

**OPTIMIZING PERFORMANCE OF DELAY
TOLERANT NETWORK (DTN) ROUTING
PROTOCOLS IN POST DISASTER
RESPONSE NETWORKS**



By

Momina Khan
(01-244102-066)

Supervised By:

Mr. Fazl-e-Hadi

**DEPARTMENT OF COMPUTER SCIENCES
BAHRIA UNIVERSITY
ISLAMABAD**

Session 2010-2012

A thesis
presented to the Bahria University, Islamabad
in partial fulfillment of the requirements
for the degree of
Masters of Science
in
Telecommunication and Networking

MASTERS THESIS

OPTIMIZING THE PERFORMANCE OF DELAY TOLERANT NETWORK (DTN) ROUTING PROTOCOLS IN POST DISASTER RESPONSE NETWORKS

Prepared by: Momina Khan
(01-244102-066)

Supervised by: Mr. Fazl-e-Hadi

Abstract

To achieve performance optimization in opportunistic Delay Tolerant Networks (DTNs), most flooding based routing algorithms like Epidemic and Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET) transmit multiple copies of a message to several custodians. This results in quick depletion of valuable network resources. On the other hand quota based protocols, such as Spray and Wait, reduce routing overhead by limiting the number of redundant transmission of a message but achieve lesser throughput. This makes both these types of approaches directly inapplicable in a post disaster response network where maximum delivery ratio with minimum routing overhead is expected.

Recent research suggests that with proper buffer management policies implemented at nodes, we can intentionally optimize any one or a number of routing metrics like message delivery ratio or delay etc. Similarly we can increase the message delivery ratios of flooding based schemes in resource constrained environments without increasing routing overhead. This makes them comparable to quota based protocols in terms of routing overhead while maintaining higher delivery ratios. In this thesis we assess the performance of flooding based protocols (e.g. Epidemic and PRoPHET) with several existing buffer management schemes in order to determine their impact. We also propose a novel buffer management scheme appropriate for a resource constrained environment like the post disaster response network, where communication is enabled through small hand held devices with limited buffer sizes and transmission capabilities.

We further prove that to curtail the negative impact of buffer overflow in such congested and constrained environments, buffer management policies should carefully select the messages to be discarded. For this purpose we have proposed a dynamic threshold size based selection mechanism where the difference between the arriving message size and the available buffer space is calculated to

determine the size of the message to be dropped, in case of buffer overflow. Due to the adaptive nature of our proposed scheme, it adds randomness to the message selection process. This gives each message equal chance of being successfully delivered. Simulations show that our proposed scheme named as Size Aware Drop (SA-Drop), significantly improves the performance of the basic flooding based DTN routing protocols by increasing their message delivery probability and by controlling routing overhead.

When used with PROPHET, our proposed policy outperforms Encounter Based Routing (EBR) a quota based protocol, specifically designed for post disaster response networks. This makes it suitable for enabling communication in a post disaster scenario.

DECLARATION OF AUTHENTICATION

*I certify that the research work presented in this thesis “**Optimizing Performance of Delay Tolerant Network (DTN) Routing Protocols in Post Disaster Response Networks**” is to the best of my knowledge my own. All sources used and any help received in the preparation of this thesis have been acknowledged. I hereby declare that I have not submitted this material, either in whole or in part, for any other degree at this or any other institution.*

Momina Khan
(01-244102-066)

ACKNOWLEDGEMENTS

In the name of Allah Almighty, who is the most merciful and the most beneficent. He is the one who gave me the courage and determination to carry on when all seemed impossible.

It is my great fortune to have pursued my research under the guidance of my advisor, Sir Fazl-e-Hadi, who introduced me to the subject area, and guided me at every step of the way with his knowledge and experience. His prompt and detailed feedback greatly helped me throughout my research and inspired me to explore deeper. Our weekly meetings always kept me focused and helped me complete my work within the proposed time line.

I would also like to thank Dr. Abid Ali Minhas, for the inspiration to opt for research work in the first place. Despite his busy schedule, he has always shown keen interest in my research and answered all my queries with patience. Moreover the workshops arranged by him on Network Simulator-2 (NS-2) have been of great assistance.

My special thanks go to Miss Madeeha Basharat for arranging initial meetings with various other faculty members. I would also like to thank Dr. Imran Siddiqi, from the Research cell, for his guidance regarding issues like time lines and other research requirements.

I would also like to extend my gratitude to certain researchers across the globe i.e. Sam Nelson, Md. Yusuf Sarwar Uddin, Matthias Schwamborn, Abraham Martin and Osman Khalid, who responded promptly via emails and provided personal insight to their research works on enquiry. It is encouraging to know that intellectuals are willing to extend help to complete strangers for the promotion of literary benefits.

During any research, there come times when one feels discouraged by dead ends. It is the time when supportive friends and colleagues can boost hope and provide encouragement. I feel privileged to have such friends especially Miss Nida Akhtar, who was always positively hopeful even when I was not.

Finally I would like to extend my gratitude to Bahria University, Islamabad, for giving me this opportunity to conduct this research and to facilitate it by enabling access to valuable knowledge pool of IEEE archives.

DEDICATION

I dedicate this thesis to my family, my husband Moetesum Khurshid for his support and inspiration, my parents for their endless prayers, my in-laws for their consideration and patience and most of all to my dear children Haya and Taha, for coping with my demanding routines and occasional frustration throughout my Masters.

PROJECT IN BRIEF

Project Title: Optimizing Performance of Delay Tolerant Network (DTN) Routing Protocols in Post Disaster Response Networks

Submitted By: Momina Khan
01-244102-066
MS-(Telecommunication And Networking)

Supervised By: Mr. Fazl-e-Hadi

Start Date: September 2011

Completion Date: September 2012

Tools & Technologies: -Opportunistic Network Environment (ONE) Simulator (version 1.4.1)
-Network Simulator-2 (ns-allinone-2.37-RC7)
-Ubuntu-10.10
-VMware 7.0
-Jdk-1.6.0_31
-Jre6
-Microsoft Office Excel 2007
-Microsoft Office Word 2007
-Notepad

Operating System: Windows XP

System Specifications: Intel® Pentium® Dual-Core Processor T4200
2.0 GHz
956MB RAM
160 GB HDD

ABSTRACT

To achieve performance optimization in opportunistic Delay Tolerant Networks (DTNs), most flooding based routing algorithms like Epidemic and Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET) transmit multiple copies of a message to several custodians. This results in quick depletion of valuable network resources. On the other hand quota based protocols, such as Spray and Wait, reduce routing overhead by limiting the number of redundant transmission of a message but achieve lesser throughput. This makes both these types of approaches directly inapplicable in a post disaster response network where maximum delivery ratio with minimum routing overhead is expected.

Recent research suggests that with proper buffer management policies implemented at nodes, we can intentionally optimize any one or a number of routing metrics like message delivery ratio or delay etc. Similarly we can increase the message delivery ratios of flooding based schemes in resource constrained environments without increasing routing overhead. This makes them comparable to quota based protocols in terms of routing overhead while maintaining higher delivery ratios. In this thesis we assess the performance of flooding based protocols (e.g. Epidemic and PRoPHET) with several existing buffer management schemes in order to determine their impact. We also propose a novel buffer management scheme appropriate for a resource constrained environment like the post disaster response network, where communication is enabled through small hand held devices with limited buffer sizes and transmission capabilities.

We further prove that to curtail the negative impact of buffer overflow in such congested and constrained environments, buffer management policies should carefully select the messages to be discarded. For this purpose we have proposed a dynamic threshold size based selection mechanism where the difference between the arriving message size and the available buffer space is calculated to determine the size of the message to be dropped, incase of buffer overflow. Due to the adaptive nature of our proposed scheme, it adds randomness to the message selection process. This gives each message equal chance of being successfully delivered. Simulations show that our proposed scheme named as Size Aware Drop (SA-Drop), significantly improves the performance of the basic flooding based DTN routing protocols by increasing their message delivery probability and by controlling routing overhead.

When used with PRoPHET, our proposed policy outperforms Encounter Based Routing (EBR) a quota based protocol, specifically designed for post disaster response networks. This makes it suitable for enabling communication in a post disaster scenario.

LIST OF ABBREVIATIONS

DTN	Delay Tolerant Networking
DTNs	Delay Tolerant Networks
IP	Internet Protocol
TCP	Transport Control Protocol
UDP	User Datagram Protocol
TTL	Time-To-Live
IPNs	Inter Planetary Networks
WSNs	Wireless Sensor Networks
PDRNs	Post Disaster Response Networks
PRoPHET	Probabilistic Routing Protocol using History of Encounters and Transitivity
SnW	Spray and Wait
EBR	Encounter Based Routing
RAPID	Resource Allocation Protocol for Intentional DTN routing
ORWAR	Opportunistic DTN Routing with Window-aware Adaptive Replication
DF	Drop Front
DL	Drop Last
DR	Drop Random
DY	Drop Youngest
DOA	Drop Oldest
DLA	Drop Largest
SHLI	Evict Shortest Life Time First
MOFO	Evict Most Forwarded First
MOPR	Evict Most Probable First
LEPR	Evict Least Probable First
GBD	Global knowledge Based Drop
HBD	History Based Drop
SA-Drop	Size Aware Drop
NoF	Number of Forwards
FIFO	First In First Out
GRTR_Max	Forward if greater and sort descending by remote's predictability

LIST OF TABLES

Table 1.1: DTN Routing Protocols Resource Assumptions	11
Table 3.1: Message Summary Table at Node A and B	28
Table 3.2: Table of Notations and Their Description.....	31
Table 3.3: RWP: Table of Simulation Parameters.....	37
Table 3.4: Disaster: Table of Simulation Parameters.....	38

LIST OF FIGURES

Figure 1.1: DTN Network Layer Architecture	2
Figure 1.2: Taxonomy of DTN Routing Protocols	9
Figure 2.1: DTN Buffer Management Architecture.....	16
Figure 2.2: Taxonomy of Existing Buffer Management Schemes.....	22
Figure 3.1: Proposed Model for Estimation of Threshold Size for Message Drop.....	27
Figure 3.2: Algorithm 1 for makeRoomForMessage() Function.....	31
Figure 3.3: Algorithm 2 for getThresholdMessage() Function.....	32
Figure 3.4: Schematic Overview of the ONE Simulation Environment [48].....	34
Figure 3.5: RWP: Snapshot of Node Placement at 0 and 3600 seconds	35
Figure 3.6: Disaster: Snapshot of Node Placement at 0 and 3600 seconds	36
Figure 4.1: RWP: Comparison of Optimized Epidemic and SnW w.r.t Delivery Probability	51
Figure 4.2: RWP: Comparison of Optimized Epidemic and SnW w.r.t Overhead Ratio.....	51
Figure 4.3: RWP: Comparison of Optimized Epidemic and SnW w.r.t Buffer Time Average	52
Figure 4.4: RWP: Comparison of Optimized Epidemic and SnW w.r.t Hop Count Average.	52
Figure 4.5: RWP: Comparison of Optimized Epidemic and SnW w.r.t Messages Dropped ..	53
Figure 4.6: RWP: Comparison of Optimized Epidemic and SnW w.r.t Messages Relayed ...	53
Figure 4.7: RWP: Comparison of Optimized Epidemic and SnW w.r.t Delivery Delay	54
Figure 4.8: RWP: Impact of Varying Message Generation Rate on Delivery Probability.....	55
Figure 4.9: RWP: Impact of Varying Message Generation Rate on Overhead Ratio.....	55
Figure 4.10: RWP: Impact of Varying Buffer Sizes on Delivery Probability	56
Figure 4.11: RWP: Impact of Varying Buffer Sizes on Overhead Ratio	56
Figure 4.12: RWP: Impact of Varying Message Sizes on Delivery Probability.....	57
Figure 4.13: RWP: Impact of Varying Message Sizes on Overhead Ratio.....	57
Figure 4.14: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Delivery Probability	63
Figure 4.15: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Overhead Ratio	63
Figure 4.16: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Buffer Time Average	64
Figure 4.17: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Hop Count Average	64
Figure 4.18: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Messages Dropped	65
Figure 4.19: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Messages Relayed	65
Figure 4.20: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Delivery Delay.	66
Figure 4.21: Disaster: Impact of Varying Message Generation Rate on Delivery Probability	67
Figure 4.22: Disaster: Impact of Varying Message Generation Rate on Overhead Ratio.....	67
Figure 4.23: Disaster: Impact of Varying Buffer Sizes on Delivery Probability.....	68
Figure 4.24: Disaster: Impact of Varying Buffer Sizes on Overhead Ratio	68
Figure 4.25: Disaster: Impact of Varying Message Sizes on Delivery Probability	69
Figure 4.26: Disaster: Impact of Varying Message Sizes on Overhead Ratio.....	69
Figure A.1: Appendix i: Snapshot of Usage of Parameter File Generation Tool for NS-2.....	79
Figure A.2: Appendix i: Snapshot of Usage of ONE Compatible Mobility Trace File Generation Tool for NS-2	80

Figure A.3: Appendix i: Snapshot of <i>oneMobilityTrace</i> File	80
Figure A.4: Appendix ii: Java Class File for Active Router.....	87
Figure A.5: Appendix ii: Java Class File for Epidemic Router with Size Aware Drop (SA-Drop)	88
Figure A.6: Appendix ii: Java Class File for Prophet Router with Proposed policy (SA-Drop)	93
Figure A.7: Appendix ii: Java Class File for Spray and Wait Router	95
Figure A.8: Appendix ii: Java Class File for EBR Router	99
Figure A.9: Appendix ii: getOldestMessage() Function to Implement DOA.....	100
Figure A.10: Appendix ii: getLargestMessage() Function to Implement DLA.....	100
Figure A.11: Appendix ii: getShortestLifeMessage() Function to Implement SHLI.....	101
Figure A.12: Appendix ii: getMaxForwardedMessage() Function to Implement MOFO	101
Figure A.13: Appendix ii: getThresholdMessage() and Modified makeRoomForMessage() Functions to Implement SA-Drop.....	102
Figure A.14: Appendix ii: Configuration File for Scenario 1: Random Waypoint.....	105
Figure A.15: Appendix ii: Snapshot of Scenario 1 Configuration File.....	105
Figure A.16: Appendix ii: Configuration File for Scenario 2: Disaster	108
Figure A.17: Appendix ii: Snapshot of Scenario 2 Configuration File.....	109
Figure A.18: Appendix ii: Snapshot of MessageStatsReport File	109

TABLE OF CONTENTS

DECLARATION OF AUTHENTICATION	iii
ACKNOWLEDGEMENTS	iv
DEDICATION	v
PROJECT IN BRIEF	vi
ABSTRACT	vii
LIST OF ABBREVIATIONS	viii
LIST OF TABLES	ix
LIST OF FIGURES	x
TABLE OF CONTENTS	xii
Chapter 1	1
INTRODUCTION	1
1.1. OVERVIEW OF DELAY TOLERANT NETWORKING (DTN)	1
1.2. APPLICATIONS	3
1.2.1. Remote Area Networks	3
1.2.2. Military Battlefield Networks.....	3
1.2.3. Energy Constrained/Sparse Wireless Sensor Networks (WSNs)	4
1.2.4. Inter Planetary Networks (IPNs)	4
1.2.5. Post Disaster Response Networks (PDRNs)	4
1.3. TAXONOMY OF DTN ROUTING PROTOCOLS	6
1.3.1. Forwarding Based Protocols.....	6
1.3.2. Replication Based Protocols.....	6
1.3.2.1 Flooding Based Protocols	6
a) Single Copy Flooding.....	6
b) Multi Copy Flooding.....	7
1.3.2.2. Quota Based Protocols.....	8
a) Non-Adaptive Limitation	8
b) Adaptive Limitation	9
1.4. PROTOCOL PERFORMANCE AND RESOURCE CONSUMPTION TRADEOFF	10
1.5. THESIS OBJECTIVES	12
1.6. THESIS CONTRIBUTIONS	13
1.7. THESIS ORGANIZATION.....	14
Chapter 2	15

LITERATURE REVIEW	15
2.1. BUFFER MANAGEMENT IN DTNs	16
2.2. TAXONOMY OF EXISTING BUFFER MANAGEMENT POLICIES	17
2.2.1. Policies without Network Wide Knowledge	18
2.2.1.1. Drop Front (DF)	18
2.2.1.2. Drop Last (DL).....	18
2.2.1.3. Drop Random (DR)	18
2.2.1.4. Drop Youngest (DY)	18
2.2.1.5. Drop Oldest (DOA)	19
2.2.1.6. Evict Shortest Life Time First (SHLI).....	19
2.2.1.7. Drop Largest (DLA).....	19
2.2.2. Policies with Network Wide Knowledge	19
2.2.2.1. Policies with Complete Network Knowledge	19
a) RAPID's Utility Based Drop	20
b) Global Knowledge Based Drop (GBD).....	20
c) History Based Drop (HBD)	20
2.2.2.2. Policies with Partial Network Knowledge.....	21
a) Evict Most Forwarded First (MOFO)	21
b) Evict Most Probable First (MOPR).....	21
c) Evict Least Probable First (LEPR).....	21
2.3. LIMITATIONS OF EXISTING BUFFER MANAGEMENT POLICIES	23
2.4. PROBLEM STATEMENT	24
Chapter 3	26
SIZE AWARE DROP (SA-DROP)	26
3.1. PROPOSED BUFFER MANAGEMENT POLICY	27
3.1.1. METHODOLOGY	28
3.1.2. ALGORITHM.....	30
3.1.3. SIMULATION SETUP.....	32
3.1.3.1. Simulation Tool.....	32
3.1.3.2. Mobility Models.....	34
a) Random Waypoint Mobility Model (RWP)	34
b) Event Driven, Role Based Disaster Mobility Model.....	35
3.1.3.3. Simulation Parameters.....	36
a) Scenario 1: Random Waypoint Scenario.....	37

b) Scenario 2: Disaster Scenario	37
3.1.4. IMPLEMENTATIONS.....	39
Chapter 4.....	43
RESULTS AND ANALYSIS	43
4.1. EVALUATION METRICS	43
4.1.1. Delivery Probability.....	43
4.1.2. Overhead Ratio.....	44
4.1.3. Buffer Time Average	44
4.1.4. Hop Count Average	44
4.1.5. Messages Dropped.....	45
4.1.6. Messages Relayed.....	45
4.2. PROTOCOLS UNDER EVALUATION	45
4.3. BUFFER MANAGEMENT SCHEMES UNDER EVALUATION	46
4.4. RESULTS AND ANALYSIS	47
4.4.1. Scenario 1: Performance Evaluation in Random Waypoint Mobility Model... 47	
4.4.1.1. Comparative Analysis	47
4.4.1.2. Performance Analysis.....	54
4.4.2. Scenario 2: Performance Evaluation in Disaster	59
4.4.2.1. Comparative Analysis	60
4.4.2.2. Performance Analysis.....	66
Chapter 5.....	71
CONCLUSION AND FUTURE WORK.....	71
5.1. FUTURE DIRECTIONS.....	72
BIBLIOGRAPHY.....	74
A) APPENDICES	79
APPENDIX I:.....	79
i) TOOLS FOR MOBILITY TRACE GENERATION IN NS-2.....	79
a) Parameter File Generation Tool for NS-2	79
b) ONE Compatible Mobility Trace Generation Tool	79
APPENDIX II:.....	81
i) PROTOCOLS CLASS FILES IN ONE SIMULATOR	81
a) ActiveRouter.java	81
b) EpidemicRouter.java with Size Aware Drop (SA-Drop)	87
c) ProphetRouter.java with Size Aware Drop (SA-Drop).....	89

d)	SprayAndWaitRouter.java.....	93
e)	EBRRouter.java	95
ii)	DROP POLICY FUNCTIONS IMPLEMENTED.....	100
a)	Drop Oldest (DOA).....	100
b)	Drop Largest (DLA).....	100
c)	Evict Shortest Life Time First (SHLI)	101
d)	Evict Most Forwarded First (MOFO)	101
e)	Size Aware Drop (SA-Drop)	102
iii)	SCENARIO CONFIGURATION FILES.....	103
a)	Scenario 1: Random Waypoint with Epidemic Router	103
b)	Scenario 2: Disaster with PRoPHET Router	106
iv)	SAMPLE OF REPORT FILE.....	109

Chapter 1

INTRODUCTION

The promise of pervasive communication has greatly influenced the infusion of wireless technology into our lives. Over the years the basic assumptions about network connectivity and applications have changed all together. In networks, where nodes are subject to frequent mobility and power outages, traditional postulations like end-to-end connectivity, short transmission delays or low data transfer error rates no longer hold. Such networks are termed as challenged networks and include a wide range of application scenarios. A Post Disaster Response Network (PDRN) is one such scenario where communication is critical for saving lives. However, enabling communication in the absence infrastructure is quite a challenge [1].

In 2003 K. Fall *et al* [2] presented a Delay Tolerant Networking (DTN) architecture to ensure reliable message delivery in such challenged networks. The architecture works by applying a store-carry-and-forward paradigm for message dissemination. A node may store and carry the message for quite a long period of time until an appropriate forwarding opportunity arises. Although effective in successful message delivery in harsh environments, these networks have introduced a new set of challenges not common in the traditional TCP/IP networks.

1.1. OVERVIEW OF DELAY TOLERANT NETWORKING (DTN)

As mentioned earlier, the introduction of Delay tolerant Networking (DTN) [3] in our communications gives rise to many challenges not present in traditional IP networks. Some key issues that need to be addressed for Delay Tolerant Networks (DTNs) include:

- *Intermittent Connectivity*- Nodes in such networks are subject to frequent failures due to their deployment in adverse surroundings. This causes the network to be frequently disconnected leaving the communications interrupted. Furthermore heterogeneous structures comprising of different underlying protocols and applications also contribute to this intermittent connectivity which is not so common in an IP network.
- *Network Partitioning*- Geographical displacement, weak signal strength or other such limiting factors, eventually lead to non existence of end-to-end connectivity between the source and the destination. These are referred to as network partitions which are quite common in DTNs.

- *High Error Rates*-As mentioned previously, elevated mobility and weak signal strength produce aggravating circumstances like short connectivity that lead to a high link-error rate in such networks. In such a scenario end-to-end reliability is quite difficult to achieve.
- *Long & Variable Delays*-The intermittent connectivity causes long and unpredictable delays. Data usually has to be stored for a long time if no path directly exists between the source and the destination node. That is why these networks are termed as “Delay Tolerant”. Depending on the mobility of the nodes and their consequent connectivity within the network, these delays can last for hours or even days.
- *Asymmetric Data Rates*- Elapse time between request and response in DTNs may be hours rather than milliseconds due to its intermittent nature. Furthermore heterogeneous device capabilities make communication between two nodes mainly asymmetric in such networks.

The DTN architecture solves most of these problems by introducing an abstraction layer in between the transport layer and the application layer, called the bundle layer [4] as shown in Figure 1.1. The objective of this overlay architecture is to route data from the source to the destination reliably without making any assumptions about the underlying networks. The bundle layer achieves this end-to-end delivery of messages through custodian transfer.

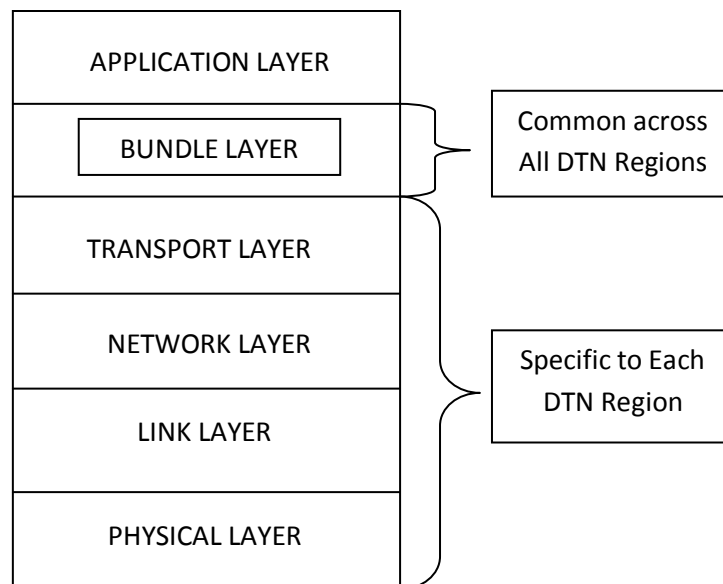


Figure 1.1: DTN Network Layer Architecture

DTN architecture recommends the use of a minimal conversational model due to unpredictable and short contact duration. To achieve message delivery with minimum end-to-end transactions the bundle layer uses large self-contained messages called bundles. Bundles

are message aggregates and can contain anything from a message fragments to several messages. DTN architecture does not define any particular length of these bundles neither does it put any upper or lower limits on their size. However the bundles are often considerably large. The reason being that due to unpredictable contact duration, a single message exchange is preferred to enable complete application interactions. As mentioned earlier, large bundle size is favorable in a store-carry-and-forward paradigm, however this adds new challenges for communication in resource constrained environments.

The bundles are propagated through the network via custody transfer which provides per hop reliability. The transfer of a bundle over a single hop is carried out by using a suitable convergence layer (such as TCP or UDP). In order to limit the distribution of bundles in a network, an attribute called Time-To-Live (TTL) is used. It is the lifetime of a bundle after which it expires and is removed from the network. Bundles can be retransmitted either at the source node or at an intermediate node provided it has the custodian rights for that bundle. The destination node may generate an acknowledgment for the source, upon receiving a bundle. These acknowledgments can also be used as anti packets for deleting the bundles already transferred to their destinations. This helps in removing unnecessary traffic from the network, thus relieving occupied valuable resources.

1.2. APPLICATIONS

Delay Tolerant Networking is gaining popularity perhaps due to the nature of applications that it addresses. Although a few real-world DTN applications have been deployed so far, yet there are some promising environments which can benefit from it. Some examples are as follows:

1.2.1. Remote Area Networks

Many remote regions in underdeveloped countries suffer from unavailability of Internet and other communication connectivity, even in this day and age. Present day technologies like wireless or satellite communications are often too expensive to install and maintain. In such environments, the notion of using human or vehicle mobility to offer intermittent connectivity is an interesting approach. DakNet [5] project is one such example. Such a network is characterized by intermittent connectivity, sparse deployment of nodes, node mobility, high propagation delay, heterogeneous and asymmetric data rates, making it a perfect DTN environment.

1.2.2. Military Battlefield Networks

Military communication networks [6] are essential to modern warfare; however these networks are also under constant stress due to enemy interference, destruction of critical nodes, or even something as simple as lack of line of sight to the receiving station. All these factors cause low transmission reliability. Data rates in such environments are relatively low. In such a setting, DTNs can deliver data services where standard radio frequency communications fail.

1.2.3. Energy Constrained/Sparse Wireless Sensor Networks (WSNs)

Wireless Sensor Networks (WSNs) is an emerging field in present day communication technologies. It is usually applied in areas without basic communication infrastructure by deploying nodes equipped with sensors. These participatory nodes usually have limited power, memory, and processing capabilities. The networks themselves are of a large scale, with hundreds of nodes per network. In most cases mobile nodes are used to collect and exchange sensor data which is then sent to a sink node within range. ZebraNet [7] is one such example. The key characteristics of these networks are power conservation, sparse deployment, low and asymmetric data rates.

1.2.4. Inter Planetary Networks (IPNs)

Inter Planetary Networks (IPNs) [8] are subject to high latencies due to predictable interruptions caused by planetary dynamics, or due to low earth orbiting satellites that periodically orbit each day. Limited visibility causes low transmission reliability and data rates are low and asymmetric. Planetary-routes can be calculated and satellite encounters are scheduled or predictable. DTNs enable data transmission and delivery while the planet or receiving object is within range or line of sight.

1.2.5. Post Disaster Response Networks (PDRNs)

Disasters take place and dissolve quickly, but their consequences are usually difficult to overcome. There may be a difference between developed and undeveloped countries, but in both cases preinstalled infrastructure can become unusable under similar influences. Rapid and coordinated first responses after disasters are essential to limit the number of casualties.

In the wake of recent catastrophic events over the past decade, where earthquakes, tsunamis and terrorist attacks have marked the world with their impacts, the need for resilient communication paradigms became vivid. *Post Disaster Response Networks (PDRNs)* [9] are usually set up when customary communication infrastructure is suddenly unavailable. In such a scenario communication usually relies on wireless ad hoc networks. People carrying

wireless hand held devices and moving on foot or in vehicles actively participate in facilitating communications. The basic aim of these networks is to provide rapid and easily deployable communication infrastructure for police and paramedics, as well as other first responders and civilians in the areas.

Besides the general challenges outlined earlier, most real life deployments like post disaster response networks are characterized by some particular challenges of their own. These include:

- *Dynamic Topology*- Network topology is highly dynamic as changes in node position are fast. Civilians are moving towards safety while responders are oscillating between disaster area and medical camps. Node population is also varying. At points of interest like disaster area or refugee camps, nodes tend to form clusters. Although node speeds are usually fast in wake of a disaster however obstacles and damaged infrastructure may cause unexpected stops or detours. Due to limited connectivity, topology information between nodes cannot be updated in due time. This means that the majority of routing decisions must be determined locally or based on historic information.
- *Heterogeneity of Devices*- Nodes participating in a post disaster response network differ in terms of their levels of storage and energy resources as well as communication capabilities. Most civilians use hand held devices like smart phones. While responders may be equipped with other devices. Different interfaces like Bluetooth or Wi-Fi with asymmetric data rates and transmission ranges participate during communication. A DTN routing algorithm must be tuned in order to provide required level of communication in such environments.
- *Congestion* – In a particular scenario of a disaster there exist chances of traffic congestion and bottlenecks in the network due to excessive message generation. This contention is mostly unavoidable as panic stricken victims are generating redundant messages while responders are trying to coordinate the rescue and relief operations. Another vital reason for this excessive message generation is the volunteer activities contributing to situational awareness. This gives rise to the need of message differentiation and congestion control mechanisms.
- *Resource Constraints* – When hand held wireless devices are at play then conserving resources is a challenge. Due to the intermittent connectivity and node mobility contact window or link duration varies across a wide range of values. Usually it is short lived. Similarly buffer sizes are limited as well. Energy is a very valuable resource in wake of power outages. While designing DTN routing protocols, it is important to keep these constraints in mind so that efficient use of these resources is made possible.

1.3. TAXONOMY OF DTN ROUTING PROTOCOLS

With all the challenges outlined in the previous section it appears that routing is the most demanding issue within a DTN [10]. Many limitations have to be taken into consideration for successful message dissemination in such networks. Traditional network routing protocols fail in these environments mostly due to the absence of instantaneous end-to-end paths. Instead DTN routing protocols carry messages in their buffers and propagate them to other nodes until they finally reach their destination. Based on their routing strategies DTN routing protocols can be divided into two main classes i.e. *Forwarding-based* protocols and *Replication-based* protocols.

1.3.1. Forwarding Based Protocols

Forwarding based protocols [11], [12] are single copy protocols which keep only one replica of a message in the network and try to forward that towards its destination. Simple forwarding means that, instead of copying the message to be transmitted to the new custodian and keeping one copy for itself, a node will delete the original copy from its own buffers as soon as transmission has been completed correctly. The unique copy of message is forwarded along a single path. Hence forwarding based protocols rely on intelligent path selection for optimal message delivery.

1.3.2. Replication Based Protocols

Replication based protocols are usually multi copy protocols that forward more than one copies into the network to increase the probability of successful message delivery. Based on the number of replicas created, there exists a trade-off between the resource usage and probability of successful message delivery. These protocols can further be divided into two classes i.e. *Flooding based protocols* and *Quota based protocols*.

1.3.2.1 Flooding Based Protocols

In simple terms flooding based protocols try to send a copy of each message to as many nodes as possible. Therefore in most routing protocols the number of allowed copies of any given message is reliant on the number of nodes in the network. We can further classify flooding based schemes broadly into *Single copy flooding* and *Multi copy flooding*.

a) Single Copy Flooding

Direct Delivery [13] and First Contact [13] are examples of single copy flooding based protocols. Both these protocols maintain one copy for each message. Direct Delivery routing

operates simply by the source node keeping its messages in its buffer until it encounter the destination nodes. In First Contact a message is forwarded along a single path by selecting the next hop randomly from available links. If connections do not exist the nodes waits and transmit the message to first available contact. Single copy flooding schemes incur the least amount of overhead ratio but fail to produce the enhanced message delivery ratios. The reason lies in the fact that with only a single copy being routed in a network prone to failures and losses, the reliability of successfully of a message is very little.

b) Multi Copy Flooding

Single copy flooding schemes minimize the number of transmissions thus consuming lesser bandwidth, buffer space and energy. However these schemes fail to achieve considerable delivery ratio especially in a congested environment. Moreover a single failure increases the probability of permanent message loss. Keeping these limitations in view researchers suggest multi copy flooding schemes where chances of message delivery are enhanced by flooding more than one copies of a message into the network. As the DTN protocols evolved and realistic assumptions about the resources and mobility started being taken into consideration, flooding strategies started seeing a transition from greedy flooding to controlled flooding.

- *Greedy Flooding*-The simplest version of flooding is performed in a “greedy” manner. Messages are flooded through the network. Each host stores a list of messages that it is sending or has received. This list of messages is then sent to all nodes that come in contact with this node, any previously unseen messages are then transmitted. Epidemic Routing [14] represents this extreme end of the flooding family. It attempts to send multiple copies of each message over multiple paths in the network. This approach is highly robust due to the fact that in case of a single node failure other nodes are sure to have a copy of the message. Additionally, Epidemic delivers each message with high probability in minimum amount of time. The reason is that it propagates message replicas over multiple paths instead of one optimal path. However in order to achieve this high message delivery ratio, Epidemic is highly resource consuming and incurs high routing overhead.
- *Controlled Flooding*-Earlier assumptions of a DTN included purely random and autonomous movements of the mobile nodes. However soon it was realized that in most realistic scenarios there is certain patterned node movement as well i.e. some nodes are better in contact to the popular destinations than the others. This gives rise to the idea of intelligent carrier selection. Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET) [15] is an example of controlled flooding routing protocol which forwards the message to a node by utilizing its encounter history and transitivity. A

metric called Delivery Predictability (DP) is maintained at each node for a known destination. Delivery Predictability is the predicted probability of any node to successfully deliver the message to its destination. Whenever a source node sends a message PROPHET selects a subset of nodes in the neighborhood to forward the message to based upon their DP ranking. Due to its controlled flooding strategy, PROPHET has higher delivery probability with much lower routing overhead than Epidemic. The idea of intelligently forwarding message replicas to only those nodes whose degree of centrality is better than the others not only improves delivery ratios but also reduces routing overhead. Still being a flooding based protocol, it can be resource consuming in certain scenarios.

1.3.2.2. Quota Based Protocols

Replication based routing protocols are termed as quota based protocols when the maximum number of copies of a message is independent of the number of nodes in the network. By design, quota based protocols bound the number of replicas of a message. This is to reduce the routing overhead incurred by flooding based schemes. Initially this limit on the number of replicas of a message was non-adaptive giving less room for further optimization. However, recently the idea of adaptive upper bound is gaining prominence which is resulting in better performance than its non-adaptive counterparts.

a) Non-Adaptive Limitation

Spray and Wait [16], a quota based protocol makes use of replication by predetermining the number of copies in a static non-adaptive way. This algorithm has two phases: the spray phase, which involves the source node distributing the copies to encountered nodes; and the wait phase, in which the nodes that are carrying the message copies follow the Direct Delivery method of routing on behalf of the source node.

A variant of Spray and Wait is called Binary Spray and Wait [16] in which a node with a number of copies of a message hands over half of its copies to the encountering node which has no copies of that particular message. This process is repeated again and again until the node is left with only a single message which is then directly delivered to the destination only.

