack of connected end-to-end paths as well as excessive high delay and data error-rates. The accessibility of mobile nodes in stressed environments can immensely affect their resources, which include central processing unit (CPU), memory and network capacity (SuvarnaPatil & Chillerge, 2014)

DTNs architecture is designed to operate efficiently in challenging environments, achieve interoperability and extend range of connectivity of complex application (SuvarnaPatil & Chillerge, 2014). The DTN applications represent a broad class of networks, which include:

1. Opportunistic Network
2. Mobile Ad-Hoc Networks connecting remote and rural communities through global position system (GPS), cellular devices and portable memory
3. Exotic Media Networks (EMN) interconnecting extra-terrestrial nodes such as satellites and deep space probes in Interplanetary Networks (IPNs).
4. Wireless Sensor Networks (WSNs) deployed to monitor wildlife tracking

2.2.1.1 Delay Tolerant Network Architecture

A typical architecture of DTN is depicted in Figure 2.1. It consists of DTN gateways and four heterogeneous regions having different communication protocols. Region A is an internet region, region B (Guo *et al.*, 2007) is rural network in remote areas consisting of a mobile gateway that ferry between gateway 3 and gateway 5, region C is the sensor network, and region D is satellite networks having a low earth orbit (LEO) link which allow cycle connection (Sun *et al.*, 2011). Nodes within the same region communicate with each other via local region communication, and those in different region communicate with other through gateway at the point of regional border (Marcus *et al.*, 1998).

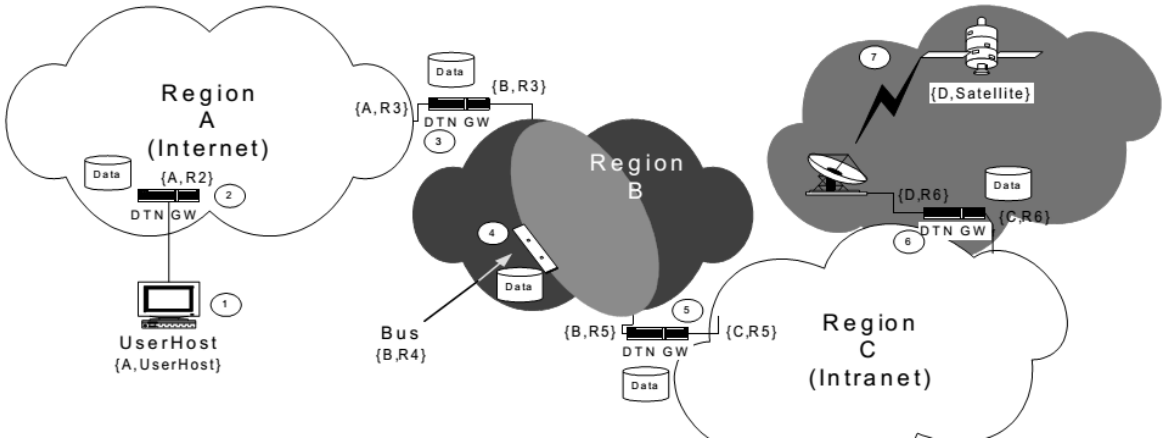


Figure 2.1: Architecture of Delay Tolerant Network (Fall, 2003)

2.2.1.2 Bundle Protocol

The DTN implements bundle layer between the transport layer and application layer in heterogeneous networks. The bundle layer provides persistent storage function as a result of store and forward mechanism and enhance interoperability among heterogeneous network, using naming mechanism based on universal resource locator (URL) and encapsulation mechanism (Marcus *et al.*, 1998). Figure 2.2, illustrates the comparison between the transmission control protocol (TCP/IP) stack and DTN protocol stack.

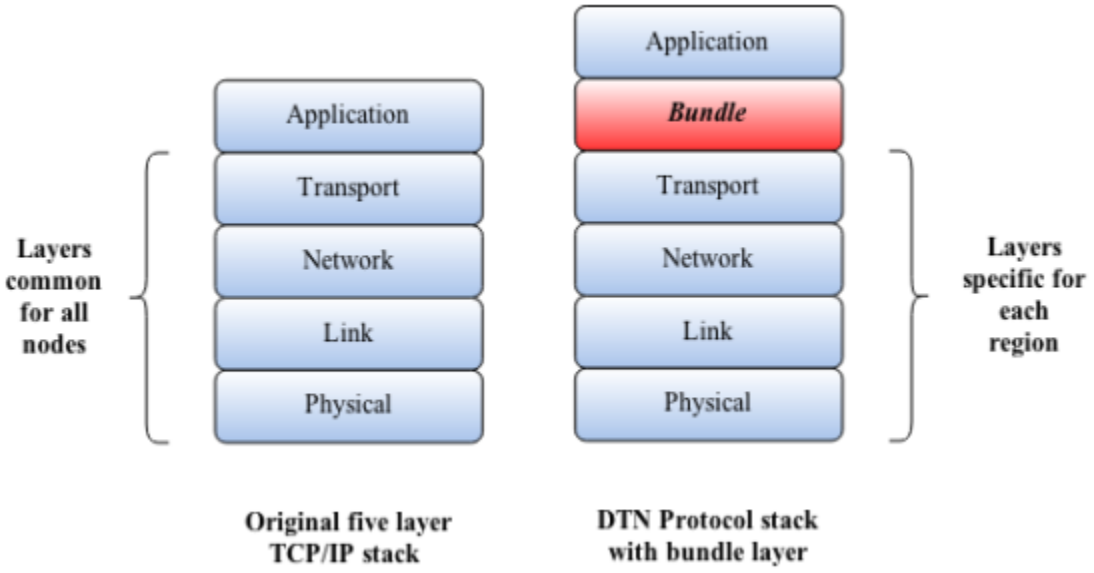


Figure 2.2: DTN Specific Stack (Voyiatzis, 2012).

The bundle protocol provides the following classes of services as implemented in the architecture of DTN (Voyiatzis, 2012):

1. Custody transfer
2. Notification of custody transfer
3. Notification of bundle forwarding
4. Return receipt

5. Delivery priority

6. Authentication

2.2.2 Delay Tolerant Networks Routing Protocols

One of the characteristics of DTN is the connectivity of nodes that is intermittent, no two nodes always have stable communication links due to the dynamic evolving network topology and end-to-end paths are not guaranteed. DTN utilizes store-carry-and-forward strategy to forward packets across source and address nodes. In order to make this a reality, the intermediate nodes provide custody services for packets and later forward them when opportunity arises (SuvarnaPatil & Chillerge, 2014).

For efficient forwarding of packets and performance of an OppNets, a route between nodes must be optimal and strategically formed. Researchers have advance several strategies used for routing, among which are classified as flooding strategies and forwarding strategies which basically reflect the epidemic routing and prophet routing (SuvarnaPatil & Chillerge, 2014).

2.2.2.1 Flooding strategies

In flooding strategy, a message is replicated severally and copies are handed over to relay nodes. The relay nodes store the message until an opportunity arises to have contact with the destination node. Researchers have proposed several strategies to optimize successful delivery of message such as epidemic routing protocol. These strategies are based on the context of mobile ad-hoc network where random mobility is an option to bring the source node in contact with destination node. Message replication is used to maximize the successful delivery of message to the desired node without any prior knowledge of the network (SuvarnaPatil & Chillerge, 2014).

Routing protocols that fall in the category of flooding scheme are discussed as follow:

1. Epidemic routing protocol

The epidemic routing protocol is a flooding-based mechanism. Every node possesses two buffers, one for storing the self-generated messages and the other one for the received messages from the neighboring nodes. Every message has a particular identity. Every node holds the record of message identities that is presently stored in its buffer called summary vector. Whenever, two nodes contact each other, they share their summary vectors with each other. The nodes compare their summary vectors and exchange those messages that they do not have with them. Immediately the exchange operation is completed, the nodes maintain the same message level in their buffers resulting to large amount of message redundancy in the networks and incurs significant demand on the network's constrained resources (Dhurandher *et al.*, 2011).

2. Spray And Wait protocol

This protocol provides measure to limit the flooding level in the OppNets. The message is delivered in two distinct phases: the spray phase and the wait phase. For each message emanating from the source node, has L duplicates widely spread all over the network by the source node to L different relays. In the spray phase if the destination was not discovered, every node holding a duplicate of the message embark on direct transmission in the Wait phase (Dhurandher *et al.*, 2011).

2.2.2.2 Forwarding strategies

Forwarding strategy requires the knowledge of the network to determine the best path of nodes between source and destination, such that a message is transferred among the intermediate nodes along this path. Many researchers have advanced many of these protocols like Probability Routing Protocol based on History of Encounter and Transitivity (PRoPHET) routing protocol

that uses history of encounter to optimize the probability of successful message delivery ratio (SuvarnaPatil & Chillerge, 2014).

The following routing protocols that fall in the same categories with the forwarding strategies are discussed in detail as follow:

1. First Contact

It routes the packets at random through any available opportunistic contact. The message is transmitted via a link that is randomly chosen amongst all the selected contacts. If none of the links is available, the packet waits until links come up and is tagged to the first available contact. This algorithm deletes the local replicate of the packet after a transfer from one node to another. This presents single copy of the packets in the network, amounting to less resource consumption and low congestion in the network. The tendency of path loops exist, if frequent contacts are established between the same node pair for a long period of time and exchanging packets between them (Dhurandher *et al.*, 2011).

2. Maxprop

It is a routing protocol designed for vehicular ad hoc networks (VANETs). The protocol transferred the message to any of the node in the network having maximum message delivery probability to the destination. It employs modified Dijkstra algorithm and history of encounters between nodes to compute the path for every message to its peers. It further utilizes the acknowledgement of delivered message to delete redundant messages from the network when buffer becomes full (Dhurandher *et al.*, 2011).

3. PRoPHET routing protocol

In PRoPHET makes use of the observations that mobile users move in a predictable pattern. When a user has visited a particular location severally, there is every tendency to revisit such

location again, and PRoPHET routing uses this information to improve routing performance. The protocol keeps records of predictability metric at every node, and this metric depicts the message delivery probability of a node to the desire destination. When two nodes are in contact, they exchange summary vector as well as delivery probability information. With this information, the update of the delivery probability information is maintained (Lindgren *et al.*, 2003).

The PRoPHET routing protocol is opted for in this research because of the fact that its communication overhead is controlled and that the protocol maintains the record of historical information of encounters as well as node movements to predict the probability that a packet will be delivered to the intended destination (Lindgren *et al.*, 2003).

2.2.3 Delay Tolerant Networks Congestion Control Schemes

The implementation of congestion control scheme in DTN is challenging due to its characteristics. Notwithstanding, several researchers have advanced various congestion control schemes which to a certain extent are critical to ensure adequate network operation and execution. A substantial conflict between traditional network and DTN is the end-to-end reliability issues, which gets the traditional network congestion control mechanism not suitable for DTN. The disruption of message in DTN occurs due to unreliable connected path transmission mechanism, which involves the operation of the retransmission mechanisms. Figure 2.3 indicates a typical congestion control phenomenon for DTN. Node A, B, C represent the source node, node D depicts the destination node, and node T represents a managed forwarding node, while nodes E and F act as relay nodes. Nodes A, E, T, D constitute link1 and nodes B, F, T, D depict link2 (Zhang & Bai, 2015).

When A and B independently send messages to node D via link1 and 2, consuming the larger portion of storage resource of node T and the storage resources is spent after a certain period of

time, then node C forwards messages to D via T, as a result of limited storage resources of node T, such message cannot be accommodated due to congestion at this time. Congestion resolution is classified into two processes, namely: congestion detection and congestion control (Zhang & Bai, 2015): A typical DTN congestion control schematic diagram is shown in Figure 2.3.

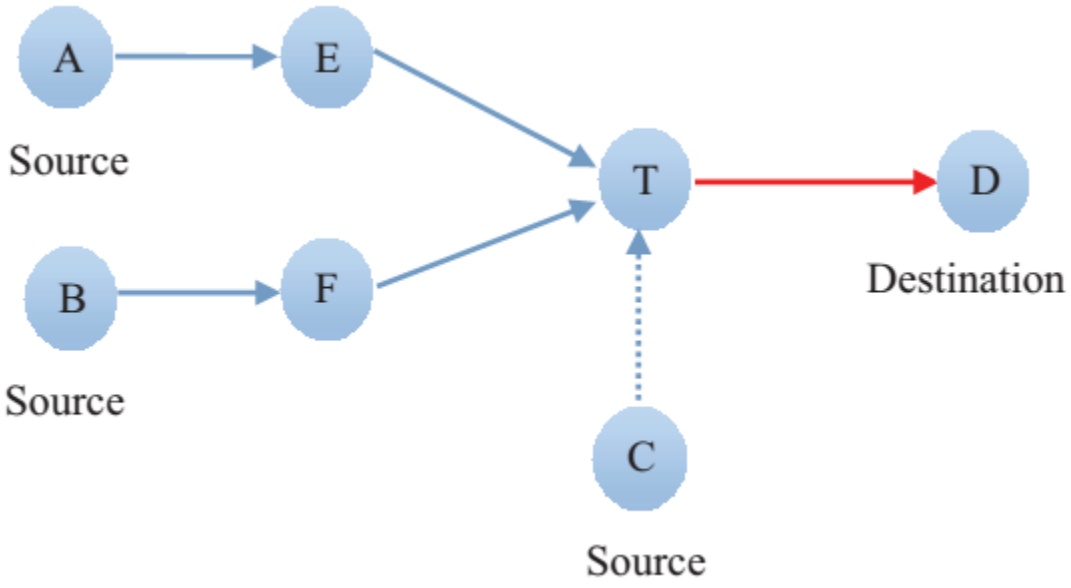


Figure 2.3: DTN Congestion Control Schematic Diagram (Zhang & Bai, 2015)

2.2.3.1 Congestion detection

It is the bedrock of congestion control both in traditional networks and DTNs. Presently, DTN uses indexes to detect congestion which includes cache queue length, transmission delay, network capacity, and rate of packet loss (Akyildiz *et al.*, 2002; Burleigh *et al.*, 2003). The symptoms are critical to congestion control and determine how effectively to detect network congestion and respond immediately. This prevents excess traffic injected into the network and avoid overloading the network link.

2.2.3.2 Congestion control

Presently, the existing congestion control mechanism is to send a copy of the same message via multiple paths independently. This enhances reliability, but then increases the link error rate and message reliable transfer is hard to guarantee. Currently, the existing mechanism for congestion control can further be classified into congestion avoidance and congestion handling. Congestion avoidance is also known as open-loop control, which is a preventive measure that does not depend on the feedback coming from the network before avoiding network congestion. This ensures high throughput state of the network (Zhang & Bai, 2015) Congestion handling is otherwise called closed-loop control, which is a passive recovery approach. The sender uses this techniques to detect the messages lost to affirm congestion and effect the necessary control measures to avert the congestion in order to restore the normal communication of the network (Zhang & Bai, 2015).

With the rapid increase in the deployment of DTN applications, a robust network congestion control measure becomes imperative to enhance efficient performance of the network. In DTN forwarding strategy, applies the mechanism of directing messages towards specific node to optimize delivery ratios and minimize delay. However, within network load these nodes may be over loaded with times. Several works in DTNs network congestion try to minimize the hop-by-hop delay and cost. This is achieved with the assumptions of unlimited transfer and storage capacity (Grundy, 2012). Most congestion control mechanism centered on buffer management (Balasubramanian *et al.*, 2007; Burleigh *et al.*, 2006) . Recently advancements have taken replication management into account(Seligman *et al.*, 2007; Tournoux *et al.*, 2009) and distribution(Nelson *et al.*, 2009; Pujol *et al.*, 2009) . Congestion avoidance mechanism has the ability to limit the nodes sending speed of its own sensory information and regulate the sending

speed of different nodes. Congestion handling deals with input/output speed, reasonable traffic scheduling amongst others (Zhang & Bai, 2015).

2.2.4 Opportunistic Networks

Opportunistic networks (OppNets) are one type of DTN applications where network contact is characterized by intermittent connectivity and connected path rarely exist between the source and destination node. As a result, TCP/IP protocol is ineffective in this kind of networking environment because link re-establishment can exist briefly at unpredictable period. Long propagation delay and extreme queueing delays are some of the representative properties of OppNets and considerable number of internet protocols that function on the assumption of quick return of acknowledgement and data are unsuitable in this type of challenged network where complete path from source to destination does not exist most of the time. Furthermore, links are subject to frequent disruption and to make the data communication feasible in an OppNets, relay nodes take custody of the message during the link breakage and forward it when connectivity resumes (Vidya & Hemanth, 2014).

The basic opportunistic network scenario is presented in Figure 2.4, which describes how transfer reliability and congestion control functions interact. A custody transfer scenario is depicted in Fig.2.4, where a message meant for node D, currently occupies the persistent buffer of node S (Figure.2.4 (a)). During its mobility, node S encounters node R as the next best hop of the message to node D based on its routing protocol requirements and forward the message to node R (Figure.2.4 (b)). S requests for custody transfer service of the message to R and set the request time-out-timer to begin (Figure.2.4(c)). On getting the custody request, R triggers its storage management mechanism to determine the impact of receiving such message in future,

and decide whether to accept or reject the custody request. In this scenario, R accepts the request (figure. 2.4(d)).

