

ABSTRACT

Title of Document: PERFORMANCE EVALUATION OF
DISRUPTION TOLERANT NETWORKS
WITH IMMUNITY MECHANISM AND
CODING TECHNIQUE

Jin Na Lee
Doctor of Philosophy, 2015

Directed By: Associate Professor Richard J. La,
Department of Electrical and Computer
Engineering

We examine the performance of a Disruption Tolerant Networks (DTNs) with an epidemic routing (ER) scheme with the coding technique and/or immunity mechanism under the various network environments. We are interested in the scenarios of opportunistic dissemination of large files. First, we study how the different implementations of the ER scheme perform in diverse network settings. We compare the performance of ER with its summary vector implemented as both a list and as a Bloom filter. Second, we examine how network coding affects the performance of the ER scheme. To this end, we investigate the performance of encoding-based routing (EBR), a variant of the ER scheme which uses random linear coding at source nodes. EBR is expected to mitigate what is commonly known as the coupon collector's problem, which arises when a large file is chopped into small fragments and then the fragments are disseminated throughout the network. We

compare this to the case where intermediate non-source nodes are allowed to create new linear combinations from the ones it already holds. Lastly, we evaluate the benefits of two different types of immunity mechanisms – one based on file ID and the other based on bundle ID – with not only the ER scheme but also two different EBR schemes in various network scenarios and settings. We also investigate the performance gain from compressing the immunity list.

By presenting and analyzing extensive simulation results, we provide information that could provide a guideline for employing each of the aforementioned techniques in routing schemes of interest in various network settings.

PERFORMANCE EVALUATION OF DISRUPTION TOLERANT NETWORKS
WITH IMMUNITY MECHANISM AND CODING TECHNIQUE.

By

Jin Na Lee

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Advisory Committee:

Associate Professor Richard J. La, Chair

Professor Armand Makowski

Associate Professor Charles Silio

Professor Gang Qu

Professor Sung Lee

© Copyright by
Jin Na Lee
2015

Acknowledgements

First and foremost, I would like to thank my Lord, who began this work and finally finished it. Completing this dissertation and my Ph.D. journey is a true blessing and God be the glory.

This dissertation would not have been finished without the help of my advisor Dr. Richard La. I would like to thank for his great guidance, continuous support and patience during my Ph.D. years. Also, I am grateful to all of my committee members for their valuable advices and encouragement as well as being on my committee: Dr. Armand Makowski, Dr. Charles Silio, Dr. Gang Qu, and Dr. Sung Lee. I am also thankful to my research mentors for their kindness, support, and guidance: Dr. Padma Mundur and Dr. Gregory Stein.

There are not enough words to describe how thankful I am to my family and friends for their tremendous support, love, encouragement, and prayers. I would like to thank my family first for their unconditional love and support throughout my whole life: dad Dr. Ho-In Lee, mom Ms. Hae-Jeung Lim, sisters Dr. Hannah Lee and Dr. Yoonnah Lee, brothers Kwangjin Lee, Dr. Sangwoo Shin and Dr. Norman Kim, and lovely nieces Christine Shin and Ydel Kim. Thank you and love you all! I would also like to thank my awesome friends for their love, prayers, help, support, and encouragement: Dr. Brenton Walker, Hyeonmi Kim, Sohl Han, Yoohee Kim, Uran Oh, Dongheon Ha, and Sungmin Eum. I would like to thank my BizFlow family for their support, patience, understanding, and prayers: Jae Ahn, Caffrey Lee, Peter Lee,

Kyungwon Kim, and others. I must also thank my family in Christ for their prayers and encouragement: Pastor Aaron Park, Kilya Park, Nara Kim, Jaeyul Kwon, Kenny Kim, Yoonsoo Lee, Youngin Lee, and my KBS family. Last but not least, I am eternally grateful to everyone who has prayed for me.

Table of Contents

Acknowledgements.....	ii
Table of Contents.....	iv
List of Tables.....	vi
List of Figures.....	vii
List of Abbreviations.....	xiii
Chapter 1: Introduction.....	1
1.1 Epidemic routing scheme.....	3
1.2 Coding technique.....	4
1.3 Immunity mechanism.....	7
1.4 Summary of key changes to the simulator and challenges.....	9
1.5 Organization.....	11
Chapter 2: Related work.....	14
2.1 Coding technique.....	14
2.2 Immunity mechanism.....	16
Chapter 3: Network scenarios.....	18
3.1 The ONE simulator.....	18
3.2 Performance metrics.....	20
3.3 Parametric scenario.....	20
3.3.1 Number of nodes.....	20
3.3.2 Transmission rate and range.....	21
3.3.3 Buffer capacity.....	22
3.3.4 Node speed.....	22
3.4 Mobility scenarios.....	22
3.4.1 Ferry Scenario.....	23
3.4.2 Manhattan in New York City Scenario.....	25
3.4.3 Island hopping within Istanbul, Turkey Scenario.....	26
3.4.4 Taxi Trace in San Francisco, CA Scenario.....	28
3.5 File traffic generation.....	29
3.6 Summary of simulation settings.....	31
4.1 Implementation Details.....	33
4.1.1 Description of the ER Scheme.....	33
4.1.2 File Handling and Scheduling Strategy.....	35
4.2 Summary Vector Implemented by the Bloom Filter.....	36
4.2.1 The optimal number of hash functions.....	37
4.3 Simulation Results.....	39
4.3.1 Ferry scenario.....	39
4.3.2 NYC scenario.....	43
Chapter 5: Encoding Based Routing Scheme.....	47
5.1 Encoding Based Routing with Source Coding.....	47
5.1.1 File Creation.....	48
5.1.2 Rank Calculation.....	53
5.1.3 Control Message.....	60
5.2 Encoding Based Routing with Network Coding.....	64
5.2.1 Implementation Details.....	64

5.2.2 Simulation Results	66
5.3 Benefit of Coding Technique.....	68
Chapter 6: Immunity Mechanism	80
6.1 <u>Bundle Based Immunity Mechanism</u>	81
6.1.1 Implementation Details	81
6.1.2 Simulation Results	82
6.1.3 Compression over the Immunity List	88
6.2 <u>UUID Based Immunity Mechanism</u>	91
6.2.1 Implementation Details	91
6.2.2 Simulation Results	92
Chapter 7: Conclusion.....	109
Bibliography	113

List of Tables

1.1	Summary of key changes to the ONE simulator.....	10
3.1	Mobility domain assignments on ferry nodes	25
3.2	Simulation settings	32
4.1	ADD and FDR of ER scheme with Bloom filter and list summary vector under the NYC scenario with the transmission range of 100m	45
4.2	ADD and FDR of ER scheme with Bloom filter and list summary vector under the NYC scenario with the transmission range of 50m.....	46
5.1	Statistics of the ferry mobility scenarios with the transmission range of 100m.....	71
5.2	Statistics of the taxi trace mobility scenarios	77
7.1	Guidance for summary vector configuration	110
7.2	Guidance for rank check configuration	110
7.3	Guidance for mobility scenario.....	111
7.4	Guidance for coding scheme in the ferry scenario.....	111
7.5	Guidance for coding scheme in the NYC scenario.....	112
7.6	Guidance for coding scheme in the Istanbul scenario.....	112
7.7	Guidance for coding scheme in the Taxi scenario.....	112

List of Figures

1.1	Organization of Chapter 4, 5, and 6.....	12
1.2	Combinations of the routing schemes and the approaches of interest	13
3.1	Screen shot of the ONE running a scenario with Istanbul map	19
3.2	Locations of stationary nodes and the mobility domains of ferry nodes in the ferry scenario.....	24
3.3	A snapshot of the locations of 30 nodes on the map of Manhattan in New York City, NY	26
3.4	A snapshot of the Istanbul scenario depicting the locations of the stationary nodes and the sub-areas	28
3.5	A taxi mobility trace in May 2008	29
4.1	FDR and ADD of the ER scheme with Bloom filter and list summary vector in the ferry scenario with 200kB/s of transmission rate and 50m of transmission range.....	40
4.2	Average contact time and total number of contacts in the ferry mobility scenarios.....	41
4.3	FDR and ADD of ER scheme with Bloom filter and list summary vector in the ferry scenario with 100kB/s of transmission rate and 50m of transmission range	42
4.4	File delivery delays of the F2-2 scenario by increasing order	43
4.5	ADD of ER scheme with Bloom filter and list summary vector under the NYC scenario.....	44
5.1	FDR of the EBR scheme varying the encoding weight in the ferry scenario when $N = 1000$	52
5.2	FDR of the EBR scheme varying the encoding weight in the ferry scenario when $N = 100$	52

5.3	FDR of the EBR scheme varying the encoding weight in the NYC scenario when $N = 1000$	53
5.4	FDR of the EBR scheme varying the encoding weight in the NYC scenario when $N = 100$	53
5.5	FDR and ADD of the EBR scheme with the Bloom filter and the list summary vector under the ferry mobility with light traffic congestion	58
5.6	FDR and ADD of the EBR scheme with the Bloom filter and the list summary vector under the ferry mobility with moderate traffic congestion.....	58
5.7	FDR and ADD of the EBR scheme with and without rank check at the relay nodes under the NYC mobility with file generation rate of 0.0005 file/s.....	59
5.8	FDR and ADD of the EBR scheme with and without rank check at the relay nodes under the NYC mobility with file generation rate of 0.00005 file/s.....	59
5.9	FDR and ADD of the EBR scheme with the Bloom filter and the list summary vector under the ferry mobility with light traffic congestion	62
5.10	FDR and ADD of the EBR scheme with the Bloom filter and the list summary vector under the ferry mobility with moderate traffic congestion.....	63
5.11	FDR and ADD of the EBR scheme with the Bloom filter and the list summary vector under the NYC mobility.....	64
5.12	FDR of the EBR with network coding scheme varying the re-encoding weight under the NYC scenario with 100MB files.....	68
5.13	FDR of the EBR with network coding scheme varying the re-encoding weight under the NYC scenario with 10MB files.....	68
5.14	FDRs of ER and two modes of EBR scheme under the Ferry scenario with light traffic.....	70