Another variation of Spray and Wait is Spray and Focus [17] in which similar spray phase as that of original Spray and Wait algorithm, is followed by a focus phase. However unlike Spray and Wait, in the focus phase a message is forwarded to different intermediate relay nodes, in order to help maximize a utility function according to a given forwarding criterion based on elapse time between two encounters.

b) Adaptive Limitation

Encounter-Based Routing (EBR) [18] is another example of quota based replication protocol. However it applies an adaptive bound on the number of allowable replications of a message. In this protocol a node records its encounter rates with its neighbors. It then intelligently decides the number of replicas of a message to transfer during a contact opportunity. The objective is to achieve high message delivery ratios with good latency performance, while maintaining low overheads. EBR aims to lower the redundant transmissions incurred while trying to deliver a message, by reducing the total number of messages exchanged. EBR facilitates this by forwarding more copies of a message to nodes that are better connected. The connectivity of a node is calculated as an exponentially weighted moving average of a node's windowed degree of centrality.

Unlike static fixed upper bound quota set by Spray and Wait during the message creation time, EBR dynamically sets bounds on the basis of node encounter value. This results in delivering a better performance.

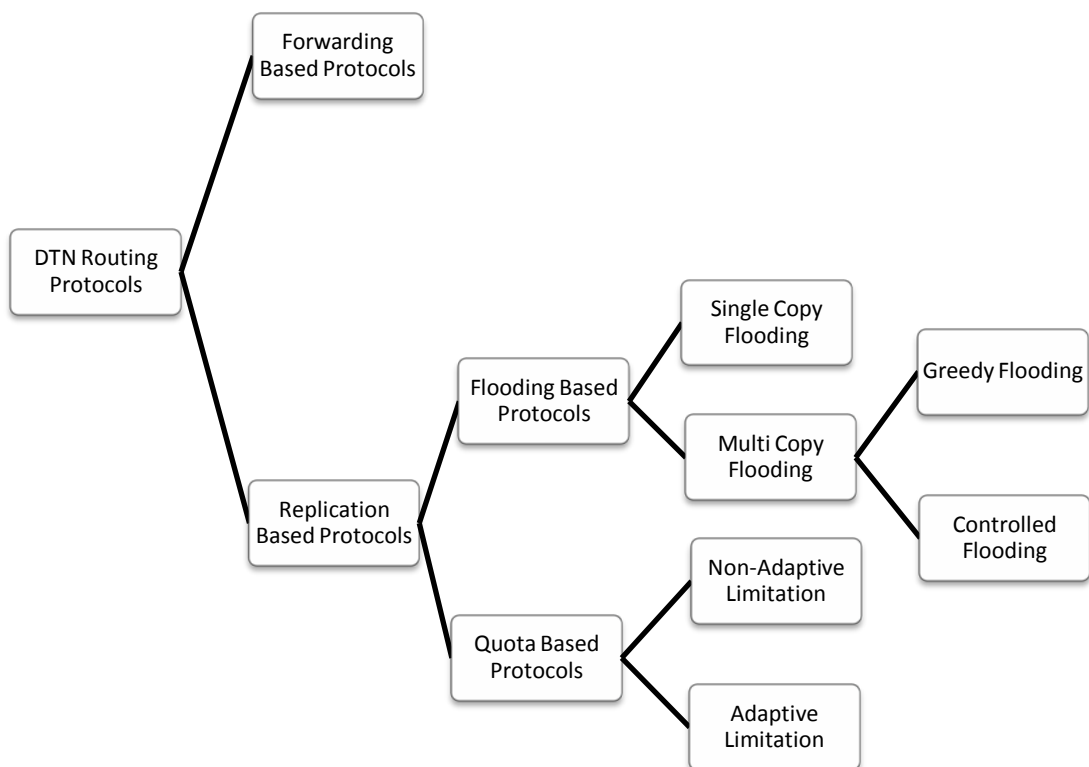


Figure 1.2: Taxonomy of DTN Routing Protocols

1.4. PROTOCOL PERFORMANCE AND RESOURCE CONSUMPTION TRADEOFF

As mentioned earlier real DTNs are often subject to severe resource constraints. Transmission bandwidth and contact duration is often limited due to the low data rates and excessive node mobility. In addition, applications involving hand held devices often use small battery-powered nodes which easily run out of energy and memory capacity. Most of all due to store carry and forward paradigm, buffer and storage constraints are the worst type.

Single copy forwarding schemes and single copy flooding schemes both attain delivery with minimum expenditure of resources. However these strategies reduce the network throughput in terms of message delivery ratio and raise its delivery delay whereas the objective is to maximize the network throughput. On the other hand, multi copy flooding based routing schemes maximize the network throughput by maximizing the message delivery ratio and minimizing message delivery delay but acquire additional network resources.

Initially designed flooding based schemes such as Epidemic are resource greedy with assumptions of unlimited bandwidth and buffer availability. Later on research emphasized the fact that in most of the real life applications such assumptions are not reasonable. Sharp performance degradation is observed when resource constraints are considered [19].

In order to achieve performance optimization with less routing overhead, the DTN research community came up with the idea of quota based replication. Several methods have been proposed in this regard such as in [20] the source makes n copies in the first phase. Each of these copies will try for a direct delivery then. Thus this algorithm can be viewed as a multi copy, two-hop scheme. The Spray and Wait and its variant Spray and Focus are other modifications of the same scheme that bound the number of copies in the network. By using fixed numbers of duplicate messages, these schemes effectively limit network resource consumption and thus attain better overhead performance than flooding based routing schemes. However, tuning the parameters in these schemes is challenging. Moreover, quota based protocols still have performance problem due to the fact that each message is delivered and transferred in the same and fixed amount of replicas, which would cause unwanted results like redundancy or insufficiency.

Another problem with initial DTN protocols is that these do not take the difference of node delivery capability into account, and simply consider that every node has the same capability to deliver data to the destination. However, in practice there are obvious differences between the delivery capabilities of nodes in a network. This gives rise to the idea of controlled

flooding where intelligent selection of relay nodes is done. PRoPHET is one such scheme which utilizes a node's contact history to make this selection. By elaborately adapting the number of duplicates for every data message and computing the message delivery probability so that data messages are forwarded to nodes with higher delivery capability, these protocols can achieve high network performance with low transmission overhead cost. Encounter Based Routing (EBR) is another example of such optimization where quota based routing is combined with probabilistic routing to increase overall network throughput. However most history based protocols have to maintain large tables for these contact histories or have to compute complex probabilities.

Another technique to conserve resources in a flooding approach is to embed additional information to the message so that the number of copies can be limited. Network coding [21], [22] and erasure coding [23], [24] are two such popular methods. In network coding decoding algorithm is embedded into the coded message blocks while erasure coding embeds redundancy into the message blocks. These schemes however, have most of the limitations of flooding based schemes and are not suitable of scenarios like a post disaster response networks. These are beyond the scope of this thesis.

Recent research considers more realistic resource assumptions while designing protocols e.g. resource aware protocols and context aware protocols like RAPID [25] and ORWAR [26] are examples of such protocols where optimization with minimum overhead is achieved through realistic assumptions of resources available. However most of these utility based protocols are in their infancy and have certain implementation issues in realistic environments. For instance RAPID requires the estimation of remaining life time of a message whereas in ORWAR, replication depends on estimation of contact widow between two nodes, both these parameters are difficult to estimate in real life applications.

Table 1.1 shows the existing replication based DTN routing protocols and their assumption on availability of buffer and bandwidth; both being either limited or unlimited.

Table 1.1: DTN Routing Protocols Resource Assumptions

DTN ROUTING PROTOCOLS	RESOURCE AVAILABILITY ASSUMPTIONS	
	BANDWIDTH	BUFFER
Epidemic [14]	Unlimited	Unlimited
Spray and Wait [16]	Unlimited	Unlimited
PRoPHET [15]	Unlimited	Limited
EBR [18]	Unlimited	Limited
RAPID [25]	Limited	Limited
ORWAR [26]	Limited	Limited

1.5. THESIS OBJECTIVES

Each DTN routing protocol has its advantages and disadvantages, it is however the scenario at hand that determines its applicability. In order to understand the design issues related to DTN routing in a particular scenario, this thesis focuses on a particular application of post disaster response networks. For a post disaster response network, high success rate together with efficient usage of the network resources is critical to successfully carry out the rescue and relief operations. Keeping these objectives in view we aim at determining the most appropriate DTN routing approach for providing maximum utilization of communication resources in this scenario.

Forwarding based approaches, as mentioned earlier, although less consuming, are limited in their effectiveness for a post disaster response network. When a single copy of a message exists in the network, a single route break is sufficient to cause a delivery to fail. Same is the case with single copy flooding based protocols. Therefore these approaches are not suitable for providing desirable results in a post disaster response network.

Greedy flooding (e.g. Epidemic) may be able to achieve high message delivery probability and minimum latency; however such schemes incur a great overhead which is not suitable for resource constrained environments like post disaster response networks. Similarly a non-adaptive quota based protocol like Spray and Wait might incur less overhead but its performance in terms of delivery might not be sufficient for such a scenario. However such schemes can be utilized as good baseline for performance evaluation.

Resource-Aware and Context-Aware routing protocols (e.g. RAPID and ORWAR) can be a good choice for routing in a post disaster scenario; however there are certain implementation issues which hamper their wide spread deployment. RAPID requires computation of more sophisticated and desired metrics such as worst-case delivery delay and packet delivery ratio at every transfer opportunity to justify the resources used. On the other hand in ORWAR the delivery of number of copies depends upon evaluation of the contact window between two nodes, which in a real world scenarios is quite difficult to estimate due to uncertainty of node's transmission range due to obstacles and interference.

Controlled flooding (e.g. PRoPHET) and adaptive quota-based protocols (e.g. EBR) are good choices for routing in such environments, as they achieve good delivery ratio with controlled overhead. PRoPHET is known to have delivered better results in most scenarios where mobility exhibits certain patterns. EBR has the advantage over others as it was specifically designed for a disaster scenario. The important question that arises here is that, can PRoPHET

maintain its superior message delivery probability with overhead ratio comparable to that of EBR in a constrained environment like a post disaster response network.

Recent studies suggest the importance of efficient buffer management to achieve these desired objectives. As mentioned earlier, in a constrained environment, frequent node buffer overflows can degrade performance of flooding based schemes due to excessive message loss/drop. In such a scenario buffer management schemes are expected to minimize the negative impact of buffer overflow by controlling redundant message drop. Hence the objective of this research is to propose, analyze, and evaluate efficient buffer management policies for achieving performance optimization of flooding based DTN routing schemes without increasing routing overhead.

The research outcomes include a detailed literature review of existing schemes, their classification and performance evaluation. It also includes design and analysis of an intelligent and efficient buffer management policy Size Aware Drop (SA-Drop) which is proposed specifically for resource constrained DTNs like post disaster response networks. The policy as we show, when applied to popular flooding based DTN protocols optimizes their performance with overhead comparable to that of quota based protocols.

1.6. THESIS CONTRIBUTIONS

The contributions of this thesis are threefold.

The first contribution of this research is to identify the peculiar characteristics of a post disaster response network. To our best knowledge very less work is available in literature that outlines the distinct features and requirements of a disaster response network.

The second contribution is to study the effects of buffer management policies on the performance of popular flooding based DTN protocols in terms of message delivery ratio and routing overhead in highly constrained environments. We analyze their performance in two different scenarios with two different mobility models i.e. Random Waypoint and Event Driven, Role Based Disaster Mobility Model and compare them with quota based protocols.

The most important contribution of this research is to determine that in case of highly constrained environments with limited buffer space and excessive message generation, size of a message is an appropriate discarding criterion. Based on this we propose an efficient buffer management scheme called Size Aware Drop (SA-Drop).

Simulation results determine that our proposed scheme Size Aware Drop (SA-Drop) not only increases the message delivery ratio but also improves the routing overhead significantly when both bandwidth and buffer space are constrained as in case of a disaster scenario.

1.7. THESIS ORGANIZATION

The thesis has been distributed as follows; Chapter 1 gives an introduction of DTN and its applications alongwith the taxonomy of its routing protocols. It highlights the delivery probability and resource consumption tradeoff of these schemes and their applicability in a particular scenario of a disaster. It also gives a brief introduction to the objectives of this study. Chapter 2 explains the importance of buffer management in achieving performance optimization of DTN protocols. It also gives the taxonomy of already existing buffer management policies and their limitations based on which we have derived our research problem statement. Chapter 3 describes our proposed solution, the methodology used and the algorithm designed. It also explains the simulation setup and implementations. Chapter 4 outlines the results of our simulations and their analysis. Finally Chapter 5 presents the conclusion and future research directions.

Chapter 2

LITERATURE REVIEW

In order to optimize the performance of resource consuming DTN protocols we must fully understand these resource constraints and their impact. The DTN environment suffers from scarcity of resources, for instance limited bandwidth, buffer space and energy. Due to these limitations the routing procedures turn out to be even more of a challenge. In a DTN framework, message transmissions occur only when nodes come in contact of each other. The node mobility limits the duration of contacts between nodes. The contact duration of the nodes in turn limits the amount of messages that can be transmitted. In basic flooding based protocols (e.g. Epidemic and PROPHET etc., the contact duration is assumed to be long enough to transmit all messages a node has. But this is not always true because of limited bandwidth. It implies that it may not be possible for a node to transmit all messages it has during the short available period of contact. Therefore it becomes necessary for the node to choose the messages to be transmitted during the period of contact.

Similar is the impact of buffer size on the performance of DTN protocols. In DTNs, to cope up with long disconnections, messages are usually buffered for a long period of time. This implies that at certain point the buffer capacity of a node will be reached. Therefore decision has to be made by the nodes to choose the messages to be dropped in favour of new ones. Again most protocols like Epidemic work with the assumption of availability of infinite buffer. But this is not the case in reality as the buffer size available at each node is considered to be limited. Redundant message drop incase of buffer overflow significantly degrades performance of most protocols.

Recent advances in application based routing designs and realistic resource assumptions have shifted the interest of the research community towards designing efficient resource management schemes besides efficient routing ones. This gives a whole new perspective to the concept of protocol performance optimization. This diverts latest research interest towards buffer management. Buffer management is a fundamental technology which controls the assignment of buffer resources. It allows message differentiation according to certain priorities or criterions. An efficient and intelligent buffer management policy is required at each contact, to decide which messages should be dropped when buffer reaches its full capacity and which messages should be transmitted first when bandwidth is limited, independent of the routing strategy being used.

2.1. BUFFER MANAGEMENT IN DTNs

The combination of long-term storage and message replication imposes a high storage and bandwidth overhead. This gives rise to scheduling and buffer management problems in a resource constrained DTN. Figure 2.1 describes basic buffer management architecture at a DTN router. Upon arrival an incoming message is classified according to the criterion implemented by that router and then stored in its buffer. At a contact opportunity, the scheduler decides the order by which messages should be transmitted as contact durations are limited. In case of buffer overflow, which messages should be discarded is decided by the drop module which uses buffer status and incoming message size. In other words scheduling strategies govern the order in which the messages are passed in case of bandwidth and contact constraints. On the other hand, buffer management policies decide which messages are deleted first in case of congestion.

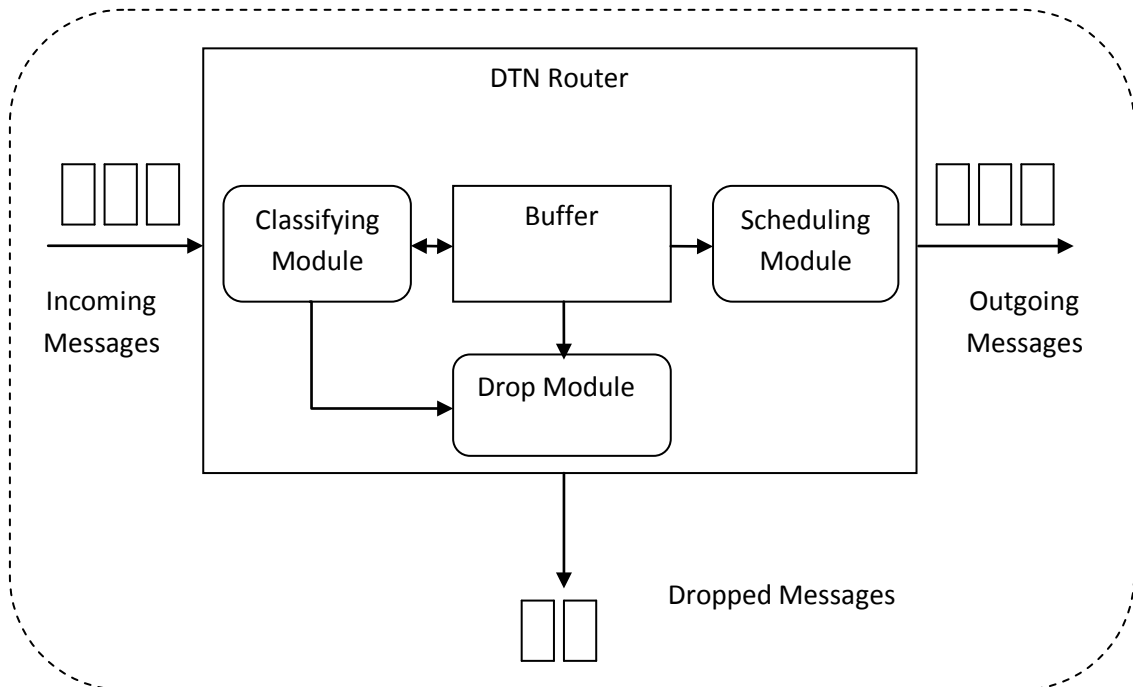


Figure 2.1: DTN Buffer Management Architecture

Several literatures [27], [28], [19] have addressed transmission scheduling and buffer management problems. However Erramilli *et al* in [29], demonstrate that it is important to study forwarding and dropping policies independently of each other. The authors conclude that the latter schemes perform better in terms of delivery rate, delay and cost. Based on their observations, the authors also state that forwarding policies have less impact in the network performance than dropping policies.

In normal conditions the drop occurs when message finds destination or its Time-To-Live (TTL) expires. However when buffers frequently become full, messages can be dropped before their lifetime expires. In our thesis we also consider the scenarios when a node needs to store new messages and it runs out of buffer space thus causing a state of congestion in the network.

The main causes of congestion in opportunistic networks are:

- Retransmission of messages that are either travelling in the network or have already been successfully delivered.
- Retransmission of messages that have been dropped before arriving at their destination resulting in bandwidth wastage.
- Discarding fragments of messages while they are on the way to the destination. Such messages are then discarded by the receiver as they will not reassemble.
- Increased overhead due to control traffic injected into the network to increase chances of successful message delivery.

In addition, such networks suffer from contention under a high traffic load. The resulting redundant and blind message drops significantly degrade performance throughput of such networks. In such scenarios carefully designed buffer management policies significantly affect the performance of routing protocols. An efficient buffer management policy aims at minimizing the negative impact of this drop.

2.2. TAXONOMY OF EXISTING BUFFER MANAGEMENT POLICIES

For increasing the delivery probability in a DTN, it is critical to drop such messages upon a full buffer which are the least likely to be delivered to the final destination. However due to the random topology of a DTN, it might be impossible to determine that. As alternatives, several buffer management schemes have been proposed which consider various message attributes in order to select the appropriate messages to drop. These existing buffer management policies can be broadly divided into two categories: *Policies without network wide knowledge* and *Policies with network wide knowledge*.

Recently there is a shift of interest towards designing efficient buffer management schemes. New attributes are being exploited for making the most appropriate selection. Hybrid schemes are also being tested where more than one attribute are being considered. However most of these new schemes are variants of the schemes mentioned below. Therefore we focus our research to the basic criteria involved in message selection.

2.2.1. Policies without Network Wide Knowledge

Some buffer management policies only depend on local information about messages (e.g. message arrival time, Time-To-Live (TTL), and size etc), to decide how to manage the messages in the buffer. These policies do not take network wide information into account. The main advantage of these schemes is that they are mostly independent of the routing strategy being applied underneath and hence can be used with a variety of protocols. Some popular examples are as follows:

2.2.1.1. Drop Front (DF)

Drop Front (DF) policy drops messages from the head of the queue. It handles the message queue in a FIFO (First In, First Out) order based on the message receive time. The message that is first to get into the queue is the first message to be discarded when the buffer overflows. This policy can result in satisfactory performance as it allows maximum stay time of messages in the buffer however it requires maximum end to end connectivity which is not much common in DTNs.

2.2.1.2. Drop Last (DL)

Like Drop Front (DF), Drop Last (DL) also uses message arriving order as criteria to drop messages. The newly received message is simply removed first from the tail of the queue. In other words the message with the minimum arrival time in the queue will be selected to drop. In this scheme the probability of dropping the source messages is extremely high. Whereas according to [28] giving priority to source messages achieves better delivery probability.

2.2.1.3. Drop Random (DR)

In this strategy, when node buffer overflows the router randomly drops the messages from the queue. It simply drops the messages by chance which means that messages would be arbitrarily deleted when the buffer overflows. Hence all messages have equal deletion priority. Such a scheme is simple but it performs poorly in a highly congested and disconnected network. Furthermore it increases redundant message drop which is not a desirable outcome in DTNs.

2.2.1.4. Drop Youngest (DY)

It drops the recently received message that is the message with the longest remaining life time is deleted first. The basic idea is that such messages have consumed the least amount of network resources at that time and dropping such messages will cause the least overhead

ratio. However source messages are more likely to be dropped in this scheme which significantly effects delivery probability.

2.2.1.5. Drop Oldest (DOA)

According to this scheme [30] the message with the longest stay time in buffer will be dropped. The basic idea is that the message with the longest time in the buffer has less probability to be forwarded to other nodes or it has already been delivered.

2.2.1.6. Evict Shortest Life Time First (SHLI)

As mentioned earlier each message has a lifetime which specifies the time till it is meaningful to its application. After this time period is over, the message is no longer useful and should be deleted from the network. According to [28], this policy drops messages with short remaining life time first since they will soon be dropped anyway. SHLI increases delivery probability of the message however it also increases message drop ratio.

2.2.1.7. Drop Largest (DLA)

Drop Largest (DLA) [31] is a recently proposed popular drop policy which utilizes the size of the message as an attribute to select messages to drop upon a full buffer. This scheme selects only the large sized messages residing in the buffer to be dropped incase of buffer overflow. This gives small sized messages more chance to be forwarded. The mentioned scheme aims at controlling unnecessary message drops thus it significantly reduce overhead and raise delivery probability thus consequently increasing network throughput to a much greater extent. Another advantage of this scheme is its ability to accommodate a large number of incoming messages with only fewer drops.

2.2.2. Policies with Network Wide Knowledge

These buffer management polices require some or complete knowledge about the network (e.g. number of nodes in the network, number of copies of message, meeting rate between two nodes etc.), to make their message discarding decision. Such policies can be further divided into two types based on the extent of their network information requirements. These are: *Policies with complete network knowledge* and *Policies with partial network knowledge*.

2.2.2.1. Policies with Complete Network Knowledge

Based on the network-wide information several buffer management methods have been proposed to enhance the network performance. Some of them are utility-based schemes,

which use the complete information related to the whole network to derive a per-utility value of a message for a certain routing metric (e.g., end-to-end delay, delivery rate) and manage messages based on that utility.

These schemes mostly produce optimal results; however acquiring timely and reliable network information is a challenge. This limits the applicability of these schemes. Some popular schemes that fall into this category are mentioned below.

a) RAPID's Utility Based Drop

RAPID or Resource Allocation Protocol for Incidental DTN routing [25] implements a buffer management scheme that considers network wide information. RAPID applies a heuristic approach which locally optimizes the marginal utility (i.e., the expected delay per message). Per message utility is the expected input of a message to the given routing metric. For example, the metric for minimizing delay is measured by adding the delay of all messages. As a result, this utility is the expected delay of the message.

In order to derive the respective message utilities, RAPID floods information about all of its copies, into the network. However this propagated information may be obsolete because of the intermittent nature of DTNs. RAPID is only a sub-optimal policy in a number of ways and can not provide the optimal network performance.

b) Global Knowledge Based Drop (GBD)

An optimal buffer management strategy is presented in [27] that manages messages in the node's buffer based on global knowledge about the network. It uses statistical learning to derive per-message utility based on node's contact history. It can be adjusted to achieve either of the two goals i.e. to minimize the average delivery delay or to maximize the average delivery rate. Nevertheless, it runs efficiently only when the mobility patterns of the nodes are deterministic, the transfer bandwidth is infinite and the message size is identical. These conditions are impractical in realistic network. Furthermore GBD is based on global knowledge about the state of the network which is difficult to acquire, hence GBD is difficult to implement.

c) History Based Drop (HBD)

As mentioned earlier acquiring global knowledge is difficult making GBD difficult to implement, thus, a deployable variant of GBD was suggested in [27] called the History Based Drop (HBD). It uses local knowledge to estimate global values. The new utilities are based on estimates of the number of nodes that have seen the message and those whom have a copy of

it. Being a variant of GBD, it too has unrealistic assumptions about the state of the network like deterministic node mobility, unlimited bandwidth and same message size. This limits the practical implementation of this approach severely.

2.2.2.2. Policies with Partial Network Knowledge

As mentioned in the previous section policies using complete network knowledge may produce optimal results but are difficult to implement. Therefore many buffer management policies are designed to improve certain metrics of message delivery based on partial network information (e.g., the number of hops, the meeting frequency between pair nodes etc), that is correlated with the messages.

a) Evict Most Forwarded First (MOFO)

Evict Most Forwarded First (MOFO) [28] as the name suggests attempts to enhance the successful dissemination of messages through the network by discarding the message that has been forwarded the maximum number of times. This enables messages with a lower hop count to travel further within the network.

In order to maximize the delivery ratio of messages, each node keeps track of the number of times each message has been forwarded to some other node. The message that has been forwarded the largest number of times is the first to be dropped. It is considered that such a message is liable to have the largest number of copies in the network. Dropping such messages gives messages with fewer copies more chances of being forwarded at a contact opportunity. MOFO increases delivery probability but it usually incurs a large overhead.

b) Evict Most Probable First (MOPR)

In this scheme [28] each message is assigned a Forwarding Predictability (FP) value, initially assigned to 0. After each forward the FP value is modified. In case of buffer overflow, the message that is dropped first is the one with the maximum FP value. It performs well but requires complex knowledge to estimate the predictability value.

c) Evict Least Probable First (LEPR)

On contrary to MOPR, Evict Least Probable First (LEPR) [28] policy drops the messages for which it has the lowest FP value. The idea is that a node is less liable to successfully deliver such a message which has a low Forwarding Predictability (FP) value. It performs well in terms of hop count, buffer time average and message drop ratio but like MOPR requires complex knowledge to calculate the FP value.

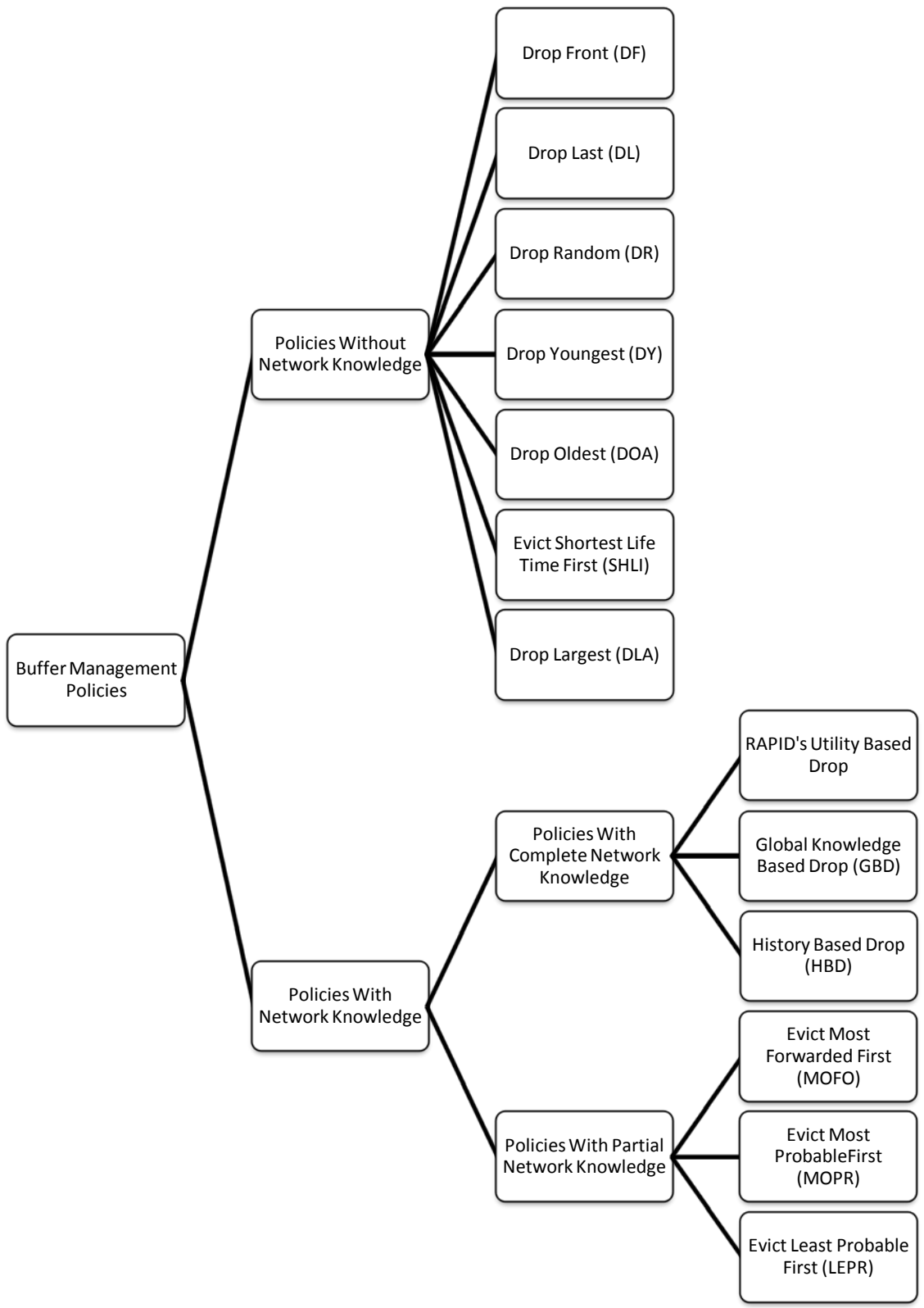


Figure 2.2: Taxonomy of Existing Buffer Management Schemes

Based on its application environment, a buffer management scheme is designed to achieve either of the two key objectives that is *maximizing message delivery ratio* or *minimizing message delivery delay*. Some of the schemes designed, focus on achieving the desired goals but incur a greater overhead. Such schemes are often not desirable in a resource constrained environment. In order to avoid such a case, a secondary objective is gaining popularity that is to control this overhead by reducing redundant message drops. N-Drop [32] and Message Drop Control (MDC) [33] are examples of such schemes. Another scheme [34] suggests that to reduce the negative effect of message drop, the message with the largest number of copies should be dropped. Both N-Drop and MDC, emphasize on the use of a threshold value to control redundant message drop. However both these schemes do not explain how to establish this threshold. Similarly determining the actual number of replicas of any message in a network is rather difficult unless predefined. This shows that there is still a lot of room for improvement and research in this field.

2.3. LIMITATIONS OF EXISTING BUFFER MANAGEMENT POLICIES

Till recently an essential issue mostly disregarded by the DTN researchers was the influence of message drop incase of buffer overflow. This is the reason why traditional buffer management schemes like Drop Front (DF), Drop Last (DL) and Drop Random (DR), do not consider any particular message attribute to make the message drop selection. Instead these schemes rely on the order in which messages arrived or are residing in the buffer. As simple as they may be, such schemes according to [28] are not suitable for DTNs. This may be due to their transition from fully connected networks to intermittently connected ones. That is why their performance is better in frequently connected networks rather than frequently disconnected ones.

Buffer management schemes like GBD and HBD utilize complete network-wide information which enables them to outperform the ones based on local information. By deriving the per message utility, these schemes can either increase the average delivery rate or decrease the delivery delay. However, most of these utility-based policies are heuristic and are not designed to provide optimal results in the opportunistic networks. Moreover topology of the opportunistic networks usually changes frequently. This makes acquiring complete network-wide information in time quite difficult. Moreover some schemes [35] use in-band control channels to exchange large metadata between nodes which incurs both delay and network overhead.

Other schemes merely take partial network-wide information into account instead of relying on deriving utility function to decide the drop policies in order to improve the performance of networks. Most of these schemes like Evict Most Forwarded First (MOFO) use distance in terms of hop count or number of forwards to determine appropriate messages to drop. Allowing messages to cover maximum distance before discarding them incurs much overhead. Other policies like [36] use acknowledgment or ACK messages or anti packets to discard copies of messages already delivered. Again flooding ACKs in the network not only incurs congestion but routing overhead as well.

Policies with local information or no network-wide information may not perform as well as those with partial or complete network wide information but they are easy to implement and incur less routing overhead. However selecting the most appropriate attribute for dropping messages is a challenging issue. Time based drop policies is a popular choice in this group. Policies like Drop Oldest (DOA), Drop Youngest (DY) and Evict shortest Life Time First (SHLI), are all examples of time based schemes. However a common problem with these schemes is the requirement for some sort of time synchronization between nodes in the network in order to determine the exact life times of the messages. However, this is a much common DTN problem.

Some policies use message priority as a criterion for selection of messages to be dropped. These are examples of policies with no network-wide knowledge as the message classification is limited to a node's vicinity. Messages have a high or low priority only at the source or transmitting node. Priority is not considered amongst inter node messages. Prioritized Epidemic (PREP) [37] is an example of such a scheme. However setting priorities amongst messages is still ambiguous.

Recently size is being considered as a selection criterion for message discard [38], [39], [40]. It does not require extra information transfer or complex computation. The key advantage of a size based schemes is the control on redundant message drop. Such schemes not only increase message delivery probability but reduce overhead making them best suited for constrained environments. Drop Largest (DLA) is a recently proposed popular size based drop policy which discards the message with the largest size. Effective as it may be this scheme is a bit naive especially when an arbitrarily large bundle size is favored in DTNs. Till now very little work has been done in this area making it interesting to observe its effects in various scenarios.

2.4. PROBLEM STATEMENT

Problems addressed in this thesis are two fold.

- As mentioned earlier, the performance of flooding based protocols severely degrades under congestion and buffer constraints due to frequent buffer overflows. It is however believed that with proper buffer management, the adverse effects of buffer overflow on message delivery probability can be alleviated. However little work has been done to substantiate this claim. In this thesis we aim to study the positive effects of buffer management on the performance of flooding based schemes both in terms of delivery probability and overhead ratio. For this purpose we have set up two scenarios using two different mobility models and two different flooding based schemes (i.e. Epidemic and PRoPHET). The resultant performance is evaluated and compared with that of two popular quota based schemes (i.e. Spray and Wait and EBR) to exhibit that with proper buffer management, flooding based schemes can outperform quota based protocols in terms of message delivery probability with comparable routing overhead. This makes them suitable for application in most resource constrained and challenging environments like a post disaster response network.
- It is also mentioned earlier that policies with complete or partial network-wide information tend to outperform those with local or no network-wide knowledge. However we believe that this may not be the case in every scenario. The key is to identify the appropriate criterion for message selection according to the requirements of the scenario. We believe that message size is a suitable criterion in case of a bandwidth and storage constrained environment, like a post disaster response network. We intend to evaluate and compare its performance with its counterparts in two different constrained environments including a disaster scenario. For this purpose we propose an effective and efficient scheme called Size Aware Drop (SA-Drop) which aims at performance optimization without incurring undue overhead. The objective is to intelligently select messages and to control redundant message drop as random or naive message drop causes excessive retransmissions and increases communication overhead. Simulation results confirm our hypothesis and thus we conclude that with adequate buffer management schemes, smaller buffers can be used without negative impact on the message delivery ratio and routing overhead, resulting in significant performance optimization of otherwise resource consuming flooding based protocols.