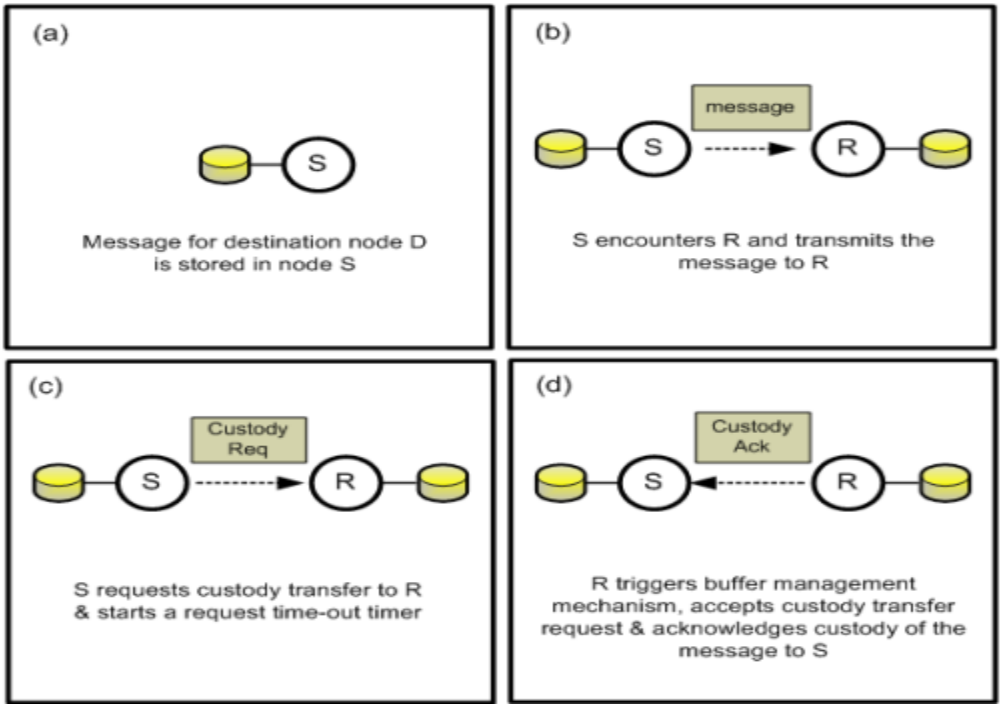


Figure 2.4: Opportunistic Network Scenario (Vidya & Hemanth, 2014)

The existence of simultaneous paths to transmit a message between a source and destination in OppNets are not assumed because the context information in which the users communicate is a vital factor to design efficient routing protocols for OppNets. Nevertheless, this data is not regularly available, especially when the users are isolated, information cannot be transmitted and efficient routing decision is not viable. Node mobility and local forwarding techniques are exploited the context information issue in order to facilitate data transfer. The intermediate nodes store the data, carry it, claiming the merit of node mobility, and forward it when opportunistic contact exists. Opportunistic networks are regarded as one of the most interesting advancement of the MANET. In OppNets, mobile nodes are equipped to communicate with each other once they are within communication range, and are not

required to possess prior knowledge of the network topology, which is fundamental to traditional MANET routing protocols. Routes are established dynamically, and packets are enroute between the source and the destination. In message transfer, any available relay nodes are utilized opportunistically as the next best hop that can bring the message closer to the final destination. The applications of OppNets is mostly feasible in environment that is tolerant to long delay and high error rate (Vidya & Hemanth, 2014).

2.2.5 Token Based Congestion Control

Token-based congestion control (TBCC) is a congestion avoidance strategy based on proactive measure that only allows the node holding valid token to inject message into the network, which is similar to the token ring. However, TBCC differs from token ring, such that it requires token to inject data into the network rather than performing per hop transmitting of messages. Token is independent of routing algorithm used and are distributed evenly across all the network nodes. The aim of TBCC is to monitor the amount of traffic entering the network with the network capacity such that a node accommodates as much data as it can forward. The token is retrieved when message leaves the network either favorably or unfavorably in one of the following ways (Coe & Raghavendra, 2010):

1. Messages arrive their destination.
2. Messages are discarded due to insufficient storage space at a node
3. Messages are dropped due to time-to-live (TTL) timer expiration

The flow chart for TBCC is shown in Figure 2.5

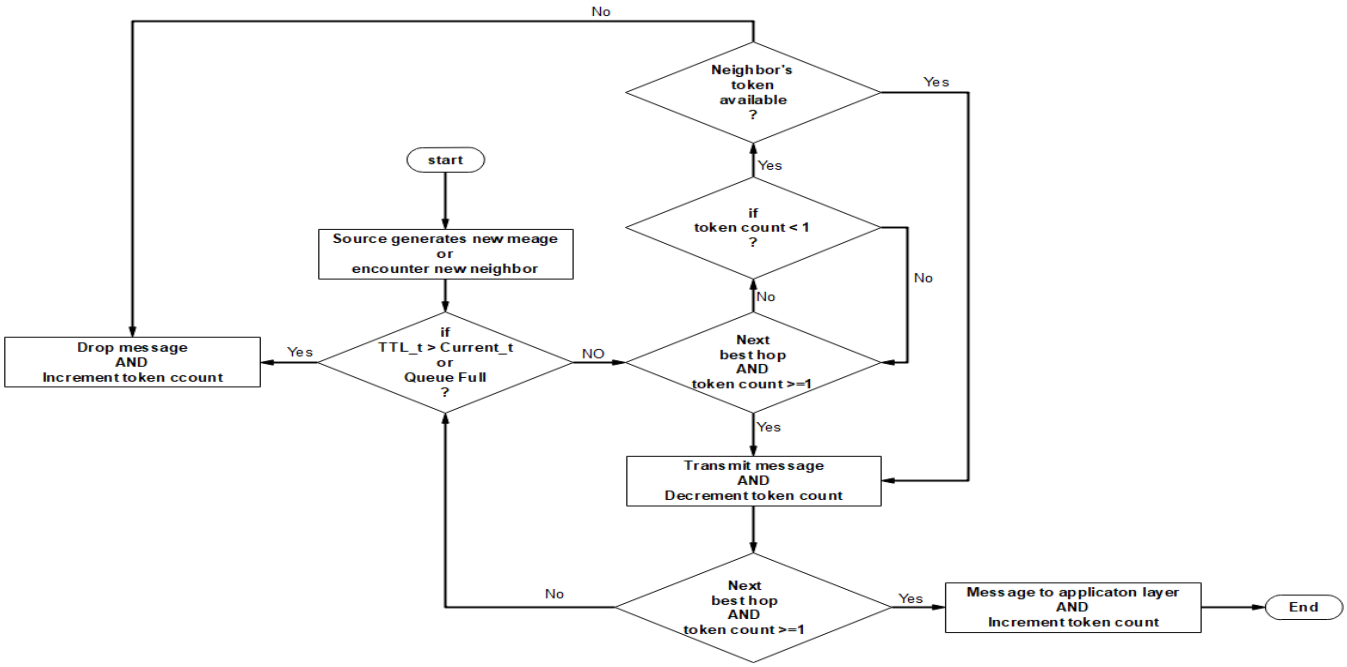


Figure 2.5: Flowchart of TBCC (Patil & Penurkar, 2015)

The number of token to put across the opportunistic network nodes is equal to the queue size of a node as given in (Coe & Raghavendra, 2010):

$$\text{Queue size} = \frac{buffersize}{messagesize} \tag{2.1}$$

2.2.6 Adaptive Forwarding Strategy

Adaptive forwarding strategy is an algorithm that redirect the traffic on the network from the more congested nodes to congestion free nodes of the network and drop the message in case, the buffer is full for all nodes, by considering the largest available free buffer space and the least migration cost during the time of increasing congestion(Radenkovic & Grundy, 2012). The buffer space threshold and the migration cost threshold (which describe the modification carried out on TBCC when congestion occurs) are given as (Li *et al.*, 2009; Yang *et al.*, 2012):

$$MC_{ith} = TC + SC \tag{2.2}$$

$$BS_{ith} = Li(\max) + \text{lc} \tag{2.3}$$

where:

MC_{ith}: Migration cost threshold

TC: Transmission cost

SC: Storage cost

$Li(\max)$: Maximum message length

lc: Packet overhead length

BS_{ith}: Buffer space threshold

2.2.7 Opportunistic Network Environment (ONE) Simulator

ONE is an open source java-based discrete event simulator design for research works in DTN environment that can be customized. It provides the necessary codes implementation for some DTN routing protocols and has inbuilt interface for importing mobility traces from real-world traces or other mobility generators. It can be configured for simulating different scenarios quickly in a flexible manner. The simulator has the potential to generate several reports ranging from node movement, message passing and general statistics. It has different routing schemes such as (epidemic, spray and wait, prophet, etc.) for message forwarding when contact opportunity is available. The simulation environment of ONE simulator, as in Figure 2.6, has the potential of (Keränen & Ott, 2007):

1. Producing node movement by utilizing different movement models
2. Routing messages between nodes with different DTN routing algorithms

3. Visualization of node mobility and message passing during the simulation process in real time with the aid of its Graphical User Interface (GUI). Figure 2.6. Illustrates the ONE simulator overview.

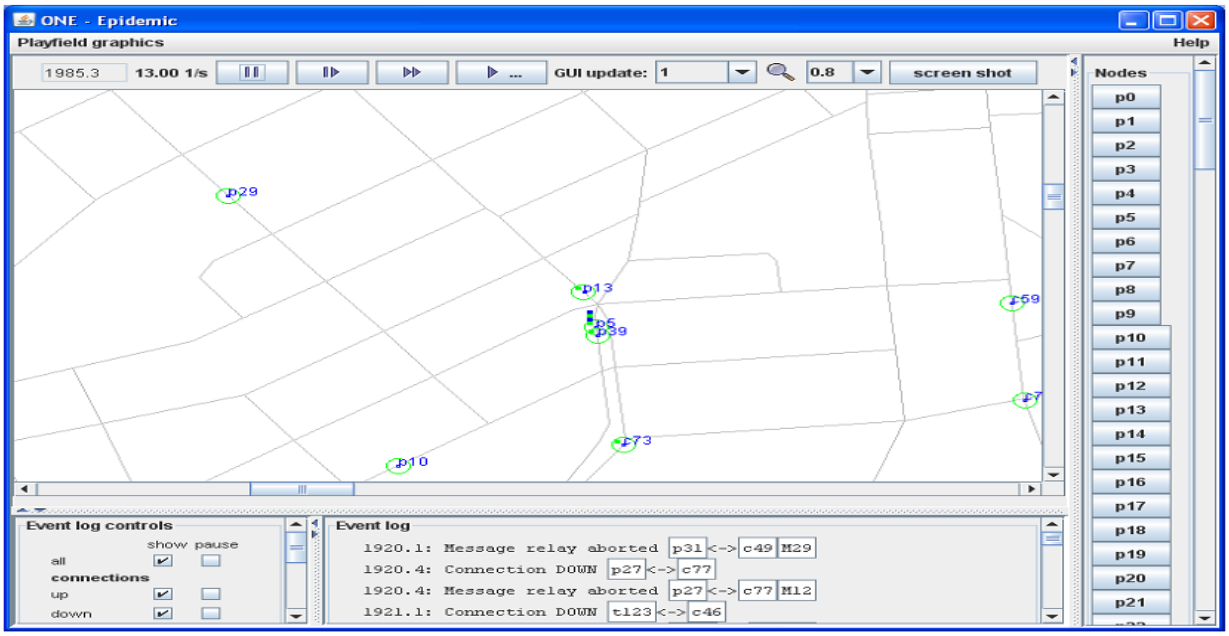


Figure 2.6: ONE Simulator Overview (Keränen & Ott, 2007)

The ONE simulator (GUI) enables interactive inspection of what happens during simulation process in the following ways (Keränen & Ott, 2007):

1. Potential to display simulations in real time showing node locations, active contacts as well as the messages carried by the nodes.
2. Utilizing map-based mobility constraint model, indicating all map paths.

2.2.8 Research Performance Metrics

This aspect considers the following performance metrics and their significance in OppNets congestion control as they affect the overall network performance (Coe & Raghavendra, 2010):

1. Dropped messages: is the average number of messages discarded when network is under congestion. This has the potential to degrade the packet delivery rate of a network and hence reduces the network performance.
2. Network transit times: is the average contact time from the first hop to the desired destination nodes. Opportunistic Networks are characterized by high and variable delay, the shorter the time elapse for an algorithm, the better its performance. Equation 2.4 describe the network transit time.

   Network transit time = Average latency – contact time of the first node (2.4)

2.3 REVIEW OF SIMILAR WORKS

This section presents the review of similar works pertinent to the research.

Yun *et al.* (2010) proposed congestion control mechanism based on Average Forwarding Number based on Epidemic Routing (AFNER). Packets were forwarded in non-fragmented message bundles arranged in ascending order in the buffer in accordance to their forwarding number. In addition, packets were also forwarded based on FIFO order. The message arrived at the intended destination node only if the forwarding number of packets was greater than or equal to the average forwarding number of packets presently occupying the entire buffer. The AFNER and epidemic routing protocols were simulated using NS-2 simulator and the results showed that AFNER outperformed the epidemic routing protocol in terms of delivery ratio, end-to-end latency and average forwarding number. However, this control strategy was epidemic routing protocol dependent, which generated unrestricted copies of the same messages that wasted network resources and invariably increased congestion in the network.

Coe and Raghavendra (2010) presented token-based method for congestion control aimed at monitoring the total amount of data entering the network to the total network capacity. The state of the previous congestion control strategies by different researchers were presented followed by the TBCC algorithm. The greatest connected components (GCC) metric was introduced to provide additional control over network connectivity characteristics. Token was uniformly distributed across all the network nodes initially and then randomized. The TBCC was simulated using discrete event simulator (DES). This mechanism did well in terms of maximizing total network transit time and minimizing average dropped messages. However, deleting a message that has traversed several hops in the network resulted to waste of bandwidth, buffer resources and also induced the retransmission node for retransmission of such message when the need arised.

Das *et al.* (2011) proposed proactive congestion control strategy based on local information and economic concept. In their work, the node computed the price and dropping probability of a bundle being dropped from the node in order to facilitate the acceptance and rejection decision making for the bundle. The concept of price denoted the ranks for each bundle at the node based on the bundle and node's properties. The cash-in-hand represented the present buffer capacity of the node to store more bundles and confidence level computed the degree of confidence of node to forward the buffered bundles to their respective next hops. Comparison was made against the predefined thresholds for cash-in-hand, price and confidence level to aid the decision-making processes of the scheme. The performance of the scheme was evaluated using DTN SIM2 and the simulation results showed that the scheme did well in terms of delivery ratios. However, the decision-making processes of the mechanism was computationally intensive and hence responded to congestion very slowly.

Radenkovic and Grundy (2012) presented a unified adaptive forwarding and adaptive replication into a common congestion framework called congestion aware forwarding and replication (CAFREP) for challenged networks. The mechanism has the ability to decrease the level of traffic in the congested part of the network and at the same time redirect the traffic from more to less congested part of the network. Combined utility heuristic consisted of social driven heuristic, resource heuristic, and ego network heuristic were exploited to detect and predict the parts of the network liable to congestion. The algorithm was evaluated over three real world connectivity and global position system traces from CRAWDAD: Infocom 2006, Sassy 2011 and San Francisco Taxi Cab 2011 that have different mobility and connectivity patterns. Two application scenarios were used to generate different traffic patterns such as publish subscriber Podcasting application and Facebook social networking application. The CaFRep was simulated via ONE simulator and adapts well to the dynamics of all the three data sets. It kept high success ration and availability, low delay and minimize packet loss rate and outperformed the other protocols across all data sets such as PRoPHET, spray and focus, competitive adaptive protocol, encounter based routing and retiring replicas. However, sending an adaptive number of message copies to the next suitable hop had to incurring more packet overhead, thereby consuming the limited resources of the network.

An *et al.* (2012) considered congestion level based on end-to-end acknowledgment strategy to remove redundant message-copies from the resource constrained DTN. In their work, active and passive forwarding approaches were identified and formed the basis for the proposed acknowledgements scheme transmission. In passive forwarding, acknowledgement will only be sent to the node that buffered a corresponding copy of the successfully delivered message to the destination node. For active forwarding, acknowledgement can be transmitted to the receiving

node, even if the node did not store a corresponding copy of the message. This scheme has adaptive potential to adjust the forwarding mode of acknowledgements based on the congestion level. The scheme was simulated via ONE simulator and the results outperformed other algorithms such as active end-to-end acknowledgment and the passive end-to-end acknowledgment in terms of message delivery rate and queuing delay. However, due to the intermittent connectivity nature of the DTN network, the concept of acknowledgement experienced lagging leading to congestion identification mistakes.

Lo and Lu (2012) worked on dynamic congestion control based routing for DTNs consisted of both quota-replication routing and message moving techniques. The mechanism used meta-data approach to exchanged information among nodes consisted of neighbor's buffer occupancy table (NBOT) and contact probability table (CPT). In this work, individual node has the potential to keep the record of NBOT and CPT. the NBOT maintains the record of the present buffer occupancy of each neighboring node in the network that were within the same communication coverage. The CPT kept the records of contact probability of each node encountered before and updated them each time nodes encountered one another. Average buffer occupancy of the neighboring nodes was used as a local measurement for congestion level by referring to its NBOT. Different congestion levels were identified and their corresponding threshold defined. Performance evaluation was done via ONE simulator and the results showed that the proposed mechanism outperformed the storage routing and retired replicate mechanism. However, the settings for the different congestion level parameters were computationally time consuming and the relay cost was high under lightly loaded network.

Wang *et al.* (2012a) proposed active congestion control for DTN based on following routing algorithm to control node's movement. In the method, node routing evolved around cycle of

regular routing, following relationship establishment, relationship release and lastly regular routing. During cycle movement, the routing scheme processed the message setup, message delivery and deleting operation. The mechanism was simulated using ONE simulator. The results presented showed reduction in congestion in DTN and improved message successful delivery as compared with random routing algorithm. However, for the current node to follow the movement of next-hop node until a particular message was transmitted or deleted due to its time-to-live expired led to wastage of limited bandwidth resource, time and energy, since the current node lacked enough storage space to receive the message during the message forwarding process.

Wang *et al.* (2012b) described simulated annealing and regional movement congestion control routing approach based on the principle of message deletion operation to resolve the occurrence of congestion in DTN environment. Comprehensive survey of previous works on congestion control mechanisms was presented. In their work, regional characteristic of node movement and message delivery were taken into account. The simulated annealing algorithm leveraged which message to delete when congestion happened, considered existence period of message, regional characteristic of node movement, and message transit delivery. The regional characteristics constituted the cumulative number of encounter with other nodes in the process of node movement and determination of experienced transit node as well as destination node before successful delivery of message used qualitative approach. The realization of the algorithm and obtained optimization results for the mechanism was done via the simulated annealing. ONE simulator was used to simulate the proposed mechanism and the results showed reduction in network congestion ratio as well as improved message successful delivery ratio. However, the operational sequence of simulated annealing was time demanding and multi-tasking making the scheme to slowly respond to congestion and increase delivery delay.

Wang *et al.* (2012b) presented ant colony optimization techniques (ACO) based on the congestion control routing algorithm to address the problems of congestion in DTNs. In this scenario, message was regarded as an ant, the pheromone was attributed to massage generation and the transfer node random movement termed as the selected path by ant. The preying behavior of an ant was attributed to the DTN routing protocol, and the ratio of spare space to the node storage was used as the congestion guideline for heuristic value. The mechanism combined the concept of globalization of node encounter historical records with the congestion ratio of each node to select transfer node in message process. For each message successfully delivered to destination node, pheromone was added to each transfer node in a feedback mode and pheromone adjusted to reflect the ratio of current hop number from source node to destination node. The algorithm was evaluated using ONE simulator and results presented showed improvement in congestion ratio and message successful delivery ratio. However, the summation of concentration of pheromone and heuristic as congestion guidelines demanded much decision-making time, which makes the algorithm to respond to congestion untimely.