5.15	FDRs of ER and two modes of EBR scheme under the Ferry scenario with moderate traffic.....	71
5.16	FDRs and ADDs of ER and two modes of EBR scheme under the NYC scenario with 100MB files.....	73
5.17	FDRs and ADDs of ER and two modes of EBR scheme under the NYC scenario with 10MB files.....	73
5.18	FDRs and ADDs of ER and two modes of EBR scheme under the NYC scenario with 100MB files.....	73
5.19	FDRs and ADDs of ER and two modes of EBR scheme under the NYC scenario with 10MB files.....	74
5.20	FDRs and ADDs of ER and two modes of EBR scheme under the Istanbul scenario with 24 100MB files	76
5.21	Figure 5.21. FDRs and ADDs of ER and two modes of EBR scheme under the Istanbul scenario with 325 10MB files	76
5.22	FDRs and ADDs of ER and two modes of EBR scheme under the Istanbul scenario with 24 100MB files.....	76
5.23	FDRs and ADDs of ER and two modes of EBR scheme under the Istanbul scenario with 325 10MB files.....	77
5.24	FDRs and ADDs of ER and two modes of EBR scheme under the taxi trace scenario with 100MB files.....	79
5.25	FDRs and ADDs of ER and two modes of EBR scheme under the taxi trace scenario with 10MB files.....	79
6.1	FDRs of ER scheme with and without BBI under the ferry scenario when the network is moderately congested with large files.....	83
6.2	FDRs of ER scheme with and without BBI under the ferry scenario when the network is moderately congested with small files.....	84
6.3	FDRs and ADDs of ER scheme with and without BBI under the ferry scenario when the network is lightly congested with large files.....	84
6.4	FDRs and ADDs of ER scheme with and without BBI under the ferry scenario when the network is lightly congested with small files.....	85

6.5	FDRs of ER scheme with and without BBI under the NYC scenario.....	86
6.6	ADDs of ER scheme with and without BBI under the NYC scenario.....	86
6.7	FDR and ADD of the EBR scheme with and without the BBI in the ferry scenario with transmission rate of 200kB/s, range of 100m, file size of 100MB, and file generation 0.00005 file/s.....	87
6.8	FDR and ADD of the EBR scheme with and without the BBI in the NYC scenario with transmission rate of 200kB/s, range of 50m, file size of 100MB, and file generation 0.00005 file/s.....	88
6.9	FDR and ADD of the ER scheme with BBI that the compression technique is employed and not employed in the ferry scenario.....	90
6.10	FDR and ADD of the ER scheme with BBI that the compression technique is employed and not employed in the NYC scenario.....	90
6.11	FDR of the ER scheme with and without UBI under the ferry mobility when transferring 100MB files.....	93
6.12	FDR of the ER scheme with and without UBI under the ferry mobility when transferring 10MB files.....	93
6.13	FDR and ADD of the ER scheme with and without UBI under the NYC mobility when transferring 100MB files.....	94
6.14	FDR and ADD of the ER scheme with and without UBI under the NYC mobility when transferring 10MB files.....	94
6.15	FDR and ADD of the ER scheme with and without UBI under the NYC mobility when transferring 10MB files in a lightly congested network...	94
6.16	FDR of the ER scheme with and without UBI under the Istanbul mobility with two different transmission rates (Left: 200kB/s, Right: 500kB/s).....	95
6.17	ADD of the ER scheme with and without UBI under the Istanbul mobility with light traffic.....	96
6.18	FDR and ADD of the ER scheme with and without UBI under the taxi trace when transferring 100MB files.....	97
6.19	FDR and ADD of the ER scheme with and without UBI under the taxi trace when transferring 10MB files.....	97

6.20	FDR and ADD of the source coding enabled EBR scheme with and without UBI under the ferry mobility.....	98
6.21	FDR of the source coding enabled EBR scheme with and without UBI under the ferry mobility when transferring different size of files (Left:100MB, right: 10MB).....	98
6.22	FDR and ADD of the source coding enabled EBR scheme with and without UBI under the NYC mobility when transferring 100MB files....	99
6.23	FDR and ADD of the source coding enabled EBR scheme with and without UBI under the NYC mobility when transferring 10MB files.....	99
6.24	FDR and ADD of the source coding enabled EBR scheme with and without UBI under the NYC mobility and light traffic.....	99
6.25	FDR and ADD of the source coding enabled EBR scheme with and without UBI under the Istanbul mobility when transferring 100MB files	100
6.26	FDR and ADD of the source coding enabled EBR scheme with and without UBI under the Istanbul mobility when transferring 10MB files..	100
6.27	FDR and ADD of the source coding enabled EBR scheme with and without UBI under the taxi trace mobility when transferring 100MB files.....	101
6.28	FDR and ADD of the source coding enabled EBR scheme with and without UBI under the taxi trace mobility when transferring 10MB files	101
6.29	FDR and ADD of the network coding enabled EBR scheme with and without UBI under the ferry mobility.....	102
6.30	FDR and ADD of the network coding enabled EBR scheme with and without UBI under the NYC mobility.....	103
6.31	FDR and ADD of the network coding enabled EBR scheme with and without UBI under the Istanbul mobility.....	103
6.32	FDR and ADD of the network coding enabled EBR scheme with and without UBI under the taxi trace mobility.....	103
6.33	FDR of three routing schemes with UBI enabled under the ferry scenario when transferring 100MB files.....	105
6.34	FDR of three routing schemes with UBI enabled under the ferry scenario when transferring 10MB files.....	105

6.35	FDR and ADD of three routing schemes with UBI enabled under the NYC scenario with moderate traffic congestion.....	106
6.36	FDR and ADD of three routing schemes with UBI enabled under the NYC scenario with light traffic congestion.....	106
6.37	FDR and ADD of three routing schemes with UBI enabled under the Istanbul scenario with moderate to high traffic congestion.....	107
6.38	FDR and ADD of three routing schemes with UBI enabled under the Istanbul scenario with light to moderate traffic congestion.....	107
6.39	FDR and ADD of three routing schemes with UBI enabled under the taxi trace scenario when transferring 100MB files.....	108
6.40	FDR and ADD of three routing schemes with UBI enabled under the taxi trace scenario when transferring 10MB files.....	108

List of Abbreviations

ADD	Average Delivery Delay
BBI	Bundle Based Immunity
BIM	Bundle Immunity Message
DTN	Disruption Tolerant Network
ER	Epidemic Routing
EBR	Encoding Based Routing
FDR	File Delivery Ratio
JOSM	Java OpenStreetMap
LNC	Linear Network Coding
LZW	Lempel-Ziv-Welch
RMBM	Routed Map-Based Movement
TTL	Time To Live
SPMBM	Shortest Path Map-Based Movement
UBI	UUID Based Immunity
UIM	UUID Immunity Message
UUID	Universally Unique Identifier
WKT	Well Known Text

Chapter 1: Introduction

Disruption tolerant networks (DTNs) are networks where link disruption may occur due to the sparsity of mobile nodes, the limit of wireless radio range or energy resources, noise, and other factors. Unlike traditional multi-hop wireless networks, DTNs are disconnected or partitioned most of the time. As a result one cannot assume the existence of contemporaneous end-to-end connections between sources and their intended destinations. Interplanetary communication, military ad hoc networks, wildlife tracking sensor networks, and vehicular ad hoc networks are several example use cases for DTNs.

Because of their intermittent connectivity, these types of challenged networks cannot be supported by the traditional computer network solutions (e.g., the Internet) that assume the availability of an end-to-end route between a source and its destination(s). Instead, nodes must exchange messages (“bundles” in the DTN literature) in an opportunistic manner when they meet each other in order to deliver messages to their destinations. Thus, researchers began to work on establishing a separate framework [2, 7] rather than extending the traditional network. A number of routing solutions for DTNs also have been proposed to cope with the frequent and unpredictable connectivity interruptions. Epidemic routing (ER) [29] is one of these routing solutions, which is based on the idea of replication. To maximize the probability of successful deliveries, the ER scheme produces many copies of every message. Each relay node forwards each message to every other node it meets. This creates a lot of redundancy in the network, but at the price of high resource requirements.

In this dissertation, we are specifically interested in the problem of disseminating large files in a DTN environment. When a file is too large to be communicated in a single contact, it must be first chopped into fragments that are small enough to be transferred to other nodes within a contact. Then the fragments are disseminated throughout the network, and the destination is required to collect a copy of each fragment and reassemble the original file. In order to enhance the performance of data routing in this kind of scenario, a form of linear network coding (LNC) has been proposed and studied.

In order to study the benefits of coding techniques for transmitting the sets of fragments in diverse network environments, we implemented an encoding based routing (EBR) scheme. EBR employs the LNC scheme on the fragments to be transmitted. It has two modes regarding where coding operations are employed. One is the source-coding mode that allows only source nodes to create new encodings, while the other is the network-coding mode that allows relay nodes to create new encodings from the encodings they already hold. We compare the performance of EBR to ER scheme under various network environments, and investigate the appropriate network settings that are suitable to employ these coding techniques.

In addition, we investigate the use of an immunity mechanism in both of routing schemes, ER and EBR scheme. The immunity mechanism is designed to stop the distribution of bundles that have already reached their destination(s). The traditional immunity mechanism works with individual bundles; when a bundle reaches its destination, the destination node releases an immunity message telling all other nodes that they can drop that particular message. We also implemented a new

immunity mechanism that operates based on the delivered file. We examine the performance gains due to the immunity mechanisms in different network settings by evaluating the performance of the ER and EBR schemes both with and without the immunity mechanisms.

As no solution is optimal in every situation, both coding techniques and the immunity mechanisms have different benefits in different network scenarios. With help of extensive simulation results, we investigate how these techniques bring different performance gains in different network settings. A goal of this dissertation is to provide a helpful guideline for utilizing these two techniques and choosing the suitable routing schemes for various network environments.