Chapter 3

SIZE AWARE DROP (SA-DROP)

As discussed in the previous sections, whenever the size of the arriving message is greater than the available space in the buffer, some messages are discarded to make room for it. However, each message drop is accompanied by wastage of valuable resources that the message has already consumed and further resource consumption due to unnecessary retransmissions. To minimize this negative impact of buffer overflow, it is important to control redundant message drop. For achieving this purpose, various schemes apply various criteria to determine which messages should be dropped. Time based schemes use message Time-To-Live (TTL) to make this decision, assuming that a message with short TTL will soon be removed from the network so dropping such a message is the most logical choice. Other schemes consider distance travelled by the message in the network. Such schemes assume that a message passed through a larger number of hops is either likely to have any one of its copies already delivered or is less likely to reach to destination at all.

Recently J. Whitbeck *et al* in [41] determine that given a certain maximum delay and node mobility, bundle size has a major impact on the delivery probability. We believe that size can be a useful attribute for message drop selection especially in resource constrained environments. Dropping messages based on their size can avoid unnecessary drops and consequently reduce excessive retransmissions which contribute to small delivery probability and greater overhead due to increased resource consumption.

Size based drop policies are a recent development [40], [38], [39]. Q. Ayub *et al* in [42] suggest that smaller messages cover maximum distance in one time slot than larger ones. Due to bandwidth constraints, large sized messages may not be completely transferred during a single contact. This may lead to partial transmissions and fragmentation, both leading to increased resource consumption and overhead ratio. S. Rashid *et al* in [43] suggest optimization of Epidemic by proposing a scheduling and dropping scheme based on this idea. Smaller messages are selected first for transmission while larger messages are dropped first in case of buffer overflow. As mentioned in the previous chapters, due to frequent disconnections and excessive node mobility in DTN architecture, the bundles are often significantly large to enable complete application interactions with a single message exchange. Therefore always selecting larger messages to drop is not favorable for their chances of successful delivery.

3.1. PROPOSED BUFFER MANAGEMENT POLICY

In this thesis we propose a size based policy which determines a threshold size in order to select a message to be discarded in case of buffer overflow. The basic rationale behind this scheme is that by determining the exact buffer space requirement we can easily select a message of appropriate size to be discarded. By doing so, we can control excessive message drop and avoid biasness towards any particular sized message.

Our proposed scheme Size Aware Drop (SA-Drop) implies a simple but effective mechanism to determine this threshold size. If the size of the incoming message is greater than the available buffer space, our scheme follows few simple steps. It first calculates the difference between the incoming message size and the available buffer space. By doing so it estimates a threshold value which it considers as the threshold size. Now any message residing in the buffer of the receiving node, which is equal to or greater than this threshold size is selected for the drop. Figure 3.1 shows the model for estimation of threshold size for message selection.

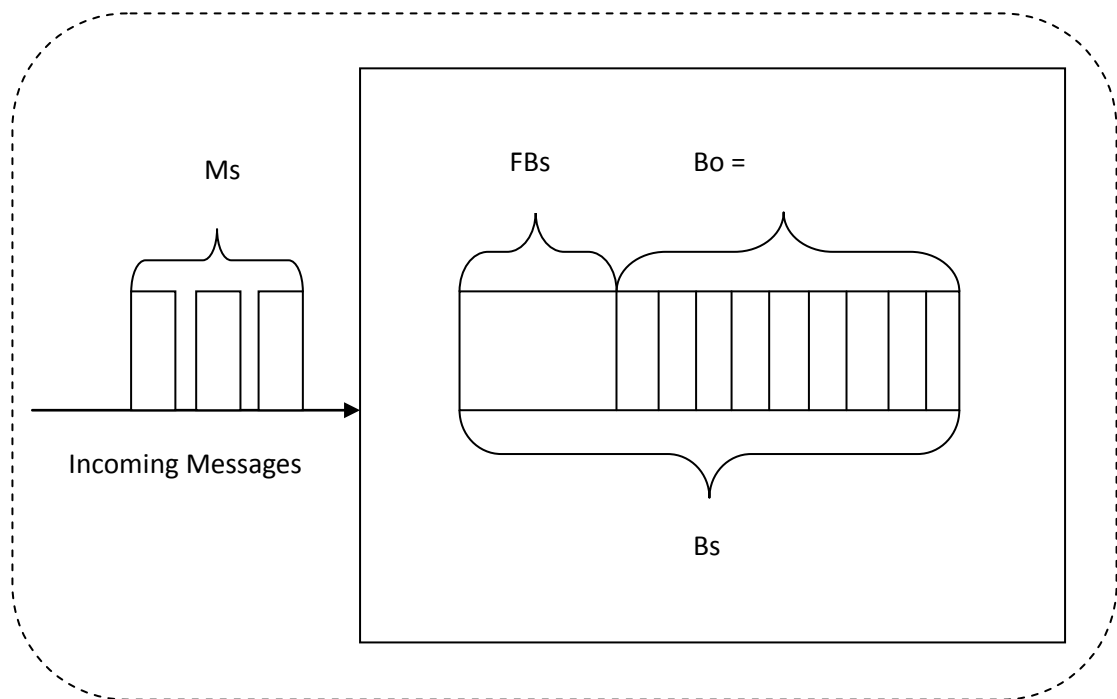


Figure 3.1: Proposed Model for Estimation of Threshold Size for Message Drop

3.1.1. METHODOLOGY

The purpose of any buffer management policy is to drop a message when a new message arrives at the node and existing storage is unable to carry it. We consider a snapshot of two intermittently connected mobile nodes A, B having M buffered messages; BSA represents the size of node A's buffer while BSB represents that of Node B's buffer size. For ease of comparison we assume both buffer sizes to be equal to 2 MB. According to [4] each bundle carries information like the Message Identity (Mid), Message Size (MS), Message Time-To-Live (TTL) and Destination (D). Additional information can also be augmented in the header like the Number of Forwards (NoF). Nodes usually maintain information like Message Arrival Time (AT) of the messages in their buffers. Nodes exchange messages according to some scheduling policy whenever they are within transmission range of each other. For ease of understanding we will consider First In First Out (FIFO) as our scheduling policy.

In order to explain the methodology of our scheme, we assume a network model where the two nodes encounter each other with the rate following a Poisson distribution. This rate λ is defined as the average number of encounters in time T. Mathematically this rate of encounter can be described as $\lambda = 1/E [U]$ where E [U] is the average meeting time. Similar assumption has been made in [27].

Table 3.1: Message Summary Table at Node A and B

Node	Mid	MS	NoF	AT	RTTL
A			20	15s	60min
			12	18s	32min
			16	30s	120min
			1	12s	40min
			10	25s	220min
			18	23s	240min
B			27	20s	32min
			23	10s	24min
			12	24s	120min
			11	19s	360min
			14	13s	150min
			21	32s	110min

We now consider some popular dropping mechanisms in order to understand their working in comparison to our own. Let us assume that both nodes A and B maintain the Table 3.1. We selected Drop Oldest (DOA), Drop Largest (DLA), Evict Shortest Life Time First (SHLI) and Evict Most Forwarded First (MOFO) schemes to explain their behavior in case of buffer overflow. As mentioned earlier when Node A comes in contact with Node B, Node A starts transferring messages to Node B in FIFO order. This assumption is subject to already known

assumptions like source messages and the messages destined for B will be delivered first and messages already present in B will not be again transferred to it. However we have assumed unique messages to simplify our scenario. So the first message to be transferred to B will be the one that arrived first (AT = 12s) at A which is M4 of size 800 KB. Buffer space available at Node B is 100 KB which is insufficient to accommodate the incoming message M4. Node B has to discard some messages to make room for it.

Incase of Drop Oldest (DOA), the message that Node B will discard first will be the one that arrived the earliest that is M8 of size 100 KB (AT = 10s). The message drop sequence at Node B will be {M8 (AT=10s), M11 (AT=13s), M10 (AT=19s)}.

Next we will observe the mechanism of Evict Most Forwarded First (MOFO). According to MOFO, the message that Node B will discard first will be the one that is forwarded for the most number of times that is M7 as it was forwarded 27 times. At Node B, the drop sequence of messages will be {M7 (NoF=27), M8 (NoF=23), M12 (NoF=21), M11 (NoF=14), M9 (NoF=12)}.

Evict Shortest Life Time First (SHLI) discards messages with shortest remaining Time-To-Live (TTL). TTL is updated each time a message is forwarded by subtracting the time it spent at the node. Hop count is increased with every forward while TTL is decreased. A message with TTL equal or less than zero is removed from the network automatically. Hence SHLI makes use of the fact that a message nearing its TTL should not affect the network throughput much. The message drop sequence at Node B with SHLI will be {M8 (RTTL=24min), M7 (RTTL=32min), M12 (RTTL=110min), M9 (RTTL=120min)}.

Accordingly we observe the dropping sequence of Drop Largest (DLA). With DLA Node B will drop the largest message in its buffer. The message dropped at Node B with DLA will be {M10 (Ms=750 KB)}. It is clearly evident that number of drops is less than the rest. However dropping the largest messages every time the buffer overflows, results in their costly retransmission with chances of being dropped again soon enough.

Finally we consider the working of our proposed scheme Size Aware Drop (SA-Drop) which considers already available buffer size alongwith the size of the incoming message. The available buffer space at Node B is 100 KB and the size of the incoming message is 800 KB. SA-Drop first computes the threshold size by calculating the difference between the arriving message size and the free buffer space.

$$= (2 \text{ MB} - 1.9 \text{ MB}) = 100 \text{ KB}$$

$$= (800 \text{ KB} - 100 \text{ KB}) = 700 \text{ KB}$$

Afterwards, our proposed algorithm scans the Node B's buffer and drops all those messages which are equal to or greater than the *Threshold Size (Ts)* that is 700KB. Hence the message selected for drop at Node B will be {M9 (Ms = 700 KB)} and not the largest that is M10 (Ms = 750).

Although the message drop in our scheme appears to be equivalent to that of DLA policy. However it does not hold the same biasness towards the larger sized messages in every case. Provided the difference between the free buffer space and the incoming message size is lesser like 300 KB then the message threshold value will be 300 KB. This gives fair chance to all sized messages to reach to destination. As an additional optimization, if all buffered messages are not lying within the *Threshold Size (Ts)* then message with the largest size will be dropped.

3.1.2. ALGORITHM

In this section we will present the basic algorithms constituting our entire drop policy SA-Drop. Two main functions called the *makeRoomForMessage()* and *getThresholdMessage()* are used in order to implement our proposed drop policy. *makeRoomForMessage()* is already included in the *ActiveRouter()* class in the ONE simulator which calls upon the function for the implemented drop policy. However we have made certain modifications in this function which include the estimation of *Threshold Size (Ts)* whose value is then passed on to the *getThresholdMessage()* function. Hence first we present the algorithm for modified *makeRoomForMessage()* function followed by the algorithm of our contributed

getThresholdMessage() function. The notations used and their respective descriptions are given in the Table 3.2.

Table 3.2: Table of Notations and Their Description

NOTATIONS	DESCRIPTION
MS	Message Size
Mid	Message Identity
Mai	Messages in Node A's Buffer
MBi	Messages in Node B's Buffer
$i=(1,2,3,\dots,n)$	Message Number
FBs	Free Buffer Size
Ts	Threshold Size
TM	Threshold Message
DMs	Dropped Message Size
LM	Largest Message

```

Algorithm 1 makeRoomForMessage( )


---


Let A and B be nodes
Let MAi and MBi be messages held by nodes A and B where  $(i=1,2,3,\dots,n)$ 
On contact node A sends MAi to node B in FIFO order
At node B:
Begin
    Ms ← MAi.Size()
    if ( Ms > FBs) then
        Ts ← Ms – FBs
        while ( Ms > FBs)
            do
                TM ← getThresholdMessage(Ts)
                deleteMessage(TM.id)
                DMs ← TM.Size()
                FBs ← FBs+DMs
            end while
        end if
    end

```

Figure 3.2: Algorithm 1 for makeRoomForMessage() Function

The key benefit of using our proposed method is that each node can use local information to decide which message to discard without appending any extra information to the header like Number of Forwards (NoF). Moreover it helps to avoid ambiguities as in time based drop policies due to non synchronization of clocks at all the nodes in the network. Our scheme can be used with any of the known DTN routing protocols thus making it suitable for a number of applications. Our proposed mechanism operates by using only local information and thus is easy to implement. Simulations confirm that it is effective across the network.

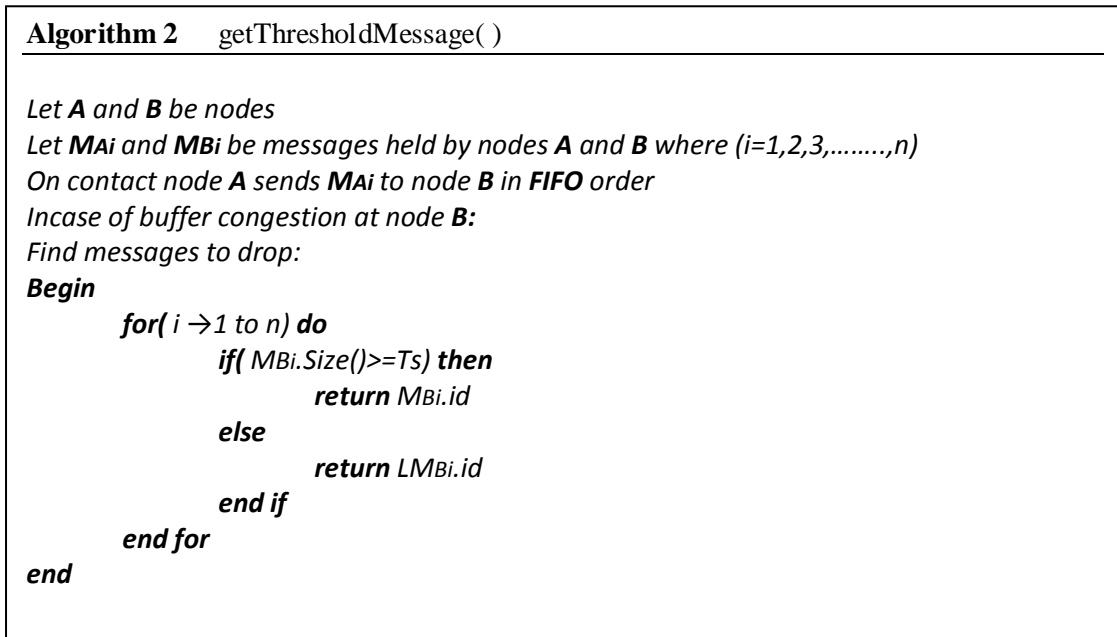


Figure 3.3: Algorithm 2 for `getThresholdMessage()` Function

3.1.3. SIMULATION SETUP

In order to confirm the effectiveness of our proposed scheme SA-Drop, we carry out a number of simulations.

3.1.3.1. Simulation Tool

Studies confirm that the performance of DTN routing protocols is highly dependent on the characteristics and mobility of the nodes involved. Relatively a new paradigm, performance evaluation of DTN protocols requires apt simulation tools. The Opportunistic Networking Environment (ONE) [44] simulator is exclusively designed for this purpose. It is a discrete, Java based, event simulator which offers a wide set of capabilities in a single framework for simulating DTN protocols in numerous scenarios. For installation and system requirements refer to [44]. The key features of the ONE simulator that make it suitable for DTN evaluation and analysis are as follows:

- It already includes some popular DTN protocols and offers a rather simple framework for users to implement their own protocols by extending the *ActiveRouter()* class. There are six popular DTN routing protocols included in this simulator i.e. 1) Direct Delivery, 2) First Contact, 3) Epidemic, 4) Spray-And-Wait, 5) PRoPHET and 6) MaxProp [45].
- It allows users to easily create different scenarios based upon different movement models. The simulator has several popularly used movement models varying from synthetic

models to real movement traces, integrated in it. Three synthetic movement models are included in the movement module which are 1) Random movement, 2) Map-constrained random movement, and 3) Human behavior based movement. Movement data can be also be imported from an external source.

- The ONE simulator utilizes the concept of location, communication range and connection throughput. The movement models provide the node position data that is used for determining whether any two nodes can communicate or exchange messages. This information can also be exported to external simulators (e.g. NS-2 and DTNSim-2) or it can be imported from them too.
- The ONE simulator provides interactive visualization (both in batch and GUI mode) and post-processing tools support for the protocol performance evaluation. This makes the ONE simulator an integral part of a real-world DTN test bed. The basic idea is that protocols can be evaluated under different settings which can be tuned to match the projected scenario as realistically as possible. A configuration file enables the users to design their own specific scenario settings. The ONE simulator reads the configuration file and generates an output trace file or report file. A number of report file options give users the ease to analyze various performance effects of parameter variations.
- The ONE simulator is designed in a modular structure to allow easy extension of various functionalities with the help of well-defined interfaces.
- Functionalities of other simulators such as NS-2 [46] or DTNSim-2 [47] can also be used in combination with the ONE simulator. Mobility and connectivity data can be exported to other programs by the report module. Similarly, the results of routing simulations can be imported back into the ONE by using the external scripts. Real-world traces can be incorporated into the ONE simulator from other mobility generators with ease. The reason for this is that it has flexible input and output interfaces. It can also generate mobility traces for other simulators to process.

At the present, the ONE simulator is able to create numerous simulation scenarios for DTN performance analyses. However it is a relatively young tool and is constantly evolving. The present versions of the ONE simulator have certain limitations as well.

- The simulation time is increased in finite steps. If a transfer is complete or there is a disconnection due to node mobility, the simulator assumes that the event has happened either before or after one complete time step.
- The size of the network and some specific mobility models can affect the processing capabilities of the system and influence simulation speed indirectly. This limits the size of the scenarios that can be simulated by using the ONE simulator. However we can still

simulate a scenario with one thousand nodes at a speed of over 10 simulated seconds per second on a regular machine.

- At the moment the ONE simulator does not take into consideration the lower layer (e.g. physical and MAC) support. Due of this, in many scenarios, the ONE can generate contact times which can be too optimistic.

Despite these certain limitations this simulator is still widely being used by the DTN research community for the design and evaluation of DTN routing protocols. Therefore we will also be opting for this simulator for our thesis. Figure 3.4 gives a schematic overview of the ONE simulation environment.

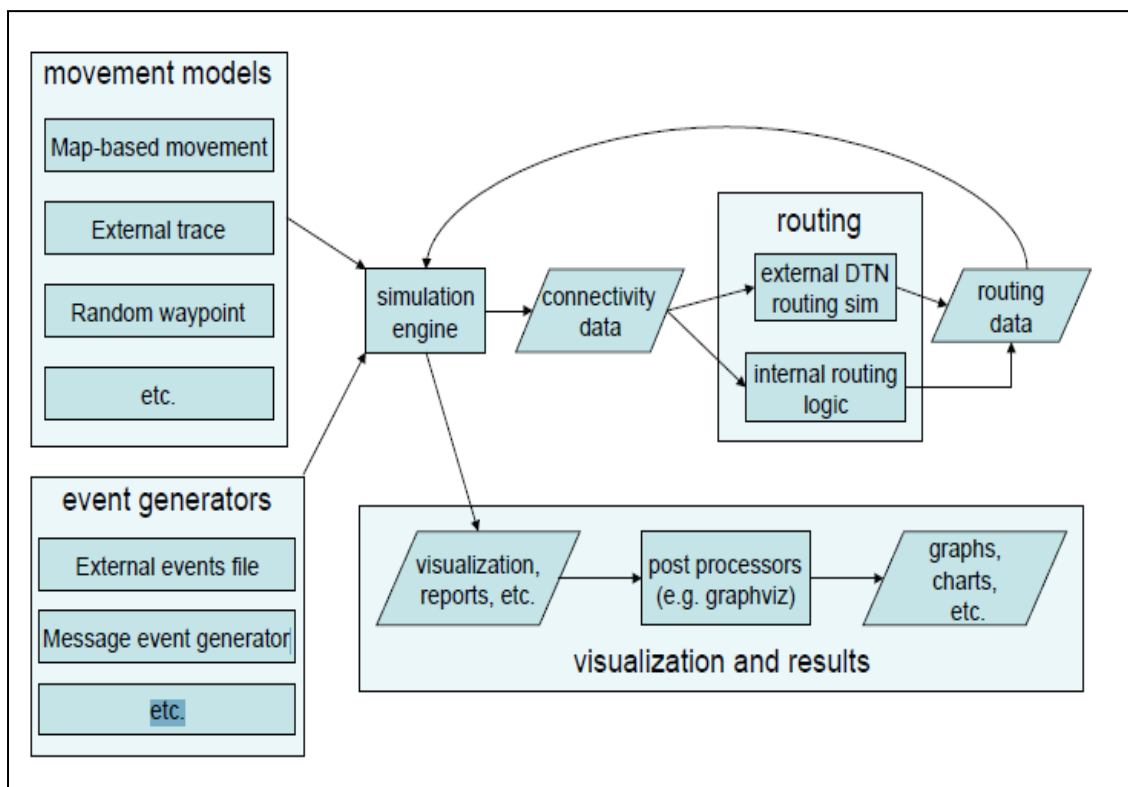


Figure 3.4: Schematic Overview of the ONE Simulation Environment [48]

3.1.3.2. Mobility Models

To evaluate the performance of our proposed scheme SA-Drop we carried out our simulations in two different scenarios. One with a synthetic mobility model called Random Waypoint Mobility Model [49] and the other is a circumstantial model which simulates the nodes movement patterns in a disaster scenario, called the Event Driven, Role Based Disaster Mobility Model [50].

a) Random Waypoint Mobility Model (RWP)

Our first scenario uses Random Waypoint (RWP) Mobility Model. It is a popular synthetic model used to mimic pedestrian movements. Scenarios with different network densities or velocities can easily be generated in it. Initially a node stays for a certain period of time at a single location and afterwards chooses a random destination with a uniform speed. After arriving at the destination, the node pauses for a specific time period before starting the process again. Figure 3.5 shows the placement of nodes in Random Waypoint at the start and the end of the simulation. Like most synthetic models Random Waypoint Mobility Model assumes independent and identically random movements of the mobile nodes. This results in a memory less property. We use this model to establish base line results.

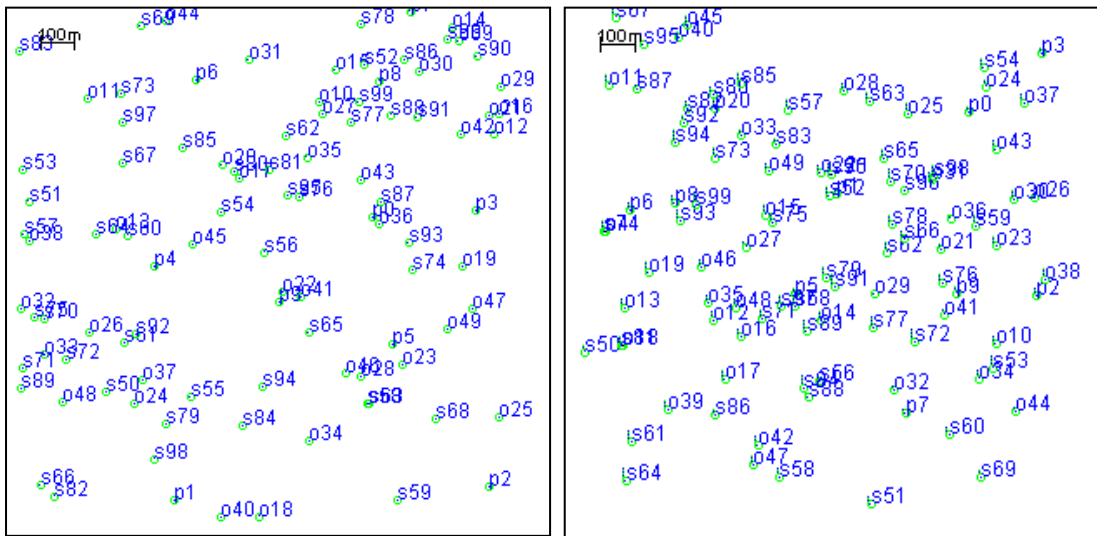


Figure 3.5: RWP: Snapshot of Node Placement at 0 and 3600 seconds

b) Event Driven, Role Based Disaster Mobility Model

The drawback of using a traditional synthetic mobility model like RWP is that such models do not capture the actual node movement as in real life applications. The independent and identical movement assumption of such models is too simple and straightforward. Their performance predictions may react differently in actual deployments. Real mobility traces like CRAWDAD [51] are also available however these do not fit for our specific scenario of a disaster. For this purpose we required a near realistic synthetic model that simulated movement of nodes in a disaster. Very little work has been done in this area and not many known disaster mobility models exist. We came across three well designed mobility models namely the *Bonnmotion Disaster Mobility Model* [52], *Post Disaster Mobility Model* [53] and *Event Driven, Role Based Disaster Mobility Model* [50]. After evaluating the strengths and weaknesses of all three of them, we decided to use the Event Driven, Role Based Disaster Mobility Model for our disaster scenario simulation.

One of the advantages of this model is that it captures the distinct movement patterns of roles in real life disaster scenario (i.e. civilians, ambulances and police cars). Each role reacts differently to an external event as is the usual case in a realistic scenario. The civilians are moving away from the disaster event area while ambulances and police cars approach it. Ambulances oscillate between the event area and the hospitals or relief centers while police cars stay at or patrol the disaster event area. Four hospitals are setup at the four corners of the simulation area. Their placement effects the movement of the mobile nodes. The basic advantage of this model is that it allows users to tune the disaster area as per desire. Users can adjust the number of civilians, the ambulances and the police cars. A snapshot of the node placement at the start and the end of the simulation in this model is shown in Figure 3.6. Four events of intensity 10000 to 20000 occur throughout the simulation which can be seen by the clustering of nodes. The damage radius is kept at 0.1 % of the intensity of the event while event horizon is 2% of its intensity as per the original model parameters.

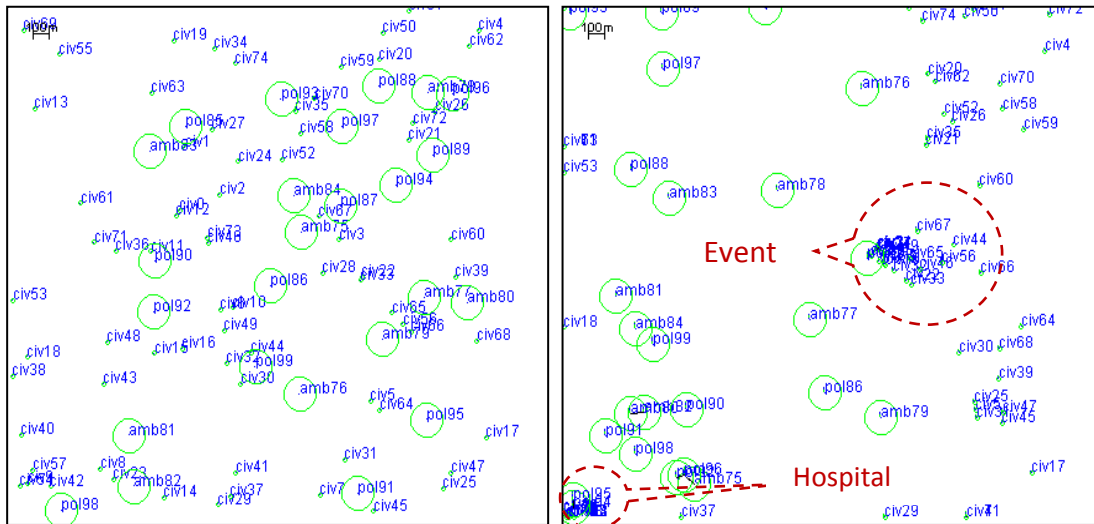


Figure 3.6: Disaster: Snapshot of Node Placement at 0 and 3600 seconds

3.1.3.3. Simulation Parameters

As mentioned in the previous section we carry out simulations in two distinct scenarios. One represents a pure opportunistic environment where nodes move randomly, without any prior knowledge of the network. We use Random Waypoint Mobility Model to simulate such an environment. The second scenario mimics the movement of nodes in a disaster. We use Event Driven, Role Based Disaster Mobility Model for this scenario. As mentioned in the previous sections the ONE simulator allows users to tune a scenario by means of configuration files. Two configuration files were created, one for Random Waypoint scenario and the other for the disaster scenario. Some common parameters in both these configuration files are explained here. Three groups of nodes are deployed. The total number of nodes that are

considered for communication is 100. Since we are simulating a constrained environment hence buffer sizes are randomly chosen to be 3MB, 4MB and 5MB for each corresponding group in the RWP scenario while 2MB, 3MB and 4MB respectively for the disaster scenario. Message size is uniformly distributed between 100 KB and 1MB while message generation interval is kept at [15s, 25s] to simulate congestion. Message Time-To-Live (TTL) is kept 60 minutes in both scenarios, which is equal to simulation run time. This is to ensure that all messages get equal priority to reach to destination. The simulation is run for 3600 seconds in both scenarios.

a) Scenario 1: Random Waypoint Scenario

In our first scenario we implement Random Waypoint Mobility Model. The network considered here has random behavior and no knowledge about the network is known a priori. Epidemic is the better choice of routing in such kind of networks. We compare performance of optimized Epidemic with Spray and Wait. This mobility model is already integrated in the ONE simulator as part of its synthetic mobility models. Table 3.3 gives simulation parameter for this scenario.

Table 3.3: RWP: Table of Simulation Parameters

PARAMETERS	VALUES		
	GROUP 1	GROUP 2	GROUP 3
Number of nodes	10	40	50
Movement Model	Random Waypoint		
Buffer Sizes (MB)	3MB	4MB	5MB
Group Interface	Bluetooth		
Transmission Speed	250 kbps		
Transmission Range(m)	10m		
Nodes Speed(m/s)	0.5-1.5(m/s)		
Message Time-To-Live (TTL)	60 min		
Message Creation interval	15s-25s		
Message Sizes	100kB-1MB		
Area of simulation	1500mx1500m		

As mentioned earlier nodes are divided into three groups. However all groups simulate pedestrian movement where nodes are moving with speed of 0.5 m/s to 1.5 m/s. Number of nodes and buffer sizes are varying in each group. Group 1 comprises of 10 nodes with buffer size equal to 3MB. The remaining two groups comprise of 40 and 50 nodes respectively with buffer sizes equal to 4MB and 5MB respectively. The interface chosen for all groups is the Bluetooth interface with bandwidth equal to 250 kbps and transmission range of 10 m. Group wait time is 0 to 120 seconds. The simulation area is 1500 m x 1500 m.

b) Scenario 2: Disaster Scenario

Our second scenario mimics that of a disaster one. As mentioned earlier we implemented a disaster mobility model called the Event Driven, Role Based Disaster Mobility Model. Table 3.4 gives the simulation parameters for this scenario.

Table 3.4: Disaster: Table of Simulation Parameters

PARAMETERS	VALUES		
	GROUP 1 (Civilians)	GROUP 2 (Ambulances)	GROUP 3 (Police)
Number of nodes	75	10	15
Movement Model	Event Based, Role Driven Disaster Mobility		
Buffer Sizes (MB)	2MB	3MB	4MB
Group Interface	Bluetooth	Wi-Fi	Wi-Fi
Transmission Speed	250 kbps	10 Mbps	10 Mbps
Transmission Range(m)	10m	100m	100m
Nodes Speed(m/s)	1-2(m/s)	7-10(m/s)	7-10(m/s)
Message Time-To-Live (TTL)	60 mins		
Message Creation interval	15s-25s		
Message Sizes	100kB-1MB		
Area of simulation	3000mx3000m		

As mentioned earlier nodes are divided into three groups to simulate role based movement. Group 1 represents civilian movement and consists of 75 nodes. These nodes move with the speed of 1 m/s to 2 m/s are node movement in disaster is slightly faster. The interface chosen for this group is the Bluetooth interface with bandwidth equal to 250 kbps and transmission range of 10 m. Buffer size of Group 1 is kept small that is 2 MB. The movement of these nodes is slightly clustered around the event area and the hospitals as civilian mobility is somewhat restricted in a disaster scenario. These nodes are fleeing from the disaster area and moving towards the hospitals. 60% of civilians are randomly chosen as curious civilians. They stop at event horizons to look at the event. The remaining two groups simulate responders' movements. One group represents ambulances and has 10 nodes. These nodes oscillate between hospitals and the event area. The other group consists of 15 nodes and represents police car movement. They visit the event area and the hospitals and even stay there for sometimes. The minimum speed of nodes in these groups is kept at 7 m/s while maximum speed is 10 m/s. The interface for these nodes is chosen to be Wi-Fi interface with bandwidth equal to 10 Mbps and transmission range is kept at 100 m. One of these groups is assigned an arbitrary buffer size of 3 MB while the other is assigned 4 MB to simulate a constrained environment.

In this model, four events randomly occur. The first event occurs between 100 and 200 seconds while the second event occurs between 125 and 225 seconds. Similarly the third event occurs between 150 and 250 seconds while the fourth and the final event occurs

between 175 and 275 seconds. After each event, a radio contact randomly occurs between 40 to 80 seconds. This is simulated by four message generation events. The first event occurs at 240s till 280s, the second event occurs at 265s till 305s, the third event occurs at 290s till 330s and the final event occurs at 315s till 355s. This ensures uniform distribution of message generation throughout the simulations. The simulation area is kept at 3000 m x 3000 m.

Since PRoPHet utilizes node's contact history to determine a better relay, traces generated by this mobility model are better suited to fully evaluate this protocol's capabilities. Therefore we map plain PRoPHET, Encounter Based Routing (EBR) and optimized PRoPHET with existing and our proposed buffer management policies in this mobility model.

3.1.4. IMPLEMENTATIONS

As mentioned earlier the ONE simulator comes with the implementation of six popular DTN protocols by default. Therefore we are utilizing Epidemic, PRoPHET and Spray and Wait implementations already incorporated in the ONE simulator. For class file source code of each refer to Appendix (ii). Moreover the default drop policy already implemented in the *ActiveRouter()* class is the Drop Oldest (DOA), hence we use it *per se* (Appendix (ii)). We However in order to complete our analysis we need to incorporate the following modules:

- Encounter Based Routing (EBR) protocol.
- Event Driven, Role Based Disaster Mobility Model.
- Drop policies:
 - Drop Largest (DLA)
 - Drop Front (DF)
 - Evict Most Forwarded First (MOFO)
 - Evict Shortest Life Time First (SHLI)
 - Our proposed policy Size Aware Drop (SA-Drop)

As mentioned earlier, Encounter Based Routing (EBR) protocol is developed by Sam Nelson *et al* in [18]. Its class file is available as user contributions at the ONE simulator home page. We downloaded it from there and added it to the routing module of the ONE simulator. Class file source code for EBR is available in Appendix (ii).

Next we need to implement the Event Driven, Role Based Disaster Mobility Model again presented by Sam Nelson *et al* in [50]. The implementation of Event Driven, Role Based Disaster Mobility Model is a bit more complex. This model cannot be directly implemented in the ONE simulator; however its traces can be imported as part of *ExternalMovement()* class. The designers of this model have implemented it in Network Simulator-2 (NS-2). Two tools

are required as an extension to NS-2 in order to generate movement traces of this model. Both these tools are available at the home page of their developer.

The first tool is a parameters file generator *paramGen.cc* that creates a properly formatted parameters file. This tool allows the user to customize settings by defining following parameters:

- Size of the network,
- Simulation runtime,
- Number of civilians, ambulances and police.

Following information is contained in the output parameters file:

- Simulation area size and runtime
- Objects and events coordinates
- Speeds for the objects
- Number of curious civilians
- Parameters for Random Walk
- Trigger and radio contact times for the events
- Intensities of events to determine the event horizons and their damage zones

Detailed usage for this tool is given in Appendix (i):

Usage: `./paramGen > paramFile`

The second tool is the mobility trace generator *disasterSimONE.cc*. This tool accepts the parameters file generated by the first tool as input and runs the complete simulation.