Zhang *et al.* (2014) proposed message deleting and transferring mechanism as a measure for congestion control in opportunistic network. In the work, the congested node computed the storage value of each message as a function of forwarding probability and message time-to-live. The message with the lowest storage value was deleted. The relay node calculated the receiving value of the message in line with the forwarding probability as well as its free buffer space prompting the congested node to transfer the message with maximum receiving value. The proposed algorithm was simulated through ONE simulator and the performance of the proposed algorithm was compared with Drop-oldest, Drop-front, Refuse to receive message algorithm in terms of arrival rate, end-to-end delay and communication overhead under different network

sizes. It outperformed all of them in term of message arrival rate. However, for the neighboring nodes to compute its receiving value and communicate back a reply message to the congested node before taking an appropriate action resulted in significant wastage of time thereby increased the network delay.

Akestoridis *et al.* (2014) introduced the congestion control with adjustable fairness (CCAF) for opportunistic networks. Normalized utility value and normalized residual space were exploited to determine the nodes as a relay node to forward the message, taking cognizance of the remaining storage space as the nodes contacted each other. This approach has the potential to dynamically adjust the tunable parameter through the social preference of the nodes to provide the trade-off between efficiency and fairness, which matched the desired network performance. Utility-based routing protocol was combined with the custody acceptance criteria for optimal fairness in the network, regardless of the offered traffic load level. The algorithm was evaluated through custom event-driven simulator and the obtained results outperformed the autonomous congestion control and Fair Route's congestion control module in terms of delay, delivery ratio and overhead ratio. However, whenever the utility value of the contacting nodes was the same it made forwarding decisions difficult, which in turn affected the network performance.

Kun *et al.* (2014) presented an improved socially aware congestion control algorithm (SACC) to address the issue of network congestion in DTN. In their method, social features and congestion level of nodes were utilized to formulate the social congestion metric. As congestion happened, message with minimum social link was dropped rather than random dropping. Several assumptions were made for the algorithm to work. In this scheme, social congestion metric was utilized to identify the forwarding capacity of the node and messages were forwarded to the nodes having higher social congestion metric. The message dropping strategy was designed to

drop messages with the least forwarded probability. The proposed SACC was simulated using ONE simulator and the results showed an improvement in delivery probability, decreased dropping probability, and reduced the overhead. However, this mechanism incurred longer delay as presented in the result, which adversely affected the overall network performance.

Wei *et al.* (2014) described congestion control strategy based on multiple attributes decision-making (MADM) and buffer management to address the DTN congestion problem. In their work, congestion control was modeled as a function of multiple attributes decision making in order to determine message forwarding set and its transmission order. They identified different congestion factors in DTN and exploited them in the design. An attribute matrix was developed consisted of congestion factors and message identity as row and column respectively. The matrix was normalized to discard irregularities and entropy computed. Diversification of entropy was determined and the weight for the different congestion factor computed using entropy method. Message utility value was utilized to determine the degree of congestion incurred by some message to a particular node. The effect of congestion of each message was stored in a specific node and evaluated by message utility metric. Utility- buffer management was used to drop the message having the least impact on the overall network performance. The performance evaluation of the proposed congestion control mechanism was carried out using ONE simulator and the results presented showed improvements in terms of delivery delay, delivery overhead and average delivery delay. However, the decision-making parameters such as weights of congestion factors, forwarding set determination and its transmission order and buffer management scheme require much computational time making the algorithm to respond to congestion very slowly.

Patil and Penurkar (2015) discussed congestion avoidance strategy epidemic (CASE) routing as a technique to resolve the issue of network congestion in DTN. Some assumptions were made, such that nodes other than destination node were creating messages; individual massage has unique identification number that can be generated by any node with varying size. The algorithm for the proposed mechanism was presented and message priority taken into account. Comparative analysis of AFNER, N-Drop and TBCC were outlined. The merits of congestion control using proactive approach over the reactive type were given. However, the proposed strategy was protocol dependent that replicated uncontrolled message copies in the network resulting into wastage of network-limited resources.

Silva *et al.* (2015) proposed directed site-bond percolation theory based DTN congestion model and studied the network behavior under congestion. In their work, the network was represented in graphical form (V, E) where V represented sites and E denoted edges of the graph otherwise called bond. In the model, they established that there existed a path between the source to destination node consisted of blocked sites, which represented DTN nodes whose buffers are congested and unblocked sites represented DTN nodes that are not congested. The active contact between DTN nodes was termed open bond. The model determined the node's buffer occupancy as an indication for congestion in the network. ONE simulator was used to simulate the model and the results showed that the DTN behavior under congestion was identical using both epidemic and first contact routing protocol and considered average buffer block time and average buffer occupancy. However, the model decision-making parameters took a lot of computation time, which eventually make the model's variables, and functions responded to congestion untimely.

Stewart *et al.* (2016) developed one-copy DTN protocol to address the issue of congestion, shortest path routing and operate effectively at high load network. The mechanism used cost estimate and least congested path to route packets from source node to destination node. In their work, the modeled path congestion and routing connectivity was characterized by convoluted parameters such as proximity measure and net destination queue waiting time by minimizing delivery cost were used. The mechanism was simulated using ONE simulator. The results presented showed maximization of packet delivery probability. However, the decision making parameters, which include proximity measure, net destination queue waiting time, and the modeled estimated delivery cost as exponential function demand much computational time thereby making the algorithm to respond to congestion untimely.

Silva *et al.* (2016) presented smart DTN congestion control (Smart-DTN-CC) technique to address congestion problem. In the work, Q-learning approach was employed which is a variant of reinforcement learning technique. They built representations of the node's state taken cognizance of the Q-values in congestion control decision making and allow node to use different action selection strategies. Buffer occupancy threshold was employed to determine the present congestion level and activate the respective state transition. ONE simulator was used to implement the smart-DTN-CC strategy and the results presented showed improvement in terms of average delivery ratio and latencies. However, the Q- learning approach required training period, and the decision-making parameters needed computational time to update the reward function for each transition made by node making the approach to respond to congestion untimely.

From the aforementioned limitations from the review of similar works and literature, it is evident that TBCC drops a good number of messages whenever the buffer overflows without any

consideration, which is the major limitation of the algorithm, which in turn affects the overall performance of the network. The mTBCC algorithm addressed this by rerouting the incoming message to the nearest congestion free node using adaptive forwarding mechanism, which takes into consideration of the available largest free buffer size and the least migration cost from the congested node.

CHAPTER THREE

METHODS

3.1 Introduction

In this chapter, the methods, and procedures used for the successful completion of this research are explained and the token-based congestion control algorithm TBCC and the modified token-based congestion control algorithm mTBCC were developed for opportunistic networks.

3.2 Methodology

The methodology adopted in this research is described as follows:

1. Replication of TBCC using the following steps:
    a. Initialization of token using queue size
    b. Mechanism to confirm source node encounters new neighbor(s)
    c. Establish that source node generates new message
    d. Determine whether the queue state of the node is full, if yes, drop the message. Otherwise, transmit the message to next best hop.
    e. Determine whether the message time-to-live timer expires, if yes, drop the message. Otherwise, transmit the message to next best hop.
2. Development of the mTBCC using the following steps:
    a. Repeat steps a-c of (1)
    b. Determine whether the queue state of the node is full; redirect the message to any intermediate node whose queue state is not full. If buffer is full for all nodes, drop the message.
    c. Repeat step 1(e)

3. Setting up the opportunistic network environment simulation scenario using ONE with Eclipse:
    a. Configuring the Java Virtual Machine and setting up the environment variable
    b. Setting the simulation workspace and compiling the ONE simulator.
4. Modeling of the opportunistic network using:
    a. Deployment of 60 mobile nodes in a Helsinki area test-bed (4500 x 3400m)
    b. Configuration of the nodes to be greatest connected components (GCC) inner nodes and greatest connected components (GCC) outside nodes
5. Simulation of the TBCC and mTBCC using ONE on the model of step(4) using the following:
    a. Setting up the simulation parameters.
    b. Run the simulation
    c. Obtain results
6. Comparison of the performance of the modified token-based congestion control algorithm with that of the conventional token-based congestion control algorithm in terms of dropped message and network transit time.

3.3 Replication of the token-based congestion control

The processes involved in the replication and development of TBCC for opportunistic network are detailed in the following sub-sections.

3.3.1 Initializing token based parameter

The performance of TBCC depends on the appropriate selection of its parameters (number of token, queue size, node density, buffer space, message size, migration cost, transmission cost and

storage cost). For the purpose of replication and implementation of TBCC algorithm, some of the token based parameters such as number of token, queue size, node density, message size and buffer space were used as contained in the work of Coe and Raghavendra (2010). The selection was reported to perform well. The node density was varied using GCC metric expressed in percentage. The amount of token put into the network is given in equation 2.1

3.3.2 Token based congestion control scenario settings

The use of token-based congestion control was called from the scenario setting of the ONE using the following code.

Scenario.name = Helsinki_TBCC_QS-100-salawu

This code triggers the graphical user interface of the ONE simulator as shown in Figure 3.1, showing the simulation process of the token-based congestion control mechanism in the ONE simulator. The program class was developed by incorporating the TBCC algorithm module on PRoPHET routing module as in (Appendix A)

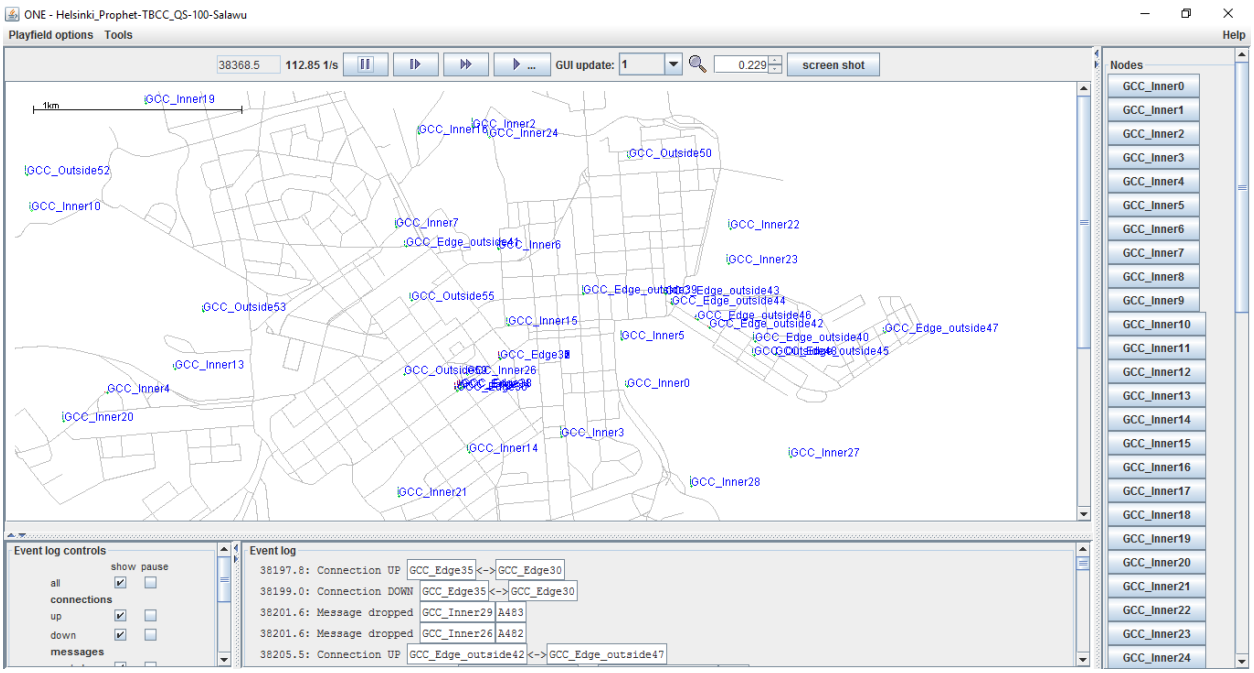


Figure 3.1: Simulation Process of Token Based Congestion Control

3.4 Development of a modified token-based congestion control

For the modified token-based congestion control algorithm, all the procedures itemized in methodology 3.2(a) to (c) of step 1of this work were repeated and thereafter step 1(e) was repeated after the modification has taken place. At the process of modification, available largest free buffer space and the least migration cost were introduced to effect efficient utilization of network resources and fairness during congestion. Figure 3.2 shows the flow chart implementation of the modified token-based congestion control algorithm.

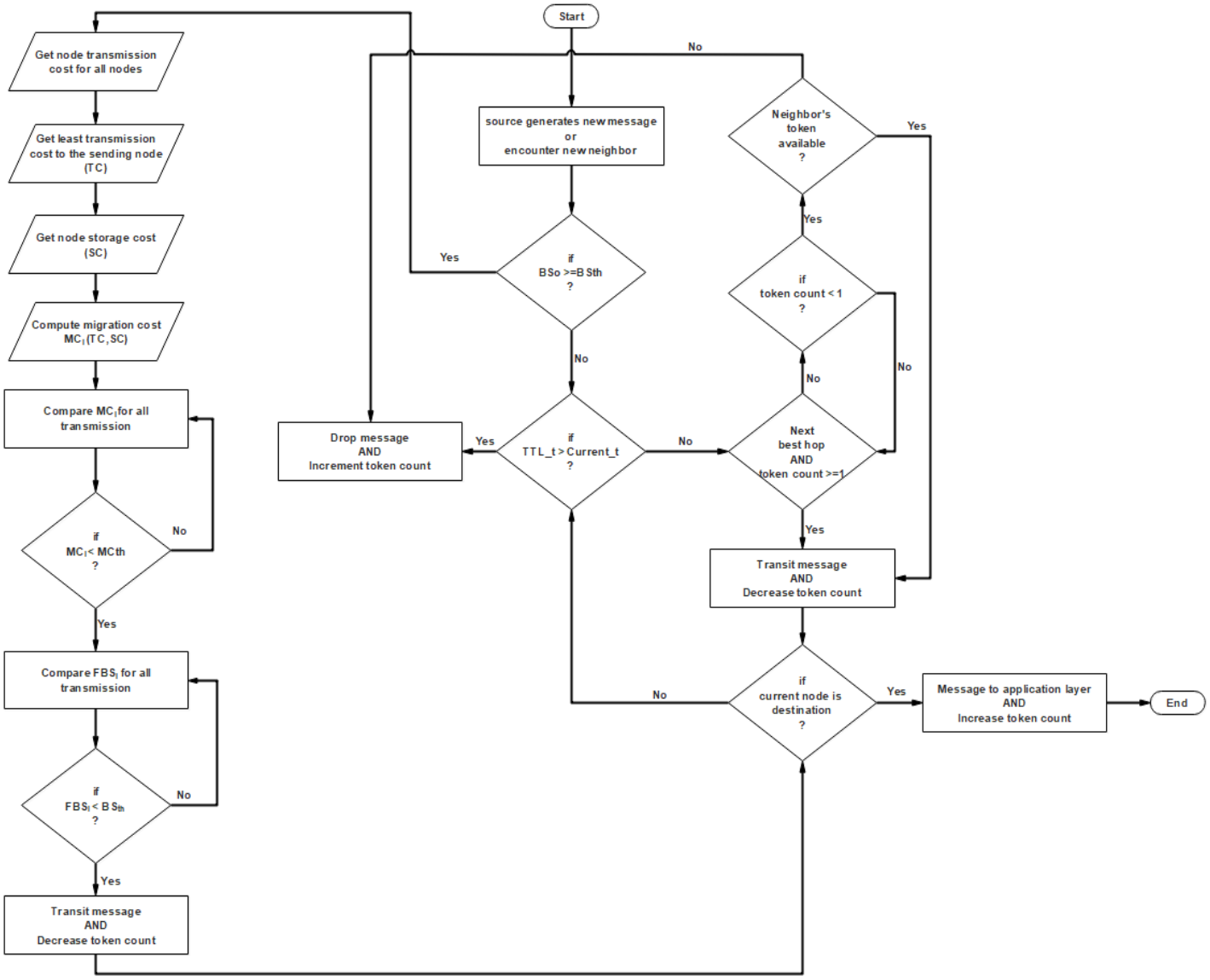


Figure 3.2: Flow chart for mTBCC

From Figure 3.2, the adaptive feature takes effects from the first decision box as illustrated by the arrow. It entails several steps, which include; getting node transmission cost for all nodes, obtain least node transmission cost to the sending node, get the node storage value, and compute migration cost. Compare migration cost for all nodes as well as free buffer size and take the necessary action.

3.4.1 Modified token based congestion control scenario settings

The leverage of modified token-based congestion control as the means to resolve congestion problem in Opportunistic Network was invoked from the scenario setting of the ONE through the following code:

Scenario.name = Helsinki_mTBCC_100-salawu

This code triggers the graphical user interface of the ONE simulator as shown in figure 3.3, showing the simulation process of the modified token-based congestion control mechanism in the ONE simulator. The event log provides an update of what happens to nodes at any period in the simulation. The program class was developed by incorporating the mTBCC algorithm module on prophet routing module as in (AppendixB).