1.1 Epidemic routing scheme

In order to deal with the expected lack of contemporaneous end-to-end paths through the network, the routing algorithms in a DTN must be compatible with the opportunistic “store-carry-forward” model, which relies on the mobility of nodes to physically move data through a network. The most common technique used to achieve reliable data delivery is replication. That is, sending identical copies of a bundle over multiple relay nodes at the expense of high resource requirements and redundant transmissions of same bundles in the network.

The ER scheme is a replication-based scheme [29]. It propagates data rapidly at every node-node contact, similar to the way an epidemic of disease might spread. Whenever a bundle-carrying node meets a new node that does not have a copy of the bundle, the carrier node is said to “infect” the new node by forwarding a copy of the bundle. Then, the newly infected node behaves in the same way to other non-infected

nodes. The bundle is finally delivered to the destination when the destination first encounters an infected node.

While this routing algorithm floods the entire network with multiple copies of a bundle, it mitigates the amount of unnecessary data transmission by maintaining a data structure, called a summary vector, at each node. A node's summary vector indicates which bundles are stored in its local buffer. Whenever two nodes get in contact, they exchange their summary vectors first. Then, they figure out what bundles are not stored in the buffer of the other node, and forward only those bundles that the neighboring node does not already have. This has the same effect as unrestrained flooding of messages; given sufficient contact time, both nodes will end up having the same list of bundles.

We implemented the summary vector for the ER router using two different data structures. One is a simple list of the bundle identifiers (IDs), and the other is a Bloom filter. A Bloom filter is a space-efficient probabilistic data structure used for the membership test with risk of the false positives. The idea of utilizing a Bloom filter for the summary vector was proposed in [29] to reduce the space overhead associated with the summary vector. We examine the performance of the ER scheme with these two different implementations over various network settings when nodes transmit large files (sets of fragments) to others.

1.2 Coding technique

The use of coding has been considered as a promising technique that can improve the performance of communication systems and networks [21, 30]. From erasure coding [17, 19, 30] to many forms of network coding such as random linear

network coding [15, 21, 33], various coding techniques have been explored in the DTN community as well. We study how coding solves the problem of disseminating large files in an opportunistic network, where the coupon collector's problem would be an issue if plain fragmentation were used [6].

When LNC [14] is employed on a large file to be transferred in the EBR scheme, the file is first broken into a fixed number of equal sized blocks, or **chunks**. These chunks are then used to generate **encodings**, each of which is a linear combination of the chunks. Along with each encoding, an **encoding vector** that specifies the list of coefficients used to generate the encoding from the chunks. The source generates distinct encodings by taking different linear combinations of the chunks. The source transmits a stream of distinct encodings instead of the original chunks from the file. The destination can recover the file when it has collected a sufficient number of linearly independent encodings. Regardless of which encodings were received by the destination, typically the number of encodings required to reconstruct the file is close to the number of chunks used by the source with high probability. Compared to sending out the original chunks from the source, which may suffer from the coupon collector's problem, the file would be recovered more quickly at the destination when coding is employed.

We implement a routing scheme utilizing this coding technique, which we name the EBR scheme. It has two different modes depending on where the coding operation is performed. The first mode is the source coding that is built on top of the ER scheme. In this mode, only source nodes generate encodings and the relay nodes only need to employ the ER scheme (i.e., forward a copy of the encodings to other

nodes after exchanging the summary vector first). The other mode is the network coding in which the coding can be performed at the relay nodes as well as the source node. Relay nodes generate new encodings from the encodings they are carrying and transmit them rather than just forwarding what they have in the buffer. With this mode, nodes do not exchange the summary vectors or check the bundle list to select which bundle to send.

Besides investigating the benefits of both coding techniques in the ER scheme in different network environments, we also study each of the coding schemes in detail for better understanding of these techniques. For the EBR with source coding scheme, we examine the effects of coding weight on the performance. When a new encoding is generated from either original chunks or existing encodings, the weight is the number of items summed together to create the new encoding. Using higher weight has the potential to improve the statistical performance of the code, but requires more internal processing at the nodes creating the encodings. Also, as we did with the ER scheme, we study how different implementations of the summary vector affect the performance of the EBR scheme. Furthermore, we examine what we call the “*rank check*” feature. With this feature, every node maintains an encoding matrix, and figures out whether a newly received encoding is linearly independent of the encodings the node already holds in its buffer. If an encoding is redundant (i.e., not linearly independent from its existing inventory), the receiving node drops it. Also, nodes stop receiving encodings if they reach the full rank for the file. We compare the performance of the EBR scheme with and without this feature. For the EBR with

network coding scheme, we examine the effect of re-coding weight on the performance.

Eventually, we evaluate the performance of these two modes of EBR scheme by comparing the result of ER scheme under diverse network settings. We study how the coding technique performs in different network environments and which network setting would be suitable for this technique to be adopted.

1.3 Immunity mechanism

An immunity mechanism [3, 15, 27, 33] has been proposed in order to mitigate the storage requirement of replication-based routing protocols. The immunity mechanism is a means of reducing additional circulation of the unwanted copies of delivered bundles in the network. Through this mechanism, the copies of delivered bundles are removed from the buffer at the nodes that become immunized to the delivered bundles (i.e., notified of their delivery). It has been shown that this immunity mechanism improves the performance of existing routing protocols in many scenarios by reducing resource consumption, while at the same time increasing the bundle delivery ratio and decreasing the delivery latency.

Since we are interested in transferring files using collections of small bundles, we implemented two different types of immunity mechanisms: one based on the file and the other based on bundle. The immunity mechanism based on the file generates a new immunity message when a whole file is successfully delivered to the intended destination. Because the immunity message associated with the delivered file contains the universally unique identifier (UUID) assigned to the file, we refer to this immunity mechanism as UUID-based immunity (UBI). The other immunity

mechanism operates as each individual bundle is delivered to the destination. In this immunity mechanism, the immunity message includes the unique ID of the delivered bundle, and we call it bundle-based immunity (BBI). We evaluate the benefits of these immunity mechanisms under various scenarios not only for ER scheme but also for the two different modes of EBR scheme.

While the immunity mechanism is intended to facilitate nodes to better utilize the limited contact times as well as the buffer space, we demonstrate that it could hurt the performance as the size of immunity list increases. Especially, since the BBI generates the immunity messages per the delivered bundle, the number of immunity messages in the network could grow very large. If the size of immunity list is very large, it is possible for nodes to waste most of contact time for exchanging the immunity messages. As a solution to this issue, we propose to compress the immunity list to reduce overhead of the control messages. We study how the compression over the BBI benefits the ER.

1.4 Summary of key changes to the simulator and challenges

We use the ONE simulator [11] for simulations. The simulator has been developed for DTN research. However, it does not include every protocol for DTNs. Furthermore, many features that we are interested in are not supported in the original ONE simulator. Therefore, we made several major modifications and added new features to the simulator in order to investigate the benefits from the coding scheme and the immunity mechanism. Key changes to the simulator are summarized in Table 1.1.

Component	Description of modification or new implementation
Link sharing scheme between nodes	When two nodes get in contact, a link is established between them. They share the link to exchange bundles with each other. However, the link sharing rule implemented in the original ONE simulator results in unfair communication between nodes. A node that gets on the channel first monopolizes the link until it runs out of bundles to send. In order to balance the nodes' communication, we modified the code so that nodes take turns.
Link scheduling	No link scheduling mechanism is implemented in the ONE simulator. When a node has links established with multiple nodes at the same time, it selects one among the links and communicates only on the link. Therefore, we added general link scheduling schemes for multiple links; Round Robin scheduling (default), and random scheduling.
Bundle scheduling	In addition to the random and FIFO bundle scheduling schemes that are supported in the ONE simulator, we implemented Round Robin scheduling.
Control message	A control message is a special bundle that is exchanged between nodes before the real data transfer as a part of a routing protocol. Unlike normal bundles, it should not be stored in the buffer. However, we still need to account for the transmission time of a control message. Since the ONE simulator does not support control messages, we added this to our implementation.
Summary vector exchange	The ER scheme implemented in the ONE simulator works different from the original ER protocol proposed in [29]. First of all, nodes do not exchange summary vectors. Instead, they send every bundle in the buffer like Flooding

	[27]. When a node transfers a bundle that already resides in the other node's buffer, the simulator does not count the time used for sending the bundle and the other node drops it (i.e., denies the transfer). However, this implementation does not take account of the overhead of exchanging the summary vector, which could affect the performance depending on network settings. Therefore, we modified the code to exchange summary vectors first when nodes meet each other. Furthermore, we implemented it with two different data structures – list and Bloom filter.
Bundle fragmentation	We newly implemented the proactive bundle fragmentation using event generator. Each fragment becomes a single bundle with a UUID representing the original bundle or file and its index stored in the extension block. Also, every node can recover the original large bundle when it collects all of fragment bundles.
EBR scheme	Coding scheme is added to the ONE simulator. Nodes can generate encoding bundles and decode them as well. There are two modes in the EBR depending on which nodes perform coding operations.
Immunity mechanism	The original ONE simulator does not support any immunity mechanism. Two different immunity mechanisms - BBI and UBI - are implemented with both ER and EBR. Immunity mechanisms are configurable.
Report module	Necessary report modules for analyzing the results are newly added to the simulator.

Table 1.1 Summary of key changes to the ONE simulator

Besides the code changes to the simulator itself, we also developed programs and scripts for the simulator, which are related to the mobility generation and result analysis.

There are several major challenges we ran into while we were implementing these components. First, the routing protocols provided in the simulator are not implemented as proposed in terms of node behavior. For example, nodes do not exchange the summary vector in the ER scheme that comes with the simulator. In

order to simulate the routing protocols according to their original design, we implemented routing schemes of our interest from the scratch. Second, many aspects of the simulator are oversimplified, which could result in several issues that could significantly affect the simulation results. Therefore, we made necessary modifications to the simulator in order to create realistic settings.

1.5 Organization

We focus on investigating the benefits from two techniques, namely the immunity mechanism and the coding technique, under diverse network environments when transmitting large files. The goal of our study is to provide a guideline for choosing the right combinations or routing scheme depending on the network settings. To this end, we implement routing schemes by implementing coding techniques and immunity mechanisms as an extension to the ER scheme. Then, we investigate the performance of three routing schemes: the ER scheme and EBR scheme with source coding and network coding, with and without the immunity mechanisms through extensive simulations in diverse network settings.