Detailed usage for this tool is also given in Appendix (i):

Usage: `disasterSimONE [-d] < paramFile > oneMobilityTrace`

A ONE simulator compatible mobility trace file called *oneMobilityTrace* is the output of this tool. This trace file is imported in the ONE Simulator by using the External Movement Model (refer to Appendix (i)).

Usage:

`ExternalEvents.class= ExternalMovement`

`ExternalMovement.file = oneMobilityTrace`

`Group.movementModel=ExternalMovement`

Finally we implement all the drop policies required for the comparison purposes including our own proposed policy SA-Drop. We implement the rest of the policies by overriding the original function for Drop Oldest (DOA) called *getOldestMessage()* in the *ActiveRouter()* class file. As the name suggests it uses *getReceiveTime()* function to return the oldest message with in the buffer.

```
for (Message m : messages)
{
    if (oldest.getReceiveTime() > m.getReceiveTime())
    {
        oldest = m;
    }
}
return oldest;
```

Following the same pattern we implement *getLargestMessage()* by using *getSize()* function for Drop Largest (DLA) policy. We implement *getMaxForwardedMessage()* by using *getHopCount()* for Evict Most Forwarded First (MOFO) policy. In original MOFO policy an extra bit is appended to the header which contains a variable called the Number of Forwards (NoF) of each message. At each hop the NoF is incremented. In our implementation we call upon the number of hops for which the message has been forwarded. The numbers of hops indicate the number of forwards. In Drop Front (DF) we do not make any comparisons and simply drop the first message returned from the buffer.

Implementations for Evict Shortest Life Time First (SHLI) and our own policy SA-Drop are slightly more complicated. A function called *getTtl()* returns the Time-To-Live (TTL) of a message in the buffer. It is decremented at every forward. However to determine the remaining Time-To-Live (RTTL) of a message residing in a buffer at any instance, we have to subtract the time it spends in the buffer before it is dropped from its updated TTL. Since TTL value is given in minutes and simulation clock time values are given in seconds so we convert TTL value into seconds. We can convert receive time value into minutes as well to achieve similar results. Afterwards we can easily compare the RTTL values of all the messages in the buffer to determine which message to discard.

$$RTTL = (getTtl()*60)-getReceiveTime()$$

Finally we implement our own policy SA-Drop. In order to do so we override two functions that are *makeRoomForMessage()* which calls the dropping function when buffer at the receiving node is full and the function for selecting the message to drop. In

makeRoomForMessage() function we determine the threshold value as described earlier in the algorithm section. This value is then passed as an extra parameter to the message selection function which is in our case called the *getThresholdMessage()*. In this function the sizes of buffered messages are compared with the threshold size and any message equal to or greater than that value is returned to be discarded.

```
for (Message m : messages)
{
    if (m.getSize() >= thresholdsize)
    {
        threshold = m;
    }
}
return threshold;
```

For detailed source codes of all these drop policies refer to Appendix (ii).

Chapter 4

RESULTS AND ANALYSIS

In this chapter we discuss and analyze the results of our simulations described in the previous chapter. The primary goal of our evaluation is to demonstrate that with an appropriate buffer management scheme a flooding based routing protocol can maintain its high message delivery ratio, while incurring considerably low overhead. To demonstrate this, we first present the metrics used in our evaluation. In the next section we carry out comparative analysis of our proposed buffer management scheme i.e. Size Aware Drop (SA-Drop) with existing schemes. To demonstrate its effectiveness on the performance of flooding based protocols (e.g. Epidemic and PRoPHET), we simultaneously compare the results with that of popular existing quota based protocols (e.g. Spray and Wait and EBR) in similar scenarios. Finally we demonstrate the impact of congestion and buffer constraints on delivery probability and overhead ratio achieved by implementing various buffer management schemes including our own SA-Drop.

4.1. EVALUATION METRICS

In order to evaluate the performance of various buffer management strategies, six metrics have been chosen. Among these delivery probability and overhead ratio are our primary concerns. The rest are used to explain the optimization phenomenon of these two parameters.

4.1.1. Delivery Probability

Message delivery probability is one of the most important metrics for evaluation of DTNs because, in such networks, inability to deliver messages within an acceptable amount of time is fairly common. In other words, messages are subject to long delays and can easily be lost. Delivery probability can be defined as the percentage of the total number of messages generated within a given time period that are successfully delivered to the final destination. Basically it is the ratio of number of messages delivered over number of messages created. High probability means that more messages are successfully delivered to the destination. Once created, a message travels through the network trying to reach its destination. In order to do so it consumes valuable resources. Any failure or message loss can cause all these resources to go to waste, hence incurring more overhead. As successful delivery justifies resource consumption, the objective is to increase delivery probability.

4.1.2. Overhead Ratio

One of the design goals of a DTN is to reduce the number of transmissions per message. The number of transmissions mainly depends on the routing strategy employed by the protocol. As a result some protocols transmit more messages than others. This excessive message transmission requires more computational and storage resources resulting in excessive energy and resource consumption. In other words overhead ratio determines the percentage of resources consumed to process the transmission and storage of a message. In case of buffer management policies, some target at achieving high delivery probability at the cost of producing high overhead. Such schemes are not efficient for most resource constrained practical DTN implementations. Objective of an efficient buffer management scheme is to minimize the value of this overhead ratio. Mathematically, it can be represented as:

4.1.3. Buffer Time Average

Delay Tolerant Networks (DTNs) are commonly characterized by long and variable delays. As a result these networks adopt a store-carry-and-forward paradigm to enable message dissemination and to ensure their successful delivery to their destinations. Consequently the delivery probability of a message highly depends upon the time it spends in the buffer of the intermediate node. Studies suggest that increase in the buffer time average guarantees the message transmission towards next hop which might be able to make progress towards destination. Hence the longer a message stays in the buffer, the greater its chances of being successfully delivered to the destination. The schemes that allow the messages to reside longer in the buffers result in better delivery probability. Buffer time average is basically the average time that messages stay in the buffer at each node and can be calculated as follows:

4.1.4. Hop Count Average

In opportunistic networks like DTNs, the delivery of a message is accomplished via multiple hops. On each hop the message consumes network resources such as bandwidth, buffer space and energy. Again the number of hops is mostly dependant on the routing strategy used yet

the objective of a buffer management scheme in a resource constrained environment is to maintain a smaller hop count average. The successful transmission of messages to their destination with minimum number of hops is considered an efficient use of network resources. Hop count is a variable carried in the header of the message and is incremented at every hop

4.1.5. Messages Dropped

Another key objective of any buffer management scheme is to control the number of messages dropped. Controlling message drop has two fold advantages; first the messages get more residing time in the buffer which enhances their delivery probability. Secondly, when a message is dropped, it is a waste of resources (i.e. bandwidth, buffer, and energy) which it has consumed during its mobility hence less drops avoids unnecessary resource wastage.

4.1.6. Messages Relayed

Messages relayed are the number of messages successfully forwarded by a node. On one hand successful message transmission is a sign of better bandwidth utilization as partial transmissions cause bandwidth wastage. However too many transmissions can exhaust resources as well. In this study messages relayed are considered as the replication required for successfully delivering a message to its destination. Therefore, we can say that number of transmissions gives the apparent consumption of resources in terms of buffer space, bandwidth and power. So lesser number of messages relayed is an objective.

Another metric usually considered in DTNs is latency which refers to the various kinds of delays that are usually incurred in processing of network data. In DTNs latency represents the interval between the time a message is generated and the time it is received. Many DTN applications like Inter-Planetary Internets (IPNs) tolerate long waits. In a post disaster scenario, the element of emergency is important; however successful message delivery is more critical than quick dissemination. Further more in a disaster scenario network is comparatively small and localized and mobile agents are usually repetitive in their movements in and out of the network. Therefore the objective of our study is not to minimize latency but to maximize delivery. However we will demonstrate the effects of our scheme on latency as well.

4.2. PROTOCOLS UNDER EVALUATION

We have selected four protocols for the purpose of our evaluation. Two of these protocols are flooding based (i.e. Epidemic and PRoPHET). These protocols are known to achieve high

delivery ratios but incur relatively high overhead as well. Epidemic is usually used in a highly opportunistic environment where nodes are blind and autonomous (e.g. like Random WayPoint). As mentioned earlier its resource assumption is unlimited and therefore its performance degrades considerably in a resource constrained environment. With similar resource assumptions, Spray and Wait (SnW), a quota based protocol reduces this resource consumption by limiting the number of replicas. The objective is to optimize the performance of Epidemic protocol by employing efficient buffer management, so that it maintains its high delivery probability with overhead comparable to Spray and Wait. Epidemic routing protocol is repeatedly used as a baseline for evaluation in numerous studies.

The second flooding based protocol used for performance evaluation is the Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET). As mentioned in earlier sections, PRoPHET is a controlled flooding based protocol where optimizing is achieved through intelligent selection of relay nodes. However due to its resource assumption, its performance also significantly degrades in a resource constrained environment. Unlike Epidemic it is usually used in scenarios where node movement is not entirely random neither completely deterministic but rather exhibit certain patterns like a more realistic disaster scenario. Again due to its resource consumption it incurs greater overhead to achieve high delivery probability. Encounter Based Routing (EBR), another quota based protocol is specifically designed for a disaster scenario. For fairness, we evaluate PRoPHET's performance with buffer management in comparison to that of EBR's performance in a disaster scenario.

4.3. BUFFER MANAGEMENT SCHEMES UNDER EVALUATION

As mentioned earlier studies suggest that drop policies have much greater effect on a protocol's performance than forwarding policies. Therefore careful analysis of message discarding schemes is our primary research goal. To isolate the effect of drop policies, we use same forwarding schemes for all comparisons.

In case of Epidemic, we implement its existing and proposed buffer management policies with First In First Out (FIFO) forwarding strategy to establish a fair comparison. In this scheme messages that arrived first are transmitted first. The buffer management schemes that are being evaluated for Epidemic are Drop Oldest (DOA) which drops message residing for the longest period of time in the buffer and Drop Largest (DLA) which discards message with the largest size, until enough space is available. Finally we implement our proposed drop policy i.e. Size Aware Drop (SA-Drop) which considers both the arriving message's size and available buffer space to determine a threshold size for selecting a message to be dropped.

We implement PROPHET with GRTR_Max forwarding scheme, where messages are transmitted in the decreasing order of remote node's predictability. Two popular queuing policies of PROPHET are selected for comparison. These include Evict Most Forwarded First (MOFO) which considers maximum number of forwards to discard a message from a node's buffer. The second policy is the Evict Shortest Life Time First (SHLI) policy which discards a message with the minimum remaining Time-To-Live (TTL). Finally we implement our drop policy i.e. Size Aware Drop (SA-Drop).

4.4. RESULTS AND ANALYSIS

To demonstrate the effectiveness of our proposed drop policy Size Aware Drop (SA-Drop), we perform two groups of simulations on each of the two mobility models scenarios. First to illustrate its effectiveness against existing policies, we vary the message dropping strategies used with either type of flooding based protocol. Meanwhile their performance is simultaneously compared with that of a quota based protocol without any buffer management, in similar conditions. Following this comparative evaluation, we evaluate how existing policies and our proposed scheme SA-Drop react to two significant constraints that are bandwidth constraints due to varying message generation rate and buffer constraints due to varying buffer sizes. Finally to show the impact of message size on each policy, we generate traffic of varying message sizes. The varying performance of each policy under similar conditions but varying message sizes justifies the use of message size as an important drop criteria.

4.4.1. Scenario 1: Performance Evaluation in Random Waypoint Mobility Model

We first evaluate the performance of our scheme SA-Drop against its popular existing counterparts (i.e. DOA and DLA) in the traditional Random Waypoint Mobility Model by using Epidemic as baseline. As mentioned earlier two groups of simulations are carried out; the first for comparative analysis with other policies and with quota based protocols and the second for performance analysis under constraints.

4.4.1.1. Comparative Analysis

In the first group of simulations, we compare the performance of Epidemic with our proposed SA-Drop policy and that of Epidemic with other popular buffer management policies and Spray and Wait. We determine the extent of optimization achieved in terms of both our primary performance metrics that are delivery probability and overhead ratio as well as others (e.g. buffer time average, hop count average, messages dropped etc). Similar metrics are used for the performance comparison with a quota based protocol (i.e. Spray and Wait).

As previously mentioned, two existing buffer management policies for Epidemic namely Drop Oldest (DOA) and Drop Largest (DLA) are used for comparison with our proposed Size Aware Drop (SA-Drop) policy. Figure 4.1 shows the affect of these schemes on the delivery probability. To demonstrate the effect of optimization achieved by intelligent buffer management, we have considered the performance of Epidemic with naive Drop Front (DF) also. It is seen that original Epidemic protocol (i.e. with DF) performs slightly better than Spray and Wait due to its flooding nature. It is however observed that by applying intelligent buffer management schemes (e.g. DOA, DLA and SA-Drop) the delivery probability is further enhanced. With DOA and DLA, Epidemic achieves 20% and 27% delivery probability enhancement respectfully than Spray and Wait. However Epidemic with SA-Drop achieves maximum performance optimization in a similar environment. Approximately 33% improvement in delivery probability is achieved.

Another important optimization metric of this research is overhead ratio. As mentioned earlier the objective is to achieve maximum delivery probability with minimum overhead ratio. Quota based protocols like Spray and Wait reduce overhead by limiting the number of copies being flooded into the network. However we observe in Figure 4.2 that by applying suitable buffer management policies to a highly resource consuming flooding based protocol like Epidemic, its overhead can be significantly reduced. Epidemic with naive DF incurs maximum overhead. The main reason is its redundant and iterative message drop. It is clearly seen that by applying carefully designed buffer management schemes that intelligently select messages to be discarded reduce overhead ratio as well. DOA reduces this overhead by 18% however it is still significant when compared with Spray and Wait. DLA further reduces this overhead by 23%. However maximum reduction in overhead ratio incurred by Epidemic is achieved when SA-Drop is applied. The overhead ratio of Epidemic with DF is reduced by approximately 30% when it is implemented with SA-Drop making it comparable to that of Spray and Wait.

In Figure 4.3, we observe the effect of existing policies and proposed Size Aware Drop (SA-Drop) for Epidemic alongwith the effect of Spray and Wait on buffer time average. Spray and Wait protocol, as the name suggests stores messages for a slightly longer period of time after the spray phase, before further replication. This is to allow the first copies to reach destination and avoid unnecessary replication. Due to this reason the buffer time average of messages at each node is more than that of Epidemic. Epidemic with DOA policy also allows messages to reside for a longer period of time in the buffer thus achieving greater buffer time averages. Epidemic with DF, DLA and our proposed scheme SA-Drop, achieves relatively smaller buffer stay time averages. The reason for this is that DF iteratively drops messages while both

size based schemes drop messages based on the requirements of buffer space. None of these schemes take time into consideration due to which relatively newer messages can be dropped as well. Thus as a result the overall buffer time average of messages is relatively smaller as compared to Spray and Wait and Epidemic with DOA. However in this scenario, against popular belief, buffer time average does not seem to have much effect on the delivery probability.

In Figure 4.4 we analyze the influence of hop count average on the optimization of Epidemic protocol. As mentioned earlier it is believed that minimum value of hop count ensures less overhead. Although this claim holds true in the case of Spray and Wait which has the least hop count average and minimum overhead. However in case of Epidemic with buffer management schemes we observe varying results. Epidemic with DOA has the maximum hop count average and comparatively more overhead as well. However Epidemic with DF which incurred maximum overhead ratio appears to have comparatively less hop count average. Epidemic with DLA has the least hop count average amongst all buffer management schemes being analyzed including our proposed SA-Drop. Although Epidemic with SA-Drop produced least overhead ratio amongst all schemes its hop count average is slightly more than that of DF and DLA. Hence we can carefully state that although hop count average does influence the overhead ratio its impact is not that significant in some cases.

As mentioned earlier, redundant message drop is another cause of increased overhead and prime objective of any buffer management scheme is to control the number of messages dropped. Figure 4.5 shows that message drop due to various buffer management schemes. It is seen that DF causes maximum number of messages to be dropped thus increasing overhead. The rest of the schemes (i.e. DOA, DLA and SA-Drop) reduce this message drop significantly consequently leading to reduced overhead ratio. This is due to selective behavior rather than blind drop of messages. However message drop due to our scheme SA-Drop is slightly greater than the other policies. This behavior is due to smaller simulation time, as the performance surely enhances after some time span. Overall Spray and Wait has least number of messages dropped which is again due to lesser number of initial copies of the message. Drop Largest (DLA) has the least message drop ratio among the other buffer management schemes. This could be due to the fact that it has a tendency to accommodate sudden increase in buffer requirement. A single drop can accommodate a number of consecutive new arrivals but again it is a bit unfair for the large sized messages.

Figure 4.6 shows the effect of each scheme on number of messages relayed in order to achieve successful delivery. Spray and Wait again has the least amount of messages relayed while Epidemic alongwith its buffer management policies has a higher amount of relay. These

results are due to the nature of these protocols. It is however interesting to see that in our case even though message relay slightly increases yet it resulted in better delivery probability and lower overhead. The reason could be that excessive message relaying degrades performance in terms of overhead when message delivery is unsuccessful. Then all the resources consumed by the message are wasted. However if the message is successfully delivered to its destination then the resource consumption can be justified. Epidemic with DF has the lowest delivery probability with highest overhead ratio which may have resulted from its maximum number of message relaying. DOA has the least amount of messages relayed due to longer buffer time average of messages. DLA and SA-Drop closely follow behind.

Figure 4.7 shows the delivery delay experienced by messages in each case. Spray and Wait exhibits the least delay followed by Epidemic with DOA policy while rest of the schemes focus on improving delivery probability rather than minimizing delay hence they show almost equal delay.

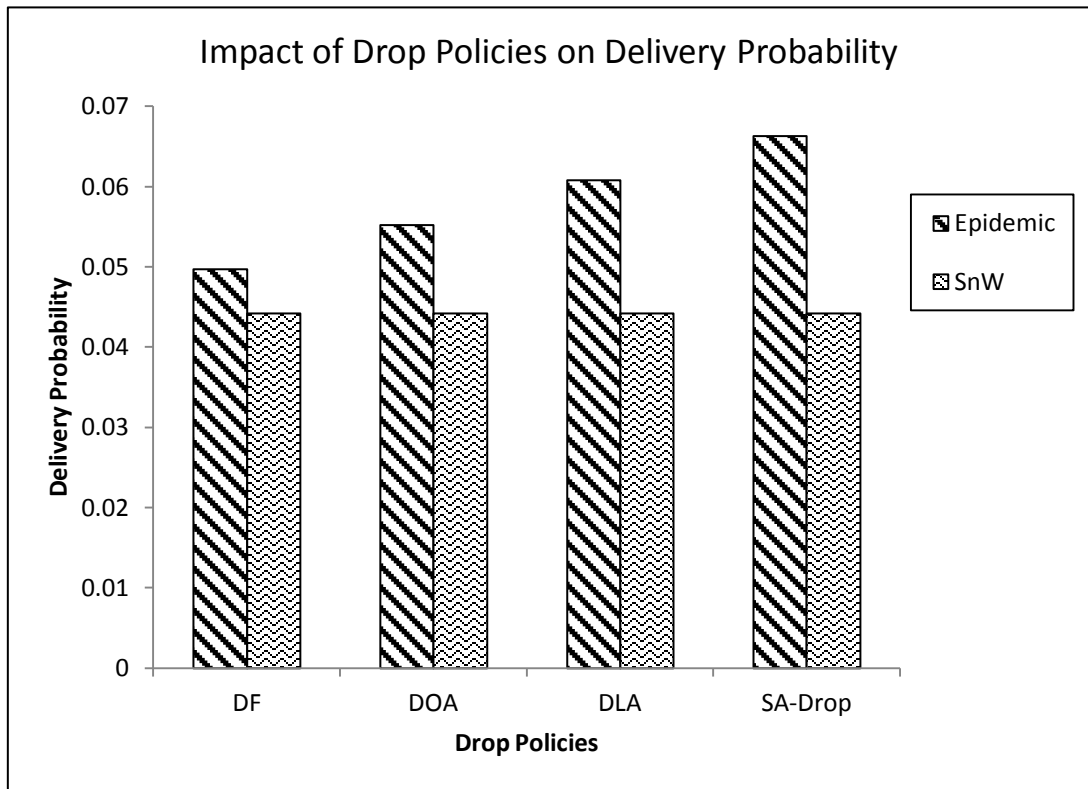


Figure 4.1: RWP: Comparison of Optimized Epidemic and SnW w.r.t Delivery Probability

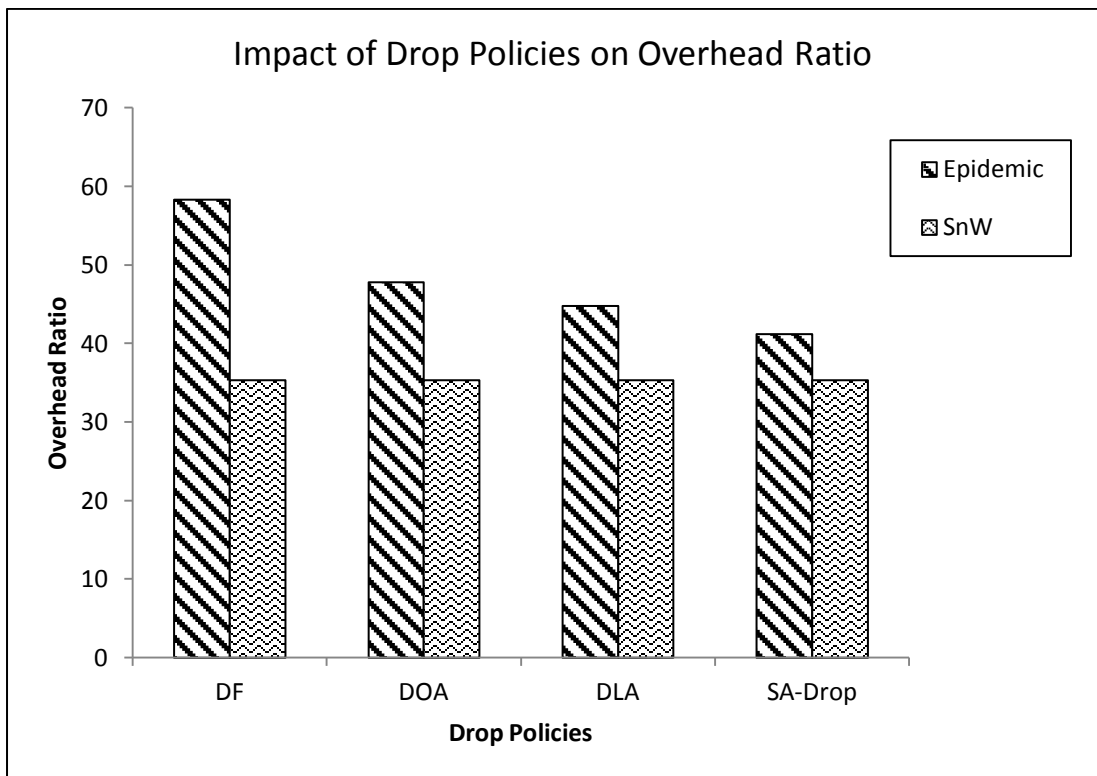


Figure 4.2: RWP: Comparison of Optimized Epidemic and SnW w.r.t Overhead Ratio

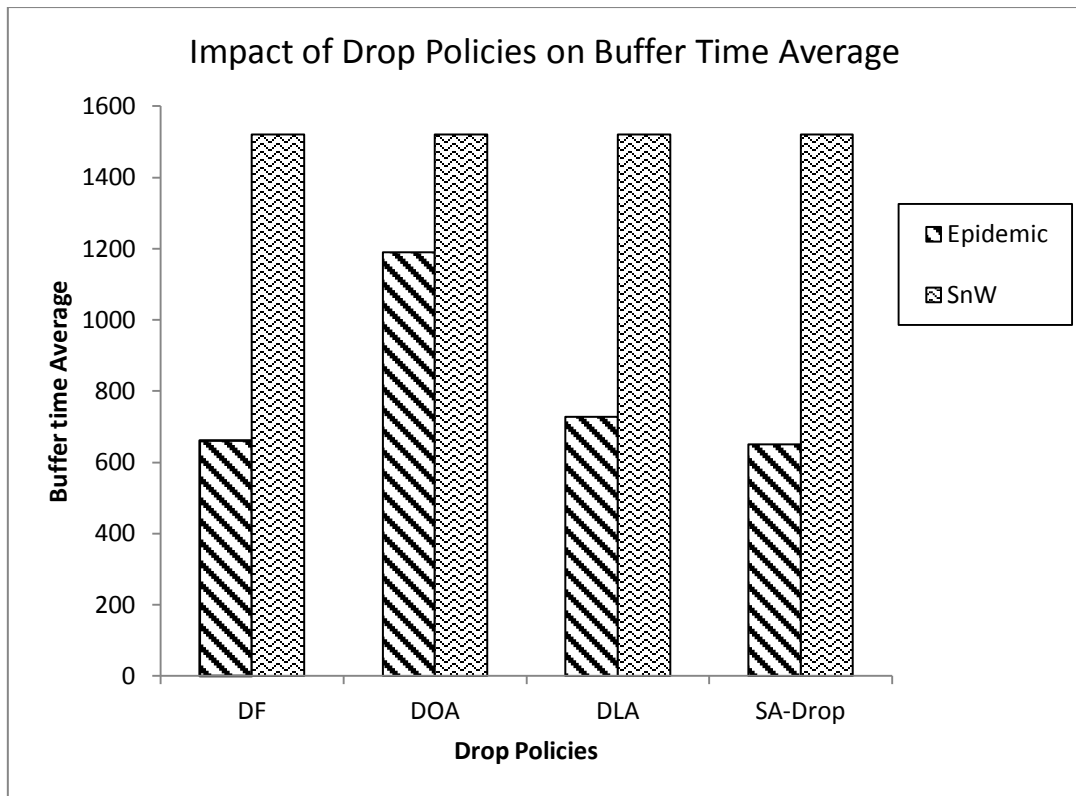


Figure 4.3: RWP: Comparison of Optimized Epidemic and SnW w.r.t Buffer Time Average

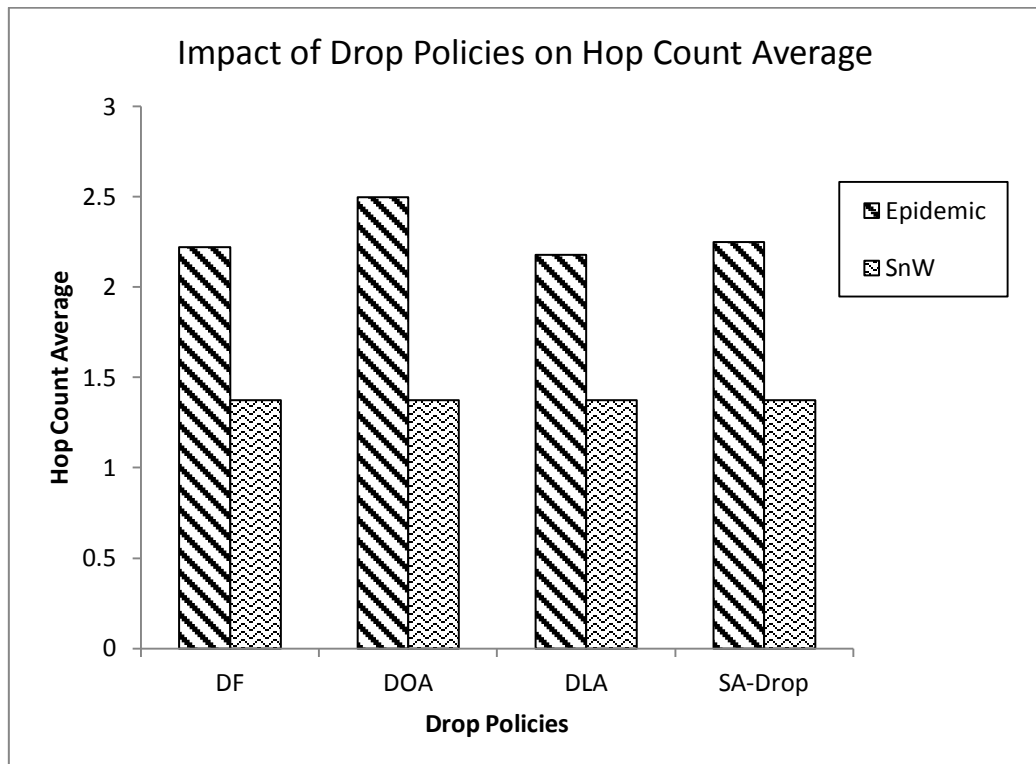


Figure 4.4: RWP: Comparison of Optimized Epidemic and SnW w.r.t Hop Count Average

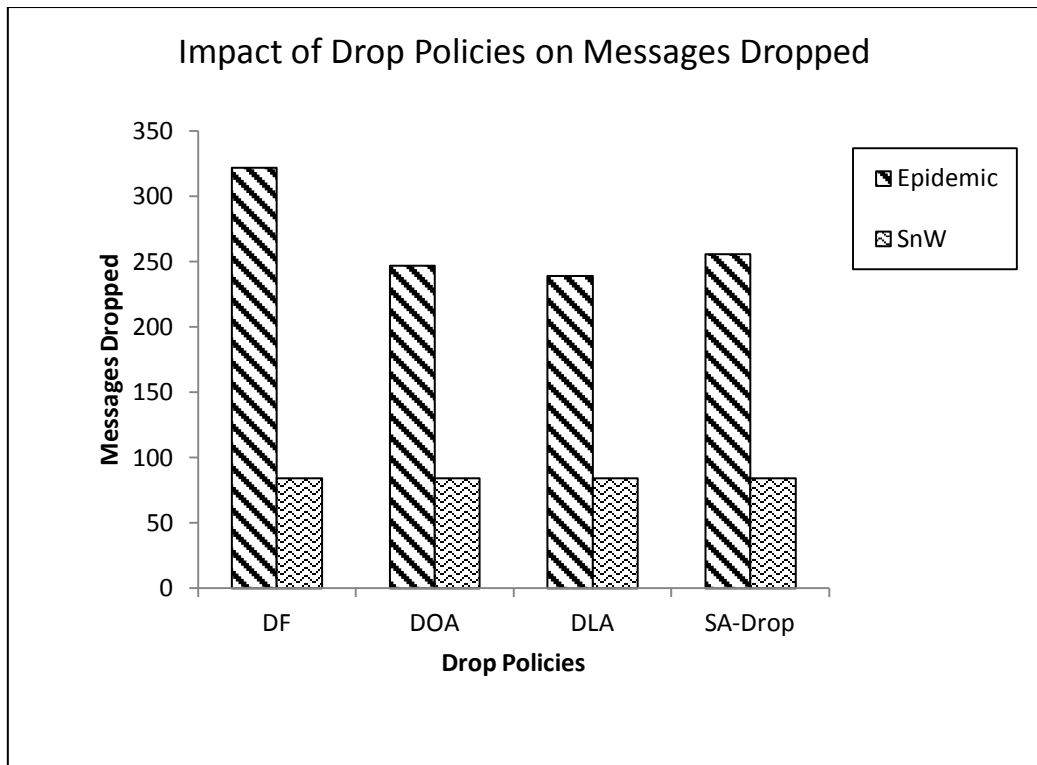


Figure 4.5: RWP: Comparison of Optimized Epidemic and SnW w.r.t Messages Dropped

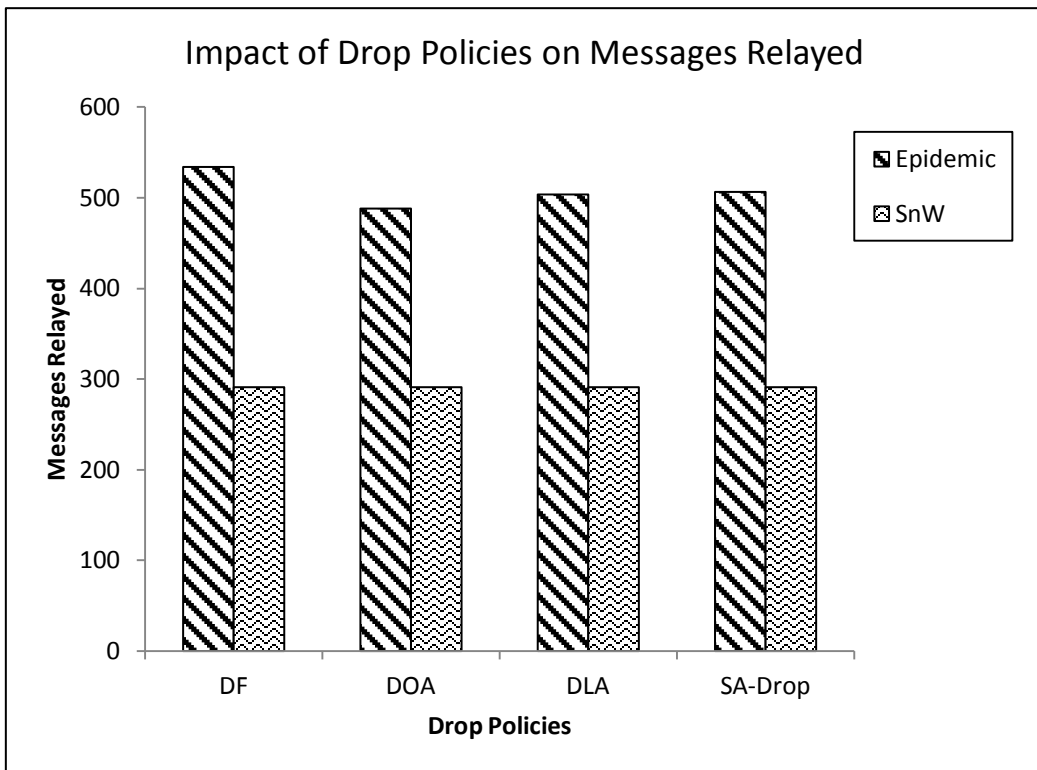


Figure 4.6: RWP: Comparison of Optimized Epidemic and SnW w.r.t Messages Relayed

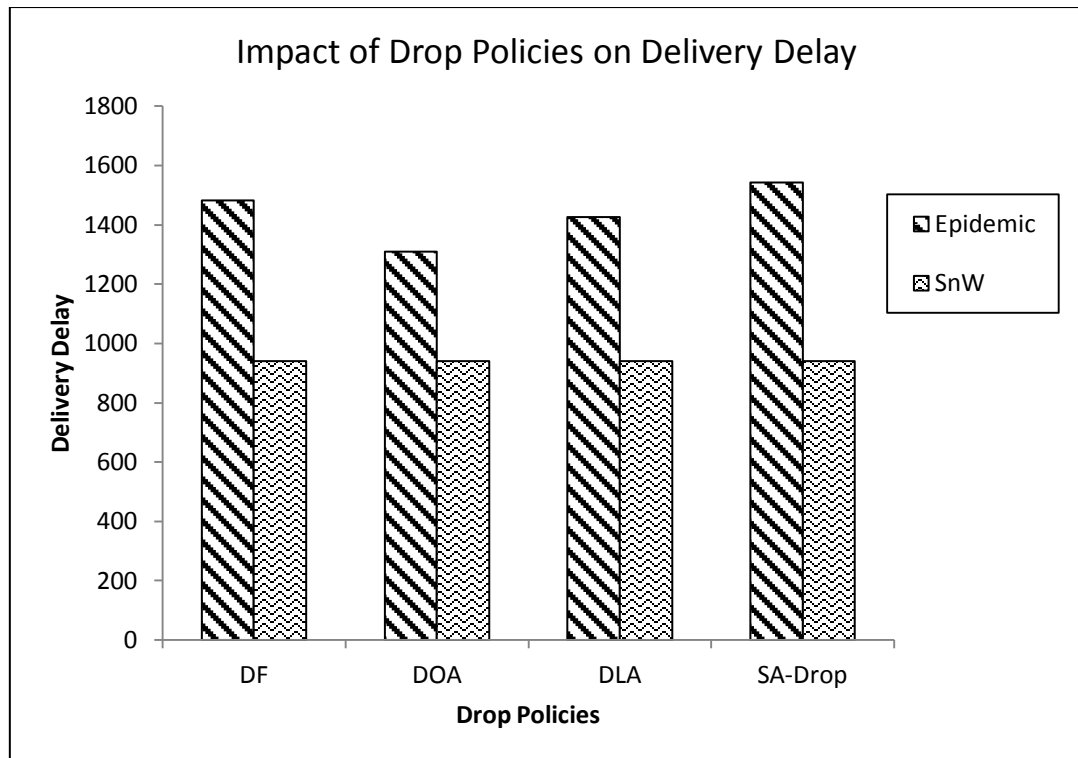


Figure 4.7: RWP: Comparison of Optimized Epidemic and SnW w.r.t Delivery Delay

4.4.1.2. Performance Analysis

In the second group of simulations, we evaluate the performance of our proposed scheme Size Aware Drop (SA-Drop) and the existing buffer management schemes by varying the rate of message generation and the size of buffers available. By varying these parameters, we determine their impact on delivery probability and overhead ratio incurred by these schemes. The results clearly show that our proposed scheme SA-Drop outperforms the rest of the popular buffer management schemes in terms of delivery probability and overhead ratio, in a highly constrained environment. Thus it proves the effectiveness of its appropriate selection mechanism and its efficient impact on the scarce network resources. We first present the impact of message generation rate on delivery probability and overhead ratio. The second is the impact of buffer sizes on both these performance parameters. The schemes chosen for comparison are Drop Oldest (DOA), Drop Largest (DLA) and our proposed scheme Size Aware Drop (SA-Drop) implemented with Epidemic protocol.