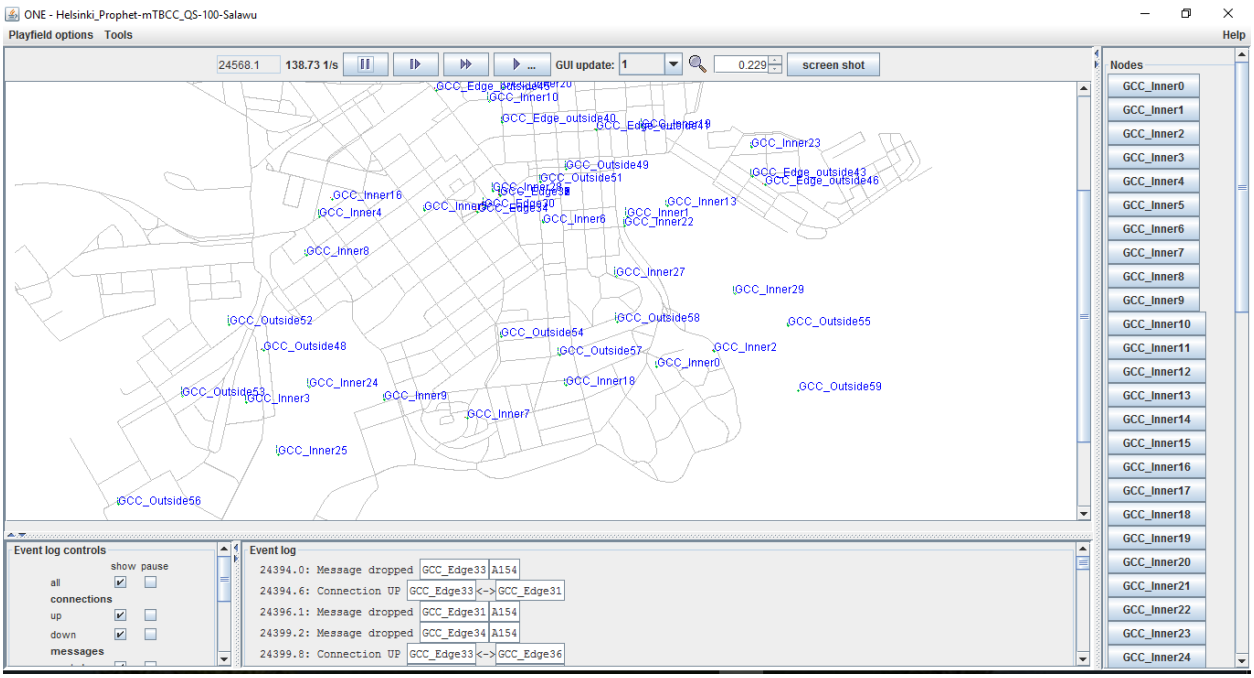


Figure 3.3: Simulation Process of Modified Token Based Congestion Control

3.5 Installation and Configuration

The zipped open source code of ONE (version ONE_1.5.1-RC) was downloaded from http://www.netlab.tkk.fi/tukkimus/dtn/theone/down/one_1.5.1-RC, extracted and compiled into java eclipse Integrated Development Environment (IDE). The simulator was configured through Eclipse IDE for java Developers downloaded from http://www.eclipse.org/downloads/ to debug and run the ONE project. Meanwhile, 'Netbeans' can equally be employed for the configuration of ONE simulator and the drawback it has is that it consumes too much of computer resources and possesses slower running speed than the Eclipse (Liu & Chen, 2013). Likewise, it can be downloaded from the link http;//www.netbeans.org/community/releases/73/index.html.

The downloaded eclipse IDE was unzipped to a folder to run the Eclipse application. Two important jars were imported into the ONE to compile which include DTNConsoleConnection.jar, and ECLA. Jar. The 'DTNConsoleConnection.jar' provides access for DTN to communicate with the simulated scenarios in DTN. ECLA.jar was employed to simulate network interface hardware and finish the simulation within the link layer.

3.6 Opportunistic network modelling

The network modeled in ONE consisted of sixty (60) mobile nodes. Data communication was assumed to occur between mobile phones and similar devices that possess up to 20MB RAM free to store the messages. Nodes were considered as pedestrians with speed of 4 to 5m/s moving into inner GCC and out of the inner GCC. Bluetooth transmission speed of 2MBps and a transmission range of 5meter was applied. The message size ranges between 500KB to 3MB. In addition, event generation ranges from 25-35 seconds and time-to-live of 5hours was used.

3.7 Simulation model

The Opportunistic Network was simulated through ONE simulator in accordance with the work of (Coe & Raghavendra, 2010) as a set of mobile nodes. The nodes are constrained to move randomly. The simulation environment consisted of sixty (60) mobile nodes in a Helsinki area of 4500 x3400m. Network connectivity was increased in terms of GCC expressed in percentage from 50% to 80%. The token-based congestion control algorithm and the modified token-based congestion control algorithm were simulated under the same scenario as in section 3.6

3.8 The PRoPHET routing protocol

The PRoPHET routing protocol was used to route packets amongst the relay nodes to the destination nodes, once the next best hop node has been discovered according to the routing protocol's requirements. Both conventional token-based congestion control and the modified token-based congestion control strategies were programed into the prophet routing protocol. The Appendices A and B illustrate the program class for TBCC and mTBCC, respectively. The scenario settings for the PRoPHET routing protocol is captured in Appendix C.

3.9 Visualization

The graphical user interface (GUI) provides real-time visualization of the simulation carried out as shown in Figures 3.1 and 3.3 respectively, which includes nodes location, number of messages carried by each node, connection among nodes, inner GCC and outer GCC nodes. It has provision for log of events in the simulation that can be selected depend on interest for further inspection. The entire simulation can be designed to exhibit pause function when interesting events happen. In addition, the visualization can be zoomed interactively.

3.10 Validation of the congestion control strategies

The same simulation environment applied to the conventional token-based congestion control algorithm was used to simulate the mTBCC algorithm. Many researchers in opportunistic networks use the Helsinki Region to validate results due to the fact that opportunistic network environment simulator originated from the region. A sample of such work include the work of (Keränen & Ott, 2007; Verma & Srivastava, 2012; Yahaya *et al.*, 2015). A section of Helsinki downtown area shown in Figure 3.4 was applied (4500 x 3400m). Data communication was assumed to take place between modern mobile telephones and similar devices (E.g. Personal digital assistant). Devices possess up to 20MB free of RAM for storing messages. Users travel on foot, trams and simulation was performed on 60 mobile nodes. Mobile nodes have the same speed as well as pause times. Inner GCC nodes and Outside GCC nodes move with random speed of 4.5 to 5m/s, observing a pause time of 0 to 120s. An assumption of 5m range, 2Mbits Bluetooth was made. Mobile users generate messages on average of once per minute per node and message time-to-live of 5hours was utilized. The message size applied was uniformly distributed between 500KB to 3MB.



Figure 3.4: Helsinki Region Overview (Yahaya *et al.*, 2015)

3.11 Performance evaluation

The performance of the conventional and the modified token-based congestion control algorithm was evaluated using the dropped message and network transit time. The dropped message was obtained from the Message Statistical Report folder of the ONE simulator and the network transit time computed using equation (2.4).

3.11.1 Percentage improvement

To show that modified token based significantly improved the results when compared with the conventional token-based congestion control, the percentage improvement was computed using:

$$\text{Performance improvement} = \frac{TBCC - mTBCC}{TBCC} X100\% \quad (3.1)$$

3.11.2 Relevant equations

The other relevant equations pertinent to this research include queue size, migration cost, buffer space threshold and network transit time have been discussed in equations 2.1 to 2.4 respectively.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

In this section, the performance of the token-based congestion control algorithm and that of the modified token-based congestion control mechanism for OppNets are discussed and relevant results reported. A number of screen shot describing the condition of the nodes during simulation are generated, presented and discussed. In addition, the effectiveness of the modified algorithm is also demonstrated via several data simulation and the results compared.

4.2 Result of the token-based congestion control strategy

The token-based congestion control strategy was implemented on the sixty mobile nodes test case. The network is segmented into inner GCC and the outside GCC. Whenever the outside GCC nodes come into the communication range of the inner GCC nodes, they exchange their delivery probability and buffer statistics. Nodes update their matrices based on historical encounter records of each node. The ONE simulator was configured to generate a report for all the simulations carried out and keep the results in a named directory by exploring the following command:

Report.nrofReports = 1

Report.reportDir = reports/Salawu/TBCC

Report.report1 = Message Stats Report

The ONE simulator keeps the results for the simulations in the directory called report/Salawu/TBCC. The summary of the results obtained for dropped message is shown in Table 4.1 and that of network transit time presented in Tables 4.2-4.6 respectively.

Table 4.1: Dropped Message Result for TBCC

Connected	QS-10	QS-20	QS-30	QS-40
50%	22893	17012	16292	14769
60%	11136	9402	8500	6190
70%	3389	2962	2762	1484
80%	1174	954	802	707

From Table 4.1, it can be seen that as the values of queue size increases from 10 to 40, a corresponding decrease in dropped messages is obtained for the different values of the GCC expressed in percentage increases from 50% to 80%.

The plot of dropped message against the GCC expressed in percentage using the Helsinki simulation region is depicted in Figure 4.1.

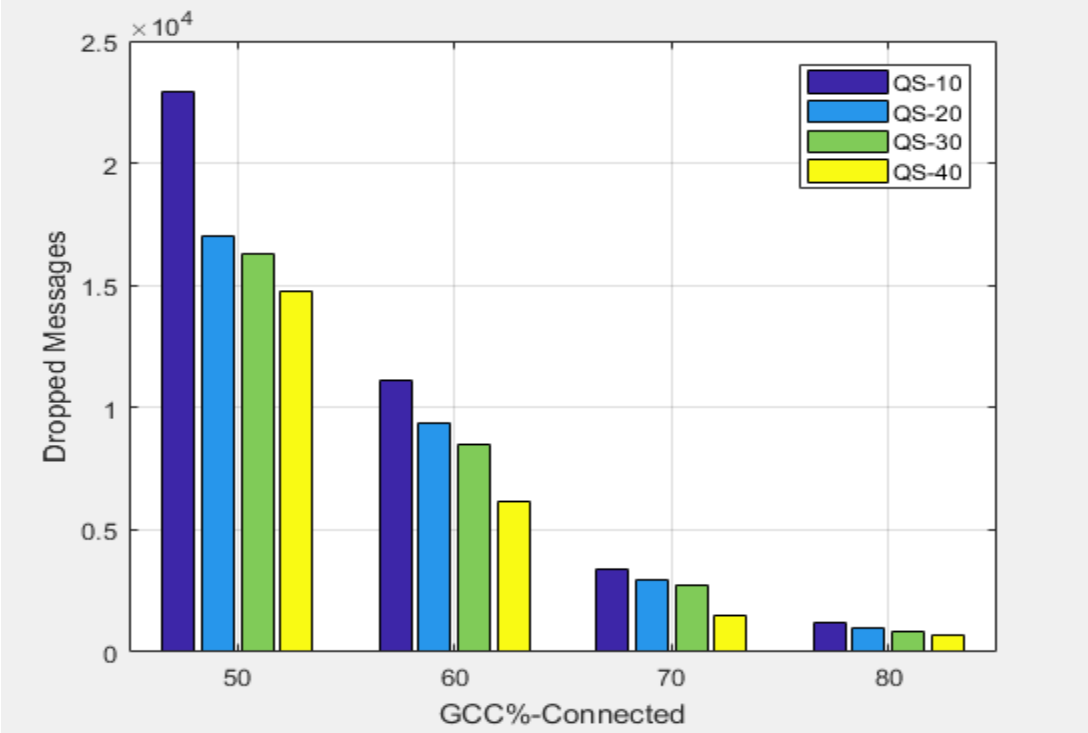


Figure 4.1: Dropped Message versus Percentage GCC for TBCC

From Figure 4.1, It can be deduced that as the network connectivity is increased in terms of GCC expressed in percentage from 50% to 80%, the amount of dropped messages are decreasing, regardless of the queue size.

Table 4.2: Network Transit Time Result for TBCC at 50% GCC

Initial Number of token/node	10	20	40	60	80	100
TBCC NTT(Secs)	2859.69	3466.32	4930.32	6048.83	6048.24	6048.30

Table4.3: Network Transit Time Result for TBCC at 60% GCC

Initial Number	10	20	40	60	80	100

of token/node						
TBCC NTT(Secs)	2501.51	2590.92	2750.34	2825.75	2825.62	2825.15

Table4.4: Network Transit Time Result for TBCC at 70% GCC

Initial Number of token/node	10	20	40	60	80	100
TBCC NTT(Secs)	1739.38	1860.30	2200.32	2459.41	2459.38	2459.42

Table 4.5: Network Transit Time Result for TBCC at 80% GCC

Initial Number of token/node	10	20	40	60	80	100
TBCC NTT(Secs)	1549.76	1735.95	2060.15	2329.08	2329.43	2329.18

Tables 4.2 to 4.5, illustrate that network transit time increases with the increase in token from 10 to 60, after which there is no significant changes in the network transit time at token of 80 and 100 because the network is saturated with token. The little differences observed in the network transit time decimal points for the token of 80 and 100 exist because of the self-fish node in the network.

Figures 4.2 to 4.5 are the plot of the network transit time from 50% to 80% GCC for TBCC with various values of initial number of token per node using Helsinki simulation environment.

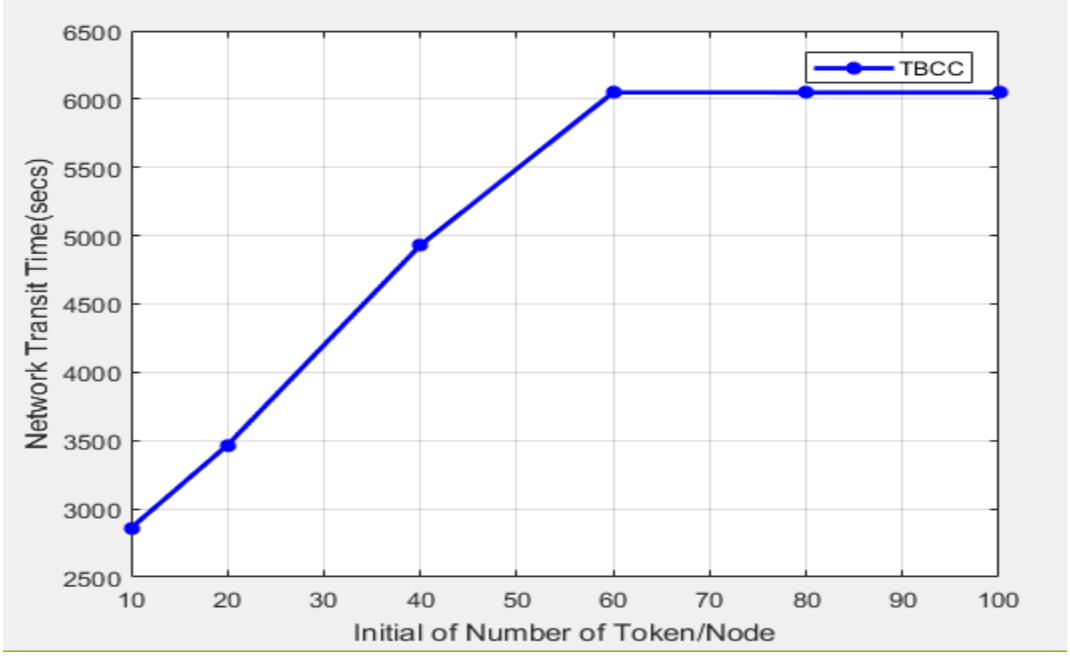


Figure 4.2: Network Transit Time versus Number of Token per Node for TBCC at 50% GCC

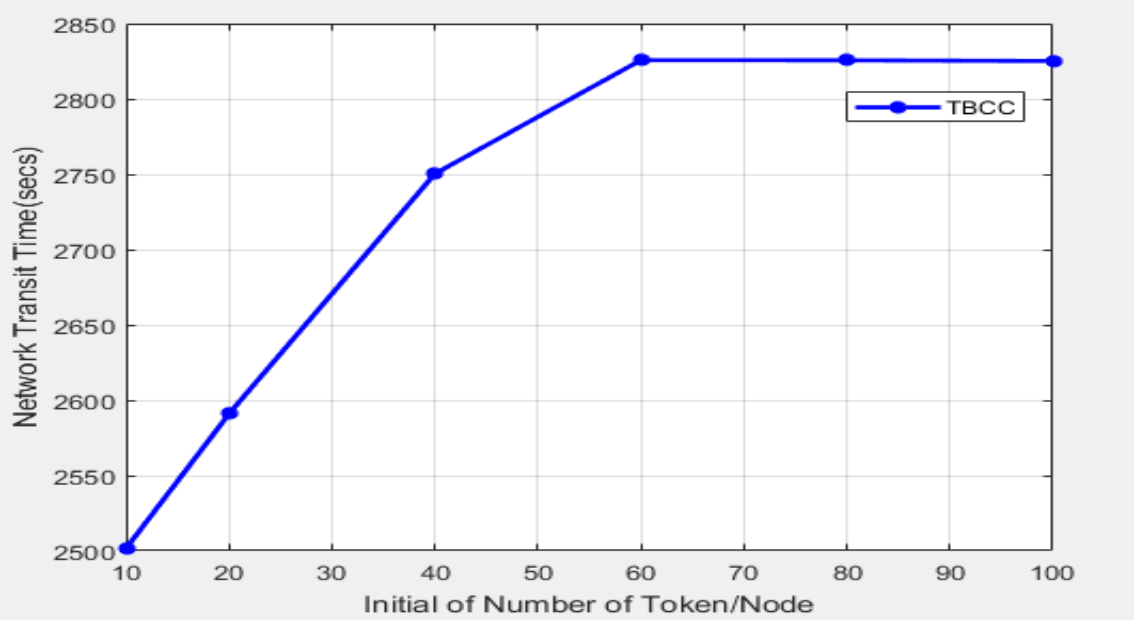


Figure 4.3: Network Transit Time versus Number of Token per Node for TBCC at 60% GCC

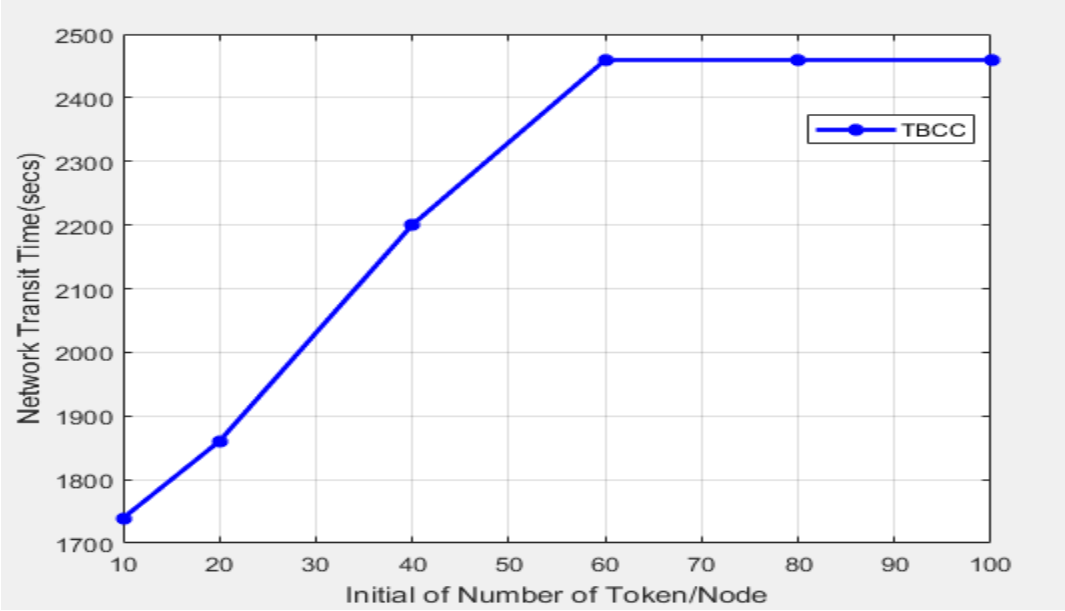


Figure 4.4: Network Transit Time versus Number of Token per Node for TBCC at 70% GCC