The rest of the dissertation is organized as follows: Chapter 2 introduces the related literature. In Chapter 3, we introduce the ONE simulator and describe the network scenarios used for the research. From Chapter 4 to Chapter 6, we study the ER scheme, the coding technique, and the immunity mechanisms. Then, the conclusion is follows in Chapter 7. Figure 1.1 depicts the details of each chapter, and Figure 1.2 shows the combinations of the routing scheme and the different implementations and/or improvement approaches to be evaluated in this dissertation.

Note on terminology: Here are some of terms used throughout the dissertation. *Chunk* denotes one of the file fragments to be transferred. *Encoding* is the result of the coding operation on chunks (i.e., a linear combination of the chunks). *Encoding vector* is a vector of the coding coefficients, which contains the information of which chunks are used for generating the corresponding encoding. *Innovative encoding* means that the encoding is linearly independent of other encodings in a set. That is, an encoding is innovative if adding the corresponding encoding vector to an existing set of encoding vectors increases the rank of the set. On the other hand, *redundant encoding* is an encoding that is not innovative. That is, its encoding vector is not linearly independent of the existing set of encoding vectors. *Re-encoding vector* is a vector that is created when a relay node performs re-coding on encodings.

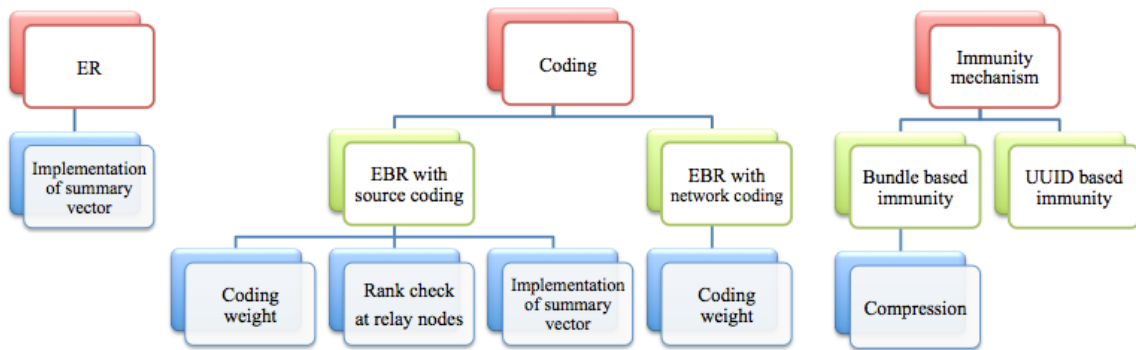


Figure 1.1. Organization of Chapter 4, 5, and 6

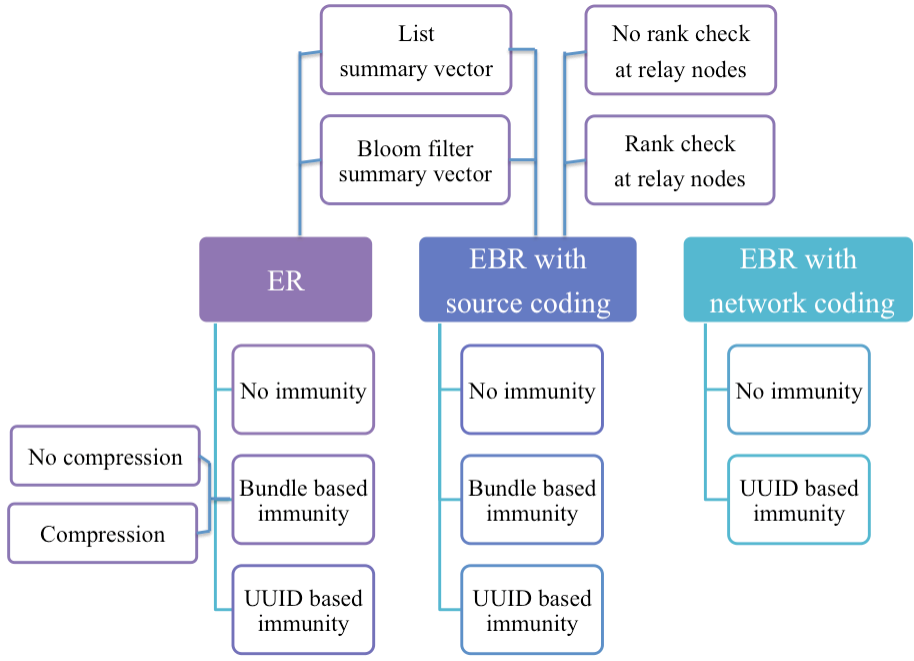


Figure 1.2. Combinations of the routing schemes and the approaches of interest

Chapter 2: Related work

In this chapter, we introduce the existing research related to the techniques of interest, opportunistic network coding and the immunity mechanism. Then, we discuss the limitations of these works with the motivation of this dissertation.

2.1 Coding technique

Assume that there is a source that wants to send a very large file (e.g. video) to a destination in a DTN. The file must be chopped into small pieces that are transferable during a single contact. Thus, the source sends out a large number of chunks (i.e., small pieces) containing a small part of the file to deliver. The destination needs to receive all of those chunks to reconstruct the original file. When a flooding protocol is used, collecting all of the distinct chunks can be modeled as a coupon collector's problem [6] with the assumption of the independence of message arrivals at the destination. It takes very little time to collect the first few chunks, while it takes a long time to collect the last few chunks. Hence, the delivery delay could be very large. The expected number of chunks needed at the destination is $O(N \log N)$, where N is the number of chunks generated by the source [6]. A natural question is “How can we improve the performance of delivery delay in this situation?”

Coding techniques have been considered for improving the performance of DTN routing schemes. Existing work [16, 33, 35] has shown that network coding [1] improves the performance of average delivery delay over a set of messages with a

relatively small amount of overhead in many scenarios. Gkantsidis and Rodriguez [8] study the effectiveness of using network coding for the distribution of large data. They also investigate the performance of source coding, which allows coding operations only at the source. Their work suggests that network coding improves download rates compared to source coding and no coding (plain fragmentation) in a heterogeneous network. Wang et al. [30] propose a coding-based routing algorithm that adapts the source coding approach to a simple replication-based algorithm. They show that the source coding improves the worst case delay.

Going back to the situation of interest, sending a large file in a DTN can be converted to a case of transferring a set of messages, in which coding technique could be used as an enhancement to the existing routing protocols. Then, how much better does coding technique perform as compared to replication, and in what particular network scenarios? Would the coupon collector's problem be mitigated at all by the coding technique? If so, when does it deliver the most benefits? Throughout this research, we investigate the benefits of coding when sending large files in a DTN, trying to find an answer to these questions.

The main difference from the previous studies of coding technique in DTNs would be the range of investigation with respect to network settings. Although all of the existing works investigate the benefits of coding given a type of challenged network, there has been no research considering multiple network characteristics. For instance, the node mobility examined in [35] for studying the benefits of random linear coding is only the random waypoint model. However, we evaluate the performance of the routing protocol with and without coding in different network

settings, in order to see when and how much the coding improves or hurt the performance. The results will provide insights towards a better understanding of utilizing the coding technique when transferring large files in DTNs.

2.2 Immunity mechanism

Haas and Small [9] discuss the impact of discarding obsolete information in the context of an infostation model for sensor network applications. They propose and study several strategies (called IMMUNE, IMMUNE_TX, and VACCINE) that discard a packet when it is delivered, but keeps an identifier (ID) of the packet which they call an “antipacket”. Each strategy has a different method for using antipackets. Another related study is the work by Zhang et al. [34] on ER and its variants. They study the performance of basic ER and its variations with the immunity mechanisms introduced in [9]. In addition, Matsuda and Takine [20] analyze the performance of (p, q)-ER with VACCINE. Mundur et al. [18] also propose an algorithm called ER with immunity, and study the performance improvement over the ER scheme when the buffer constraint exists.

All of these studies except [18] are based on a mathematical model, either a Markov chain model [9, 20] or ordinary differential equations (ODEs) [34]. In addition, there is no existing study considering the scenarios where a set of fragments needs to be transmitted. Furthermore, these researches have limited performance evaluation in different network settings. Just as no single routing algorithm fits perfectly in every situation, the immunity mechanism(s) may also not be able to improve the performance in every network environment. Depending on network characteristics, mobility scenario, routing protocol, and so forth, the benefit of the

immunity mechanism may vary to a large extent. However, all of the existing works study the immunity mechanism with a specific network setting. For example, the mobility scenario chosen in [18] is only random waypoint, which is not realistic for most DTN applications. Throughout this dissertation we investigate the performance under multiple sets of varying network settings.

Moreover, existing studies regarding the immunity mechanism do not consider its overhead such as bandwidth consumed by the immunity mechanism.

Chapter 3: Network scenarios

In this chapter, we describe the various network settings we used for the performance evaluation. In order to thoroughly investigate the benefits of each approach of interest, we carried out an extensive simulation-based study in a number of different network settings. For each network scenario, except for the ones using real trace data, we generate 10 runs with random initial seeds and take the average.

This chapter describes the network environments we consider. The rest of this chapter is organized as follows: Section 3.1 briefly introduces the ONE simulator, which is the tool we use for the simulations. The performance metrics that we are interested in are presented in Section 3.2. Section 3.3 specifies the parameters and the values we chose for creating diverse network scenarios with other settings such as mobility scenarios and traffic generations. Section 3.4 describes four different mobility scenarios, each of which has distinguishing characteristics from other ones. Settings of the traffic generation scenarios are presented in Section 3.5. The summary of network settings is shown in a table in Section 3.6.

3.1 The ONE simulator

The ONE simulator [11] is a widely used simulation tool that is specifically designed for evaluating DTN routing protocols. With this simulator, we can construct various network settings relatively easily. One of its attractive features is the map and mobility feature. It allows using real maps (e.g., Google map) and the mobility scenarios based on the map. Figure 3.1 shows the ONE simulator running with the Istanbul city map. It supports not only programmed mobility scenarios such as

random movements on the map, but also supports the real traces collected by field tests (e.g. UMass DieselNet [4]) by importing the trace data into the simulated environments. In this dissertation, we explore both of these types of mobility patterns, the map-based mobility, and the real traces for the performance evaluation.