Figure 4.8 demonstrates the impact of message generation rate on the delivery probability of various buffer management schemes for Epidemic in a Random Waypoint Mobility Model. It is clearly evident that delivery probability in all cases decreases with the increase in the number of messages that is in congestion.

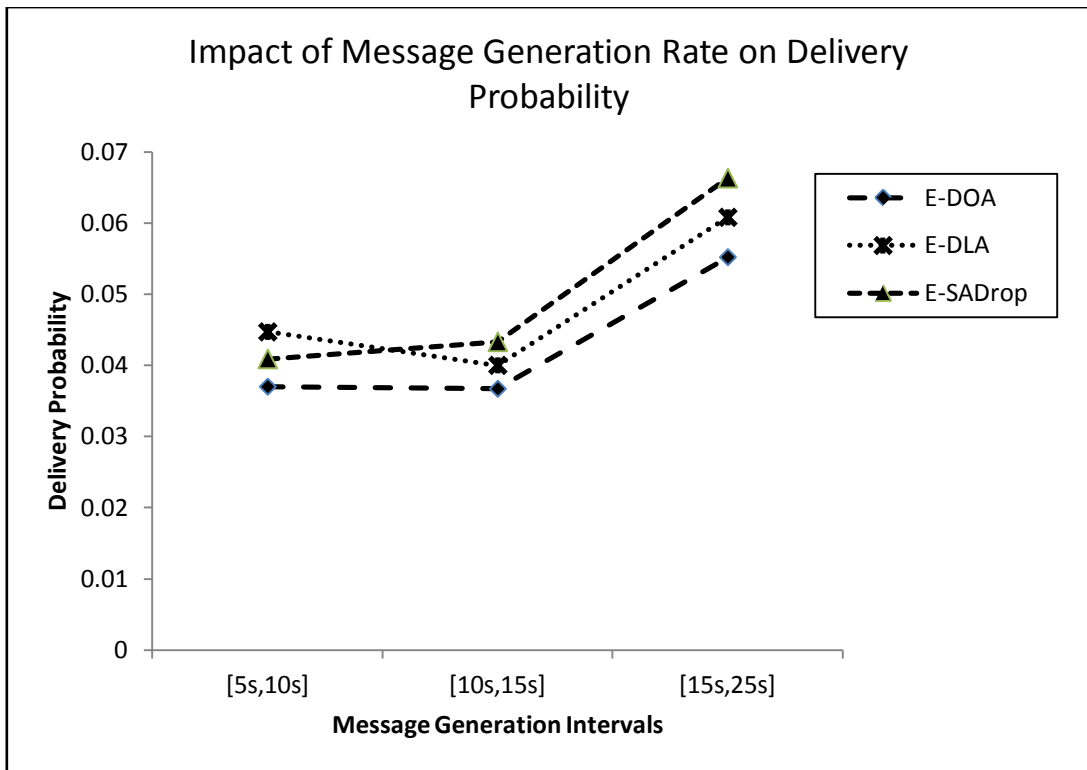


Figure 4.8: RWP: Impact of Varying Message Generation Rate on Delivery Probability

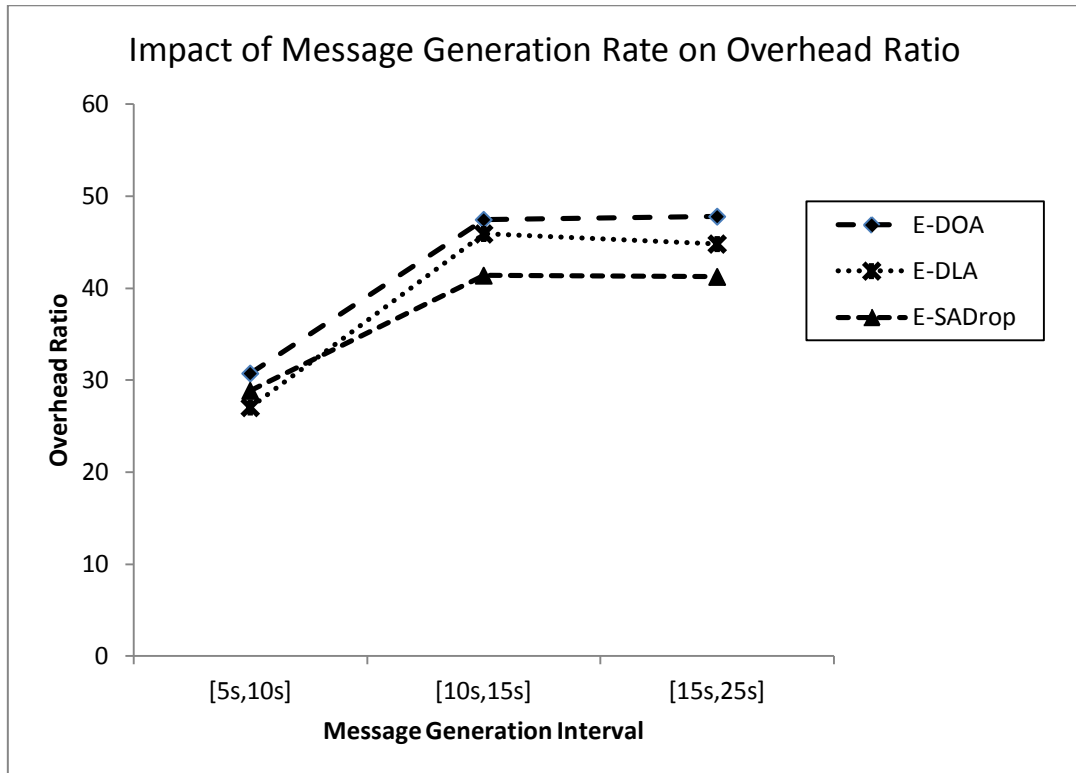


Figure 4.9: RWP: Impact of Varying Message Generation Rate on Overhead Ratio

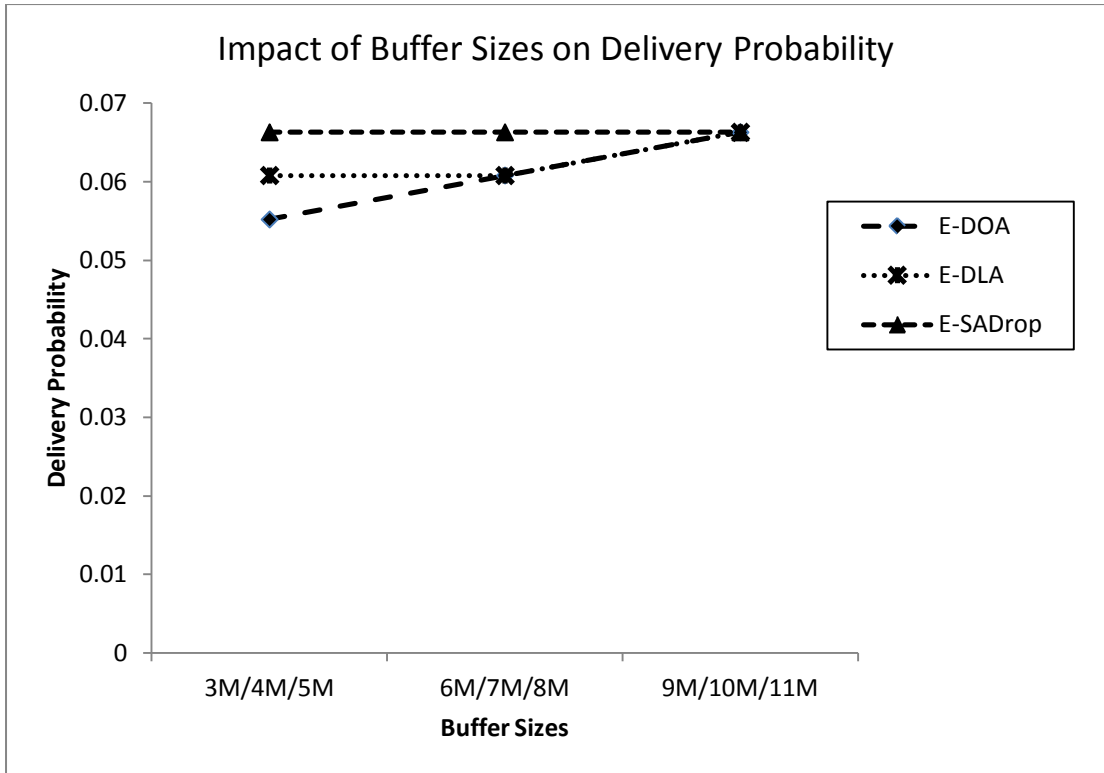


Figure 4.10: RWP: Impact of Varying Buffer Sizes on Delivery Probability

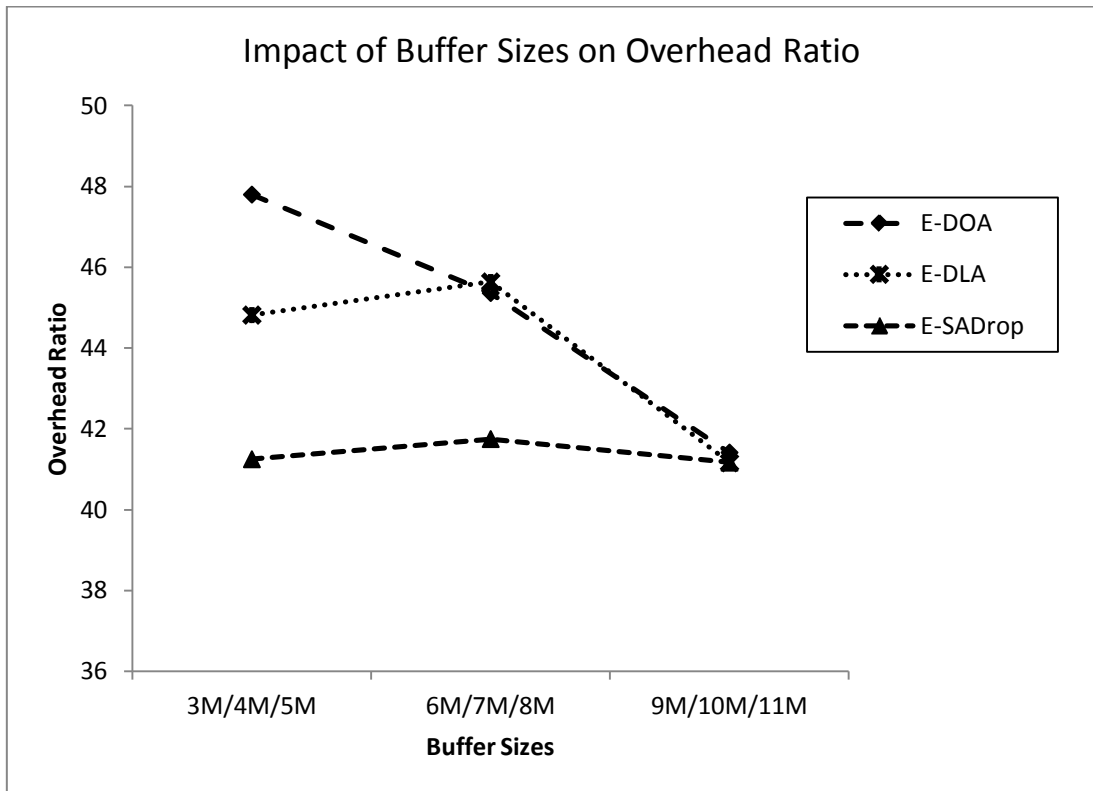


Figure 4.11: RWP: Impact of Varying Buffer Sizes on Overhead Ratio

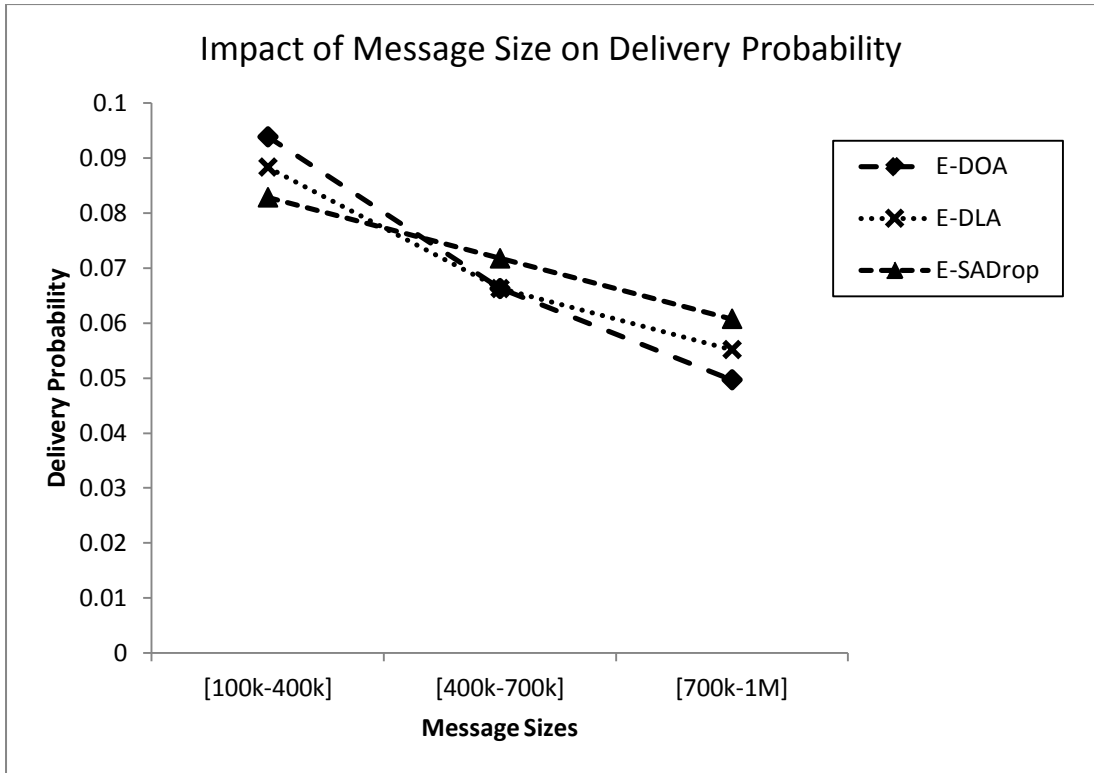


Figure 4.12: RWP: Impact of Varying Message Sizes on Delivery Probability

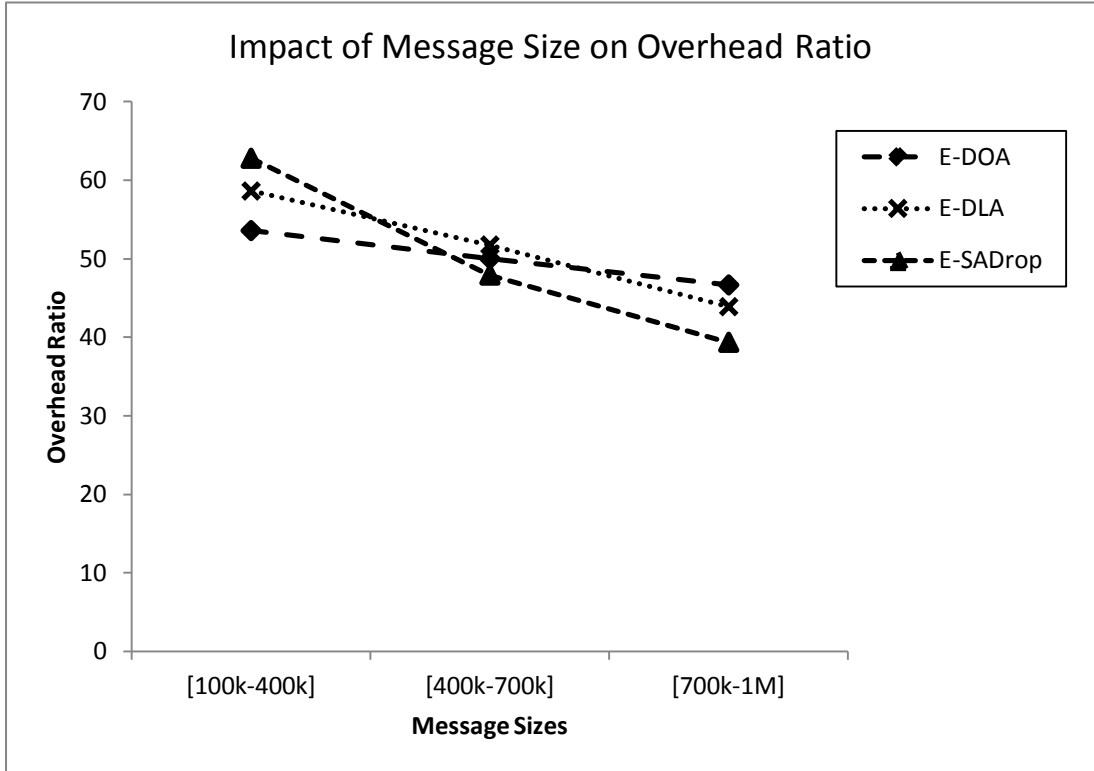


Figure 4.13: RWP: Impact of Varying Message Sizes on Overhead Ratio

Three message generation intervals are used to regulate the number of messages created. In the first interval that is [5s, 10s] a new message is created after every 5 to 10 seconds which simulates a highly congested environment with more number of messages being created in a shorter time span.

Similarly a message is created in 10 to 15 seconds and 15 to 25 seconds in [10s, 15s] and [15s, 25s] intervals correspondingly. It is observed that both size based schemes (i.e. DLA and SA-Drop) outperform the time based scheme DOA in all three message generation rates supporting the fact that size is an appropriate message attribute for message discard in case of highly congested and constrained environments. Furthermore our scheme SA-Drop outperforms DLA in [10s, 15s] and [15s, 25s] scenarios while in extremely congested environment of [5s, 10s] DLA slightly improves delivery probability. The reason being its tendency to accommodate sudden fluctuations in buffer requirement since the maximum sized message drop may accommodate several incoming messages.

Similarly in Figure 4.9, the impact of message generation rate on the resource consumption that is the overhead ratio is demonstrated. It is interesting to see that against our common expectation the overhead ratio in this scenario decreases in a more congested environment whereas popular belief suggests that it should have been increasing. A logical explanation for this behavior may lie in the scenario at hand. Due to highly opportunistic nature of the Random Waypoint mobility model successful message delivery is highly dependant on the number of messages traveling in the network. Increasing message generation rate ensures greater number of successful deliveries which consequently lead to justified resource consumption thus lesser overhead. In a less congested environment message loss reduces delivery ratio and consequently increases resource wastage leading to more overhead. However in any case less overhead is an objective which is clearly achieved by our proposed policy SA-Drop. Resource consumption incase of DOA is maximum in all the cases while SA-Drop outperforms DLA in all scenarios except the highly congested one where it is approximately equivalent to that of DLA. This again confirms the authenticity of our claim that size is the appropriate attribute for use in any congested environment.

In Figure 4.10, we observe the impact of varying buffer sizes of the three groups of nodes participating in the communication. In the original scenario one group has buffer size equal to 3M while the other two groups have buffer sizes equal to 4M and 5M. We observe that by increasing each group's buffer sizes the delivery probability of messages increases for each buffer management scheme. However an interesting effect is that our proposed scheme Size Aware Drop (SA-Drop) maintains its message delivery probability in all three environments whereas the performance of both DOA and DLA degrade with increased buffer constraints.

Our scheme outperforms others in more constrained scenarios while the rest catch up to it in a less congested one. This effect supports our claim that our scheme is more suitable for constrained environments which in most real life applications is a more appropriate assumption.

Similarly in Figure 4.11 we observe that in a less constrained environment that is with greater buffer sizes all schemes impose almost comparable overhead. However as the environment becomes more constrained the other schemes start incurring more overhead due to excessive resource consumption whereas Size Aware Drop (SA-Drop) maintains minimum overhead ratio. This further confirms the utility of our proposed scheme in such scenarios.

Figure 4.12 and 4.13 show the impact of various sized messages on the two metrics that are being evaluated. We observe that message delivery probability is increased when message sizes are small. However with smaller messages number of transmissions also increases thus increasing the routing overhead. When message sizes are increased routing overhead decreases due lesser transmissions however message delivery probability decreases as expected. Hence it shows that policies that favor either extreme of the message sizes only focus on optimizing any one of the two metrics. Our scheme allows all messages equal opportunities to be delivered. This randomness results in better overall optimization of message delivery probability and routing overhead ratio.

We extracted some general observations regarding the performance of these buffer management schemes from our experimentation. It is observed that naive schemes like DF and time based schemes like DOA do not perform well in constrained environments. This is due to the fact that smaller sized buffers are subject to overflow sooner than larger sized ones. On the other hand both size based schemes i.e. DLA and SA-Drop significantly increase the delivery likelihood of messages while incurring only minimum amount of overhead. This makes them an appropriate choice for employment in most realistic DTNs. However, since it is relatively a novel supposition, only a little amount of work has been done in this area. This area has much potential for future research work.

4.4.2. Scenario 2: Performance Evaluation in Disaster

The behavior of Epidemic protocol with and without buffer management is observed in Random Waypoint Mobility Model which is commonly used to determine the performance base line. However realistic assumptions, regarding the mobility of nodes gives more realistic behavioral insights about the performance of any scheme. For this purpose we have created another scenario using a synthetic mobility model with more realistic mobility patterns called the Event Driven, Role Based Disaster Mobility Model [50]. In this scenario we observe the

effect of buffer management policies on the performance of flooding based protocol (i.e. PRoPHET) and compare it with that of an adaptive quota based protocol Encounter Based Routing (EBR). Encounter Based Routing (EBR) is specifically designed for enabling efficient communication in post disaster response networks and its designers have utilized same disaster mobility model mentioned above. As previously stated two groups of simulations are carried out, one for the purpose of comparison between optimized PRoPHET and EBR while the second for the performance evaluation of existing buffer management schemes and our proposed Size Aware Drop (SA-Drop) under congestion and storage constraints. For comparison two popular existing buffer management schemes already implemented with PRoPHET are used namely the Evict Most Forwarded First (MOFO) and Evict Shortest Life Time First (SHLI).

4.4.2.1. Comparative Analysis

In the first group of simulations, PRoPHET is evaluated under the same criteria as Epidemic that is to improve delivery probability with the least overhead ratio. The results show similar overall optimization with slight behavioral variations. Figure 4.14 demonstrates that PRoPHET with buffer management outperforms EBR in terms of delivery probability. PRoPHET with DF and our proposed SA-Drop achieves approximately 32 % performance enhancement than EBR while PRoPHET with MOFO and SHLI achieves 20% and 29% performance enhancement respectively. Although DF has delivery ratio equivalent to our scheme yet the important question is whether it improves overhead ratio to similar extent as well.

Figure 4.15 shows that all buffer management policies except our proposed policy SA-Drop incur overhead greater than EBR. MOFO incurs maximum overhead which is 53% greater than that of EBR. Closely following is the overhead incurred by SHLI which is almost 39% greater than that of EBR. Finally DF also produces considerable overhead ratio that is almost 38% more than EBR. With this much overhead, its enhanced delivery probability comes at quite a cost of resources making it inappropriate for resource constrained environments. With same delivery probability our proposed scheme SA-Drop reduces the overhead ratio by approximately 40% than EBR. An interesting observation is that both SHLI and MOFO produce the highest overhead ratio which signifies that time and hop count may be an inefficient attribute to consider while discarding a message from the buffer in such a scenario.

In order to understand the impact of each scheme on our two prime performance evaluation parameters, we next study their impact on our secondary parameters (i.e. buffer time average, hop count average, number of messages dropped and number of messages relayed). In Figure

4.16 we observe the buffer time average of messages with all these schemes. As expected with EBR, messages spend a longer time in the node's buffer. This is again due to the forwarding paradigm of this protocol. On the other hand in PRoPHET with all buffer management policies, message buffer time average is less than that of EBR. Both DF and MOFO demonstrate a very small message buffer time average while with SHLI policy message buffer time average is slightly increased. However with our proposed policy SA-Drop, messages stay in the buffer for much longer period of time. SA-Drop policy increases message buffer time average by 60% than DF, 52% than SHLI and almost 66% than MOFO. It is however 38% less than that of EBR but still produce greater delivery probability than it. As observed the message buffer time average is the smallest with MOFO which contributes to its reduced delivery probability.

Figure 4.17 shows the impact of all these schemes on the hop count average. Both EBR and PRoPHET with our proposed policy SA-Drop have minimum hop count averages resulting in minimum overhead ratios. However an interesting observation is that PRoPHET with MOFO has the least hop count average yet incurs maximum overhead which supports our baseline assumptions that perhaps against popular belief hop count average may have little impact on resource consumption. Both SHLI and DF have considerably higher hop count averages and consequently more overhead ratios as well.

Figure 4.18 shows the comparison between the numbers of messages dropped with each scheme. We observe that our proposed scheme SA-Drop significantly reduces the number of messages dropped as compared to other existing buffer management schemes. It successfully reduces message drop by 59% than DF, 57% than SHLI and 64% than MOFO. This is primarily due to the fact that instead of blind drop it specifies a threshold value as a selection criterion. This significantly reduces overhead and contributes to more successful deliveries. Closely following in performance is EBR which also has a lesser number of messages dropped. However this is more due to controlled replication than intelligent buffer management.

Figure 4.19 demonstrates the effect of each scheme on the number of messages relayed. It is again observed that PRoPHET with our proposed buffer management scheme SA-Drop greatly reduces number of transmissions required for successful delivery of messages. This consequently results in the least amount of overhead incurred. SA-Drop reduces unnecessary replication by 62% as compared to DF, 60% as compared to SHLI and 67% as compared to MOFO. It even has 11% less message relay than EBR. This results in better utilization of bandwidth which is also a valuable resource in such environments.

Finally we observe the latency induced by each scheme. As mentioned earlier the key objective of our proposed scheme is to maximize delivery probability with least amount of overhead. It does not focus on minimizing delivery delay. We observe a slight increase in the delivery delay of messages in case of our proposed scheme, SA-Drop. However it is comparable with delay occurring due to other schemes.

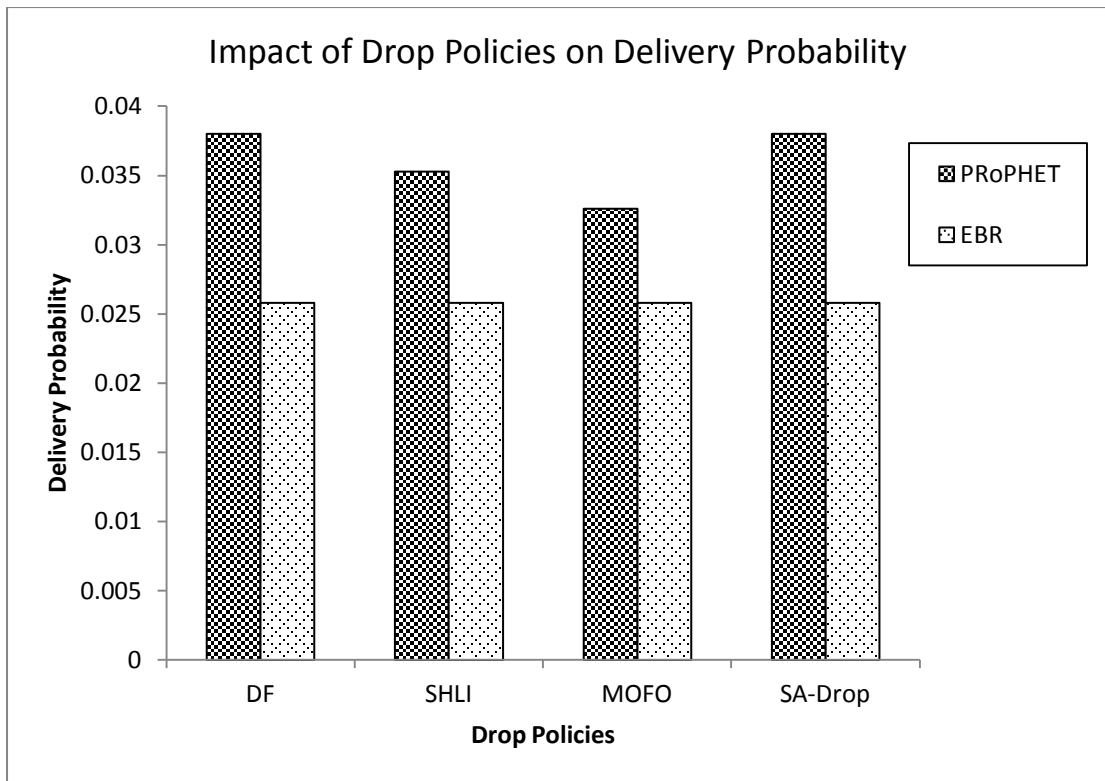


Figure 4.14: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Delivery Probability

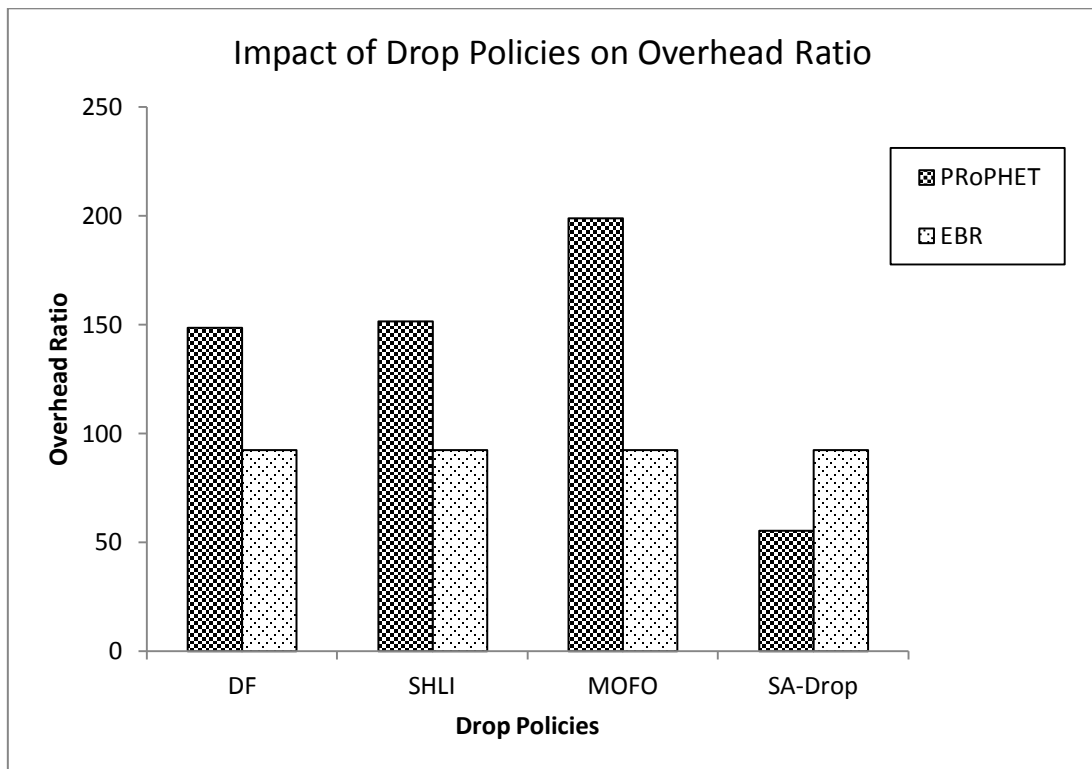


Figure 4.15: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Overhead Ratio

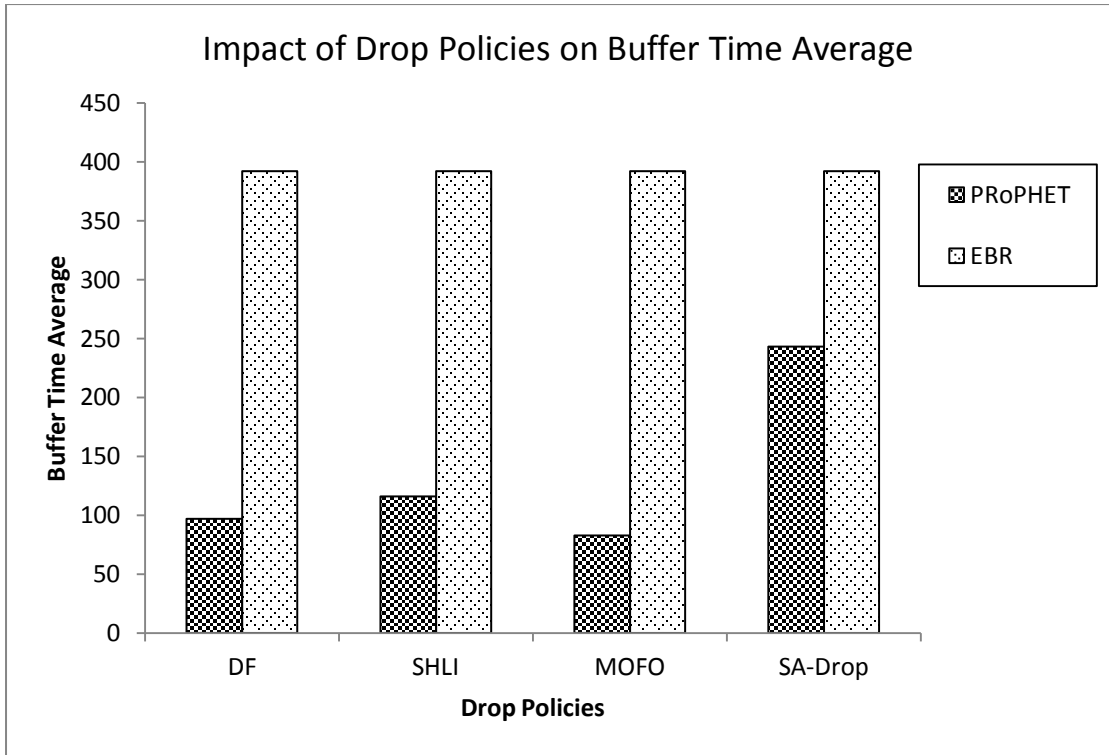


Figure 4.16: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Buffer Time Average