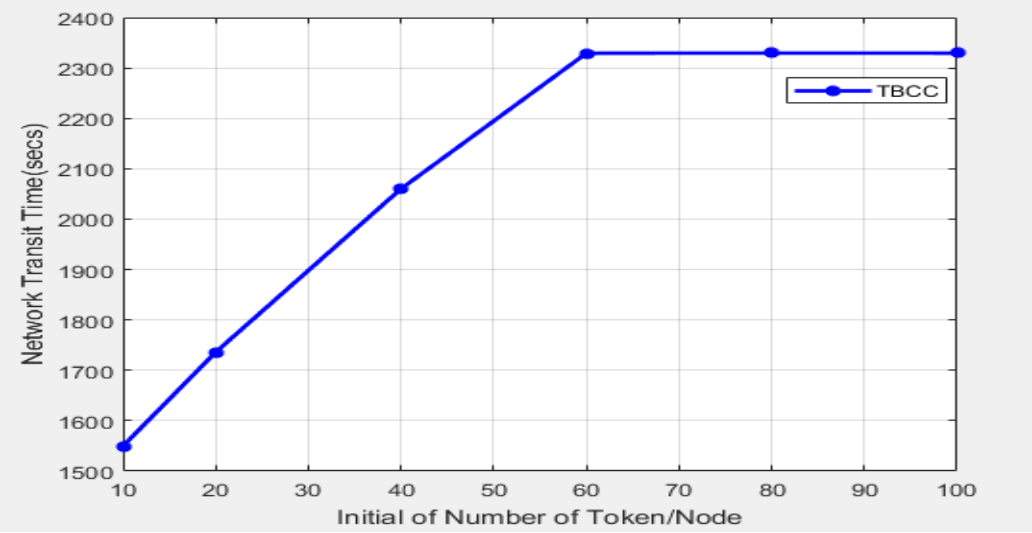


Figure 4.5: Network Transit Time versus Number of Token per Node for TBCC at 80% GCC

It is observed from Figure 4.2 to 4.5 that the network transit time increases linearly as token increases from 10 to 60, after which the network transit time becomes constant for the token of 80 and 100. These occur due to the saturation of the network with token.

4.3 Result of the modified token-based congestion control strategy

The modified token-based congestion control with adaptive forwarding strategy was implemented on the sixty mobile nodes test case. The network connectivity was increased from 50% to 80% in terms of GCC expressed in percentage. The network is segmented into inner GCC and the outside GCC. Whenever the outside GCC nodes come into the communication range of the inner GCC nodes, they exchange their delivery probability as well as buffer statistics. Nodes update their matrices based on historical encounter records of each node. The ONE simulator was configured to generate a report for all the simulations carried out and keep the results in a named directory by exploring the following command:

Report.nrofReports = 1

Report.reportDir = reports/Salawu/mTBCC

Report.report1 = Message Stats Report

The ONE simulator keeps the results for the simulations in the directory called report/Salawu/mTBCC. The summary of the results obtained for the dropped message are shown in Table 4.6 and that of network transit time presented in Tables 4.7-4.10 respectively.

Table 4.6: Dropped Message Result for mTBCC

Connected	QS-10	QS-20	QS-30	QS-40
50%	21504	15467	15031	14599
60%	10737	8828	8293	5460
70%	3037	2830	2733	1436
80%	756	720	710	702

From Table 4.6, it indicates that as the values of queue size increases from 10 to 40, it produces a corresponding decrease in dropped messages for the different values of GCC expressed in percentage from 50% to 80%.

Figure 4.6 depicts the plot of dropped message for the mTBCC algorithm using Helsinki simulation region.

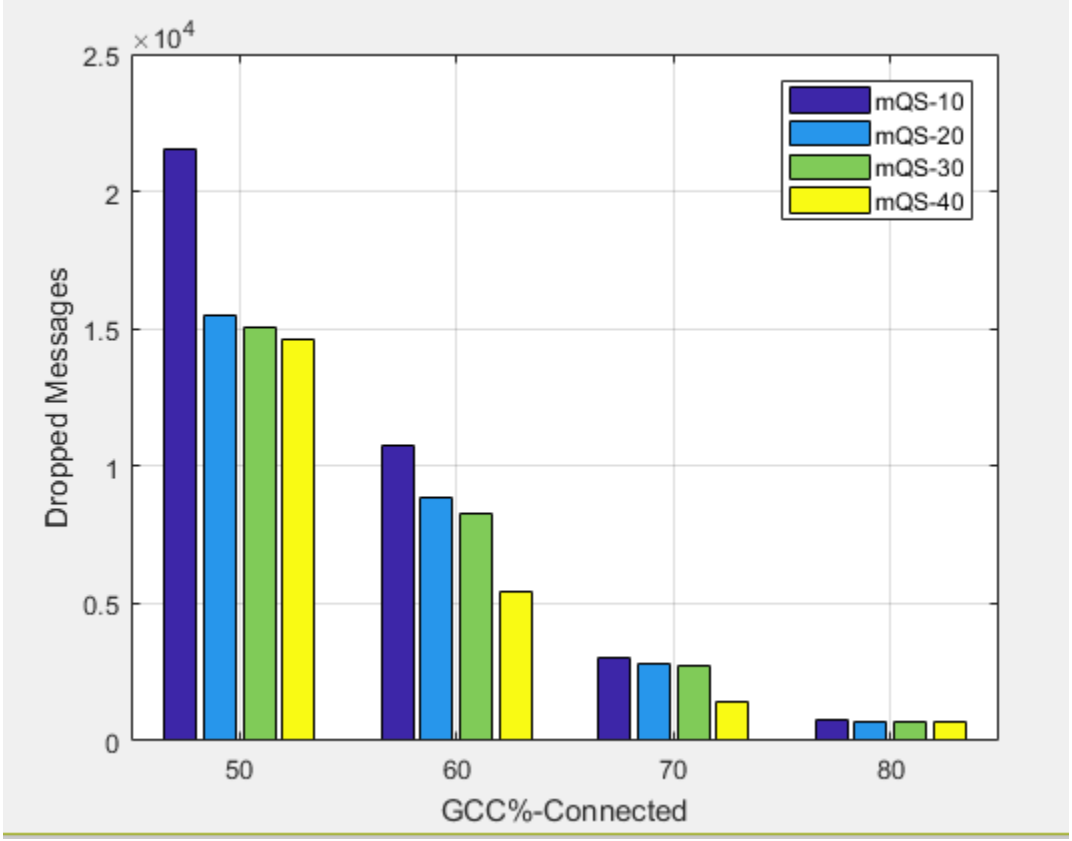


Figure 4.6: Dropped Message versus Percentage GCC for mTBCC

From Figure 4.6, It can be seen that as the GCC expressed in percentage increases network connectivity from 50% to 80%, the amount of dropped messages are in decreasing order, irrespective of the queue size.

Table 4.7: Network Transit Time result for mTBCC at 50% GCC

Initial Number of token/node	10	20	40	60	80	100
mTBCC NTT(Secs)	2566.49	3031.41	4406.23	5717.84	5717.57	5717.87

Table 4.8: Network Transit Time Result for the mTBCC at 60% GCC

Initial Number of token/node	10	20	40	60	80	100
mTBCC NTT(Secs)	2431.29	2510.18	2680.19	2781.61	2781.58	2781.57

Table 4.9: Network Transit Time Result for the mTBCC at 70% GCC

Initial Number of token/node	10	20	40	60	80	100
mTBCC NTT(Secs)	1669.66	1760.25	2010.28	2380.20	2380.34	2380.28

Table 4.10: Network transit Time Result for the mTBCC at 80% GCC

Initial Number of token/node	10	20	40	60	80	100

mTBCC NTT(Secs)	1339.91	1543.74	1950.54	2206.32	2206.61	2206.11

Tables 4.7 to 4.10, show that network transit time increases as token increases from 10 to 60, beyond which there is no substantial variation in the network transit time at token of 80 and 100, because the network is saturated with token. Further, the little differences observed in the network transit time decimal points for the token of 80 and 100 occur due to the effect of self-fish node in the network.

Figures 4.7 to 4.10 are the plot of the network transit time from 50% to 80% GCC for mTBCC algorithm with different values of initial number of token per node using Helsinki simulation environment.

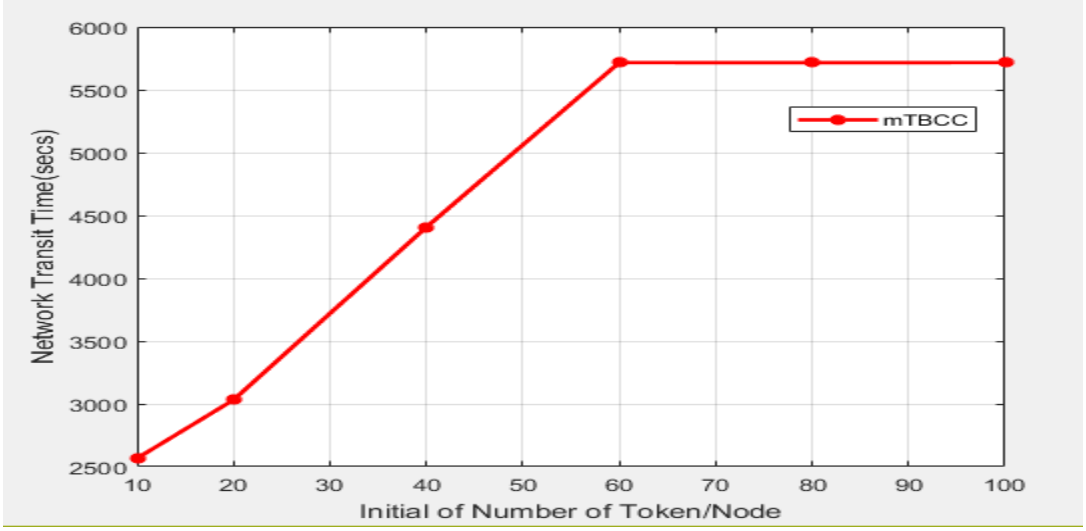


Figure 4.7: Network Transit Time versus Number of Token per Node for mTBCC at 50% GCC

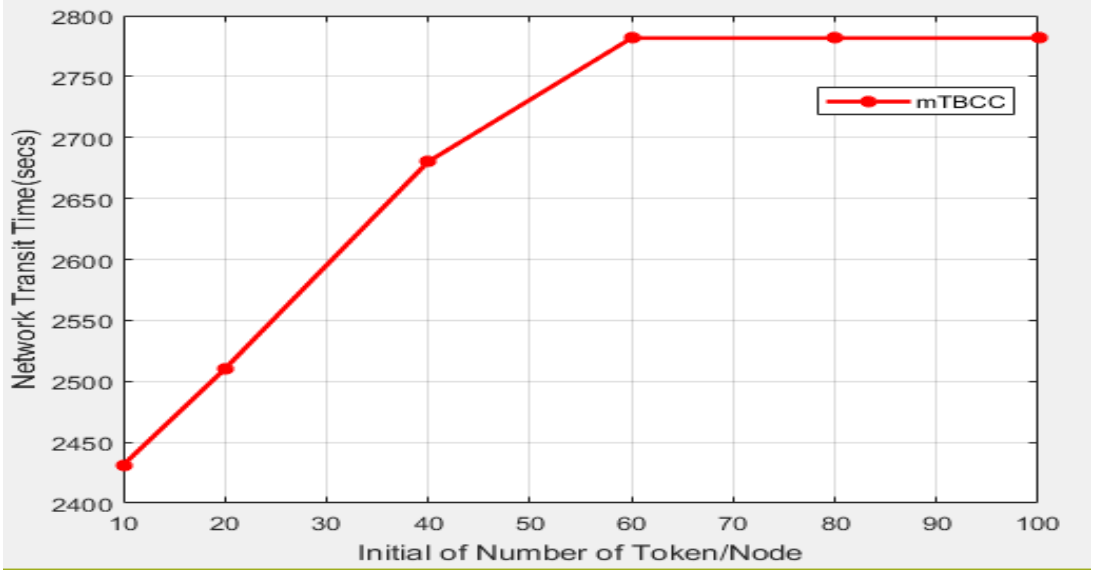


Figure 4.8: Network Transit Time versus Number of Token per Node for mTBCC at 60% GCC

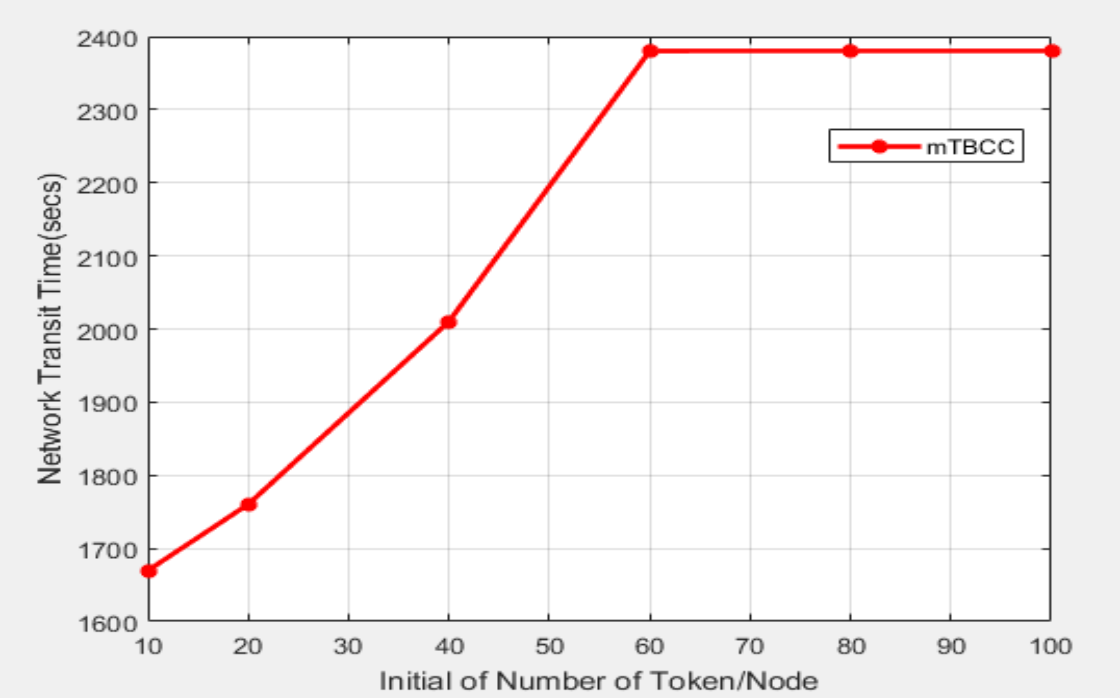


Figure 4.9: Network Transit Time versus Number of Token per Node for mTBCC at 70% GCC

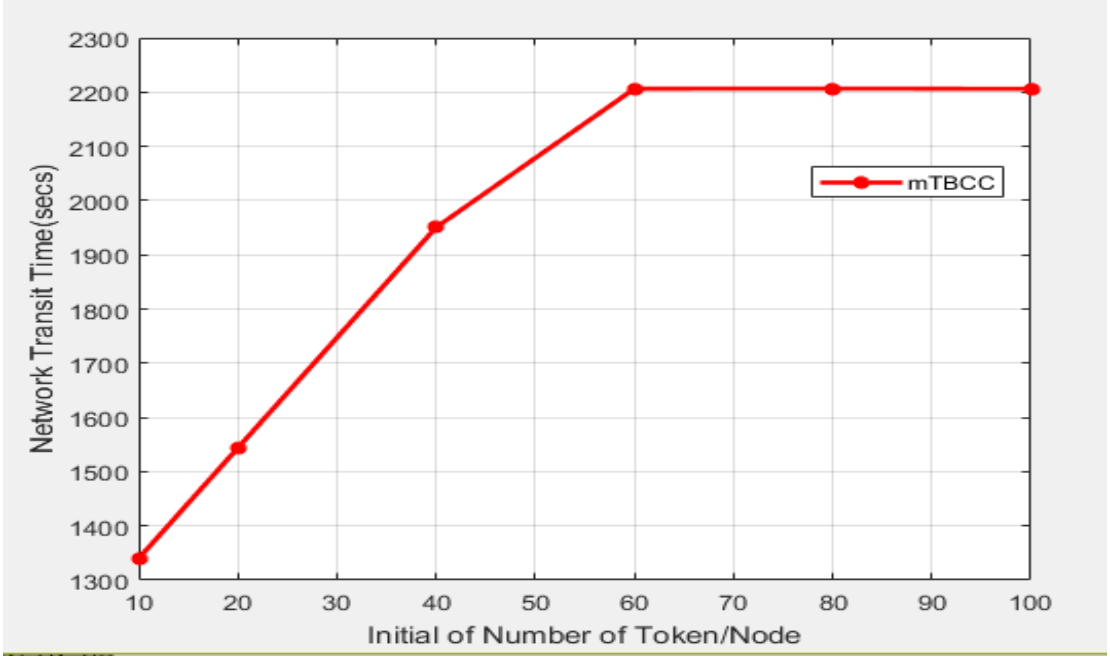


Figure 4.10: Network Transit Time versus Number of Token per Node for mTBCC at 80% GCC

It is evident from Figure 4.7 to 4.10 that the network transit time increases linearly as token increases from 10 to 60, after which it becomes constant for the token of 80 and 100, due to the saturation of the network with token.