As stated in Section 1.4, the ONE simulator does not come with all of routing protocols proposed in the research fields or other features. Thus, we made necessary modifications to the simulator in order to study the techniques of interest. Details of the implementations will be presented in relevant Chapters.

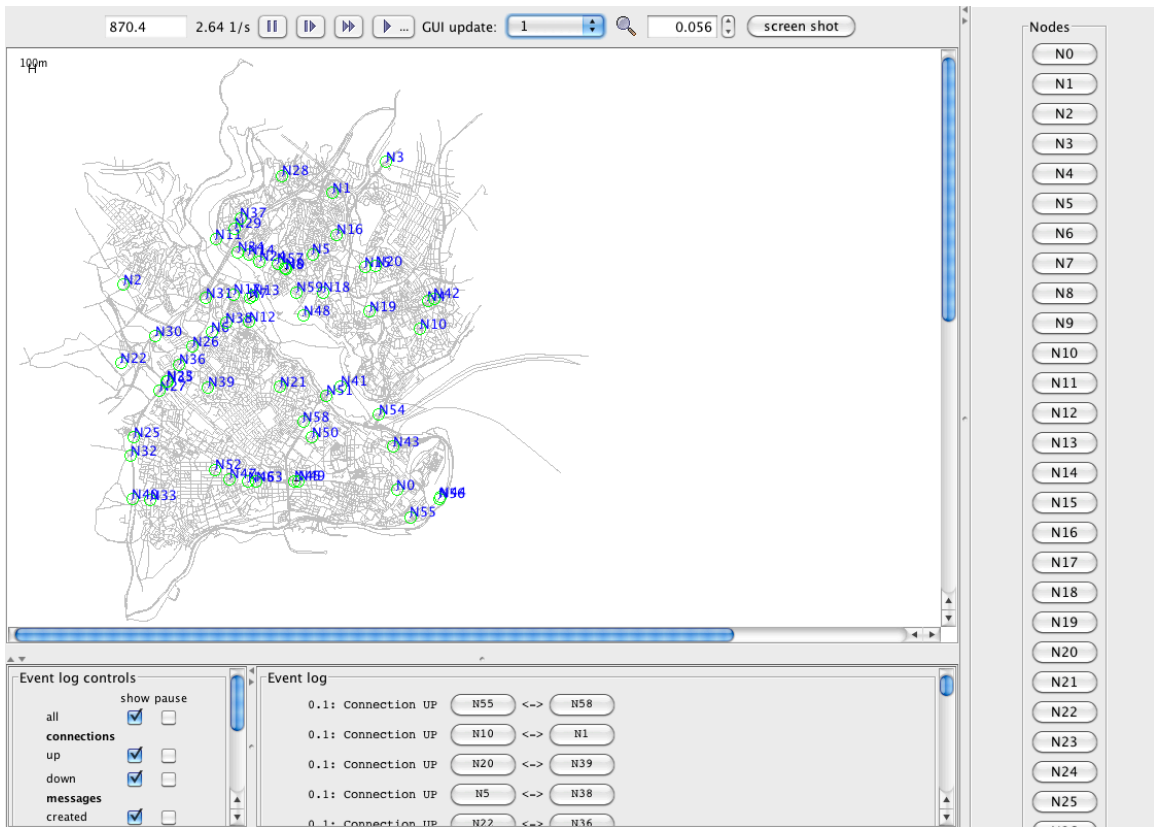


Figure 3.1. Screen shot of the ONE running a scenario with Istanbul map

3.2 Performance metrics

For studying each of the approaches of interest, we will focus on the following performance metrics. First, we are interested in evaluating the file delivery ratio (FDR) that is the fraction of files successfully delivered to their destinations. When we say that a file is delivered, it means that the destination received every chunk of the file or enough encodings to recover the file (i.e., the number of linearly independent encodings is the same as the number of chunks of the file). Second, we will compute the average delivery delay (ADD) for transmitting files, which is the time between when the source starts to generate encodings or to create chunks and when the destination finishes the reconstruction of the file by collecting enough encodings or all of chunks.

3.3 Parametric scenario

As we mentioned earlier, performance could vary widely depending on different network settings even with same technology or identical routing scheme. Besides the mobility scenarios and the message traffic generation scenarios, there are parameters that are important when setting up a network environment. In this section, we explain the parameters we consider along with the reason that they matter respect to the performance.

3.3.1 Number of nodes

This represents the node density, which directly affects the network connectivity. The connectivity of the network critically affects the performance

metrics such as the delivery ratio and delivery latency. In particular, the delivery delay will be longer as the network becomes sparser.

We vary the number of nodes not only over different mobility scenarios, but also with same mobility scenario in order to see how the node density affects the performance in various network environments.

3.3.2 Transmission rate and range

The properties of the node include the transmission range and rate.

Transmission range also affects the network connectivity. When the transmission range of nodes gets larger, the network is connected better with the same number of nodes. This means that the chance that nodes contact another node becomes higher. As a consequence, the probability that messages are delivered to their destination increases.

Transmission rate means how fast nodes transmit data to another node. It affects the number of bundles that can be transferred to the other node during a contact. Obviously, the delivery ratio and the delivery delay will be affected by the transmission rate.

In this dissertation, we assume that nodes communicate with each other using Wi-Fi, which is the most popular local area wireless technology. Even though modern Wi-Fi using extensions to IEEE 802.11 allows maximum data rates higher than 100 megabit per second (e.g. 802.11n), we set the transmission rate much lower because in practice goodput tends to be much lower than the max transmission rate. Depending on the scenario, we choose the rates from 100 kilobyte per second (kB/s) to 500 kB/s accordingly. Also, for the transmission range, we assume that all of nodes

are moving around outdoors. We set the transmission range to 50m, 100m, and 150m as we explore different connectivity of each mobility scenario.

3.3.3 Buffer capacity

Buffer capacity is one of the most important resource constraints in DTNs. Depending on the buffer size, the stay time of bundles in the buffer varies. If bundles are dropped from the buffer due to overflow, the delivery ratio could suffer. In recent years, however, storage capacities have increased considerably, and this is not a restrictive constraint anymore. Therefore, we do not consider buffer capacity in this dissertation, and assume that there are no bundle drops due to buffer congestion.

3.3.4 Node speed

How fast nodes are moving affects the network dynamics. When nodes move, the network connectivity also changes according to their location changes. As nodes move faster, the probability that nodes meet another node in a limited time gets increased, while the contact time between nodes decreases. We set different node speeds and the pause time in different mobility scenarios based on what we believe are realistic speed of vehicles.

3.4 Mobility scenarios

As we mentioned before, the ONE simulator supports the map-based mobility. Utilizing this feature of the simulator, we construct three different mobility scenarios for the performance evaluation. As the ONE simulator understands map data in Well Known Text (WKT) format, we generate map data files by exporting the real-world map using Java OpenStreetMap Editor (JOSM)[36] for the first two mobility

scenarios. In addition, we created a simple map using OpenJUMP [37], a Geographic Information System (GIS) program. For the last mobility scenario, instead of the map-based mobility, we use real trace data that are processed to be readable by the ONE simulator.

With map-based mobility, nodes move only on paths and routes defined in the map data. We use the Shortest Path Map-Based Movement (SPMBM) for the first two mobility scenarios, and the Routed Map-Based Movement (RMBM) for the third scenario. With SPMBM, nodes are initially placed at random locations. Then, each node selects a random point on the given map and moves towards the chosen point following the shortest route decided by the Dijkstra's shortest path algorithm. With RMBM, nodes follow pre-determined routes. At the beginning, nodes are placed somewhere on the route assigned to them. Then they select a next point on the route, and start moving to the target point along the route. With both of map-based mobility schemes, nodes move at a uniformly assigned speed among the pre-configured speed range. When the node reaches its target point, it pauses for a random amount of time uniformly distributed over pre-configured pause time, selects a new next point and a new speed, and repeats the same process.

3.4.1 Ferry Scenario

Ferry mobility scenarios are very popular in the DTN research community, where a set of nodes called ferries are responsible for carrying messages (i.e., bundles) for all nodes in the networks. With this ferry scenario, we evaluate the performance under a simple topology with low node density where not all of nodes meet each other.

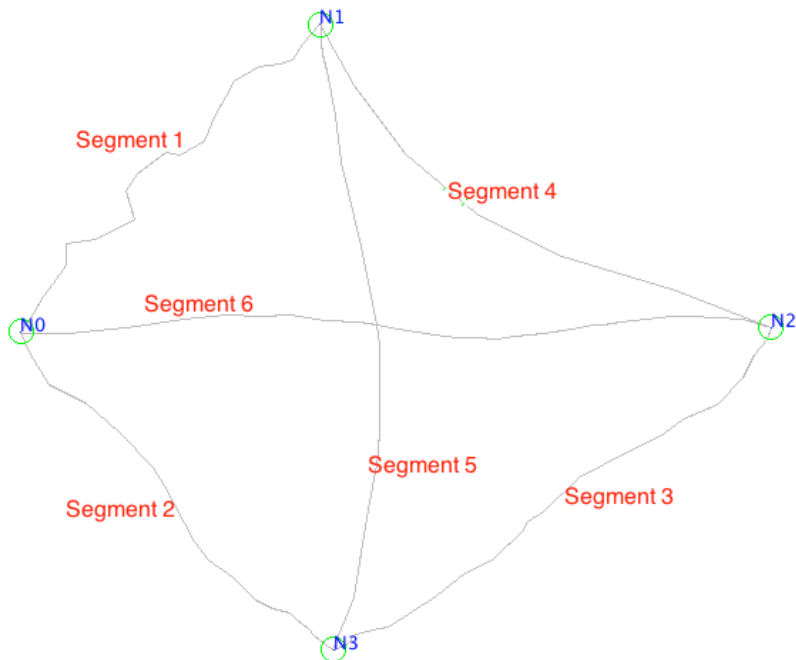


Figure 3.2. Locations of stationary nodes and the mobility domains of ferry nodes in the ferry scenario

We generate a simple diamond shape map using OpenJUMP as shown in Figure 3.4. At each vertex of the diamond, there is a stationary node. Every ferry node is moving along the pre-assigned routes. Since not only the number of ferries, but also the domains of ferries' mobility could affect the performance, we carry out multiple combinations of ferry routes as shown in Table 3.1. Columns under 'Segments on which each ferry moves' show the pre-defined routes for each ferry. For example, second column of the third row presents that three ferries move on the segments $\{1, 5\}$, $\{3, 5\}$, and $\{6\}$, respectively.