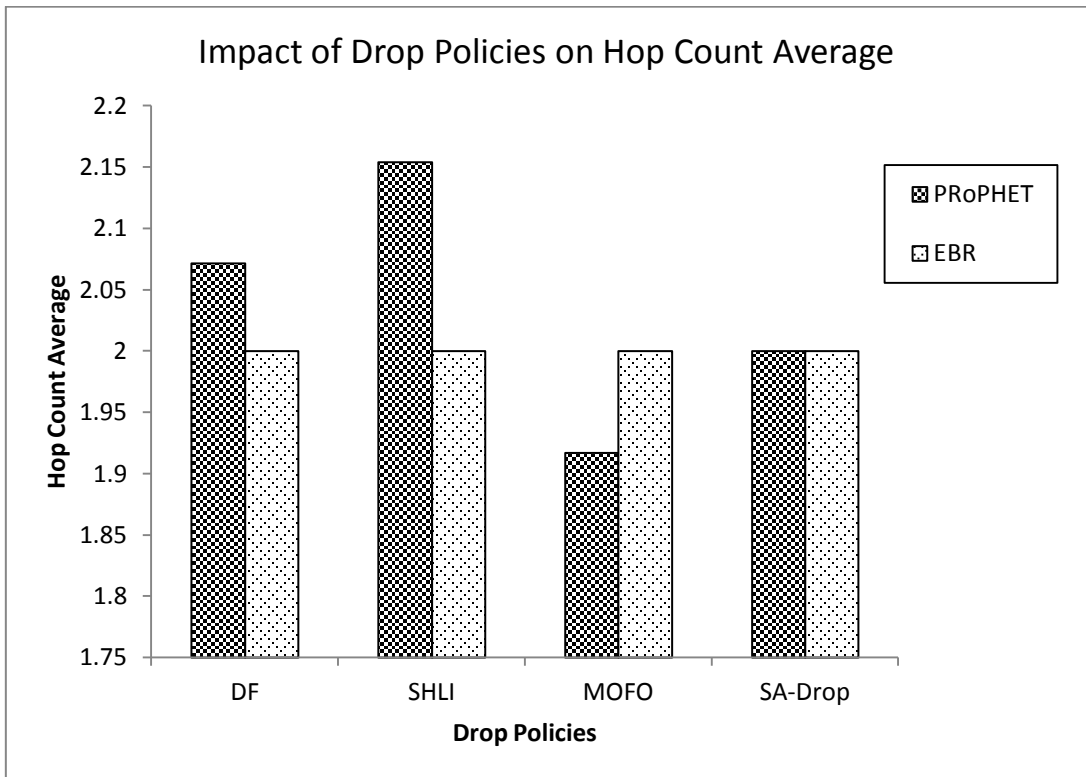


Figure 4.17: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Hop Count Average

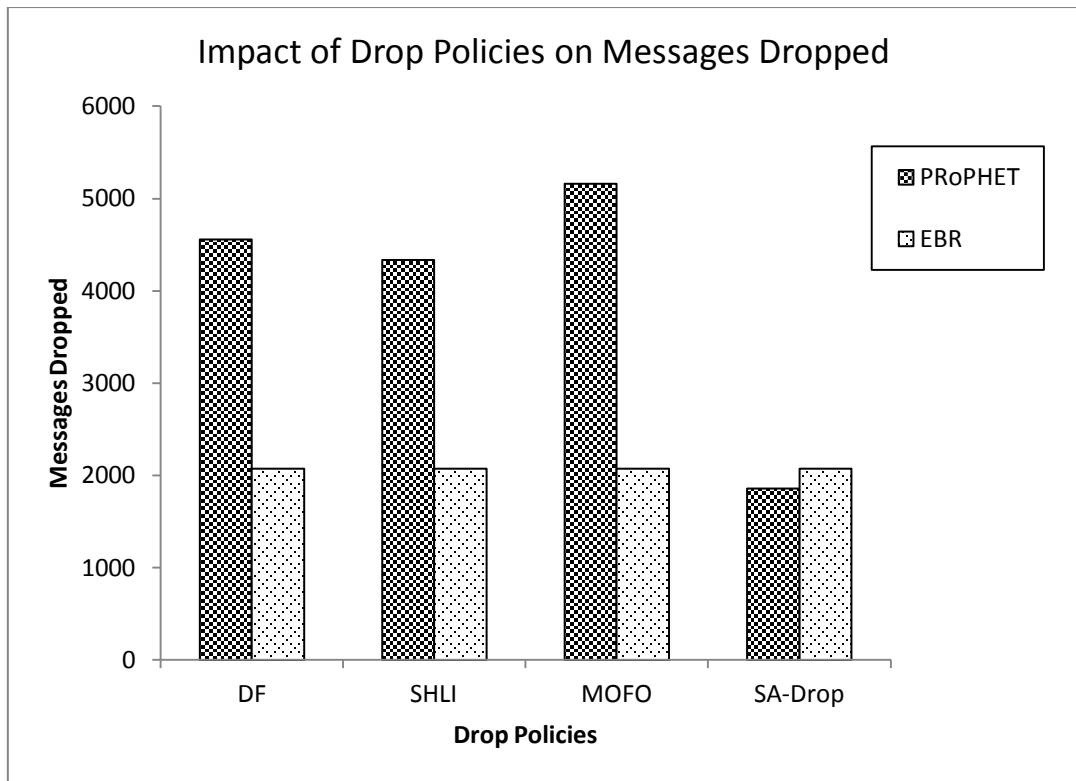


Figure 4.18: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Messages Dropped

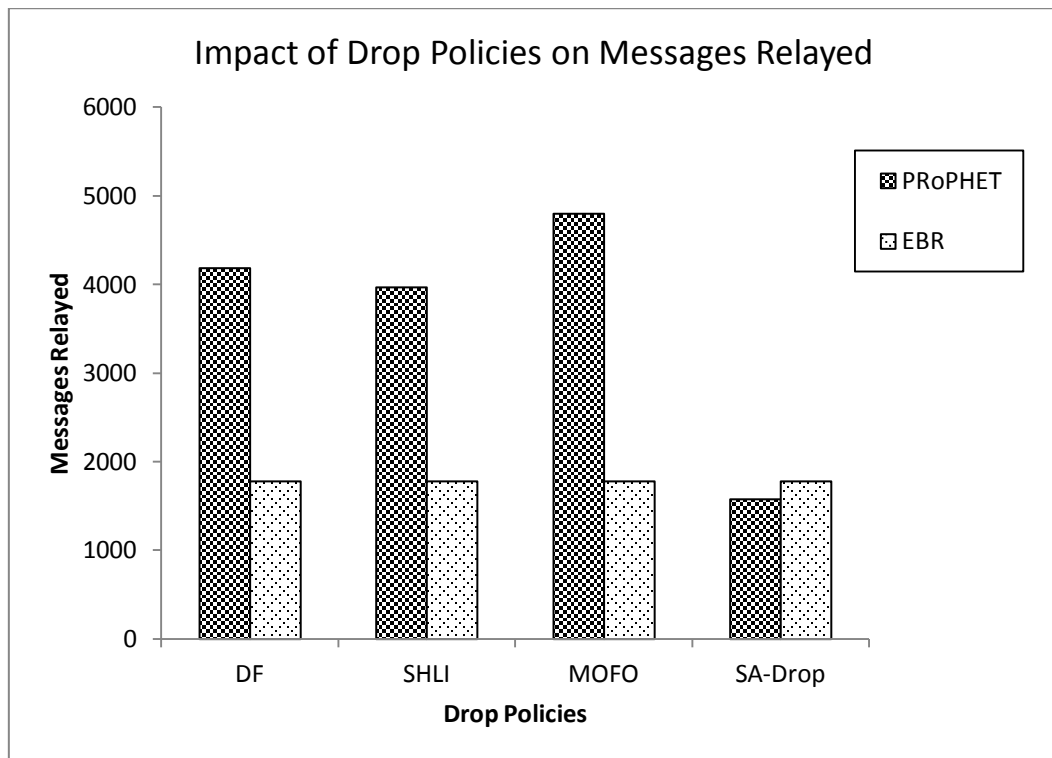


Figure 4.19: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Messages Relayed

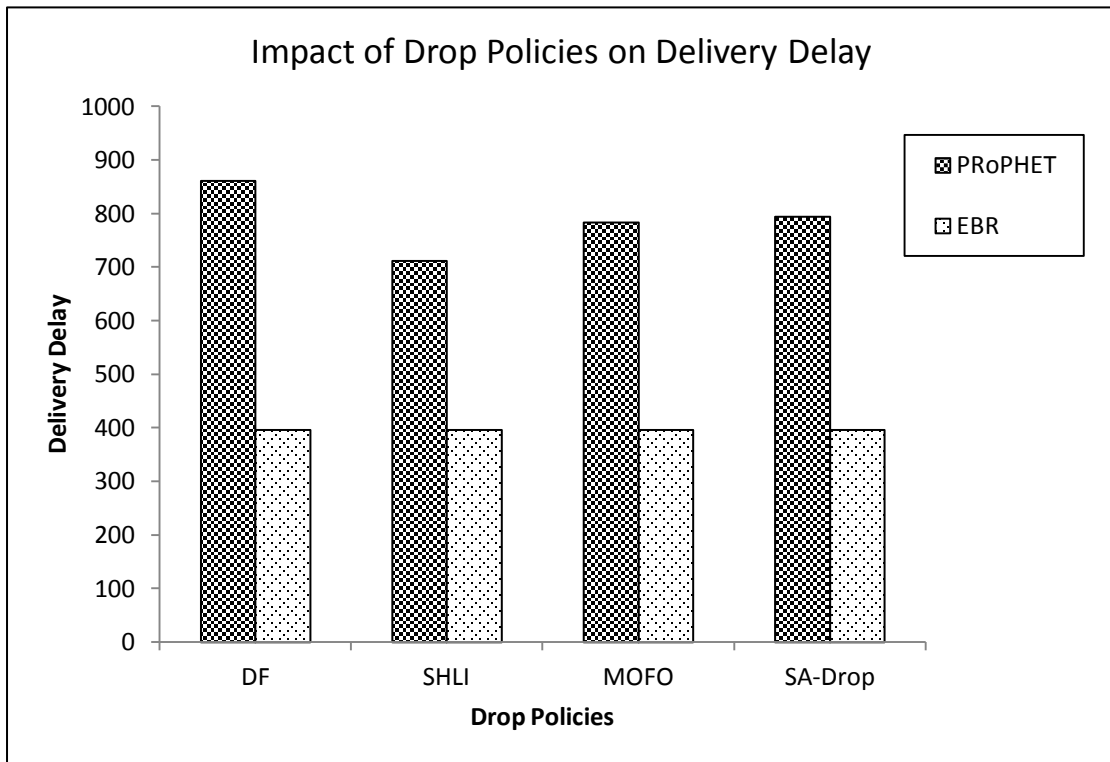


Figure 4.20: Disaster: Comparison of Optimized PRoPHET and EBR w.r.t Delivery Delay

4.4.2.2. Performance Analysis

In the second group of simulations, we compare the performance of our proposed scheme Size Aware Drop (SA-Drop) and the existing buffer management schemes under varying conditions. For this purpose we vary the rate of message generation and the size of buffers available. We first present the impact of message generation rate on delivery probability and overhead ratio. In the second set of simulations we determine the impact of buffer sizes on both these performance parameters. The schemes chosen for comparison are Evict Shortest Life Time First (SHLI), Evict Most Forwarded First (MOFO) and our proposed scheme Size Aware Drop (SA-Drop). All are implemented with PRoPHET protocol.

Figure 4.21 shows the impact of message generation rate on delivery probability. Three different message generation intervals are used to simulate slightly congested, congested and highly congested environments. The intervals applied are [35s, 45s], [25s, 35s] and [15s, 25s] respectively. It is observed that in a slightly less congested scenario MOFO outperforms SHLI and SA-Drop which give comparable results in the same environment. However there is a sharp degradation in the delivery probability of MOFO as the message generation rate is increased. Although a gradual decrease in the delivery probability of the rest of the schemes is also observed with smaller message generation interval yet the delivery probability of SA-Drop outperforms the others in a highly congested environment.

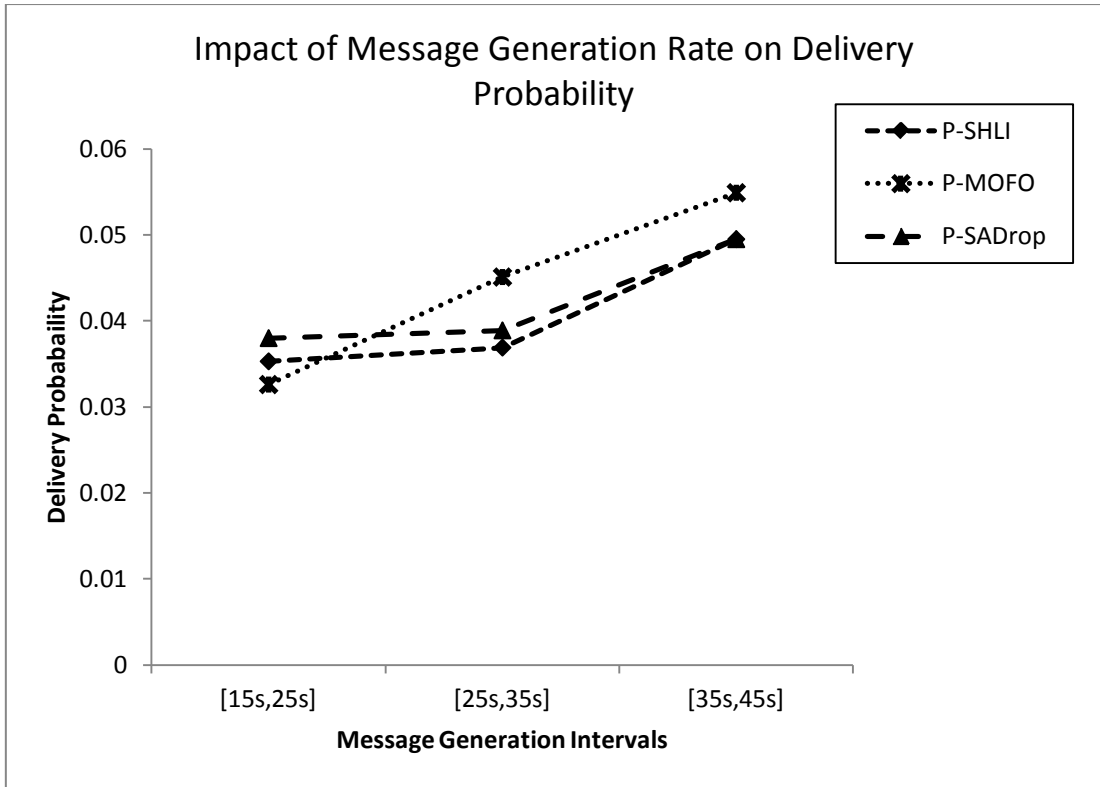


Figure 4.21: Disaster: Impact of Varying Message Generation Rate on Delivery Probability

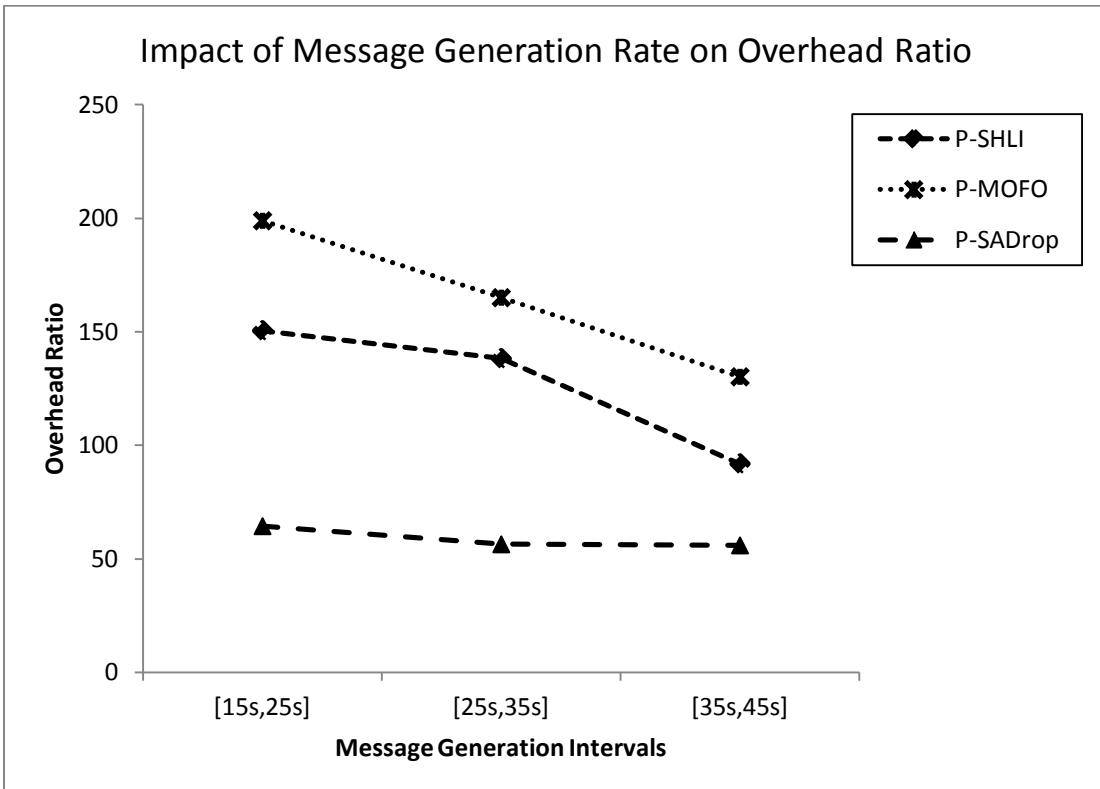


Figure 4.22: Disaster: Impact of Varying Message Generation Rate on Overhead Ratio

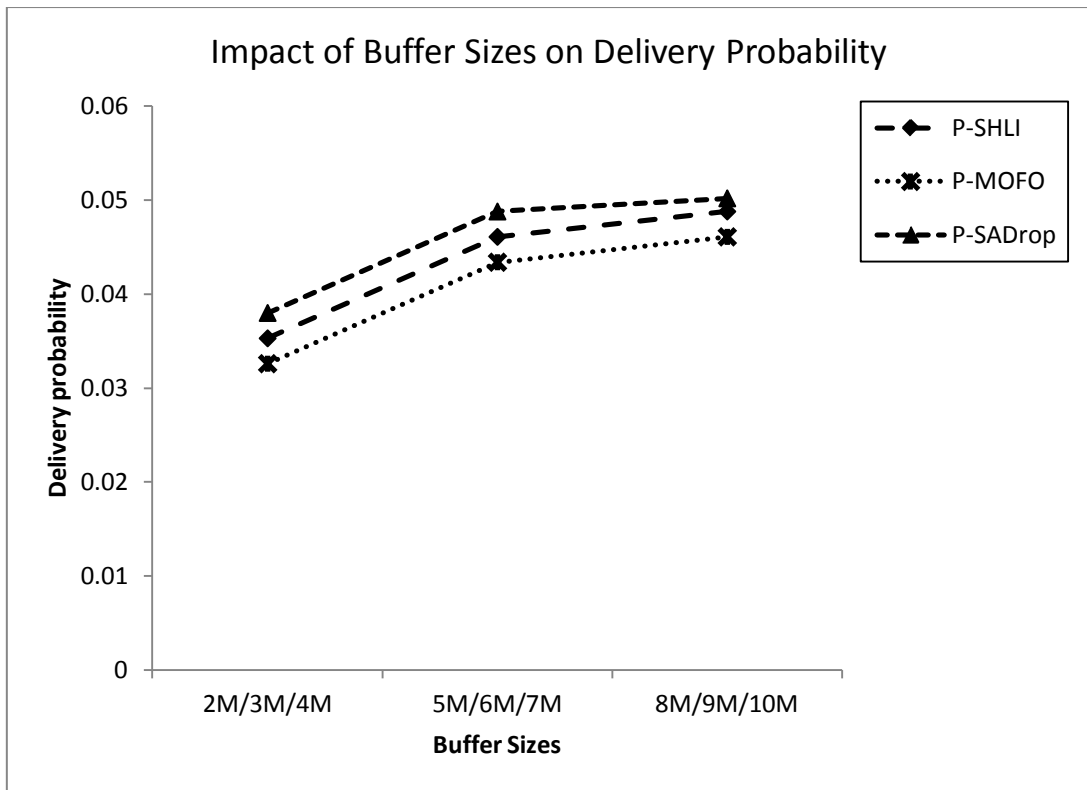


Figure 4.23: Disaster: Impact of Varying Buffer Sizes on Delivery Probability

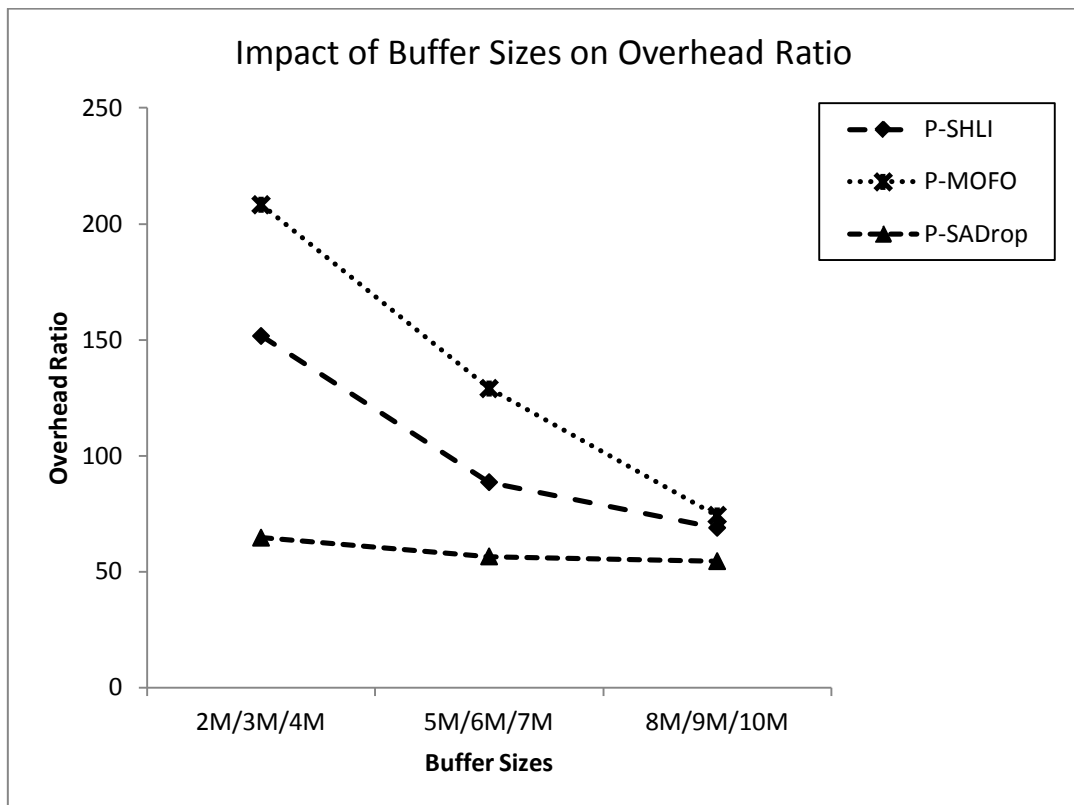


Figure 4.24: Disaster: Impact of Varying Buffer Sizes on Overhead Ratio

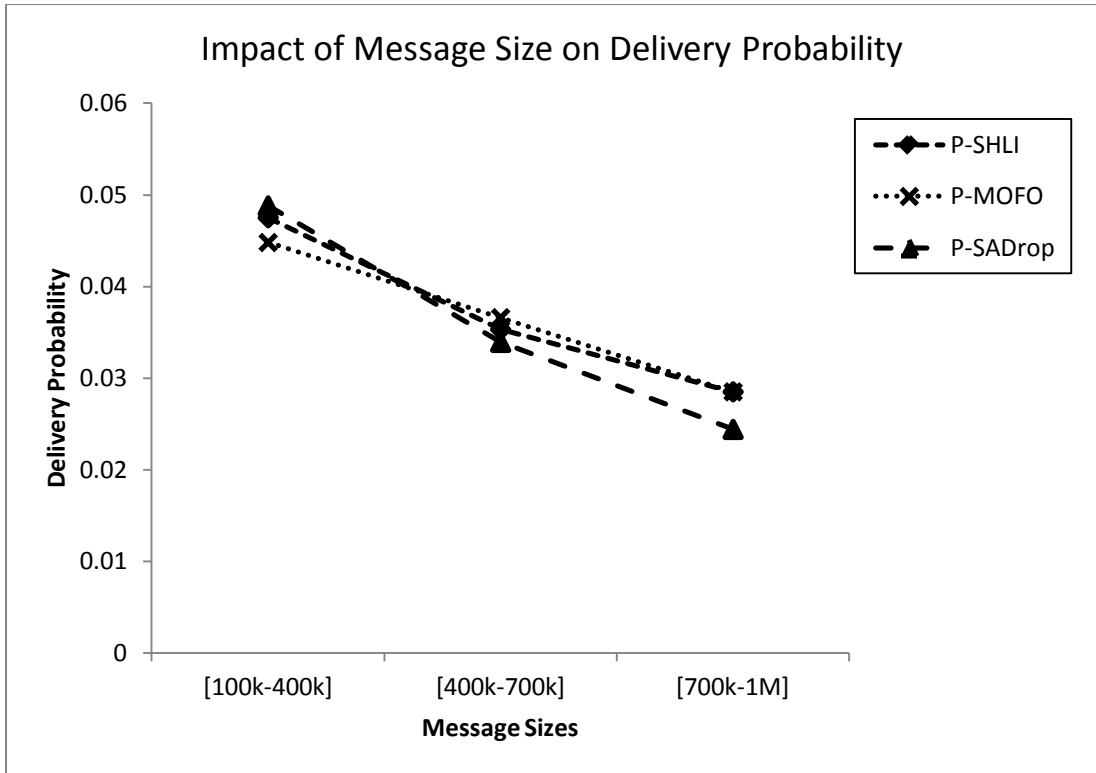


Figure 4.25: Disaster: Impact of Varying Message Sizes on Delivery Probability

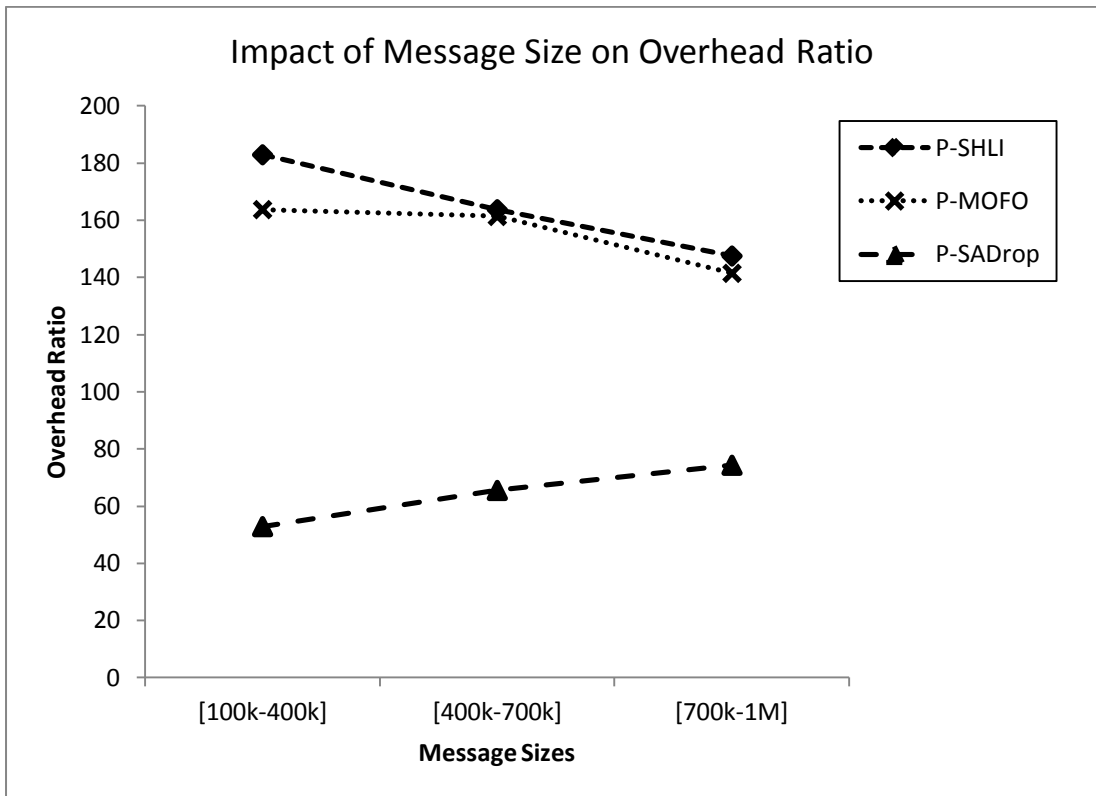


Figure 4.26: Disaster: Impact of Varying Message Sizes on Overhead Ratio

Figure 4.22 shows the impact of increased message generation rate on the overhead incurred by the three buffer management schemes. As expected the overhead ratio of all these schemes increases with the increase in the number of messages injected into the network. However our proposed scheme SA-Drop not only maintains its low overhead ratio but also outperforms SHLI and MOFO by 57% and 68% respectively in a highly congested scenario.

Figure 4.23 demonstrates the impact of buffer constraints on the delivery probability. For this purpose the groups' buffer sizes are varied from 2M/3M/4M to 5M/6M/7M to finally 8M/9M/10M to simulate a highly constrained, slightly constrained and not as constrained environment. As expected the delivery probability of messages increases with the increase in buffer sizes and decreases with more constrained buffer sizes. However the delivery probability resultant by our scheme SA-Drop in all cases is more than that due to rest of the schemes.

Figure 4.24 shows the impact of buffer constraints on the overhead ratio incurred by the three schemes under evaluation. Overhead significantly increases with the decrease in buffer size in both SHLI and MOFO. However due to less resource consumption resultant from our proposed policy Size Aware Drop (SA-Drop), the overhead ratio of our scheme not only is maintained but is the least in all three cases.

Finally Figure 4.25 and 4.26 show the impact of message size on the given metrics. As seen in the first scenario as the traffic of large sized messages increases, the message delivery probability decreases due to short contact durations and bandwidth restrictions. On the other hand routing overhead also decreases with the increase of message size. In this particular scenario, unlike other schemes the overhead ratio of our scheme increases with the increase in the message sizes. However, it is still a lot less than that incurred by the other schemes.

These performance evaluations support our assumption that size is a more appropriate message discarding criteria in a highly congested and constrained environment like a disaster scenario where popular time based and hop based schemes fail to produce the desired results.

The reported results emphasize on the authenticity of our claim that in both Random Waypoint and Disaster scenarios, if there is scarcity of resources both in terms of bandwidth and buffer, our proposed scheme Size Aware Drop (SA-Drop) produces the most desirable results. It also alleviates the effects of congestion greatly resulting in better performance optimization of basic flooding based protocols as compared to quota based protocols.

Chapter 5

CONCLUSION AND FUTURE WORK

So far much attention has been given to improving routing strategies in Delay Tolerant Networks (DTNs), while research on buffer management is still in its infancy. Using simulations based on both random and near realistic movements, we demonstrate that an efficient buffer management policy can optimize the performance of flooding based DTN routing protocols in terms of delivery probability and overhead ratio. The objective is to reduce the overhead incurred by flooding based protocols while maintaining or enhancing their superior delivery ratios. We compare the performance of two popular flooding based protocols (i.e. Epidemic and PROPHET) with various buffer management schemes under two different scenarios. The results demonstrate an improvement in delivery probabilities with significant reduction of overhead ratios. For comparison, two popular quota based protocols (e.g. Spray and Wait and Encounter Based Routing (EBR)) are used. It is clearly observed that efficient buffer management can reduce the overhead incurred by these popular flooding based protocols to values comparable with their quota based counterparts while still maintaining higher delivery ratios.

Existing buffer management policies are classified on the basis of their network information requirements. While policies requiring complete network information perform better than those that utilize partial or no network knowledge, they are not favorably adopted due to their implementation difficulties. On the other hand policies requiring local information are more popular. These policies use certain attributes to make selection of messages to be discarded. The aim of this research is to determine which attribute produces the most desirable results in congested and constrained environments. While comparing the performance of various buffer management schemes we conclude that size is an appropriate criterion when the node's buffer overflows. It is especially advantageous in a highly congested and constrained environment. Schemes that use other criteria like Time-To-Live (TTL) or Hop Count do not produce the desirable effects to alleviate congestion. Furthermore their high resource consumption makes them inappropriate for application in resource constrained environments like a disaster scenario.

We also propose an efficient buffer management policy Size Aware Drop (SA-Drop) which estimates the difference of the available buffer size and the size of the incoming message and then determine a threshold size for message discard. We evaluate its performance with both Epidemic and PRoPHET in two different scenarios. In Random Waypoint scenario, our proposed policy SA-Drop outperforms existing Drop Oldest (DOA) and Drop Largest (DLA) policies by 17% and 8% respectively in terms of delivery probability while it reduces the overhead ratio by 14% and 8% respectively. In the second scenario where near realistic movements of nodes in a disaster are simulated SA-Drop when implemented with PRoPHET outperforms both Evict Shortest Life Time First (SHLI) and Evict Most Forwarded First (MOFO) by 7% and 14% respectively in terms of delivery probability. In this scenario SA-Drop significantly reduces overhead ratios by 64% and 72% respectively as compared to SHLI and MOFO.

We also determine the impact of congestion and buffer constraints on the performance of our proposed scheme. The results conclude that in a more congested and constrained environment our scheme maintains its enhanced performance while others exhibit serious degradation both in terms of delivery probability and overhead ratio.

5.1. FUTURE DIRECTIONS

As mentioned earlier it is a rather fresh area of research with little work done. In this study we isolate the impact of drop policies on the performance of flooding based routing protocols by implementing same message scheduling/forwarding (i.e. FIFO and GRTR_Max) policies with each buffer management scheme. An interesting future direction is to evaluate the performance of our proposed scheme with other scheduling/forwarding policies. Designing a new scheduling scheme based on size, for further optimization is an interesting extension to our work.

In this thesis we only compare the performance of optimized flooding based protocols with that of existing quota based protocols. We have not compared their performance with resource aware and context aware protocols like RAPID and ORWAR. It is interesting to see how our scheme performs when compared with these protocols.

As mentioned earlier Delay Tolerant Networking (DTN) is expected to become an increasingly important paradigm in future communication. An important future research issue in DTN will be buffer management. In this study, we propose a buffer management policy and provide a comparative review of existing buffer management schemes. Due to encouraging results we believe that great potential lies in the inspection and exploration of

this area further. More attributes can be identified and new policies can be proposed with local or partial network knowledge that can effortlessly be implemented in other real life DTN applications.