4.4 Comparison of the results

The comparative analysis of the various performance index for both the modified token-based congestion control and the conventional token-based congestion control are presented as follow:

4.4.1 Comparison of mTBCC and TBCC performance for dropped message

Four different values of queue size were used to determine the queue state of the nodes as well as varying the GCC expressed in percentage for different values of network connectivity as shown in Table 4.11

Table 4.11 shows different dropped messages for different queue sizes of node at different values of network connectivity expressed in terms of GCC in percentage.

Table 4.11: Dropped message comparison for mTBCC and TBCC

connected	TBCC (QS-10)	mTBCC (QS-10)	TBCC (QS-20)	mTBCC (QS-20)	TBCC (QS-30)	mTBCC (QS-30)	TBCC (QS-40)	mTBCC (QS-40)
50%	22893	21504	17012	15467	16292	15031	14759	14599
60%	11136	10737	9402	8828	8500	8293	6190	5460
70%	3389	3037	2962	2830	2762	2733	1484	1436
80%	1174	759	954	720	802	710	707	702

It is observed in Table 4.11 that dropped message decreases as network connectivity increases in terms GCC expressed in percentage from 50% to 80% regardless of the queue size and mTBCC algorithm significantly reduced the amount of dropped messages as compared to the conventional TBCC algorithm

Figure 4.11 to 4.14 are the plots of comparison of dropped messages for different values of queue size for both mTBCC algorithm and conventional TBCC algorithm at different values of GCC expressed in percentage using Helsinki simulation environment.

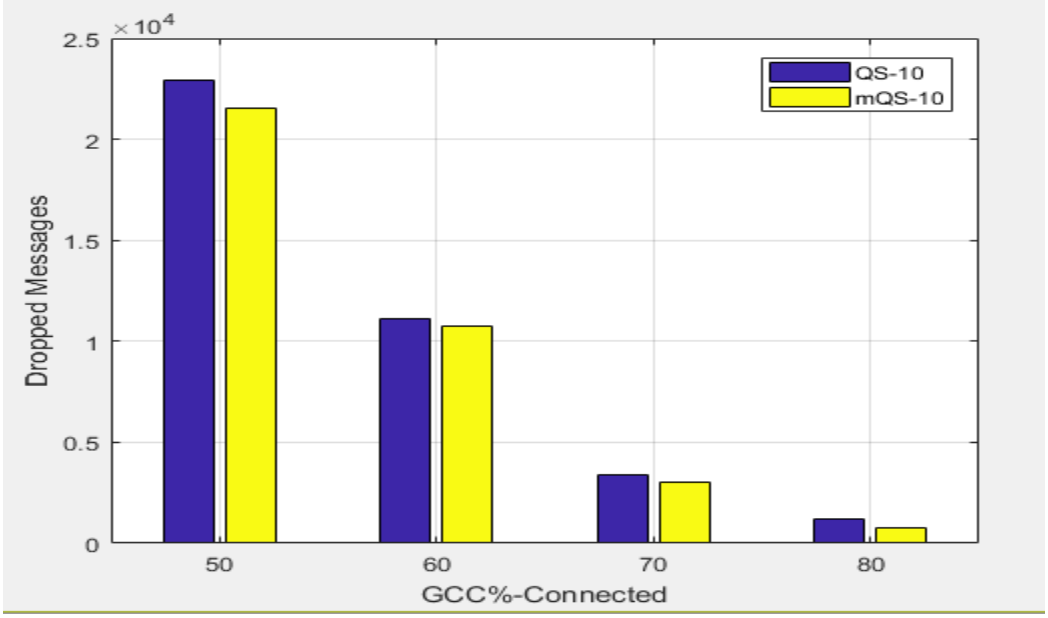


Figure 4.11: Comparison of Dropped Messages versus GCC for mTBCC and TBCC at (QS-10)

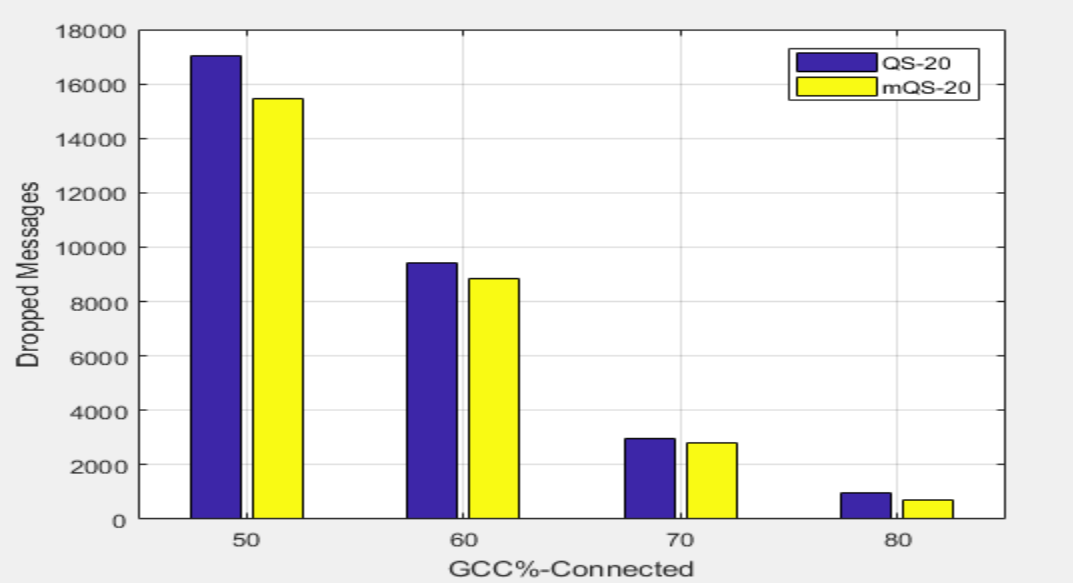


Figure 4.12: Comparison of Dropped Messages versus GCC for mTBCC and TBCC at (QS-20)

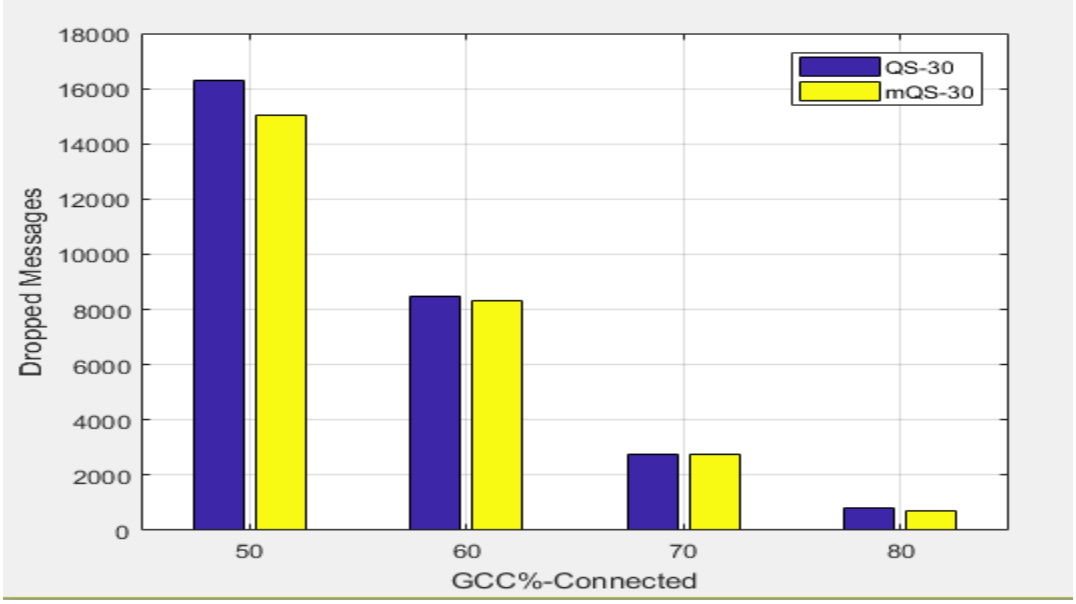


Figure 4.13: Comparison of Dropped Messages versus GCC for mTBCC and TBCC at (QS-30)

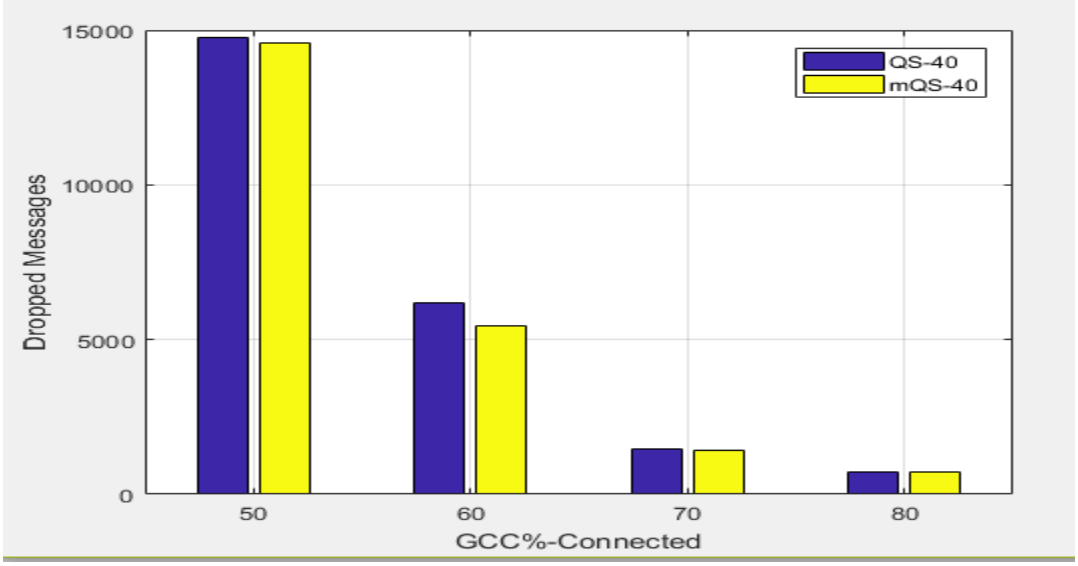


Figure 4.14: Comparison of Dropped Messages versus GCC for mTBCC and TBCC at (QS-40)

From Figure 4.11 to 4.14, it is observed from the bar charts that the modified token-based congestion control algorithm significantly minimized the number of dropped messages regardless of the queue size as compared with conventional token-based congestion control algorithm as the network connectivity is increased in terms of GCC expressed in percentage from

50% to 80%. This occurs due to the fact that mTBCC algorithm has effective resource utilization and fairness by rerouting the message from more congested to congestion free node of the OppNet.

4.4.1.1 Percentage improvement for dropped message

The percentage improvement for dropped messages for both modified token-based congestion control algorithm and the conventional token-based congestion control algorithm is presented in the Tables 4.12 to 4.15.

Table 4.12: Dropped message improvement of mTBCC over the TBCC at (QS-10)

Connected	QS-10	mQS-10	Dropped message Improvement (%)
50%	22893	21504	6.07
60%	11136	10737	3.58
70%	3389	3037	10.39
80%	1174	756	35.60

An average of 13.91 dropped message improvement was gotten over the conventional TBCC

Table 4.13: Dropped message improvement of mTBCC over the TBCC at (QS-20)

Connected	QS-20	mQS-20	Dropped message Improvement (%)
50%	17012	15647	8.02
60%	9402	8828	6.11
70%	2962	2830	4.46
80%	954	720	24.53

An average of 10.78% dropped message improvement was obtained over the conventional TBCC

Table 4.14: Dropped message improvement of mTBCC over the TBCC at (QS-30)

Connected	QS-30	mQS-30	Dropped message Improvement (%)
50%	16292	15031	7.74
60%	8500	8293	2.44
70%	2762	2733	1.05
80%	802	710	11.47

An average of 5.68% dropped message improvement was achieved over the conventional TBCC

Table 4.15: Dropped message improvement mTBCC over the TBCC at (QS-40)

Connected	QS-40	mQS-40	Dropped message Improvement (%)
50%	14769	14599	1.15
60%	6190	5460	11.79
70%	1484	1436	3.23
80%	707	702	0.71

An average of 4.22% dropped message improvement was obtained over the conventional TBCC

4.4.2 Comparison of mTBCC and TBCC performance for network transit time

Six different values of token per node were used to enable any node holding a valid token to inject message into the network at four different values of GCC expressed in percentage to increase the network connectivity as presented in the Table 4.16

Table 4.16: Comparison of Network transit time for mTBCC and TBCC

Initial number of token/node	50% GCC Connected		60% GCC Connected		70% GCC Connected		80% GCC Connected	
	TBCC (Secs)	mTBCC (Secs)	TBCC (Secs)	mTBCC (Secs)	TBCC (Secs)	mTBCC (Secs)	TBCC (Secs)	mTBCC (Secs)
10	2859.69	2566.49	2501.51	2431.29	1739.38	1669.66	1549.76	1339.91
20	3466.32	3031.41	2590.92	2510.18	1860.30	1760.25	1735.95	1543.74
40	4930.32	4406.23	2750.34	2680.19	2200.32	2010.28	2060.15	1950.54
60	6048.83	5715.84	2825.75	2781.61	2459.41	2380.20	2329.08	2206.32
80	6048.24	5715.57	2825.62	2781.58	2459.38	2380.34	2329.43	2206.61
100	6048.30	5715.87	2825.15	2781.57	2459.42	2380.28	2329.18	2206.11

Figure 4.15 to 4.18 are the plots of the network transit time from 50% to 80% GCC for different Initial number of token per node using Helsinki simulation environment.