In the ferry scenario, there is one source and one destination. N0 is the source, and N2 is the destination in Figure 3.4. Ferry nodes move at the speed given by a random variable uniformly distributed over $[10, 30]$ mile per hour (mph). Their pause

time is given by random variables uniformly distributed over [10, 120] seconds. The minimum pause time is larger than other mobility scenario considering that ferries tend to stay longer for loading and unloading data when they meet another node.

Total simulation time is a week as well.

Number of ferries	Scenario name	Segments on which each ferry moves
2	F2-1	{1, 5}, {3, 5}
	F2-2	{1, 2}, {3, 4}
3	F3-1	{1, 5}, {3, 5}, {6}
	F3-2	{1, 2}, {3, 4}, {6}
6	F6-1	{1, 2}, {3, 4}, {5}, {6}, {1, 3, 5}, {2, 4, 5}
	F6-2	{5}, {6}, {1, 4}, {2, 3}, {1, 3, 6}, {2, 4, 6}

Table 3.1. Mobility domain assignments on ferry nodes for each scenario

3.4.2 Manhattan in New York City Scenario

In this scenario, nodes are moving on the map of Manhattan in New York, NY, which is generated using the JOSM editor as described above. Figure 3.3 shows the snapshot of the simulator running the Manhattan scenario with 30 nodes.

The Manhattan scenario represents a random movement in an urban area. Within the small area, nodes keep moving slowly and communicating with each other. Node speeds are given by independent random variables uniformly distributed over [1.5, 3.5] mph to model pedestrian mobility. In this scenario, nodes are homogeneous and move according to the same mobility scheme, SPMBM described earlier, with the same speed range and pause time range. Pause times are also random variables uniformly distributed over [0, 120] seconds. Furthermore, every node can generate files destined to any node in the network (i.e., every node can be a source or destination of files).

In the Manhattan scenario, we vary the number of nodes from 30 to 60, and then to 90. We run the simulations for a week or 604800 seconds.

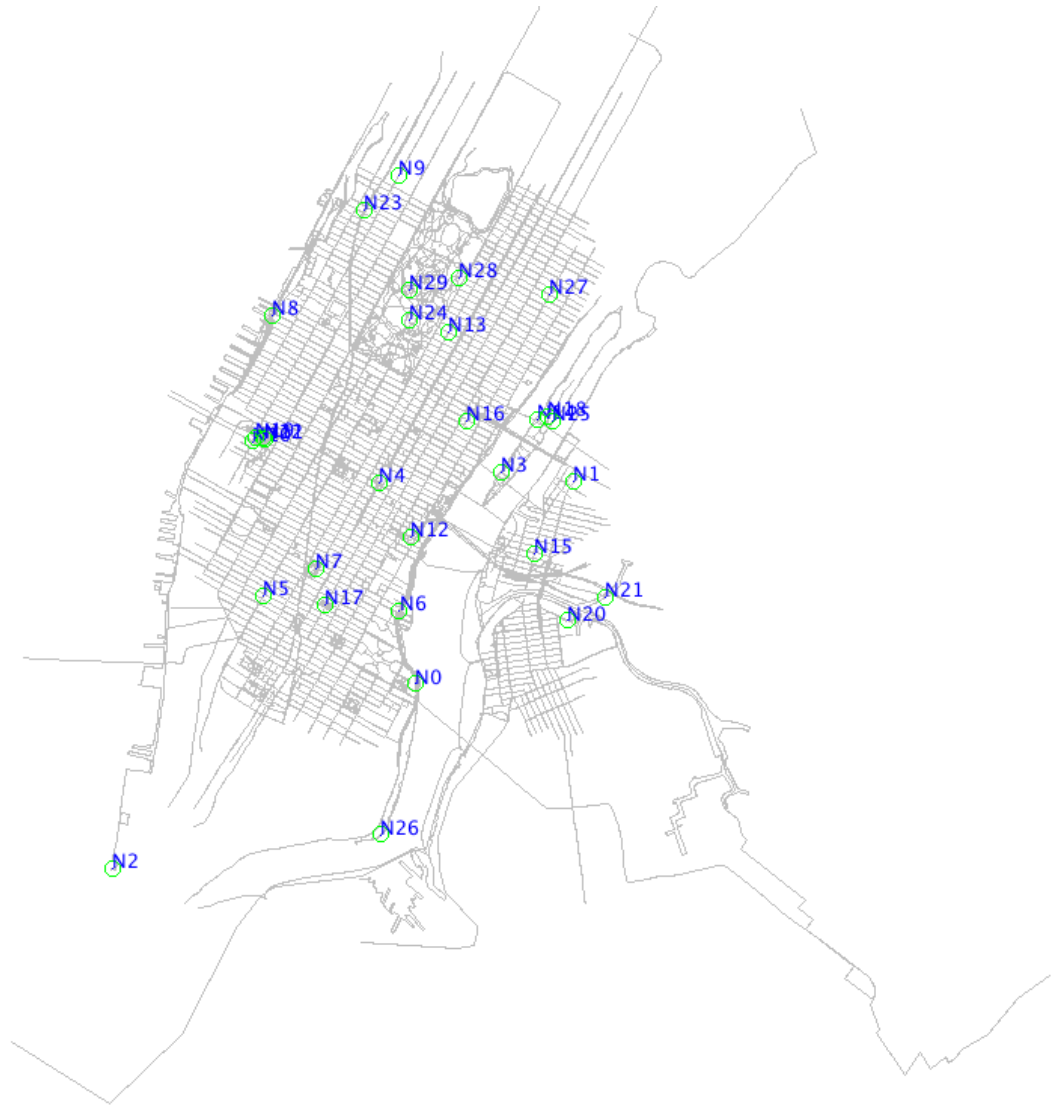


Figure 3.3. A snapshot of the locations of 30 nodes on the map of Manhattan in New York City, NY

3.4.3 Island hopping within Istanbul, Turkey Scenario

Island hopping scenario is suggested as a mobility model for a partitioned network in [25] based on an observation that in many practical settings, spatial node

distributions are very heterogeneous, and there are some concentration points of high node density. We create a mobility scenario based on this idea. Indeed, in the real life, nodes are not moving around over the whole network field. Rather, nodes tend to have a small sub-area where they tend to stay, and communicate mainly with other nodes that share the sub-area.

In the Istanbul scenario, we create three sub-areas each of which is overlapped at the edge. In each sub-area, there is one stationary node, which generates files to other stationary nodes in a different sub-area. Also, the number of moving nodes in the sub-areas is same. We conduct simulations with the number of moving nodes in a sub-area of 9, 19, and 29. Thus, total number of nodes in the network is 30, 60, and 90 respectively. Figure 3.3 is a snapshot of the Istanbul scenario with 60 nodes showing the map of Istanbul city and the locations of the stationary nodes (with red circle) as well as other mobile nodes.

Moving nodes follow the SPMBM as in the Manhattan scenario. The node speed is set to mimic the speeds of cars in Istanbul. Considering the traffic congestion in the city, we use minimum speed of 22.5 mph and maximum speed of 36 mph. The pause time is the same as in the Manhattan scenario, a random variable uniformly distributed over $[0, 120]$ seconds. Total simulation time is also a week, or 604800 seconds.

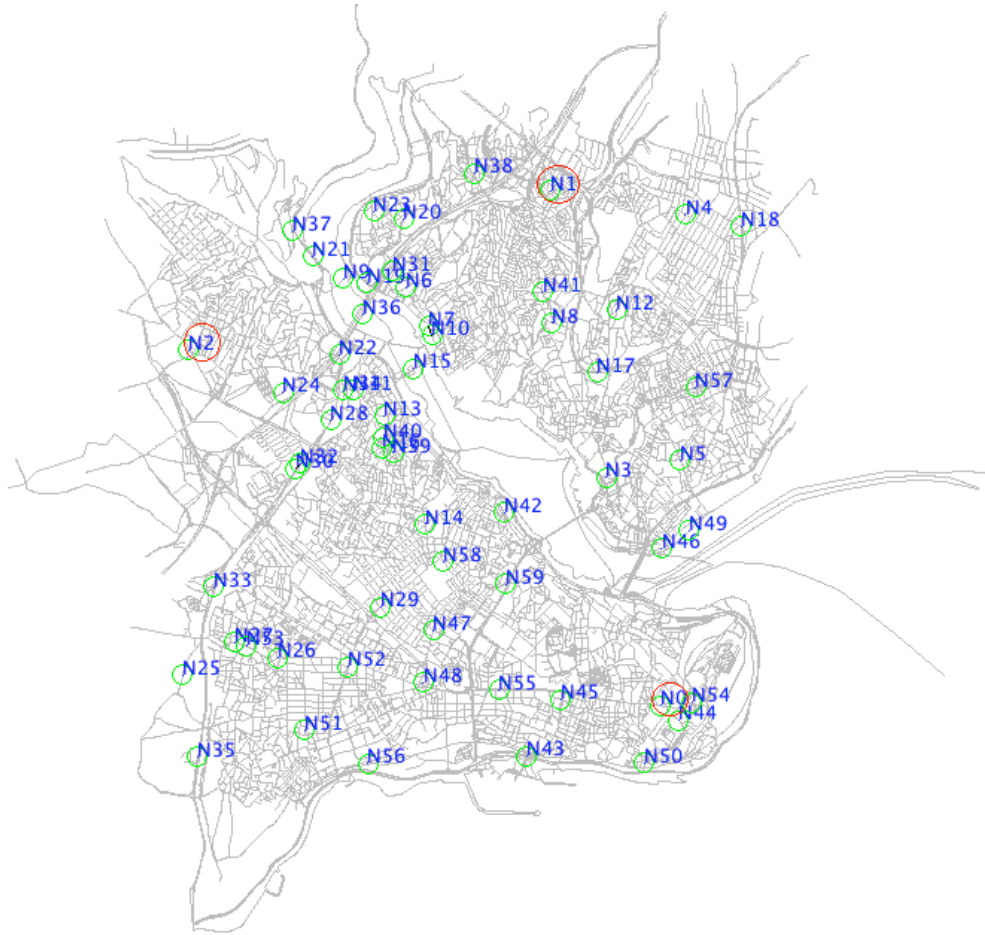


Figure 3.4. A snapshot of the Istanbul scenario depicting the locations of the stationary nodes and the sub-areas