BIBLIOGRAPHY

- [1] Th. Luckenbach, Th. Risse, Th. Kirste, and H. Kirchner A. Meissner, "Design Challenges for an Integrated Disaster Management Communication and Information System," in *IEEE Workshop on Disaster Recovery Networks*, New York, June 2002.
- [2] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets," in *Proc. of ACM SIGCOMM '03*, New York, NY, USA, 2003, pp. 27–34.
- [3] F. Warthman. (2003) "Delay-Tolerant Networks (DTNs) - A Tutorial". [Online]. <http://www.dtnrg.org/docs/tutorials/warthman-1.1.pdf>
- [4] K. Scott and S. Burleigh. (July 2006) Bundle Protocol Specification (Internet draft). IRTF.
- [5] R. Fletcher, A. Hasson A. Pentland, "DakNet: Rethinking Connectivity in Developing Nations," *Computer*, Vol. 37, No. 1., pp. 78-83, 2004.
- [6] Disruption Tolerant Networking. [Online]. [\[Online\]. Available: http://www.darpa.mil/ato/solicit/DTN/](http://www.darpa.mil/ato/solicit/DTN/)
- [7] Hidekazu Oki, Yong Wang, Margaret Martonosi, Li-Shiuan Peh, Daniel Rubinstein Philo Juang, "Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet," in *Proc. of ASPLOS-X*, October 2002, pp. 96–107.
- [8] A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott and H. Weiss S. Burleigh, "Delay-Tolerant Networking: An Approach to Interplanetary Internet," *IEEE Communications Magazine*, Vol. 41, pp. 128-136, 2003.
- [9] Gianluca Iannaccone, Jayanthkumar Kannan, Fernando Silveira, Nina Taft, Kevin Fall, "A Disruption-Tolerant Architecture for Secure and Efficient Disaster Response Communications," 2010.
- [10] K. Fall and R. Patra S. Jain, "Routing in a Delay Tolerant Network," *ACM SIGCOMM Computer Communication Review*, vol. 34, pp. 145-158, 2004.
- [11] L. Li, and P. Ward E. Jones, "Practical Routing in Delay-Tolerant Networks," in *Proc. of ACM Chants Workshop*, August 2005, pp. 237–243.
- [12] T. Friedman, and V. Conan J. Leguay, "DTN Routing in a Mobility Pattern Space," in *Proc. of ACM Chants Workshop*, August 2005, pp. 276–283.
- [13] T., Psounis, K. Spyropoulos, "Single-Copy Routing in Intermittently Connected Mobile Networks," in *Proc. of Sensor and Ad Hoc Communications and Networks SECON*, October 2004, pp. 235–244.

- [14] T. Vahdat and D. Becker, "Epidemic Routing for Partially Connected Ad Hoc Networks," Duke University, Durham, NC, Tech. Rep. CS-2000-06, July 2000.
- [15] A. Doria, O. Scheln and A. Lindgren, "Probabilistic Routing in Intermittently Connected Networks," in *Proc. of MobiHoc 03*, 2003.
- [16] Konstantinos Psounis, Cauligi S. Raghavendra and Thrasyvoulos Spyropoulos, "Spray And Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks," in *Proc. of the 2005 ACM SIGCOMM workshop on Delay-Tolerant Networking*, 2005.
- [17] Konstantinos Psounis, Cauligi S. Raghavendra and Thrasyvoulos Spyropoulos, "Spray And Focus: Efficient Mobility-Assisted Routing for Heterogeneous and Correlated Mobility," in *Proc. of Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops*, 2007.
- [18] Mehedi Bakht, Robin Kravets and Samuel C. Nelson, "Encounter-Based Routing in DTNs," in *Proc. of INFOCOM 2009, IEEE*, April 2009, pp. 846 - 854.
- [19] G. Neglia, J. Kurose, and D. Towsley X. Zhang, "Performance Modeling of Epidemic Routing," *Computer Networks, Elsevier, Vol. 51, No. 10*, pp. 2867-2891, July 2007.
- [20] P. Nain, G. Koole and R. Groenevelt, "The Message Delay in Mobile Ad Hoc Networks Performance Evaluation," *Performance Evaluation - Performance 2005, Vol. 62*, no. 1-4, pp. 210–228, October 2005.
- [21] J. Widmer, J.Y. and L. Boudec, "Network Coding for Efficient Communication in Extreme Networks," in *Proc. of ACM SIGCOMM Workshop on Delay Tolerant Networking (WDTN)*, 2005.
- [22] B. Liang, B. Li and Y. Lin, "Performance Modeling of Network Coding in Epidemic Routing," in *MobiOpp'07*, June 2007.
- [23] M. Demmer, R. Patra, K. Fall and S. Jain, "Using Redundancy to Cope with Failures in a Delay Tolerant Network," *SIGCOMM'05*, pp. 22–26, August 2005.
- [24] S. Jain, M. Martonosi, K. Fall and Y. Wang, "Erasure Coding Based Routing for Opportunistic Networks," in *Proc. of ACM SIGCOMM Workshop on Delay Tolerant Networking (WDTN)*, 2005.
- [25] Brian Neil Levine, Arun Venkataramani and Aruna Balasubramanian, "DTN Routing As a Resource Allocation Problem," in *Proc. ACM SIGCOMM*, August 2007.
- [26] Gabriel Sandulescu and Simin Nadjm-Tehrani, "Opportunistic DTN Routing with Window-aware Adaptive Replication," in *Proc. of the 4th Asian Conference on Internet Engineering AINTEC '08*, NY, USA, 2008, pp. 103-112.

- [27] C. Barakat, T. Spyropoulos and A. Krifa, "Optimal Buffer Management Policies for Delay Tolerant Networks," in *Proc. of IEEE Fifth Annual Communication Society Conference on Sensor, Mesh and Ad Hoc Communication and Networks (SECON)*, 2008.
- [28] A.Lindgren and K. S. Phanse, "Evaluation of Queuing Policies and Forwarding Strategies for Routing in Intermittently Connected Networks," in *Proc. of IEEE COMSWARE*, January 2006, pp. 1-10.
- [29] V. Erramilli and M. Crovella, "Forwarding in Opportunistic Networks with Resource Constraints," in *CHANTS '08 Proc. of the third ACM workshop on Challenged networks*, San Francisco, California, USA, September 2008, pp. 41-48.
- [30] A. Fagg, and B. Levine J. Davis, "Wearable Computers as Packet Transport Mechanisms in Highly-Partitioned Ad-Hoc Networks," in *Proc. of Fifth International Symposium on Wearable Computers*, 2001, pp. 141–148.
- [31] Qaisar Ayub and Sulma Rashid, "Effective Buffer Management Policy DLA for DTN Routing Protocols Under Congestion," *International Journal of Computer and Network Security*, Vol 2, No 9, pp. 118-121, Sep 2010.
- [32] Ling Zhao, Zhanjun Liu, Qilie Liu and Yun Li, "N-DROP Congestion Control Strategy Under Epidemic Routing In DTN," in *IWCMC '09 Proc. of the 2009 International Conference on Wireless Communications and Mobile*, CHONGQING 400065, CHINA, 2009, pp. 457-460.
- [33] Sulma Rashid, M.Soperi, Mohd Zahid and Qaisar Ayub, "Buffer Scheduling Policy for Opportunistic Networks," *International Journal of Scientific & Engineering Research*, Volume 2, Issue 7, ISSN 2229-5518, July 2011.
- [34] Hanjin Park, Ikjun Yeom and Dohyung Kim, "Minimizing the Impact of Buffer Overflow in DTN," in *Proc. of International Conference on Future Internet Technologies (CFI)*, 2008.
- [35] Mengjiong Qian, Depeng Jin, Li Su, Lieguang Zeng and Yong Li, "Adaptive Optimal Buffer Management Policies for Realistic DTN," in *Proc. of IEEE Global Telecommunication Conference (GlobeCom)*, 2009.
- [36] Hiroshi Esaki and Sathita Kaveevivitchai, "Independent DTNs Message Deletion Mechanism for Multi-copy Routing Scheme," in *Proc. of the Sixth Asian Internet Engineering Conference (AINTEC '10)*, New York, NY, USA, 2010, pp. 48-55.
- [37] R. Hansen, R. Basu, P. Rosales Hain, R. Krishnan and R. Ramanathan, "Prioritized Epidemic Routing for Opportunistic Networks," in *MobiOpp '07: Procs. of the 1st international MobiSys workshop on Mobile opportunistic networking*, ACM, New York, NY, USA, 2007, pp. 62–66.
- [38] Sulma Rashid and Qaisar Ayub, "T-Drop: An Optimal Buffer Management Policy to Improve QOS in DTN Routing Protocols," *Journal of Computing*, Vol 2, ISSUE 10, pp.

46-50, October 2010.

- [39] A. H. Abdullah, M. Soperi Mohd Zahid, Qaisar Ayub and Sulma Rashid, "Mean Drop an Effectual Buffer Management Policy for Delay Tolerant Network," *European Journal of Scientific Research*, ISSN 1450-216X, Vol.70, No.3, pp. 396-407, 2012.
- [40] Qaisar Ayub, M. Soperi Mohd Zahid, A.Hanan. Abdullah and Sulma Rashid, "E-DROP: An Effective Drop Buffer Management Policy for DTN Routing Protocols," *International Journal of Computer Applications (0975 – 8887)*, Vol. 13, No.7, pp. 8-13, January 2011.
- [41] Vania Conan, and Marcelo Dias de Amorim John Whitbeck, "Performance of Opportunistic Epidemic Routing on Edge-Markovian Dynamic Graphs," *IEEE TRANSACTIONS ON COMMUNICATIONS*, TCOM-09-0163, 2010.
- [42] Sulma Rashid, Dr.Mohd Soperi Mohd and Qaisar Ayub, "Optimization of Epidemic Router by New Forwarding Queue Mode TSMF," *International Journal of Computer Applications*, Vol. 7, No. 11, pp. 5–8, October 2010.
- [43] Qaisar Ayub, M. Soperi Mohd Zahid ,A.Hanan. Abdullah and Sulma Rashid, "Optimization of DTN Routing Protocols by Using Forwarding Strategy (TSMF) and Queuing Drop Policy (DLA)," *International Journal of Computer and Network Security*, Vol 2, ISSUE 10, pp. 71-75, October 2010.
- [44] ONE Simulator. [Online]. <http://www.netlab.tkk.fi/tutkimus/dtn/theone/>.
- [45] B. Gallagher, D.Jensen and B.N. Levine J. Burgess, "MaxProp: Routing for Vehicle Based Disruption-Tolerant Networks," in *Proc. of IEEE Infocom*, 2007, pp. 1-11.
- [46] Network Simulator. [Online]. <http://www.isi.edu/nsnam/ns/>
- [47] DTNSim2. [Online]. [Available: http://shoshin.uwaterloo.ca/dtnsim2/](http://shoshin.uwaterloo.ca/dtnsim2/)
- [48] A., J. Ott, and T. Kärkkäinen Keränen, "The ONE Simulator for DTN Protocol Evaluation," in *Simutools '09 Proc. of the 2nd International Conference on Simulation Tools and Techniques* , 2009.
- [49] Jeff Boleng, and Vanessa Davies Tracy Camp, "A Survey of Mobility Models for Ad hoc Network Research," *Wireless Communications & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, Vol. 2, No. 5, pp. 483–502, 2002.
- [50] Albert F. Harris, Robin Kravets and Samuel C. Nelson, "Event-Driven, Role-Based Mobility in Disaster Recovery Networks," in *CHANTS*, 2007.
- [51] J., Gass, R., Crowcroft, J., Hui, P., Diot, C., and Chaintreau Scott. (January 2006) A. CRAWDAD trace: cambridge/haggle imote/infocom(v. 2006-01-31).

- [52] E. Gerhards-Padilla, M. Gerharz, M. Frank, P. Martini and N. Aschenbruck, "Modelling Mobility in Disaster Area Scenarios," in *Proc. of the 10th International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, October 2007, pp. 4–12.
- [53] M.Y.S., Nicol, D.M. , Abdelzaher, T.F. , Kravets and R.H. Uddin, "A Post-Disaster Mobility Model for Delay Tolerant Networking ," in *Proc. of the 2009 Winter Simulation Conference (WSC)*, December 2009, pp. 2785 - 2796.

A) APPENDICES

APPENDIX I:

i) TOOLS FOR MOBILITY TRACE GENERATION IN NS-2

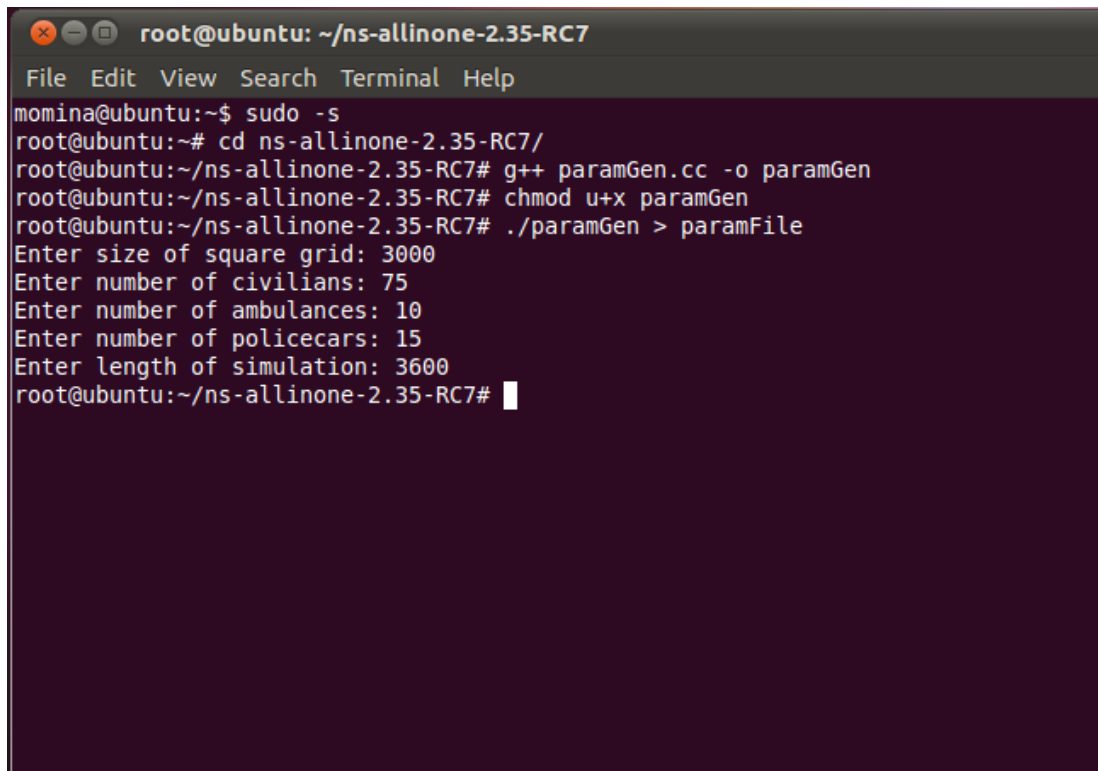
a) Parameter File Generation Tool for NS-2

Usage:

```
g++ paramGen.cc -o paramGen
```

```
chmod u+x paramGen
```

```
./paramGen > paramFile
```

A terminal window screenshot showing the execution of the parameter file generation tool. The window title is 'root@ubuntu: ~/ns-allinone-2.35-RC7'. The terminal output shows the user running 'sudo -s' to become root, then navigating to the directory 'ns-allinone-2.35-RC7'. The user then compiles 'paramGen.cc' into 'paramGen', sets permissions, and runs the tool. The tool prompts for several parameters: 'Enter size of square grid: 3000', 'Enter number of civilians: 75', 'Enter number of ambulances: 10', 'Enter number of policecars: 15', and 'Enter length of simulation: 3600'. The terminal ends with the prompt 'root@ubuntu:~/ns-allinone-2.35-RC7#'.

```
root@ubuntu: ~/ns-allinone-2.35-RC7
File Edit View Search Terminal Help
momina@ubuntu:~$ sudo -s
root@ubuntu:~# cd ns-allinone-2.35-RC7/
root@ubuntu:~/ns-allinone-2.35-RC7# g++ paramGen.cc -o paramGen
root@ubuntu:~/ns-allinone-2.35-RC7# chmod u+x paramGen
root@ubuntu:~/ns-allinone-2.35-RC7# ./paramGen > paramFile
Enter size of square grid: 3000
Enter number of civilians: 75
Enter number of ambulances: 10
Enter number of policecars: 15
Enter length of simulation: 3600
root@ubuntu:~/ns-allinone-2.35-RC7#
```

Figure A.1: Appendix i: Snapshot of Usage of Parameter File Generation Tool for NS-2

b) ONE Compatible Mobility Trace Generation Tool

Usage:

```
g++ disasterSimONE.cc -o disasterSimONE
```

```
chmod u+x disasterSimONE
```

```
./disasterSimONE [-d] < paramFile > oneMobilityTrace
```

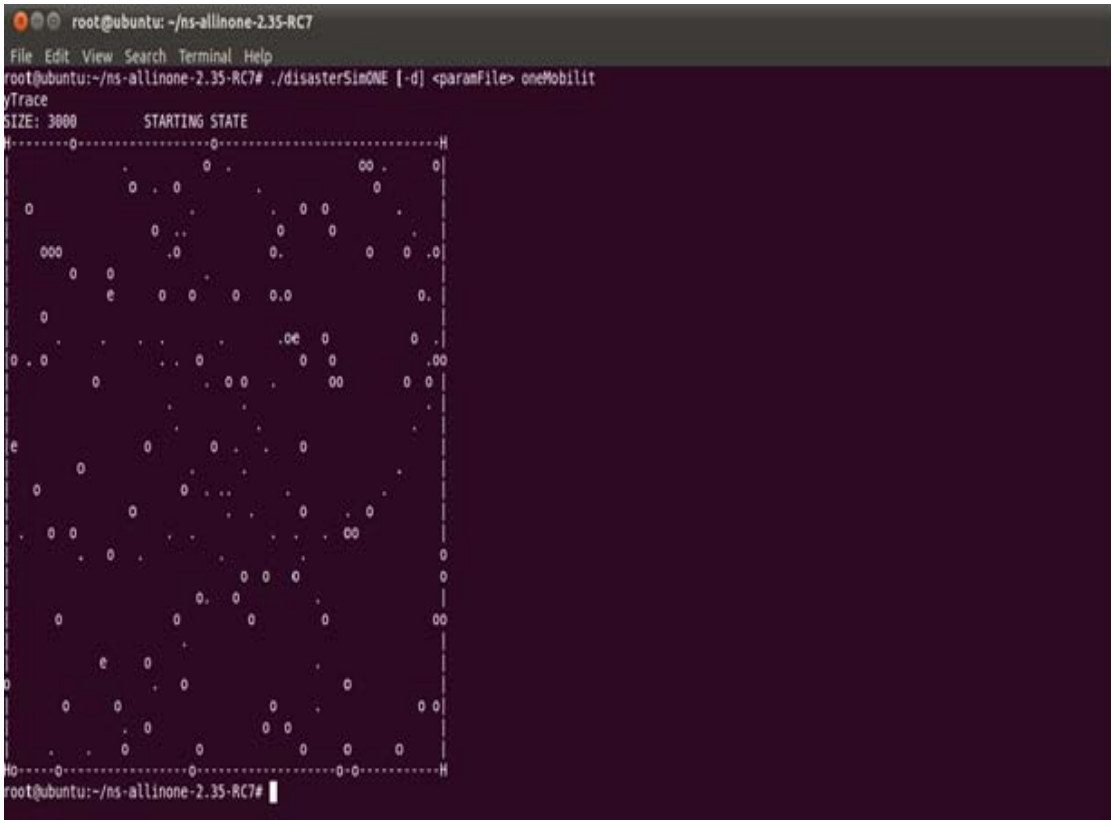



Figure A.2: Appendix i: Snapshot of Usage of ONE Compatible Mobility Trace File Generation Tool for NS-2

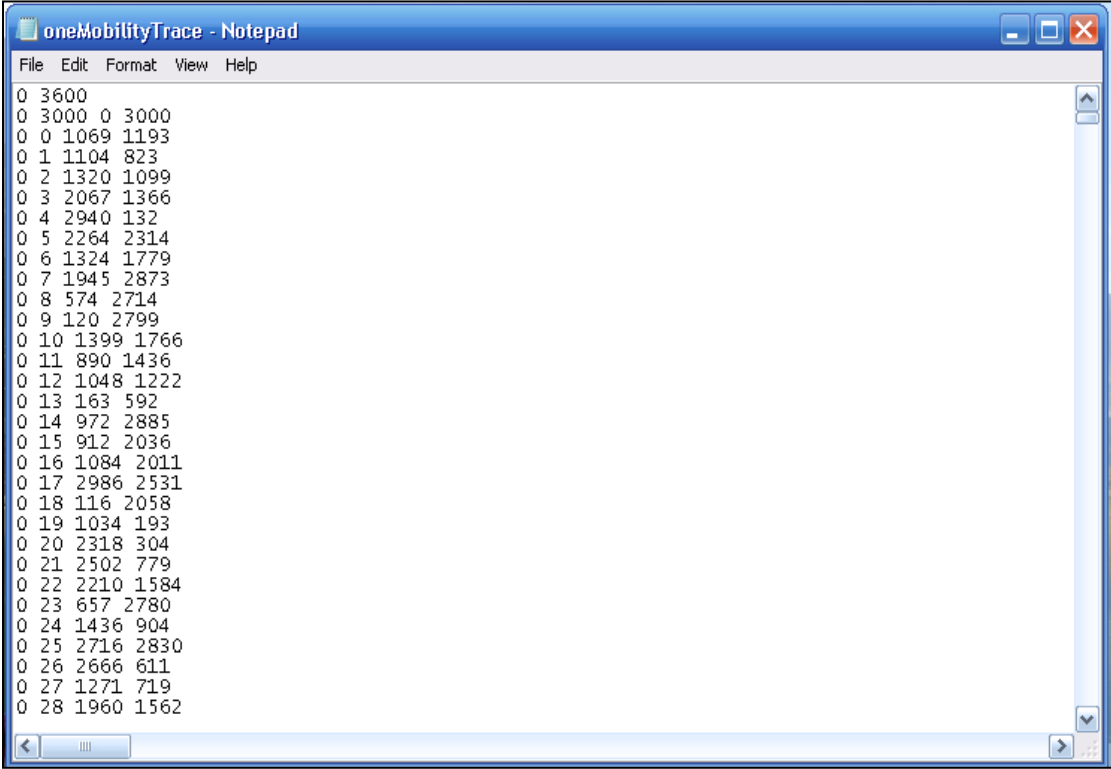


Figure A.3: Appendix i: Snapshot of oneMobilityTrace File

APPENDIX II:

i) PROTOCOLS CLASS FILES IN ONE SIMULATOR

a) **ActiveRouter.java**

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import java.util.Random;
import core.Connection;
import core.DTNHost;
import core.Message;
import core.MessageListener;
import core.Settings;
import core.SimClock;
import core.Tuple;

public abstract class ActiveRouter extends MessageRouter{
    public static final String DELETE_DELIVERED_S = "deleteDelivered";
    protected boolean deleteDelivered;
    public static final String RESPONSE_PREFIX = "R_";
    public static int TTL_CHECK_INTERVAL = 60;
    protected ArrayList<Connection> sendingConnections;
    private double lastTtlCheck;

    public ActiveRouter(Settings s){
        super(s);
        if (s.contains(DELETE_DELIVERED_S)){
            this.deleteDelivered = s.getBoolean(DELETE_DELIVERED_S);
        }
        else {
            this.deleteDelivered = false;
        }
    }

    protected ActiveRouter(ActiveRouter r) {
        super(r);
        this.deleteDelivered = r.deleteDelivered;
    }

    @Override
    public void init(DTNHost host, List<MessageListener> mListeners) {
        super.init(host, mListeners);
        this.sendingConnections = new ArrayList<Connection>(1);
        this.lastTtlCheck = 0;
    }

    @Override
    public void changedConnection(Connection con) { }
```

```

@Override
public boolean requestDeliverableMessages(Connection con) {
    if (isTransferring()) {
        return false;
    }
    DTNHost other = con.getOtherNode(getHost());
    ArrayList<Message> temp = new
ArrayList<Message>(this.getMessageCollection());
    for (Message m : temp)
    {
        if (other == m.getTo()){
            if (startTransfer(m, con) == RCV_OK){
                return true;
            }
        }
    }
    return false;
}

@Override
public boolean createNewMessage(Message m){
    makeRoomForNewMessage(m.getSize());
    return super.createNewMessage(m);
}

@Override
public int receiveMessage(Message m, DTNHost from){
    int rcvCheck = checkReceiving(m);
    if (rcvCheck != RCV_OK){
        return rcvCheck;
    }
    return super.receiveMessage(m, from);
}

@Override
public Message messageTransferred(String id, DTNHost from){
    Message m = super.messageTransferred(id, from);
    if (m.getTo() == getHost() && m.getResponseSize() > 0) {
        Message res = new Message(this.getHost(),m.getFrom(),
RESPONSE_PREFIX+m.getId(), m.getResponseSize());
        this.createNewMessage(res);
        this.getMessage(RESPONSE_PREFIX+m.getId()).setRequest(m);
    }
    return m;
}

protected List<Connection> getConnections() {
    return getHost().getConnections();
}

protected int startTransfer(Message m, Connection con) {
    int retVal;
    if (!con.isReadyForTransfer()) {
        return TRY_LATER_BUSY;
    }
    retVal = con.startTransfer(getHost(), m);
    if (retVal == RCV_OK) {
        // started transfer
    }
}

```

```

        addToSendingConnections(con);
    }
    else if (deleteDelivered && retVal == DENIED_OLD &&
m.getTo() == con.getOtherNode(this.getHost())){
        this.deleteMessage(m.getId(), false);
    }
    return retVal;
}
protected boolean canStartTransfer(){
    if (this.getNrofMessages() == 0){
        return false;
    }
    if (this.getConnections().size() == 0){
        return false;
    }
    return true;
}
protected int checkReceiving(Message m){
    if (isTransferring()) {
        return TRY_LATER_BUSY;
    }
    if ( hasMessage(m.getId()) || isDeliveredMessage(m) ){
        return DENIED_OLD;           // already seen this message ->
reject it
    }
    if (m.getTtl() <= 0 && m.getTo() != getHost()){
        return DENIED_TTL;
    }
    if (!makeRoomForMessage(m.getSize())){
        return DENIED_NO_SPACE;       // couldn't fit into buffer -> reject
    }
    return RCV_OK;
}

```

```

/*_For other drop policies these two functions are overridden*/

```

```

protected boolean makeRoomForMessage(int size){
    if (size > this.getBufferSize()){
        return false;           // message too big for the buffer
    }
    int freeBuffer = this.getFreeBufferSize();
    while (freeBuffer < size)
    {
        Message m = getOldestMessage(true);
        // don't remove msgs being sent
        if (m == null) {
            return false;       // couldn't remove any more messages
        }
        deleteMessage(m.getId(), true);
        freeBuffer += m.getSize();
    }
    return true;
}

```

```

protected void dropExpiredMessages() {
    Message[] messages = getMessageCollection().toArray(new Message[0]);
    for (int i=0; i<messages.length; i++)
    {
        int ttl = messages[i].getTtl();
        if (ttl <= 0) {
            deleteMessage(messages[i].getId(), true);
        }
    }
}

protected void makeRoomForNewMessage(int size){
    makeRoomForMessage(size);
}

protected Message getOldestMessage(boolean excludeMsgBeingSent) {
    Collection<Message> messages = this.getMessageCollection();
    Message oldest = null;
    for (Message m : messages) {
        if (excludeMsgBeingSent && isSending(m.getId())) {
            continue;
        }
        if (oldest == null ) {
            oldest = m;
        }
        else if (oldest.getReceiveTime() > m.getReceiveTime()) {
            oldest = m;
        }
    }
    return oldest;
}

protected List<Tuple<Message, Connection>> getMessagesForConnected() {
    if (getNrofMessages() == 0 || getConnections().size() == 0) {
        return new ArrayList<Tuple<Message, Connection>>(0);
    }
    List<Tuple<Message, Connection>> forTuples =
        new ArrayList<Tuple<Message, Connection>>();
    for (Message m : getMessageCollection()) {
        for (Connection con : getConnections()) {
            DTNHost to = con.getOtherNode(getHost());
            if (m.getTo() == to) {
                forTuples.add(new Tuple<Message,
Connection>(m,con));
            }
        }
    }
    return forTuples;
}

protected Message tryAllMessages(Connection con, List<Message> messages) {
    for (Message m : messages) {
        int retVal = startTransfer(m, con);
        if (retVal == RCV_OK) {
            return m;
        }
    }
}

```

```

        else if (retVal > 0) {
            return null;
        }
    }
    return null;
}
protected Connection tryMessagesToConnections(List<Message> messages,
List<Connection> connections) {
    for (int i=0, n=connections.size(); i<n; i++)
    {
        Connection con = connections.get(i);
        Message started = tryAllMessages(con, messages);
        if (started != null){
            return con;
        }
    }
    return null;
}
protected Connection tryAllMessagesToAllConnections(){
    List<Connection> connections = getConnections();
    if (connections.size() == 0 || this.getNrofMessages() == 0) {
        return null;
    }
    List<Message> messages = new ArrayList<Message>(this.getMessageCollection());
    this.sortByQueueMode(messages);
    return tryMessagesToConnections(messages, connections);
}
protected Connection exchangeDeliverableMessages() {
    List<Connection> connections = getConnections();
    if (connections.size() == 0) {
        return null;
    }
    @SuppressWarnings(value = "unchecked")
    Tuple<Message, Connection> t =
    tryMessagesForConnected(sortByQueueMode(getMessagesForConnected()));
    if (t != null) {
        return t.getValue(); // started transfer
    }
    for (Connection con : connections)
    {
        if (con.getOtherNode(getHost()).requestDeliverableMessages(con)){
            return con;
        }
    }
    return null;
}
protected void shuffleMessages(List<Message> messages){
    if (messages.size() <= 1){
        return;
    }
}
protected void shuffleMessages(List<Message> messages) {
    if (messages.size() <= 1) {
        return; // nothing to shuffle
    }
}

```

```

        Random rng = new Random(SimClock.getIntTime());
        Collections.shuffle(messages, rng);
    }
    protected void addToSendingConnections(Connection con) {
        this.sendingConnections.add(con);
    }
    public boolean isTransferring() {
        if (this.sendingConnections.size() > 0) {
            return true; // sending something
        }
        if (this.getHost().getConnections().size() == 0) {
            return false; // not connected
        }
        List<Connection> connections = getConnections();
        for (int i=0, n=connections.size(); i<n; i++){
            Connection con = connections.get(i);
            if (!con.isReadyForTransfer()) {
                return true;
            }
        }
        return false;
    }
    public boolean isSending(String msgId){
        for (Connection con : this.sendingConnections)
        {
            if (con.getMessage() == null) {
                continue; // transmission is finalized
            }
            if (con.getMessage().getId().equals(msgId)){
                return true;
            }
        }
        return false;
    }
    @Override
    public void update(){
        super.update();
        for (int i=0; i<this.sendingConnections.size(); )
        {
            boolean removeCurrent = false;
            Connection con = sendingConnections.get(i);
            if (con.isMessageTransferred()){
                if (con.getMessage() != null) {
                    transferDone(con);
                    con.finalizeTransfer();
                    removeCurrent = true;
                }
            }
            else if (!con.isUp()){
                if (con.getMessage() != null){
                    transferAborted(con);
                    con.abortTransfer();
                }
                removeCurrent = true;
            }
        }
    }

```

```

        if (removeCurrent) {
            if (this.getFreeBufferSize() < 0) {
                this.makeRoomForMessage(0);
            }
            sendingConnections.remove(i);
        }
        else {
            i++;
        }
    }
    if (SimClock.getTime() - lastTtlCheck >= TTL_CHECK_INTERVAL &&
        sendingConnections.size() == 0) {
        dropExpiredMessages();
        lastTtlCheck = SimClock.getTime();
    }
}
protected void transferAborted(Connection con) {}
protected void transferDone(Connection con) {}
}

```

Figure A.4: Appendix ii: Java Class File for Active Router

b) EpidemicRouter.java with Size Aware Drop (SA-Drop)

```

package routing;
import core.Settings;
import core.Message;
import core.SimClock;
import java.util.Collection;
import java.util.Collections;

public class EpidemicRouter extends ActiveRouter {
    public EpidemicRouter(Settings s){
        super(s);
    }
    protected EpidemicRouter(EpidemicRouter r) {
        super(r);
    }
    @Override
    public void update(){
        super.update();
        if (isTransferring() || !canStartTransfer()) {
            return;
        }
        if (exchangeDeliverableMessages() != null) {
            return;
        }
        this.tryAllMessagesToAllConnections();
    }
}

```



```

        /* SA-Drop Implemented as Drop Policy*/
    @Override
    protected boolean makeRoomForMessage(int size){
        if (size > this.getBufferSize()){
            return false;           // message too big for the buffer
        }
        int freeBuffer = this.getFreeBufferSize();
        int thresholdsize = 0;
        if (size > freeBuffer){
            thresholdsize = (size - freeBuffer);
        }
        else if (size==freeBuffer){
            thresholdsize = size;
        }
        while (size>=freeBuffer){
            Message m = getThresholdMessage(thresholdsize,true);
            if (m == null){
                return false;      // couldn't remove any more messages
            }
            /* delete message from the buffer as "drop" */
            deleteMessage(m.getId(), true);
            freeBuffer += m.getSize();
        }
        return true;
    }
    protected Message getThresholdMessage(int ts, boolean excludeMsgBeingSent)
    {
        Collection<Message> messages = this.getMessageCollection();
        Message threshold = null;
        for (Message m : messages)
        {
            if (excludeMsgBeingSent && isSending(m.getId())){
                continue; // skip the message(s) that router is sending
            }
            if (threshold == null ){
                threshold = m;
            }
            if (m.getSize()>=ts){
                threshold = m;
            }
            else if(threshold.getSize() < m.getSize()){
                threshold = m;
            }
        }
        return threshold;
    }
    @Override
    public MessageRouter replicate() {
        EpidemicRouter r = new EpidemicRouter(this);
        return r;
    }
}

```

Figure A.5: Appendix ii: Java Class File for Epidemic Router with Size Aware Drop (SA-Drop)

c) **ProphetRouter.java with Size Aware Drop (SA-Drop)**

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import core.Connection;
import core.DTNHost;
import core.Message;
import core.Settings;
import core.SimClock;
import core.Tuple;

public class ProphetRouter extends ActiveRouter {
    public static final double P_INIT = 0.75;
    public static final double DEFAULT_BETA = 0.25;
    public static final double GAMMA = 0.98;
    public static final String PROPHET_NS = "ProphetRouter";
    public static final String SECONDS_IN_UNIT_S = "secondsInTimeUnit";
    public static final String BETA_S = "beta";
    private int secondsInTimeUnit;
    private double beta;
    private Map<DTNHost, Double> preds;
    private double lastAgeUpdate;
    public 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();
    }
    protected ProphetRouter(ProphetRouter r) {
        super(r);
        this.secondsInTimeUnit = r.secondsInTimeUnit;
        this.beta = r.beta;
        initPreds();
    }
    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 oldValue = getPredFor(host);
    double newValue = oldValue + (1 - oldValue) * P_INIT;
    preds.put(host, newValue);
}
public double getPredFor(DTNHost host) {
    ageDeliveryPreds();           // make sure preds are updated before getting
    if (preds.containsKey(host)) {
        return preds.get(host);
    }
    else {
        return 0;
    }
}
private void updateTransitivePreds(DTNHost host) {
    MessageRouter otherRouter = host.getRouter();
    assert otherRouter instanceof ProphetRouter : "PROPHET only works " +
    " with other routers of same type";
    double pForHost = getPredFor(host);           // P(a,b)
    Map<DTNHost, Double> othersPreds =
    ((ProphetRouter)otherRouter).getDeliveryPreds();
    for (Map.Entry<DTNHost, Double> e : othersPreds.entrySet()) {
        if (e.getKey() == getHost()) {
            continue;
        }
        double pOld = getPredFor(e.getKey());     // P(a,c)_old
        double pNew = pOld + (1 - pOld) * pForHost * e.getValue() * beta;
        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());
        ProphetRouter othRouter = (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 = ((ProphetRouter)tuple1.getValue().
            getOtherNode(getHost()).getRouter()).getPredFor(
            tuple1.getKey().getTo());
        double p2 = ((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;
}

/* SA-Drop Implemented as Drop Policy*/

@Override
protected boolean makeRoomForMessage(int size){
    if (size > this.getBufferSize()){
        return false;           // message too big for the buffer
    }
    int freeBuffer = this.getFreeBufferSize();
    int thresholdsize = 0;
    if (size > freeBuffer){
        thresholdsize = (size - freeBuffer);
    }
    else if (size==freeBuffer){
        thresholdsize = size;
    }
    while (size>=freeBuffer)
    {
        Message m = getThresholdMessage(thresholdsize,true);
        if (m == null){
            return false;
        }
        deleteMessage(m.getId(), true);
        freeBuffer += m.getSize();
    }
    return true;
}

protected Message getThresholdMessage(int ts, boolean excludeMsgBeingSent)
{
    Collection<Message> messages = this.getMessageCollection();
    Message threshold = null;
    for (Message m : messages)
    {
        if (excludeMsgBeingSent && isSending(m.getId())){
            continue;           // skip the message(s) that router is sending
        }
        if (threshold == null ){
            threshold = m;
        }
        if (m.getSize()>=ts){
            threshold = m;
        }
        else if(threshold.getSize() < m.getSize()){
            threshold = m;
        }
    }
    return threshold;
}
}