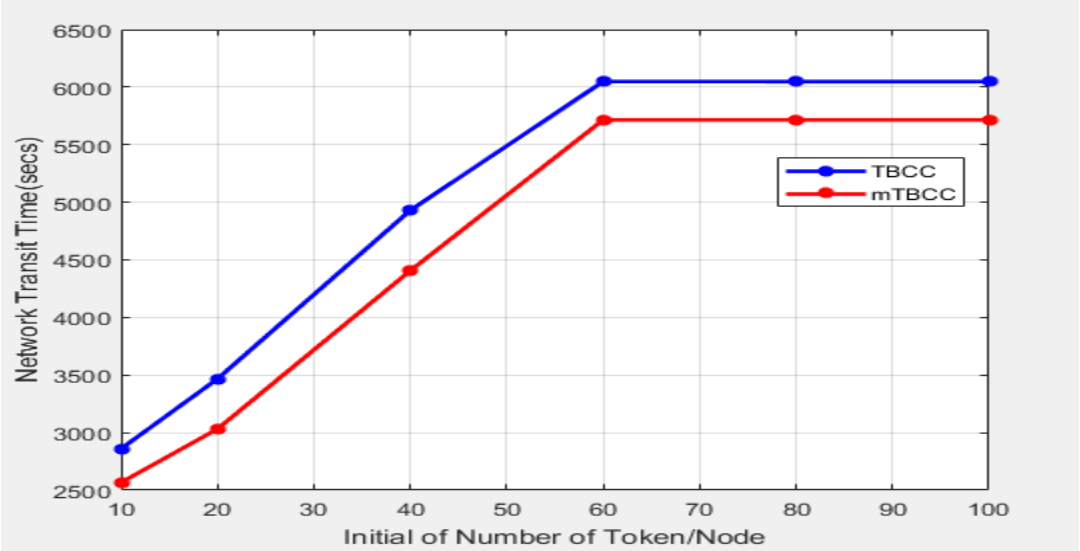


Figure 4.15: Network Transit Time versus Initial Number of Token/Node at 50% GCC

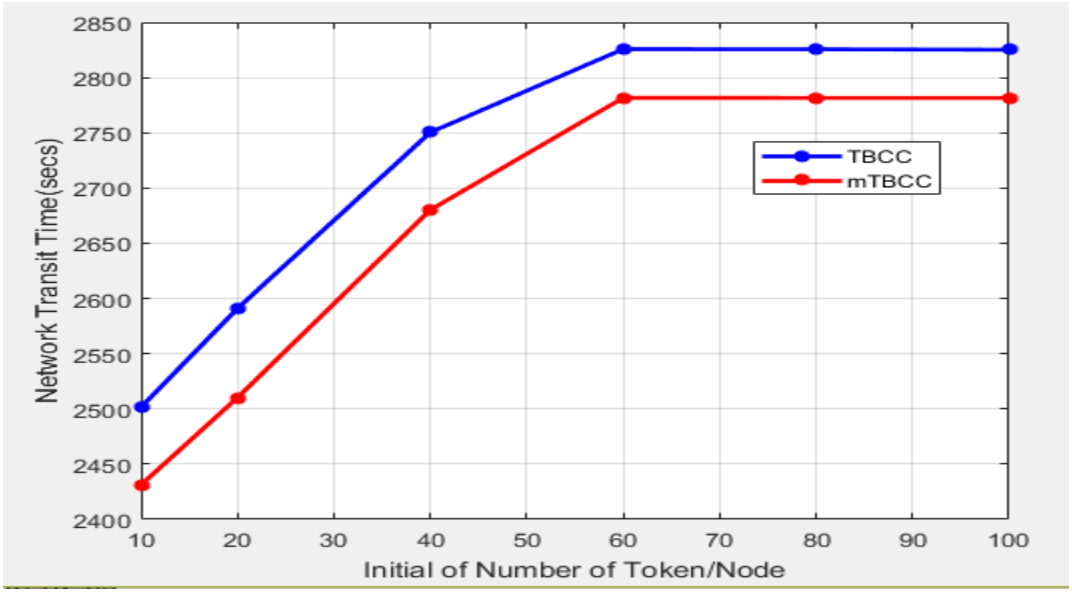


Figure 4.16: Network Transit Time versus Initial Number of Token/Node at 60% GCC

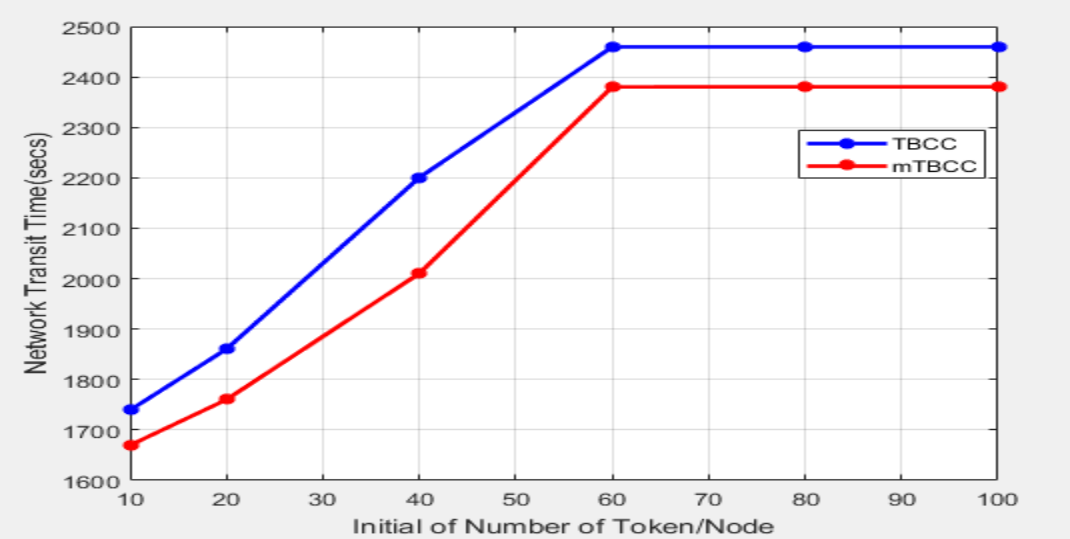


Figure 4.17: Network Transit Time versus Initial Number of Token/Node at 70% GCC

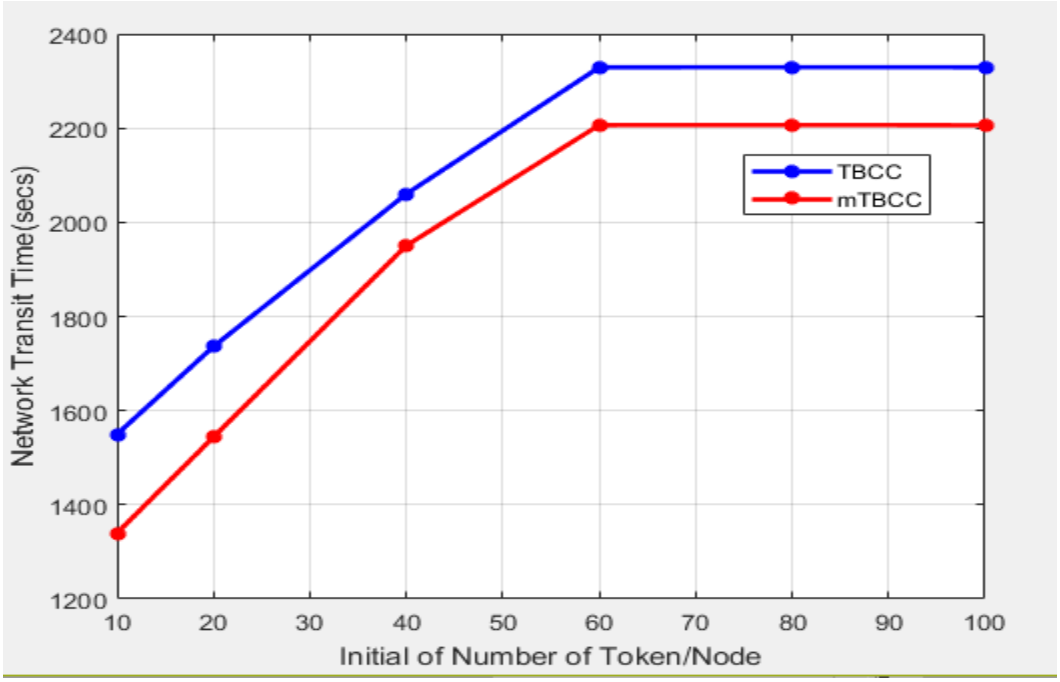


Figure 4.18: Network Transit Time versus Initial Number of Token/Node at 80% GCC

From Figure 4.15 to 4.18, it is seen that the network transit time for both algorithms increases linearly as token increases from 10 to 60 and becomes constant for both algorithms at token of 80 and 100, due to the saturation of the network with token. Further the mTBCC algorithm

significantly reduced the network transit time as compared with the conventional TBCC algorithm and this is attributed to the fact mTBCC algorithm has the potential to redirect the message from the congested node to congestion free node of the OppNet, by enhancing effective resource utilization and fairness in the network.

4.4.3 Network transit time percentage improvement

The percentage improvement for the network transit time for both modified token-based congestion control algorithm and the conventional token-based congestion control algorithm is presented in Table 4.17 – 4.20 respectively.

Table 4.17: Network Transit Time improvement of mTBCC over the TBCC at 50% GCC

Initial number of token/node	TBCC(secs)	mTBCC (secs)	NTT Improvement (%)
10	2859.69	2566.49	10.25
20	3466.32	3031.41	12.55
40	4930.32	4406.23	10.63
60	6048.83	5715.84	5.51
80	6048.24	5715.57	5.50
100	6048.30	5715.87	5.50

An average of 8.34% network transit time improvement was obtained over the standard TBCC

Table 4.18: Network Transit Time improvement of mTBCC over the TBCC at 60% GCC

Initial number of token/node	TBCC(secs)	mTBCC (secs)	NTT Improvement (%)
10	2501.51	2431.29	2.81
20	2590.92	2510.18	3.12
40	2750.34	2680.19	2.55
60	2825.75	2781.61	1.56
80	2825.62	2781.58	1.56
100	2825.15	2781.57	1.56

An average of 2.19% network transit time improvement was obtained over the standard TBCC

Table 4.19: Network Transit Time improvement of mTBCC over the TBCC at 70% GCC

Initial number of token/node	TBCC(secs)	mTBCC (secs)	NTT Improvement (%)
10	1739.38	1669.66	4.01
20	1860.30	1760.25	5.38
40	2200.32	2010.28	8.64
60	2459.41	2380.20	3.22
80	2459.38	2380.34	3.21
100	2459.42	2380.28	3.22

An average of 4.61% network transit time improvement was obtained over the standard TBCC

Table 4.20: Network Transit Time improvement of mTBCC over the TBCC at 80% GCC

Initial number of token/node	TBCC(secs)	mTBCC (secs)	NTT Improvement (%)
10	1549.76	1339.91	13.54
20	1735.95	1543.74	11.07
40	2060.15	1950.54	5.32
60	2329.08	2206.32	5.27
80	2329.43	2206.61	5.27
100	2329.18	2206.11	5.28

An average of 7.63% network transit time improvement was obtained over the standard TBCC

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 Summary

This research is aimed at the development of a modified token-based congestion control scheme with adaptive forwarding for addressing the problem of dropped messages in OppNets. Results obtained for both dropped messages and network transit time at different network connectivity via the GCC metrics expressed in percentage showed substantial minimization in the number of dropped messages and network transit time when compare with the conventional TBCC algorithm.

5.2 Conclusion

The modified token-based scheme was developed by hybridizing adaptive forwarding into the conventional token-based congestion control mechanism. The modified scheme was simulated and its results obtained compared with the conventional token-based scheme based the Helsinki simulation region. The results showed that dropped message was reduced by 13.91% at QS-10, 10.78% at QS-20, 5.68% at QS-30 and 4.22% at QS-40 respectively. In addition 8.34% at 0.5 GCC, 2.19% at 0.6 GCC, 4.61% at 0.7GCC, and 7.63% at 0.8GCC respectively reduction in network transit time, when compared with the conventional token-based congestion control using Helsinki Region. Which indicates improvement as proposed.

5.3 Limitation

The limitations associated to this research work are enumerated below:

i. Map importation into ONE simulator was a challenged, therefore, I am constrained to use the Helsinki default map.

ii. Nodes are constrained to land movement therefore flying nodes are challenged like aircraft.

iii. Due to overwhelming of buffer size for epidemic routing protocol, hence, I am constrained to use PRoPHET routing protocol.

5.4 Significant contributions

The significant contributions of this research work are itemized as follow:

i. Hybridization of token-based congestion control with adaptive forwarding mechanism called modified token-based congestion control for OppNets.

ii. The developed mTBCC showed an average improvement of 13.91% at QS-10, 10.78% at QS-20, 5.68% at QS-30 and 4.22% at QS-40 respectively over the conventional TBCC for dropped messages. In addition, 8.34% at 0.5 GCC, 2.19% at 0.6 GCC, 4.61% at 0.7 GCC, and 7.63% at 0.8 GCC respectively for network transit time, when compared with the conventional token-based congestion control using Helsinki Region.

5.5 Recommendations for further work

The following areas are recommended for further works in future:

i. Consideration of dynamic creation of token and deletion based on localized congestion detection.

ii. Take effects of selfish nodes into account in message delivery process.

iii. Extensive research efforts can be made in determining the OppNets capacity.

REFERENCES

Akestoridis, D.-G., Papanikos N., & Papapetrou E. (2014). Exploiting social preferences for congestion control in opportunistic networks. Paper presented at the IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2014

Akyildiz, I. F., Su W., Sankarasubramaniam Y., & Cayirci E. (2002). A survey on sensor networks. IEEE Communications Magazine, 40(8), 102-114.

An, Y., Huang J., Song H., & Wang J. (2012). A Congestion Level based end-to-end acknowledgement mechanism for Delay Tolerant Networks. Paper presented at the IEEE Global Communications Conference (GLOBECOM), 2012.

Balasubramanian, A., Levine B., & Venkataramani A. (2007). DTN routing as a resource allocation problem. Paper presented at the ACM SIGCOMM computer communication review.

Burleigh, S., Hooke A., Torgerson L., Fall K., Cerf V., Durst B., Scott K., & Weiss H. (2003). Delay-tolerant networking: an approach to interplanetary internet. IEEE Communications Magazine, 41(6), 128-136.

Burleigh, S., Jennings E., & Schoolcraft J. (2006). Autonomous congestion control in delay-tolerant networks: Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration.

Coe, E., & Raghavendra C. (2010). Token based congestion control for DTNs. Paper presented at the IEEE Aerospace Conference, 2010.

Das, R., Prodhan M. A. T., Kabir M. H., & Shoja G. C. (2011). A novel congestion control scheme for delay tolerant networks. Paper presented at the International Conference on Selected Topics in Mobile and Wireless Networking (iCOST), 2011.

Dhurandher, S. K., Sharma D. K., Woungang I., & Chao H.-C. (2011). Performance evaluation of various routing protocols in opportunistic networks. Paper presented at the IEEE GLOBECOM Workshops (GC Wkshps), 2011

Fall, K. (2003). A delay-tolerant network architecture for challenged internets. Paper presented at the Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications.

Grundy, A. (2012). Congestion control framework for delay-tolerant communications. University of Nottingham.

Guo, S., Falaki M. H., & Oliver E. A. (2007). Sumair Ur Rahman, Aaditeshwar Seth, Matei A. Zaharia, Usman Ismail, and Srinivasan Keshav. Design and implementation of the kiosknet system. Paper presented at the Proceedings of the International Conference on Information and Communication Technologies and Development (ICTD 2007).

Keränen, A., & Ott J. (2007). Increasing reality for dtn protocol simulations. Helsinki University of Technology, Tech. Rep.

Kun, W., Huang G., Lei S., & Bo L. (2014). An improved congestion control algorithm based on social awareness in Delay Tolerant Networks. Paper presented at the IEEE International Conference on Communications (ICC),2014.

Li, Y., Zhao L., Liu Z., & Liu Q. (2009). N-Drop: congestion control strategy under epidemic routing in DTN. Paper presented at the Proceedings of the 2009 international conference on wireless communications and mobile computing: connecting the world wirelessly.

Lin, X., Lu R., Zhang C., Zhu H., Ho P.-H., & Shen X. (2008). Security in vehicular ad hoc networks. IEEE Communications Magazine, 46(4), 88-95.

Lindgren, A., Doria A., & Schelén O. (2003). Poster: Probabilistic routing in intermittently connected networks. Paper presented at the Proceedings of The Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2003).

Liu, X., & Chen Y. (2013). Report of A DTN Simulator-THE ONE: no.

Lo, S.-C., & Lu C.-L. (2012). A dynamic congestion control based routing for delay-tolerant networks. Paper presented at the 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), 2012.

Lu, R., Lin X., Zhu H., Ho P.-H., & Shen X. (2008). ECPP: Efficient conditional privacy preservation protocol for secure vehicular communications. Paper presented at the INFOCOM 2008. The 27th Conference on Computer Communications. IEEE

Marcus, W. S., Hadzic I., McAuley A. J., & Smith J. M. (1998). Protocol boosters: Applying programmability to network infrastructures. IEEE Communications Magazine, 36(10), 79-83.

Nelson, S. C., Bakht M., Kravets R., & Harris III A. F. (2009). Encounter: based routing in DTNs. ACM SIGMOBILE Mobile Computing and Communications Review, 13(1), 56-59.

Partan, J., Kurose J., & Levine B. N. (2007). A survey of practical issues in underwater networks. ACM SIGMOBILE Mobile Computing and Communications Review, 11(4), 23-33.

Patil, P., & Penurkar M. (2015). Congestion avoidance and control in Delay Tolerant Networks. Paper presented at the International Conference on Pervasive Computing (ICPC), 2015.

Piórkowski, M., Sarafijanovic-Djukic N., & Grossglauser M. (2008). On clustering phenomenon in mobile partitioned networks. Paper presented at the Proceedings of the 1st ACM SIGMOBILE workshop on Mobility models.

Pujol, J. M., Toledo A. L., & Rodriguez P. (2009). Fair routing in delay tolerant networks. Paper presented at the INFOCOM 2009, IEEE.

Radenkovic, M., & Grundy A. (2012). Efficient and adaptive congestion control for heterogeneous delay-tolerant networks. Ad Hoc Networks, 10(7), 1322-1345.

Seligman, M., Fall K., & Mundur P. (2007). Storage routing for DTN congestion control. Wireless communications and mobile computing, 7(10), 1183-1196.

Silva, A. P., Hil M. R., Hirata C. M., & Obraczka K. (2015). A Percolation-Based Approach to Model DTN Congestion Control. Paper presented at the IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), 2015

Silva, A. P., Obraczka K., Burleigh S., & Hirata C. M. (2016). Smart Congestion Control for Delay-and Disruption Tolerant Networks. Paper presented at the 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), 2016.

Stewart, M. F., Kannan R., Dvir A., & Krishnamachari B. (2016). CASPaR: Congestion avoidance shortest path routing for delay tolerant networks. Paper presented at the 2016 International Conference on Computing, Networking and Communications (ICNC).

Sun, W., Liu C., & Wang D. (2011). On Delay-Tolerant Networking and Its Application. Paper presented at the Proceedings of 2011 International Conference on Computer Science and Information Technology (ICCSIT 2011).

SuvarnaPatil, G., & Chillerge R. (2014). Delay Tolerant Networks–Survey Paper. Int. Journal of Engineering Research and Applications, 4(2).

Thompson, N., Nelson S. C., Bakht M., Abdelzaher T. F., & Kravets R. (2010). Retiring Replicants: Congestion Control for Intermittently-Connected Networks. Paper presented at the INFOCOM.

Tournoux, P.-U., Leguay J., Benbadis F., Conan V., De Amorim M. D., & Whitbeck J. (2009). The accordion phenomenon: Analysis, characterization, and impact on dtn routing. Paper presented at the INFOCOM 2009, IEEE.

Verma, A., & Srivastava D. (2012). Integrated routing protocol for opportunistic networks. arXiv preprint arXiv:1204.1658.

Vidya, K., & Hemanth S. (2014). Routing and congestion control strategies in opportunistic networks: A survey. International Journal For Technical Research In Engineering. 2 (11), 2347, 4718.

Voyiatzis, A. (2012). A survey of delay-and disruption-tolerant networking applications. Journal of Internet engineering, 5(1).

Wang, C., Zhao B., Peng W., Wu C., & Gong Z. (2012a). Following routing: an active congestion control approach for delay-tolerant networks. Paper presented at the 15th International Conference on Network-Based Information Systems,2012

Wang, C., Zhao B., Peng W., Wu C., & Gong Z. (2012b). Routing algorithm based on ant Colony Optimization for DTN Congestion Control. Paper presented at the 15th International Conference on Network-Based Information Systems,2012.

Wei, K., Guo S., Zeng D., & Xu K. (2014). A multi-attribute decision making approach to congestion control in delay tolerant networks. Paper presented at the 2014 IEEE International Conference on Communications (ICC).

Yahaya, B., Mu M., & Garba S. (2015). Congestion control strategies on integrated routing protocol for the opportunistic network: a comparative study and performance analysis. International Journal of Computer Applications, 117(4).

Yang, Y., Han L., Xu W., Kong X., & Yan R. (2012). An advanced congestion control mechanism based on distributed storage for DTN. Paper presented at the Computer Science & Service System (CSSS), 2012 International Conference on.

Yun, L., Xinjian C., Qilie L., & Xiaohu Y. (2010). A novel congestion control strategy in delay tolerant networks. Paper presented at the ICFN'10. Second International Conference on Future Networks, 2010.

Zhang, D.-y., Yang M., & Cui L. (2014). Congestion control strategy for opportunistic network based on message values. Journal of Networks, 9(2), 1132-1138.

Zhang, X., Kurose J., Levine B. N., Towsley D., & Zhang H. (2007). Study of a bus-based disruption-tolerant network: mobility modeling and impact on routing. Paper presented at the Proceedings of the 13th annual ACM international conference on Mobile computing and networking.