3.4.4 Taxi Trace in San Francisco, CA Scenario

In this San Francisco taxi scenario, we use the real mobility trace collected from taxi cabs in San Francisco, CA in 2008 [22]. The data set contains GPS coordinates of approximately 500 taxis collected over 30 days in the San Francisco bay area as a part of the cabspotting project [21] by the Exploratorium. Each taxi is equipped with a GPS receiver and sends a location-update to a central server periodically. Figure 3.5 shows a single taxi trace (depicted in red lines) on the real map.

significantly affects performance. In particular, the size of file(s) and the rate of file generation have strong influence on the level of data traffic congestion in the network. In addition, the size of chunk segmented from a file is also important for the traffic pattern.

Considering the transmission rate of nodes, we fix the size of chunk to be 100kB regardless of a file size, which is small enough to transfer in a single contact. Therefore, if the size of a file is big, the number of chunks for the file is also large while the size of chunk remains same. For file generation, we use a Poisson process. Throughout the simulation, files are generated according to a Poisson process with a given rate. We use two different rates with two different sizes of file respectively to keep the total traffic load in the network similar. 10MB files are generated at a rate of 0.0005 file per second (file/s), and 100MB files are generated at a rate of 0.00005 file/s. Comparison of performance between these two scenarios shows the effect of file size as well as rate of file generation.

We also consider various traffic patterns. For each mobility scenario, we give a different pattern of the source and destination. For example, every node can be source and destination in the NYC scenario, while there are fixed source and destination in the ferry scenario.

We note that the key point of traffic flow pattern we focus on in this dissertation is the level of traffic congestion. In order to set different traffic congestion environment, we vary multiple parameters such as transmission rate and range, node density, node speed, traffic generation rate, and etc. However, unlike the traditional network, traffic load is very hard to estimate by these configurable

parameters in DTNs. It significantly depends on a routing scheme and the mobility. Therefore, in this dissertation, we define the traffic congestion level by FDR from the simulation result. For instance, we say that a network is highly congested when the FDR is higher than 70%.

While analyzing the results, we explore the performance of routing schemes with at least three different levels of congestion, light, moderate, and high traffic jam. We are not interested in the situations of too light congestion (i.e., all of files are delivered) or too high congestion (i.e., no file is delivered).

3.6 Summary of simulation settings

Table 3.2 shows all the parameter settings for each mobility scenario. It will be referred to throughout Chapters 4, 5, and 6 when simulation results are presented. As shown in the table, node speed, pause time, and file generation settings are fixed for each mobility scenario. Node speeds and pause times are uniformly distributed over the specified intervals. We use the same file size and file generation rate for every scenario.

Mobility scenario	Ferry		NYC		Istanbul		Taxi	
Number of nodes	4 stationary nodes + 2, 3, 6 ferries		30, 60, 90 moving nodes		3 stationary nodes + 27, 57, 87 moving nodes		150 moving nodes	
Transmission rate, range	1	100kB/s, 50m	1	100kB/s, 50m	1	200kB/s, 100m	1	100kB/s, 50m
	2	200kB/s, 50m	2	200kB/s, 50m	2	500kB/s, 100m	2	200kB/s, 100m
	3	200kB/s, 100m	3	200kB/s, 100m			3	300kB/s, 150m
Node speed	[10, 30] mph		[1.5, 3.5] mph		[22.5, 36] mph		Real trace	
Pause time	[10, 120] s		[0, 120] s		[0, 120] s			
File generation setting	1 fixed source 1 fixed destination		Every node can be source and destination		Only stationary nodes are source and destination		Every node can be source and destination	
File size, generation rate	1	10MB, 0.0005 file/s						
	2	100MB, 0.00005 file/s						

Table 3.2. Simulation settings

Chapter 4: Epidemic Routing Scheme

In this chapter, we investigate the Epidemic Routing (ER) scheme, specifically its performance with two different implementations of the summary vector. The rest of the Chapter is organized as follows: first, we describe the implementation details of the ER scheme in Section 4.1. The issues we encountered while implementing the ER scheme in the ONE simulator are also included in Section 4.1. Section 4.2 explains the details of Bloom filters and how we use one in our summary vectors. Simulation results are presented in Section 4.3.

4.1 Implementation Details

The ONE simulator comes with an implementation of six popular DTN routing protocols, and the ER scheme is one of them. However, there is a marked discrepancy between its implementation and the originally proposed ER algorithm [29], especially regarding the exchange of the summary vector. Therefore, we implemented a new version of the ER protocol in the ONE simulator. In the following sections, we describe the implementation details of the ER scheme, which is also the base for the EBR scheme.

4.1.1 Description of the ER Scheme

The ER scheme works by replicating copies of a bundle and distributing them amongst many relay nodes. Each node maintains a buffer consisting of bundles that are created by the node as well as those that were forwarded by others. For efficient processing, we store the bundles in a hash table keyed by a unique ID of each bundle. Every node also stores a bit vector, called a summary vector that indicates which

bundles are in the bundle hash table. The summary vector is exchanged at node encounters to determine which bundles need to be transferred. As we mentioned before, we implemented the summary vector with the list of bundle IDs and the Bloom filter keyed by the bundle IDs. It will be explained in detail in later sections.

The exchange of summary vectors has not been implemented in the ER protocol of the ONE simulator. Instead, a node tries to send every bundle it is carrying to the neighboring node, and the receiver simply drops a bundle if it is already stored in the buffer. This is what is commonly referred to in the DTN literature as “flooding”, and is different from true Epidemic Routing as proposed by Vahdat and Becker [29]. However, because the simulator clock does not advance during the failed tries, it works as though only bundles the other node does not have were forwarded. However, this model of ER does not allow us to evaluate the overhead due to exchange of summary vectors. As the number of bundles stored on each node increases, so should the overhead incurred by the exchange of summary vectors. Therefore, in order to capture the effect of this overhead in our simulations, we needed to implement a more realistic version of the ER scheme in the ONE simulator.

First, we implemented the exchange of the summary vector at node encounters. When two nodes come in contact, one of them sends out its summary vector first. Then, the other node also sends its summary vector to the initiator node. Unlike [29] where the node with smaller ID always initiates the communication, we select the initiator in a random fashion. In addition, in order to best utilize the limited contact time, we skip the step of requesting the bundles a node needs (i.e., requesting

bundles it does not have, but the other node has). In our implementation, right after exchanging summary vectors, nodes start to transfer real data bundles if needed. Since we assume that links are not shared at the same time, only one node can send data at a time. In the absence of any detailed scheduling algorithm, they take turns sending a bundle. This feature is also newly implemented in the ONE simulator; in the original ONE simulator, one of the nodes transfers all its bundles before the other node gets a chance to send any.

Once a node receives the summary vector from the other node, it figures out which bundles need to be transmitted and creates a separate outgoing queue for them. It sorts the queue according to selected scheduling strategy. Then, while the link is available (i.e., they are within the transmission range) and there are more bundles in the send queue, they keep forwarding bundles to the each other, alternating between them.

4.1.2 File Handling and Scheduling Strategy

The original ONE simulator does not provide any implementation of bundle fragmentation [26] or application data unit reassembly [26]. Therefore, we needed to implement these features in the ONE simulator in order to study the scenarios of transferring large files that need to be fragmented before sending.

In our implementation, when a node creates a file, it proactively breaks it into fragments (i.e., chunks). Then, the source node places each chunk in the payload of a new bundle. Every one of these new bundles also stores information regarding the chunk such as the UUID [36] of the file and the location of the chunk within the file in the extension block. In addition, a unique bundle ID is assigned to each bundle by

the following naming rule, “[creation time], [sequence number], [time to live], [end point identifier of the source node]”, where the sequence number is generated by the source node. An example of the bundle ID is “419, 11, 200000, dtn://umd.edu/N20”.

Once a chunk is placed in a separate bundle via the above procedure, it is treated as any other ordinary bundle until it is delivered to its destination. Relay nodes, as well as the source node, pay no attention to the UUID information attached to the chunk bundles, and forward chunks (in a bundle format) by only referencing their bundle ID in the summary vector to determine if they need to be transmitted to the neighboring node.

Scheduling strategy is also simple. Since every chunk is seen as just a bundle, nodes sort chunks in the outgoing queue by the bundle scheduling strategy, regardless of the file they are associated with. In this dissertation, we use the Round Robin scheduling for all scenarios.

At the destination, chunks are grouped by the file they belong to. It checks the number of chunks collected for the file. When the number of chunks delivered at the destination reaches the total number of chunks fragmented from the file, it means the destination collects all of chunks to recover the file. Then, we say the file is delivered.

4.2 Summary Vector Implemented by the Bloom Filter

A defining part of the ER protocol is the exchange of summary vectors when nodes encounter each other. In order to avoid unnecessary data transmission, nodes check which bundles need to be transmitted to the neighbor node before starting the data transfer. A summary vector is a control message that indicates which bundles the

node is carrying at the moment of encounter. We implemented the summary vector with two different types of data structure. The list of bundle IDs has been proposed and discussed in [18]. However, the Bloom filter implementation has not been studied in detail yet to the best of our knowledge.