```

```

@Override
public MessageRouter replicate() {
    ProphetRouter r = new ProphetRouter(this);
    return r;
}
}

```

Figure A.6: Appendix ii: Java Class File for Prophet Router with Proposed policy (SA-Drop)

d) SprayAndWaitRouter.java

```

import java.util.ArrayList;
import java.util.List;
import core.Connection;
import core.DTNHost;
import core.Message;
import core.Settings;
import java.util.Collection;
import java.util.Collections;

public class SprayAndWaitRouter extends ActiveRouter {
    public static final String NROF_COPIES = "nrofCopies";
    public static final String BINARY_MODE = "binaryMode";
    public static final String SPRAYANDWAIT_NS = "SprayAndWaitRouter";
    public static final String MSG_COUNT_PROPERTY = SPRAYANDWAIT_NS + "." + "copies";
    protected int initialNrofCopies;
    protected boolean isBinary;
    public SprayAndWaitRouter(Settings s) {
        super(s);
        Settings snwSettings = new Settings(SPRAYANDWAIT_NS);
        initialNrofCopies = snwSettings.getInt(NROF_COPIES);
        isBinary = snwSettings.getBoolean( BINARY_MODE);
    }
    protected SprayAndWaitRouter(SprayAndWaitRouter r) {
        super(r);
        this.initialNrofCopies = r.initialNrofCopies;
        this.isBinary = r.isBinary;
    }
    @Override
    public int receiveMessage(Message m, DTNHost from) {
        return super.receiveMessage(m, from);
    }
    @Override
    public Message messageTransferred(String id, DTNHost from) {
        Message msg = super.messageTransferred(id, from);
        Integer nrofCopies = (Integer)msg.getProperty(MSG_COUNT_PROPERTY);
    }
}

```

```

        assert nrofCopies != null : "Not a SnW message: " + msg;
        if (isBinary) {
            /* in binary S'n'W the receiving node gets ceil(n/2) copies */
            nrofCopies = (int)Math.ceil(nrofCopies/2.0);
        }
        else {
            /* in standard S'n'W the receiving node gets only single copy */
            nrofCopies = 1;
        }
        msg.updateProperty(MSG_COUNT_PROPERTY, nrofCopies);
        return msg;
    }

    @Override
    public boolean createNewMessage(Message msg) {
        makeRoomForNewMessage(msg.getSize());
        msg.setTtl(this.msgTtl);
        msg.addProperty(MSG_COUNT_PROPERTY, new Integer(initialNrofCopies));
        addToMessages(msg, true);
        return true;
    }

    @Override
    public void update() {
        super.update();
        if (!canStartTransfer() || isTransferring()) {
            return; // nothing to transfer or is currently
transferring
        }
        /* try messages that could be delivered to final recipient */
        if (exchangeDeliverableMessages() != null) {
            return;
        }
        /* create a list of SAWMessages that have copies left to distribute */
        @SuppressWarnings(value = "unchecked")
        List<Message> copiesLeft = sortByQueueMode(getMessagesWithCopiesLeft

        if (copiesLeft.size() > 0) {
            /* try to send those messages */
            this.tryMessagesToConnections(copiesLeft, getConnections());
        }
    }
    protected List<Message> getMessagesWithCopiesLeft() {
        List<Message> list = new ArrayList<Message>();
        for (Message m : getMessageCollection())
        {
            Integer nrofCopies = (Integer)m.getProperty(MSG_COUNT_PROPERTY);
            assert nrofCopies != null : "SnW message " + m + " didn't have " +
                "nrof copies property!";
            if (nrofCopies > 1) {
                list.add(m);
            }
        }
        return list;
    }
}

```

```

@Override
protected void transferDone(Connection con) {
    Integer nrofCopies;
    String msgId = con.getMessage().getId();
    Message msg = getMessage(msgId);
    if (msg == null) {
        return;
    }
    nrofCopies = (Integer)msg.getProperty(MSG_COUNT_PROPERTY);
    if (isBinary) {
        nrofCopies /= 2;
    }
    else {
        nrofCopies--;
    }
    msg.updateProperty(MSG_COUNT_PROPERTY, nrofCopies);
}
@Override
public SprayAndWaitRouter replicate() {
    return new SprayAndWaitRouter(this);
}
}

```

Figure A.7: Appendix ii: Java Class File for Spray and Wait Router

e) EBRRouter.java

```

package routing;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Set;
import java.util.HashSet;
import core.Connection;
import core.DTNHost;
import core.Message;
import core.Settings;
import core.SimClock;
import core.Tuple;

/* Route based on encounter value */

public class EBRRouter extends ActiveRouter {
    /** identifier for the intial number of copies setting {@value}*/
    public static final String NROF_COPIES = "nrofCopies";
    public static final String ALPHA = "alpha";
    public static final String UPDATE_POPINTERVAL = "updatePOPInterval";
    /** Popularity router's settings namespace {@value}*/

```



```

public static final String EBR_NS = "EBRRouter";
public static final String MSG_COUNT_PROPERTY = EBR_NS + "." + "copies";
/** Popularity counter */
private double EV = 0;
/** Current window counter */
private double windowCounter = 0;
/** Alpha */
private double alpha;
/** Initial number of copies */
private int initialNrofCopies;
/** Future time to update EV */
private double updateInterval;
private double timeToUpdate = 0;
/** IDs of the messages that are known to have reached the final dst */
private Set<String> ackedMessageIds;
/**
 * Constructor. Creates a new message router based on the settings in
 * the given Settings object.
 * @param s The settings object
 */
public EBRRouter(Settings s) {
    super(s);
    Settings EBRSettings = new Settings(EBR_NS);
    initialNrofCopies = EBRSettings.getInt(NROF_COPIES);
    alpha = EBRSettings.getDouble(ALPHA);
    updateInterval = EBRSettings.getInt(UPDATE_POPINTERVAL);
}
/**
 * Copyconstructor.
 * @param r The router prototype where setting values are copied from
 */
protected EBRRouter(EBRRouter r) {
    super(r);
    this.initialNrofCopies = r.initialNrofCopies;
    this.alpha = r.alpha;
    this.updateInterval = r.updateInterval;
    this.ackedMessageIds = new HashSet<String>();
    this.EV = r.EV;
    this.windowCounter = r.windowCounter;
}
@Override
public int receiveMessage(Message m, DTNHost from) {
    return super.receiveMessage(m, from);
}
@Override
public void changedConnection(Connection con) {
    if (con.isUp()) {
        DTNHost otherHost = con.getOtherNode(getHost());
        MessageRouter mRouter = otherHost.getRouter();
        assert mRouter instanceof EBRRouter : "EBRRouter only works "+
            " with other routers of same type";
        EBRRouter otherRouter = (EBRRouter)mRouter;
    }
}

```

```

        // exchange ACKed message data
        this.ackedMessageIds.addAll(otherRouter.ackedMessageIds);
        otherRouter.ackedMessageIds.addAll(this.ackedMessageIds);
        deleteAkedMessages();
        otherRouter.deleteAkedMessages();
        // Update current window counter
        windowCounter++;
    }
}
public double getEV() {
    return this.EV;
}
/**
 * Deletes the messages from the message buffer that are known to be ACKed
 */
private void deleteAkedMessages() {
    for (String id : this.ackedMessageIds)
    {
        if (this.hasMessage(id)) {
            this.deleteMessage(id, false);
        }
    }
}
@Override
public boolean createNewMessage(Message msg) {
    makeRoomForNewMessage(msg.getSize());
    msg.setTtl(this.msgTtl);
    msg.addProperty(MSG_COUNT_PROPERTY, new Integer(initialNrofCopies));
    addToMessages(msg, true);
    return true;
}
@Override
public void update() {
    super.update();
    if (SimClock.getTime() >= timeToUpdate) {
        EV = alpha * windowCounter + (1-alpha) * EV;
        windowCounter = 0;
        timeToUpdate = SimClock.getTime() + updateInterval;
    }
    if (isTransferring() || !canStartTransfer()) {
        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 all connected hosts collect all msgs that have a higher
    // prob. of delivery by the other host

```

```

    for (Connection con : getConnections())
    {
        DTNHost other = con.getOtherNode(getHost());
        EBRRouter othRouter = (EBRRouter)other.getRouter();
        if (othRouter.isTransferring()) {
            continue;          // skip hosts that are transferring
        }
        for (Message m : msgCollection)
        {
            if (othRouter.hasMessage(m.getId())) {
                continue;      // skip msgs that the other one has
            }
            /** Assume our popularity is x and the other router's
             * popularity is y. Let n be the number of copies of
             * the message we have. Then we want to transfer
             * floor( y/x+y * n) messages to the other router
             */
            double y = othRouter.getEV();
            double x = this.getEV();
            int n =
                ((Integer)m.getProperty(MSG_COUNT_PROPERTY)).intValue();
            /** The other nodes popularity is high enough to send message */
            if (Math.floor((y*n)/(x+y)) > 1) {
                messages.add(new Tuple<Message,
Connection>(m,con));
            }
        }
        if (messages.size() == 0) {
            return null;
        }
        // sort the message-connection
        tuples
        Collections.sort(messages, new TupleComparator());
        return tryMessagesForConnected(messages);    // try to send messages
    }
    /**
    * Comparator for Message-Connection-Tuples that orders the tuples by
    * their connection's popularity
    */
    private class TupleComparator implements Comparator <Tuple<Message, Connection>> {
        public int compare(Tuple<Message, Connection> tuple1,
            Tuple<Message, Connection> tuple2) {
            // popularity counter for the connection in tuple1
            double p1 = ((EBRRouter)tuple1.getValue().
                getOtherNode(getHost()).getRouter()).
                getEV();          // tuple2
            double p2 = ((EBRRouter)tuple2.getValue().
                getOtherNode(getHost()).getRouter()).
                getEV();          // bigger popularity comes first
            if (p2-p1 == 0) {
                return 0;
            }
            else if (p2-p1 < 0) {
                return -1;
            }
        }
    }

```

```

        else {
            return 1;
        }
    }
}
/**
 * Reduces the number of copies we have left for a message.
 * This is the SENDER's copy
 */
@Override
protected void transferDone(Connection con) {
    String msgId = con.getMessage().getId();// Get this router's copy of the message
    Message msg = getMessage(msgId);
    /** If destination was reached, delete all copies from sender */
    if (con.getMessage().getTo().equals(con.getOtherNode(getHost()))){
        this.ackedMessageIds.add(msgId);
        this.deleteMessage(msgId,false);
        return;
    }
    if (msg == null) { // message has been dropped from the buffer after..
        return; // start of transfer no need to reduce amount of copies
    }
    double x = this.getEV(); // Get nodes' EVs
    double y = ((EBRRouter)con.getOtherNode(getHost()).getRouter()).getEV();
    int n = ((Integer)msg.getProperty(MSG_COUNT_PROPERTY)).intValue();
    // Transferred floor( y/(x+y) * n), and kept rest for self
    int newCount = n - (int)Math.floor((y*n)/(x+y));
    msg.updateProperty(MSG_COUNT_PROPERTY, new Integer(newCount));
}
/** This is the RECEIVER'S copy */
@Override
public Message messageTransferred(String id, DTNHost from) {
    Message msg = super.messageTransferred(id, from);
    // If this is final dest, simply set num of copies to 1
    if (msg.getTo().equals(this.getHost())) {
        msg.updateProperty(MSG_COUNT_PROPERTY, new Integer(1));
        this.ackedMessageIds.add(id);
        return msg;
    }
    double y = this.getEV();
    double x = ((EBRRouter)from.getRouter()).getEV();
    int n = ((Integer)msg.getProperty(MSG_COUNT_PROPERTY)).intValue();
    // "from" transferred floor( y/(x+y) * n) to self
    int newCount = (int)Math.floor((y*n)/(x+y));
    msg.updateProperty(MSG_COUNT_PROPERTY, new Integer(newCount));
    return msg;
}
@Override
public EBRRouter replicate() {
    return new EBRRouter(this);
}
}

```

Figure A.8: Appendix ii: Java Class File for EBR Router

ii) DROP POLICY FUNCTIONS IMPLEMENTED

a) Drop Oldest (DOA)

```
protected Message getOldestMessage(boolean excludeMsgBeingSent){
    Collection<Message> messages = this.getMessageCollection();
    Message oldest = null;
    for (Message m : messages)
    {
        if (excludeMsgBeingSent && isSending(m.getId())){
            continue;
        }
        if (oldest == null ){
            oldest = m;
        }
        else if (oldest.getReceiveTime() > m.getReceiveTime()){
            oldest = m;
        }
    }
    return oldest;
}
```

Figure A.9: Appendix ii: getOldestMessage() Function to Implement DOA

b) Drop Largest (DLA)

```
protected Message getLargestMessage(boolean excludeMsgBeingSent){
    Collection<Message> messages = this.getMessageCollection();
    Message largest = null;
    for (Message m : messages)
    {
        if (excludeMsgBeingSent && isSending(m.getId())){
            continue;
        }
        if (largest == null ){
            largest = m;
        }
        else if (largest.getSize() < m.getSize()){
            largest = m;
        }
    }
    return largest;
}
```

Figure A.10: Appendix ii: getLargestMessage() Function to Implement DLA

c) Evict Shortest Life Time First (SHLI)

```
protected Message getShortestLifeMessage(boolean excludeMsgBeingSent){
    Collection<Message> messages = this.getMessageCollection();
    Message shortest = null;
    for (Message m : messages)
    {
        if (excludeMsgBeingSent && isSending(m.getId())){
            continue;
        }
        if (shortest == null ){
            shortest = m;
        }
        else {
            double Rttl1 = ((shortest.getTtl()*60) - shortest.getReceiveTime());
            double Rttl2 = ((m.getTtl()*60) - m.getReceiveTime());
            if (Rttl1>Rttl2){
                shortest= m;
            }
        }
    }
    return shortest;
}
```

Figure A.11: Appendix ii: getShortestLifeMessage() Function to Implement SHLI

d) Evict Most Forwarded First (MOFO)

```
protected Message getMaxForwardedMessage(boolean excludeMsgBeingSent){
    Collection<Message> messages = this.getMessageCollection();
    Message maxfor = null;
    for (Message m : messages)
    {
        if (excludeMsgBeingSent && isSending(m.getId())){
            continue;
        }
        if (maxfor == null){
            maxfor = m;
        }
        else if(maxfor.getHopCount() < m.getHopCount){
            maxfor = m;
        }
    }
    return maxfor;
}
```

Figure A.12: Appendix ii: getMaxForwardedMessage() Function to Implement MOFO

e) **Size Aware Drop (SA-Drop)**

```
protected boolean makeRoomForMessage(int size){
    if (size > this.getBufferSize()){
        return false;                // message too big for the buffer
    }
    int freeBuffer = this.getFreeBufferSize();
    int thresholdsize = 0;           // Threshold size
    if (size > freeBuffer){
        thresholdsize = (size - freeBuffer);
    }
    else if (size==freeBuffer){
        thresholdsize = size;
    }
    /* delete messages from the buffer until there's enough space */
    while (size>=freeBuffer){
        Message m = getThresholdMessage(thresholdsize,true);
                                                // don't remove msgs being sent

        if (m == null){
            return false;                // couldn't remove any more messages
        }
        /* delete message from the buffer as "drop" */
        deleteMessage(m.getId(), true);
        freeBuffer += m.getSize();
    }
    return true;
}

protected Message getThresholdMessage(int ts, boolean excludeMsgBeingSent){
    Collection<Message> messages = this.getMessageCollection();
    Message threshold = null;
    for (Message m : messages)
    {
        if (excludeMsgBeingSent && isSending(m.getId())){
            continue;                // skip the message(s) that router is sending
        }
        if (threshold == null){
            threshold = m;
        }
        if (m.getSize()>=ts){
            threshold = m;    // drop msgs equal to or greater than threshold size
        }
        else if(threshold.getSize() < m.getSize()){
            threshold = m;    // drop largest sized message incase threshold size
                               // message is not present
        }
    }
    return threshold;
}
```

Figure A.13: Appendix ii: getThresholdMessage() and Modified makeRoomForMessage() Functions to Implement SA-Drop

iii) SCENARIO CONFIGURATION FILES

a) Scenario 1: Random Waypoint with Epidemic Router

```
#
# Default settings for the simulation of Disaster Scenario
#
## Scenario settings:
# endTime is in seconds

Scenario.name = RandomWayPoint
Scenario.simulateConnections = true
Scenario.updateInterval = 1.0
Scenario.nrofHostGroups = 3
Scenario.endTime = 3600

## Interface-specific settings:
# type: which interface class the interface belongs to
# transmitRange: range of the hosts' radio devices (meters)
# transmitSpeed: transmit speed of the radio devices (bytes per second)

Group.nrofInterfaces=1

# "Bluetooth" interface

btInterface.type = SimpleBroadcastInterface
btInterface.transmitSpeed = 250k
btInterface.transmitRange = 10

# Common settings for all groups:
# movementModel: movement model of the hosts (valid class name from movement package)
# router: router used to route messages (valid class name from routing package)
# msgTtl : TTL (minutes) of the messages created by this host group, default=infinite

Group.movementModel=RandomWaypoint
Group.router = EpidemicRouter
Group.msgTtl=60

## Group-specific settings:
# groupID : Group's identifier. Used as the prefix of host names
# nrofHosts: number of hosts in the group
# bufferSize: size of the message buffer (bytes)
# interface : Type of interface used by the hosts for transmission (Bluetooth interface)
# speed : speed of hosts in a group given in (m/s)
# waitTime : How long a host stays at a position after reaching it

# Group # 1 of pedestrians

Group1.groupID = p
Group1.nrofHosts=10
Group1.bufferSize = 3M
Group1.interface1 = btInterface
Group1.speed=0.5, 1.5
Group1.waitTime = 0, 120
```



```

# Group # 2 of pedestrians

Group2.groupID = o
Group2.nrofHosts= 40
Group2.bufferSize = 4M
Group2.interface1 = btInterface
Group2.speed=0.5, 1.5
Group2.waitTime = 0, 120

# Group # 3

Group3.groupID = s
Group3.nrofHosts= 50
Group3.bufferSize = 5M
Group3.interface1 = btInterface
Group3.speed=0.5, 1.5
Group3.waitTime = 0, 120

## Message creation parameters
# How many event generators

# Class of the event generator
# Events.class = MessageEventGenerator
# Specific settings for the MessageEventGenerator class
# Creation interval in seconds (one new message every 15 to 25 seconds)
# Message sizes (100kB - 1MB)
# range of message source/destination addresses
# Message ID prefix

Events.nrof = 1

Events1.class = MessageEventGenerator
Events1.interval = 15, 25
Events1.sizeRange = 100k, 1M
Events1.hosts=0, 99
Events1.prefix=M

# World's size for Movement Models without implicit size (width, height; meters)

# How long time to move hosts in the world before real simulation

# Seed for movement models' pseudo random number generator (default = 0)

MovementModel.worldSize = 1500, 1500
MovementModel.warmup = 500
MovementModel.rngseed = 1

## Reports - all report names have to be valid report classes

# how many reports to load
# default directory of reports (can be overridden per Report with output setting)

# Report classes to load

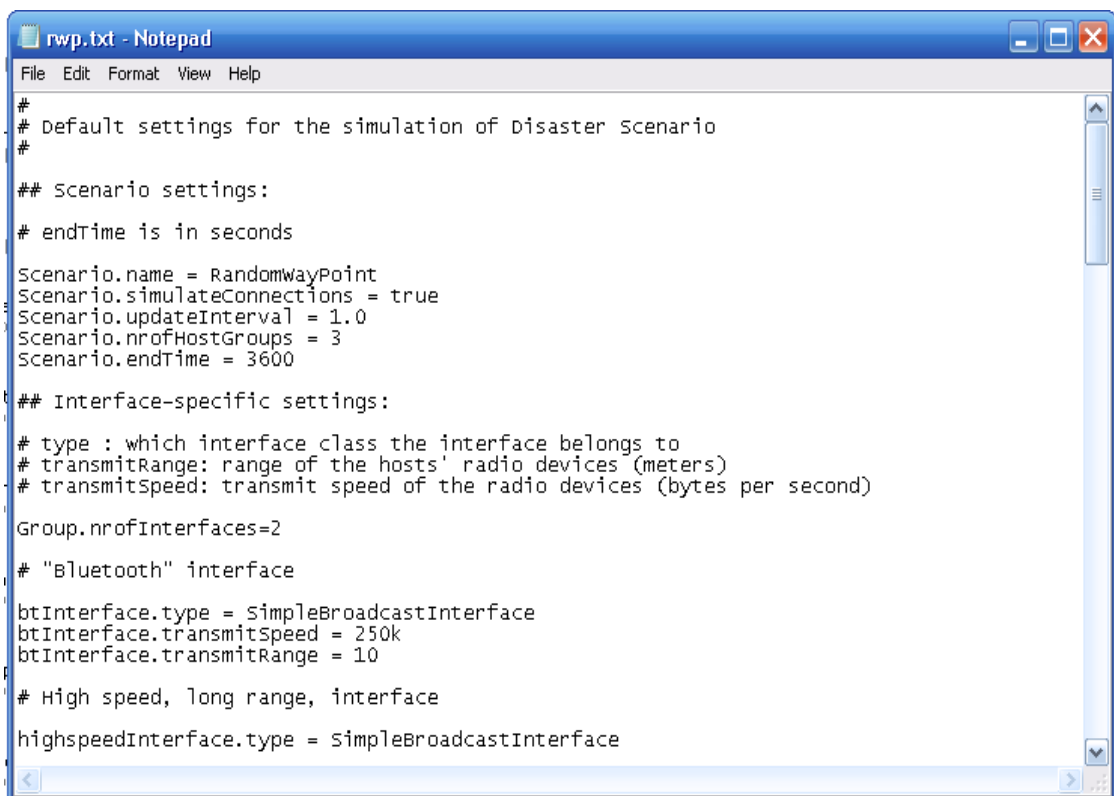
```

```
Report.nrofReports = 1
Report.reportDir = reports/Thesis/RWP
Report.report1 = MessageStatsReport

## Default settings for some routers settings
# Spray and Wait Router settings

SprayAndWaitRouter.nrofCopies = 10
SprayAndWaitRouter.binaryMode = false
```

Figure A.14: Appendix ii: Configuration File for Scenario 1: Random Waypoint



```
rwp.txt - Notepad
File Edit Format View Help
#
# Default settings for the simulation of Disaster Scenario
#
## scenario settings:
# endTime is in seconds
Scenario.name = RandomWayPoint
Scenario.simulateConnections = true
Scenario.updateInterval = 1.0
Scenario.nrofHostGroups = 3
Scenario.endTime = 3600
## Interface-specific settings:
# type : which interface class the interface belongs to
# transmitRange: range of the hosts' radio devices (meters)
# transmitSpeed: transmit speed of the radio devices (bytes per second)
Group.nrofInterfaces=2
# "Bluetooth" interface
btInterface.type = SimpleBroadcastInterface
btInterface.transmitspeed = 250k
btInterface.transmitRange = 10
# High speed, long range, interface
highspeedInterface.type = simpleBroadcastInterface
```

Figure A.15: Appendix ii: Snapshot of Scenario 1 Configuration File

b) Scenario 2: Disaster with PРоPHET Router

```
#
# Default settings for the simulation of Disaster Scenario
#
## Scenario settings:
# endTime is in seconds

Scenario.name = Disaster
Scenario.simulateConnections = true
Scenario.updateInterval = 1.0
Scenario.nrofHostGroups = 3
Scenario.endTime = 3600

## Interface-specific settings:

# type : which interface class the interface belongs to
# transmitRange: range of the hosts' radio devices (meters)
# transmitSpeed: transmit speed of the radio devices (bytes per second)

Group.nrofInterfaces=2

# "Bluetooth" interface

btInterface.type = SimpleBroadcastInterface
btInterface.transmitSpeed = 250k
btInterface.transmitRange = 10

# High speed, long range, interface

highspeedInterface.type = SimpleBroadcastInterface
highspeedInterface.transmitSpeed = 10M
highspeedInterface.transmitRange = 100

# Common settings for all groups:
# movementModel: movement model of the hosts (valid class name from movement package)
# router: router used to route messages (valid class name from routing package)
# msgTtl : TTL (minutes) of the messages created by this host group, default=infinite

Group.movementModel=ExternalMovement
Group.router = ProphetRouter
Group.msgTtl= 60

## Group-specific settings:
# groupID : Group's identifier. Used as the prefix of host names
# nrofHosts: number of hosts in the group
# bufferSize: size of the message buffer (bytes)

# Group of civilians

Group1.nrofInterfaces = 1
Group1.interface1 = btInterface
Group1.nrofHosts=75
```

```

Group1.bufferSize = 2M
Group1.groupID = civ

# Group of ambulances

Group2.nrofInterfaces = 1
Group2.interface1 = highspeedInterface
Group2.nrofHosts= 10
Group2.bufferSize = 3M
Group2.groupID = amb

# Group of police cars

Group3.nrofInterfaces = 1
Group3.interface1 = highspeedInterface
Group3.nrofHosts= 15
Group3.bufferSize = 4M
Group3.groupID = pol

## Movement model settings
# How long time to move hosts in the world before real simulation

MovementModel.warmup = 500
ExternalEvents.nrofpreloads = 500
ExternalEvents.class= ExternalMovement
ExternalMovement.file = oneMobilityTrace

## Message creation parameters
# How many event generators
# Class of the event generator
# Events.class = MessageEventGenerator
# Specific settings for the MessageEventGenerator class
# Creation interval in seconds (one new message every 15 to 25 seconds)
# Message sizes (100kB - 1MB)
# Range of message source/destination addresses
# Message ID prefix

Events.nrof = 4

Events1.class = MessageEventGenerator
Events1.interval = 15, 25
Events1.sizeRange = 100k, 1M
Events1.hosts=0, 99
Events1.prefix=S
Events1.msgTime=240,280

Events2.class = MessageEventGenerator
Events2.interval = 15, 25
Events2.size = 100k, 1M
Events2.hosts=0, 99
Events2.prefix=C
Events2.msgTime=265,305

```

```
Events3.class = MessageEventGenerator
Events3.interval = 15, 25
Events3.size = 100k, 1M
Events3.hosts=0, 99
Events3.prefix=P
Events3.msgTime=290,330

Events4.class = MessageEventGenerator
Events4.interval = 15, 25
Events4.size = 100k, 1M
Events4.hosts=0, 99
Events4.prefix=R
Events4.msgTime=315,355

# World's size for Movement Models without implicit size (width, height; meters)

MovementModel.worldSize = 3000, 3000

## Reports - all report names have to be valid report classes
# Number of reports to load
# Default directory of reports (can be overridden per Report with output setting)

# Report classes to load

Report.nrofReports = 1
Report.reportDir = reports/Thesis/Disaster
Report.report1 = MessageStatsReport

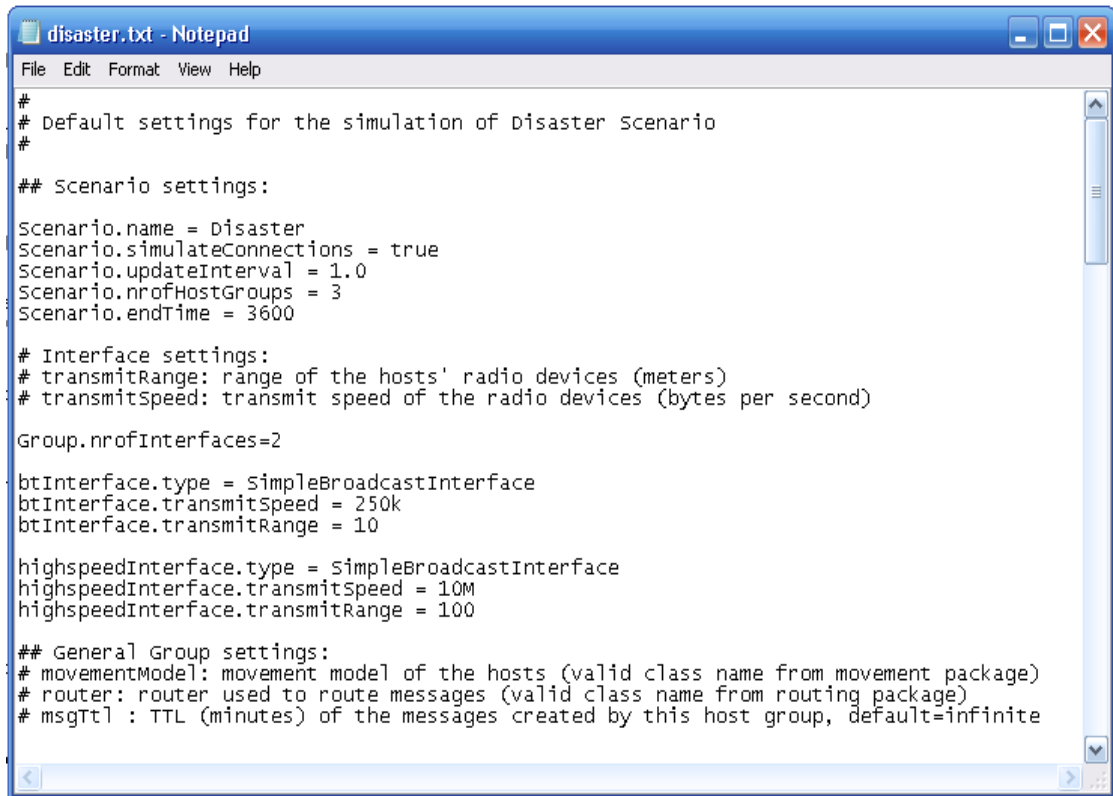
## Default settings for some routers settings
# Prophet Router settings

ProphetRouter.secondsInTimeUnit = 30

# EBR Router settings

EBRRouter.nrofCopies=20
EBRRouter.alpha=0.85
EBRRouter.updatePOPInterval=1.0
```

Figure A.16: Appendix ii: Configuration File for Scenario 2: Disaster



```
disaster.txt - Notepad
File Edit Format View Help
#
# Default settings for the simulation of Disaster Scenario
#
## Scenario settings:
Scenario.name = Disaster
Scenario.simulateConnections = true
Scenario.updateInterval = 1.0
Scenario.nrofHostGroups = 3
Scenario.endTime = 3600

# Interface settings:
# transmitRange: range of the hosts' radio devices (meters)
# transmitSpeed: transmit speed of the radio devices (bytes per second)

Group.nrofInterfaces=2

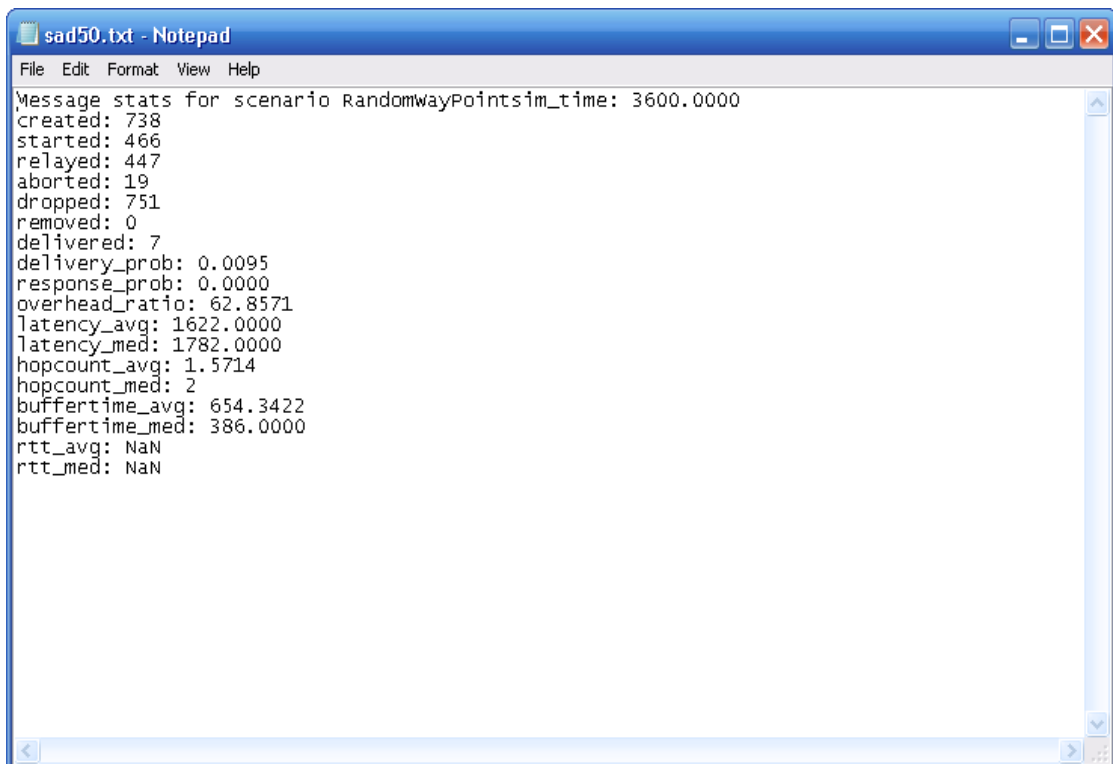
btInterface.type = SimpleBroadcastInterface
btInterface.transmitSpeed = 250k
btInterface.transmitRange = 10

highspeedInterface.type = SimpleBroadcastInterface
highspeedInterface.transmitSpeed = 10M
highspeedInterface.transmitRange = 100

## General Group settings:
# movementModel: movement model of the hosts (valid class name from movement package)
# router: router used to route messages (valid class name from routing package)
# msgTtl : TTL (minutes) of the messages created by this host group, default=infinite
```

Figure A.17: Appendix ii: Snapshot of Scenario 2 Configuration File

iv) SAMPLE OF REPORT FILE



```
sad50.txt - Notepad
File Edit Format View Help
Message stats for scenario RandomwayPointsim_time: 3600.0000
created: 738
started: 466
relayed: 447
aborted: 19
dropped: 751
removed: 0
delivered: 7
delivery_prob: 0.0095
response_prob: 0.0000
overhead_ratio: 62.8571
latency_avg: 1622.0000
latency_med: 1782.0000
hopcount_avg: 1.5714
hopcount_med: 2
buffertime_avg: 654.3422
buffertime_med: 386.0000
rtt_avg: NaN
rtt_med: NaN
```

Figure A.18: Appendix ii: Snapshot of MessageStatsReport File