Zhang, Y., & Bai X. (2015). Congestion control mechanism in Delay Tolerant Networks. Paper presented at the 5th International Conference on Electronics Information and Emergency Communication (ICEIEC), 2015

http://www.netlab.tt.fi/tutkimus/dtn/theone/down/one_1.5.1-RC Zip

http://www.eclipse.org/downloads/, http://www.netbeans.org/community/releases/73/index.html

APPENDIX A

Program class for incorporating TBCC algorithm on prophet routing protocol

```
/** SHAIBU IDRIS SALAWU
* AHMADU BELLO UNIVERSITY ZARIA
 * TBCC_PROPHET ROUTING IMPLEMENTATION */
package routing;
public class TBCC_ProphetRouter extends ActiveRouter {
        public static final double PEncMax = 0.1;
        public static double PEnc = 0.2;
        public static fiEnal double P_INIT = 0.3;
public static final double I_TYP = 1000;
        public static final double DEFAULT_BETA = 0.9;
        public static final double GAMMA = 0.98;
        Random randomGenerator = new Random();
        public static final String PROPHET_NS = "TBCC_ProphetRouter";
        public static final String SECONDS_IN_UNIT_S ="secondsInTimeUnit";
        public static final String BETA_S = "beta";
        private double beta;
```

```
private Map<DTNHost, Double> lastEncouterTime;
	private double lastAgeUpdate;
	public TBCC_ProphetRouter(Settings s) {
		super(s);
		Settings prophetSettings = new Settings(PROPHET_NS);
		secondsInTimeUnit = prophetSettings.getInt(SECONDS_IN_UNIT_S);
		if (prophetSettings.contains(BETA_S)) {
			beta = prophetSettings.getDouble(BETA_S);
		}
		else {
			beta = DEFAULT_BETA;
		}
		initPreds();
		initEncTimes();
	}
	protected TBCC_ProphetRouter(TBCC_ProphetRouter r) {
		super(r);
		this.secondsInTimeUnit = r.secondsInTimeUnit;
```

```
        this.beta = r.beta;
        initPreds();
        initEncTimes();
    }
    private void initEncTimes() {
        this.lastEncouterTime = new HashMap<DTNHost, Double>();
    }
    private void initPreds() {
        this.preds = new HashMap<DTNHost, Double>();
    }
    @Override
    public void changedConnection(Connection con) {
        if (con.isUp()) {
            DTNHost otherHost = con.getOtherNode(getHost());
            updateDeliveryPredFor(otherHost);
            updateTransitivePreds(otherHost);
        }
    }
```

```
private void updateDeliveryPredFor(DTNHost host) {
        double PEnc;
        double simTime = SimClock.getTime();
        double lastEncTime=getEncTimeFor(host);
        if(lastEncTime==0)
                PEnc=PEncMax;
        else
                if((simTime-lastEncTime)<I_TYP)
                {
                        PEnc=PEncMax*((simTime-lastEncTime)/I_TYP);
                }
                else
                        PEnc=PEncMax
                double oldValue = getPredFor(host);
double newValue = oldValue + (1 - oldValue) * P_INIT;
                preds.put(host, newValue);
                lastEncouterTime.put(host, simTime);
        }
```

```
	public double getEncTimeFor(DTNHost host) {
		if (lastEncouterTime.containsKey(host)) {
			return lastEncouterTime.get(host);
		}
		else {
			return 0;
		}
	}
	public double getPredFor(DTNHost host) {
	ageDeliveryPreds();
	if (preds.containsKey(host)) {
		return preds.get(host);
	}
	else {
		return 0;
	}
}
	private void updateTransitivePreds(DTNHost host) {
```

```
        MessageRouter otherRouter = host.getRouter();
        assert otherRouter instanceof TBCC_ProphetRouter :
                "TBCC_AF_ProphetRouter only works with other routers of same type";
        double pForHost = getPredFor(host);
        Map<DTNHost, Double> othersPreds =
                ((TBCC_ProphetRouter)otherRouter).getDeliveryPreds();
        for (Map.Entry<DTNHost, Double> e : othersPreds.entrySet()) {
                if (e.getKey() == getHost()) {
                        continue;
                }
                double pOld = getPredFor(e.getKey());
                double pNew = pOld + ( 1 - pOld) * pForHost * e.getValue() * beta;
                if(pNew>pOld)
                        preds.put(e.getKey(), pNew);
        }
}
private void ageDeliveryPreds() {
        double timeDiff = (SimClock.getTime() - this.lastAgeUpdate) /
```

```
                secondsInTimeUnit;
        if (timeDiff == 0) {
                return;
        }
        double mult = Math.pow(GAMMA, timeDiff);
        for (Map.Entry<DTNHost, Double> e : preds.entrySet()) {
                e.setValue(e.getValue()*mult);
        }
        this.lastAgeUpdate = SimClock.getTime();
}
private Map<DTNHost, Double> getDeliveryPreds() {
        ageDeliveryPreds(); // make sure the aging is done
        return this.preds;
}
@Override
public void update() {
        super.update();
        if (!canStartTransfer() ||isTransferring()) {
```

```
                return; // nothing to transfer or is currently transferring
        }
        if (exchangeDeliverableMessages() != null) {
                return;
        }
        tryOtherMessages();
}
private Tuple<Message, Connection> tryOtherMessages() {
        List<Tuple<Message, Connection>> messages =
                new ArrayList<Tuple<Message, Connection>>();
        Collection<Message> msgCollection = getMessageCollection();
        for (Connection con : getConnections()) {
                DTNHost other = con.getOtherNode(getHost());
        TBCC_ProphetRouter othRouter = (TBCC_ProphetRouter)other.getRouter();
                if (othRouter.isTransferring()) {
                        continue; // skip hosts that are transferring
                }
                for (Message m : msgCollection) {
```

```
                        if (othRouter.hasMessage(m.getId())) {
                                continue; // skip messages that the other one has
                        }
                        if((othRouter.getPredFor(m.getTo()) >= getPredFor(m.getTo())))
                        {
                                messages.add(new Tuple<Message, Connection>(m,con));
                        }
                }
        }
        if (messages.size() == 0) {
                return null;
        }
        Collections.sort(messages, new TupleComparator());
        return tryMessagesForConnected(messages);
}
private class TupleComparator implements Comparator
        <Tuple<Message, Connection>>
        public int compare(Tuple<Message, Connection> tuple1,
```

```
                Tuple<Message, Connection> tuple2) {
            double p1 = ((TBCC_ProphetRouter)tuple1.getValue().
                    getOtherNode(getHost()).getRouter()).getPredFor(
                    tuple1.getKey().getTo());
            double p2 = ((TBCC_ProphetRouter)tuple2.getValue().
                    getOtherNode(getHost()).getRouter()).getPredFor(
                    tuple2.getKey().getTo());
            if (p2-p1 == 0) {
                return compareByQueueMode(tuple1.getKey(), tuple2.getKey());
            }
            else if (p2-p1 < 0) {
                return -1;
            }
            else {
                return 1;
            }
        }
}
```

```
@Override
public RoutingInfo getRoutingInfo() {
        ageDeliveryPreds();
        RoutingInfo top = super.getRoutingInfo();
        RoutingInfo ri = new RoutingInfo(preds.size() +
                        " delivery prediction(s)");
        for (Map.Entry<DTNHost, Double> e : preds.entrySet()) {
                DTNHost host = e.getKey();
                Double value = e.getValue();
                ri.addMoreInfo(new RoutingInfo(String.format("%s : %.6f",
                                host, value)));
        }
        top.addMoreInfo(ri);
        return top;
}
@Override
public MessageRouter replicate() {
```

```
            TBCC_ProphetRouter r = new TBCC_ProphetRouter(this);

            return r;

        }

}
```

APPENDIX B

Program class for incorporating mTBCC algorithm on prophet routing protocol

```
/** SHAIBU IDRIS SALAWU
* AHMADU BELLO UNIVERSITY ZARIA
 * TBCC-AF_PROPHET ROUTING IMPLEMENTATION */
package routing;
public class TBCC_AF_ProphetRouter extends ActiveRouter {
        public static final double PEncMax = 0.2;
        public static double PEnc = 0.4;
        public static double TransmissionCost = 0;
        public static double StorageCost = 0;
        public static double MigrationCost = TransmissionCost + StorageCost;
        public static double MigrationCostThreshold = 0;
        public double ThresholdShortestDistance = 0;
        public static final double P_INIT = 0.2;
        public static final double I_TYP = 100;
        public static final double DEFAULT_BETA = 0.15;
        public static final double GAMMA = 0.98;
```

```
        Random randomGenerator = new Random();
        public static final String PROPHET_NS = "TBCC_AF_ProphetRouter";
        public static final String SECONDS_IN_UNIT_S ="secondsInTimeUnit";
        public static final String BETA_S = "beta";
        private double beta;
        private Map<DTNHost, Double> preds;
private Map<DTNHost, Double> lastEncouterTime;
        private double lastAgeUpdate;
        public TBCC_AF_ProphetRouter(Settings s) {
                super(s);
                Settings prophetSettings = new Settings(PROPHET_NS);
                secondsInTimeUnit = prophetSettings.getInt(SECONDS_IN_UNIT_S);
                if (prophetSettings.contains(BETA_S)) {
                        beta = prophetSettings.getDouble(BETA_S);
                }
                else {
                        beta = DEFAULT_BETA;
                }
```

```
        initPreds();
        initEncTimes();
    }
    protected TBCC_AF_ProphetRouter(TBCC_AF_ProphetRouter r) {
        super(r);
        this.secondsInTimeUnit = r.secondsInTimeUnit;
        this.beta = r.beta;
        initPreds();
        initEncTimes();
    }
    private void initEncTimes() {
        this.lastEncouterTime = new HashMap<DTNHost, Double>();
    }
    private void initPreds() {
        this.preds = new HashMap<DTNHost, Double>();
    }
    @Override
    public void changedConnection(Connection con) {
```

```
        if (con.isUp()) {
                DTNHost otherHost = con.getOtherNode(getHost());
                updateDeliveryPredFor(otherHost);
                updateTransitivePreds(otherHost);
        }
}
private void updateDeliveryPredFor(DTNHost host) {
double PEnc;
double simTime = SimClock.getTime();
double lastEncTime=getEncTimeFor(host);
        if(lastEncTime==0)
                PEnc=PEncMax;
        else
                if((simTime-lastEncTime)<I_TYP)
                {
                        PEnc=PEncMax*((simTime-lastEncTime)/I_TYP);
                }
                else
```

```
                        PEnc=PEncMax;
        double oldValue = getPredFor(host);
        double newValue = oldValue + (1 - oldValue) * P_INIT;
        preds.put(host, newValue);
        lastEncouterTime.put(host, simTime);
}
private void CompareMigrationCost(DTNHost host) {
double ShortestDistance = 0;
double MsgControlLength = 0;
double ThresholdShortestDistance = 0;
double MigrationCost = 0;
        if(MsgControlLength==ThresholdShortestDistance)
        PEnc=PEncMax;
        else
        if(MigrationCost<MigrationCostThreshold)
                PEnc=PEncMax;
else
        if((ShortestDistance + MsgControlLength)<I_TYP)
```

```
        {
        TransmissionCost=ShortestDistance + MsgControlLength;
        }
        else
                PEnc=PEncMax;
        double oldValue = getPredFor(host);
        double newValue = oldValue + (1 - oldValue) * P_INIT;
        preds.put(host, newValue);
        lastEncouterTime.put(host, ShortestDistance);
}
public double getEncTimeFor(DTNHost host) {
if (lastEncouterTime.containsKey(host)) {
return lastEncouterTime.get(host);
}
else {
        return 0;
}
```

```
}
public double getPredFor(DTNHost host) {
ageDeliveryPreds();
if (preds.containsKey(host)) {
return preds.get(host);
}
else {
        return 0;
        }
}
        private void updateTransitivePreds(DTNHost host) {
                MessageRouter otherRouter = host.getRouter();
                assert otherRouter instanceof TBCC_AF_ProphetRouter :
                        "TBCC_AF_ProphetRouter only works with other routers of same type";
                double pForHost = getPredFor(host);
                Map<DTNHost, Double> othersPreds =
                        ((TBCC_AF_ProphetRouter)otherRouter).getDeliveryPreds();
                for (Map.Entry<DTNHost, Double> e : othersPreds.entrySet()) {
```

```
                if (e.getKey() == getHost()) {
                        continue;
                }
                double pOld = getPredFor(e.getKey());
                double pNew = pOld + ( 1 - pOld) * pForHost * e.getValue() * beta;
                if(pNew>pOld)
                        preds.put(e.getKey(), pNew);
        }
}
private void ageDeliveryPreds() {
        double timeDiff = (SimClock.getTime() - this.lastAgeUpdate) /
                secondsInTimeUnit;
        if (timeDiff == 0) {
                return;
        }
        double mult = Math.pow(GAMMA, timeDiff);
        for (Map.Entry<DTNHost, Double> e : preds.entrySet()) {
                e.setValue(e.getValue()*mult);
```

```
        }

        this.lastAgeUpdate = SimClock.getTime();

}

private Map<DTNHost, Double> getDeliveryPreds() {

        ageDeliveryPreds();

        return this.preds;

}

@Override

public void update() {

        super.update();

        if (!canStartTransfer() ||isTransferring()) {

                return;

        }

        if (exchangeDeliverableMessages() != null) {

                return;

        }

        tryOtherMessages();

}
```

```
private Tuple<Message, Connection> tryOtherMessages() {
        List<Tuple<Message, Connection>> messages =
                new ArrayList<Tuple<Message, Connection>>();
        Collection<Message> msgCollection = getMessageCollection();
        for (Connection con : getConnections()) {
                DTNHost other = con.getOtherNode(getHost());
TBCC_AF_ProphetRouter othRouter = (TBCC_AF_ProphetRouter)other.getRouter();
                if (othRouter.isTransferring()) {
                        continue;
                }
                for (Message m : msgCollection) {
                        if (othRouter.hasMessage(m.getId())) {
                                continue;
                        }
                        if((othRouter.getPredFor(m.getTo()) >= getPredFor(m.getTo())))
                        {
                                messages.add(new Tuple<Message, Connection>(m,con));
```

```
                    }
                }
            }
            if (messages.size() == 0) {
                return null;
            }
            Collections.sort(messages, new TupleComparator());
            return tryMessagesForConnected(messages);
    }
    private class TupleComparator implements Comparator
            <Tuple<Message, Connection>> {
            public int compare(Tuple<Message, Connection> tuple1,
                    Tuple<Message, Connection> tuple2) {
                double p1 = ((TBCC_AF_ProphetRouter)tuple1.getValue().
                        getOtherNode(getHost()).getRouter()).getPredFor(
                        tuple1.getKey().getTo());
                double p2 = ((TBCC_AF_ProphetRouter)tuple2.getValue().
                        getOtherNode(getHost()).getRouter()).getPredFor(
```

```
                        tuple2.getKey().getTo());
                if (p2-p1 == 0) {
                        return compareByQueueMode(tuple1.getKey(), tuple2.getKey());
                }
                else if (p2-p1 < 0) {
                        return -1;
                }
                else {
                        return 1;
                }
        }
}
@Override
public RoutingInfo getRoutingInfo() {
        ageDeliveryPreds();
        RoutingInfo top = super.getRoutingInfo();
        RoutingInfo ri = new RoutingInfo(preds.size() +
                        " delivery prediction(s)");
```

```
            for (Map.Entry<DTNHost, Double> e : preds.entrySet()) {
                DTNHost host = e.getKey();
                Double value = e.getValue();
                ri.addMoreInfo(new RoutingInfo(String.format("%s : %.6f",
                        host, value)));
            }
            top.addMoreInfo(ri);
            return top;
        }
        @Override
        public MessageRouter replicate() {
            TBCC_AF_ProphetRouter r = new TBCC_AF_ProphetRouter(this);
            return r;
        }
}
```

APPENDIX C

Settings for Prophet Routing on Helsinki Environment

Scenario.name = Helsinki_Prophet_ mTBCC -QS-10-Salawu

Scenario.simulateConnections = true

Scenario.updateInterval = 1.0

Scenario.endTime = 43200

Scenario.nrofHostGroups = 4

btInterface.type = SimpleBroadcastInterface

btInterface.transmitSpeed = 250k

btInterface.transmitRange = 5

Group.movementModel = RandomWaypoint

Group.router = ProphetRouter

Group.bufferSize = 40M

Group.nrofInterfaces = 1

Group.interface1 = btInterface

Group.speed = 3, 5

Group.msgTtl = 300

Group.nrofHosts = 60

Group1.groupID = GCC_Inner

Group1.nrofHosts = 30

Group1.OkMaps = 4

Group1.speed = 4.5, 5

Group2.groupID = GCC_Edge

Group.nrofHosts = 10

Group2.movementModel = MapRouteMovement

Group2.speed = 4.5,5

Group3.groupID = GCC_Edge_outside

Group3.nrofHosts = 10

Group3.movementModel = shortestPathMapBasedMovement

Group3.okMaps = 1

Group3.speed = 4.5,5

Group4.groupID = GCC_Outside

Group4.nrofHosts = 12

Group4.movementModel = RandomWaypoint

Group.okMaps = 4

Group4.speed = 4.5,5

Event1.nrof = 1

Events1.class = MessageEventGenerator

Event1.interval = 25, 35

Events1.size = 500k, 1M

Event1.host = 0, 59

Events1.prefix = A

MovementModel.msgSeed = 1

MovementModel.worldSize = 4500, 3400

MovementModel.warmup = 10

MapBasedMovement.nrofMapFiles = 3

MapBasedMovement.mapFile = data/ roads.wkt

MapBasedMovement.mapFile2 = data/ pedestrian_path.wkt

MapBasedMovement.mapFile3 = data/shops.wkt

Report.nrofReports = 1

Report.warmup = 0

Reprot.reportDir = reports/ Salawu/mTBCC

Report.report1 = MessageStatsReport

ProphetRouter.secondsInTimeUnit = 1

Optimization.cellSizedMult = 5

Optimization.randomizeUpdateOrder = true

GUI.underlayImage.fileName = data/Helsinki_underlay.pnp

GUI.UnderImage.offset = 64, 20

GUI.UnderlayImage.scale = 4, 75

GUI.UnderImage.rotate = -0.015

GUI.EventLogPanel.nrofEvents = 60

APPENDIX D

Complete script for plotting dropped messages

% GCC: Greatest connected component in percentage

% DM_TBCC: Dropped messages for token based congestion control for 60 nodes

% DM_mTBCC: Dropped messages for modified token based congestion control for 60 nodes

GCC = input (‘please provide the dropped message :’);

DM_TBCC = input (‘please provide the dropped message :’);

DM_mTBCC = input (‘please provide the dropped message :’);

Bar (TBCC,mTBCC);

Title (‘Dropped messages versus GCC Percentage Connected’)

Xlabel (‘GCC % - Connected’)

Ylabel (‘Dropped messages’)

Legend (‘QS-10’,’QS-20’,’QS-30’’QS-40’,’ mQS-10’,’mQS-20’,’mQS-30’’mQS-40’)

Grind on.

APPENDIX E

Complete script for plotting network transit time

```
% Token: initial number of token per node

% NTT_TBCC: Network transit time for token based congestion control for 60 nodes

% NTT_mTBCC: Network transit time for token based congestion control for 60 nodes

Token = input ('please provide the initial number of token per nodeat different GCC percentage
connected')

NTT_TBCC= input ('please provide network transit time :');

NTT_mTBCC= input ('please provide network transit time :');

Plot (token, NTT_TBCC,'b')

Hold on

Plot (token, NTT_mTBCC,'r')

Title ('Network transit time versus Initial number of token per node at50% GCC Connected')

Xlabel ('Initial number of tiken/node')

Ylabel ('Network transit time')

Lengend ('TBCC',mTBCC)

Grind on
```