A Bloom filter is a storage-efficient probabilistic data structure that is used to test the membership of an element in a set. It experiences false positives, but it guarantees no false negatives. Since the role of the summary vector of the ER scheme is to test the membership of bundles in a neighboring node's bundle store, the Bloom filter is a good candidate data structure for the summary vector [29].

4.2.1 The optimal number of hash functions

In order to keep the probability of false positives low, we need to have a sufficiently large Bloom filter size (m) and a sufficient number of hash functions used for the membership test. Calculation of the optimal number of hash functions given the size of an entry is as follows: After inserting n entries into a Bloom filter of size m bits using k hash functions, the probability that a particular bit is still 0 is:

$$P_0 \approx \left(1 - \frac{1}{m}\right)^{kn} \quad (4.1)$$

Thus, the probability of false positive (i.e., all of the k array positions computed by the hash functions is 1) is:

$$P_{err} = (1 - P_0)^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \quad (4.2)$$

Suppose that we are given the ratio $\frac{m}{n}$ and want to optimize the number of k hash functions to minimize the probability of false positive P_{err} . We can find the minimum by taking the derivative of P_{err} with respect to k . We find the optimal k by setting the derivative to 0.

$$f = \ln P_{err} = k \cdot \ln \left(1 - e^{-\frac{kn}{m}} \right)$$

$$\frac{df}{dk} = \ln \left(1 - e^{-\frac{kn}{m}} \right) + \frac{kn}{m} \cdot \frac{e^{-\frac{kn}{m}}}{1 - e^{-\frac{kn}{m}}}$$

$$\therefore k = \frac{m}{n} \ln 2 \tag{4.3}$$

For the optimal value of k , the false positive rate is;

$$P_{err} = \left(\frac{1}{2} \right)^{\frac{m}{n} \ln 2} = (0.6185)^{\frac{m}{n}}$$

We pick 8 for $\frac{m}{n}$, the size of each entry (i.e., an entry takes 1 byte), thus, run the hash functions 7 times to keep the probability of false positives roughly 2.14%. Note that the above calculation assumes perfect hash functions that spread the elements uniformly throughout the range $\{1 \dots m\}$. [23] suggests the MD5 message-digest algorithm [24] as a hash function for the Bloom filter, claiming that it has achieved good results in practice. Our implementation also uses MD5.

4.3 Simulation Results

There is a trade-off between using the Bloom filter for the summary vector vs. using the list. Despite getting the benefit from space efficiency, the Bloom filter can result in false positives, which could adversely affect performance. Intuitively, we expect that Bloom filter implementation of summary vectors saves contact time, generally resulting in higher delivery ratio than the list type of summary vector. However, we need a better understanding of how the false positive instances affect the performance.

In this section, we evaluate the performance of the ER scheme with both types of summary vector in different network settings. We are interested in comparing the relative performance of the ER scheme with the Bloom filter type summary vector against the list type one. In order to investigate with which network setting the Bloom filter works well in the ER scheme, we use the ferry network scenario with low node density, and the NYC scenario for more complex and denser network.

4.3.1 Ferry scenario

Our experiments show that with the ER scheme, the Bloom filter type summary vector outperforms the list type when the data traffic congestion is relatively high, but not that dramatically. However, when the data traffic is light or moderate, they are usually more or less on the same level. However, in some specific conditions, the Bloom filter performs even worse than the list type. If the data traffic is too light, and all of files generated are delivered, the Bloom filter performs worse than the list with respect to the ADD. This is because destinations need to wait for the next encounter to receive few more chunks that are missed due to the false positives.

Even though false positives are not many, still destination has to collect every chunk in order to complete the file. However, we do not investigate such scenarios in detail in this dissertation.

Figure 4.1 shows the FDR and ADD of the ER scheme in the network with our second setting for transmission rate and range: 200kB/s and 50m. We used our first setting for file size and generation rate; 10MB files are generated in a Poisson process with the rate of 0.0005 file/s (refer to Table 3.2). As the left graph depicts, there is no noticeable difference in the FDR. In terms of the ADD, the mobility scenarios of F2-1 and F3-1 experience better performance with the list type of the summary vector. In other scenarios, the ADD of the Bloom filter and the list fall into the confidence interval of each other, which are calculated with 90% of confidence level.

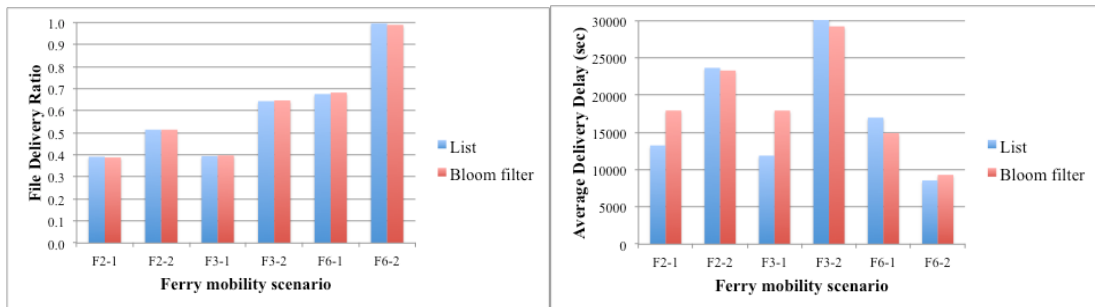


Figure 4.1. FDR and ADD of the ER scheme with Bloom filter and list summary vector in the ferry scenario with 200kB/s of transmission rate and 50m of transmission range

In order to find the reason for the debasement of the performance due to the Bloom filter, we analyze each network mobility scenario. We found that the scenarios F2-1 and F3-1 are distinct from other scenarios in some sense. First, compared to other scenarios, these two mobility scenarios tend to have longer contact times.

Second, they have fewer node encounters during the simulation than others. Figure 4.2 shows the mean contact time and the total number of contacts of every ferry scenario. In general, longer contact time helps nodes to transmit more bundles while fewer contacts are an adverse condition for delivering bundles. Considering that F2-1 and F3-1 mobility scenarios experience worse FDR, it seems that the ER with Bloom filter summary vector does not perform well in the network environment where nodes meet other nodes less frequently.

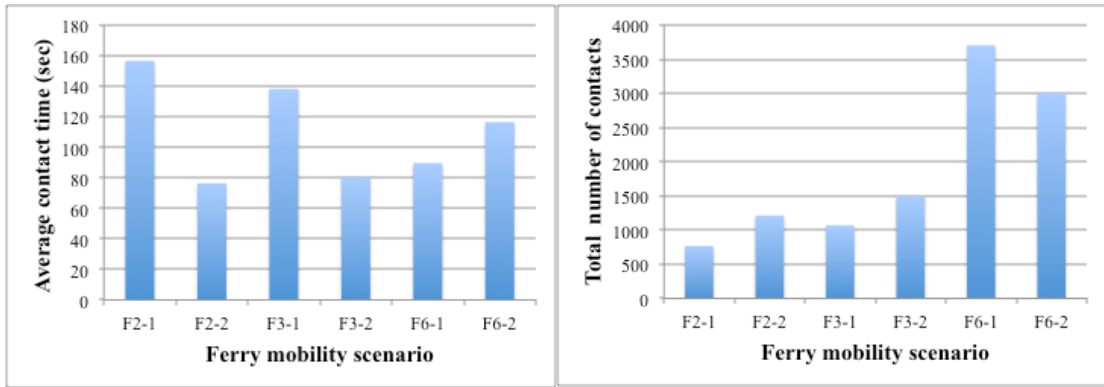


Figure 4.2. Average contact time and total number of contacts in the ferry mobility scenarios

With highly congested traffic, when using the Bloom filter implementation of the summary vector compared to the list, the performance gain from saving contact time is larger than the negative effect of the false positives in terms of FDR. Figure 4.3 shows the results when we decrease the transmission rate to 100kB/s while keeping other network settings same. With this setting, most of ferry scenarios experience low FDR (i.e., high congestion).

As is shown in the graphs, when the FDR is not distinguishable between the Bloom filter and the list such as the mobility scenarios of F2-1, F3-1, and F6-1, the ADD of the Bloom filter is smaller than that of the list type.

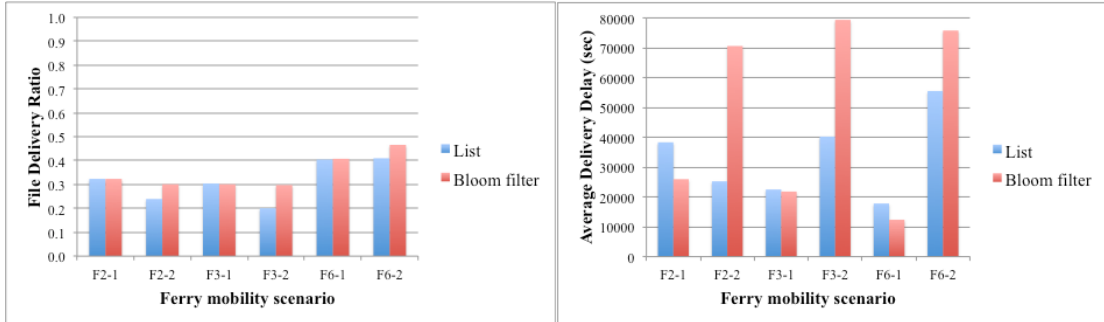


Figure 4.3. FDR and ADD of ER scheme with Bloom filter and list summary vector in the ferry scenario with 100kB/s of transmission rate and 50m of transmission range

F2-2, F3-2, and F6-2 in Figure 4.3 may suggest that the ADDs are larger for the Bloom filter. But, this is actually a consequence of the fact that the Bloom filter delivers more files that the list cannot deliver during the simulation time. Figure 4.4 plots the delivery delays of delivered files in increasing order for one of the runs. This graph tells us that if we compare the same number of delivered files with the smallest delivery delays, the Bloom filter performs better than the list. Moreover, it indicates that the ADD of the Bloom filter is affected significantly by the largest 20 percent of the delivery delays due to a sharp increase in the delivery delays for the last 17 files in the plot.

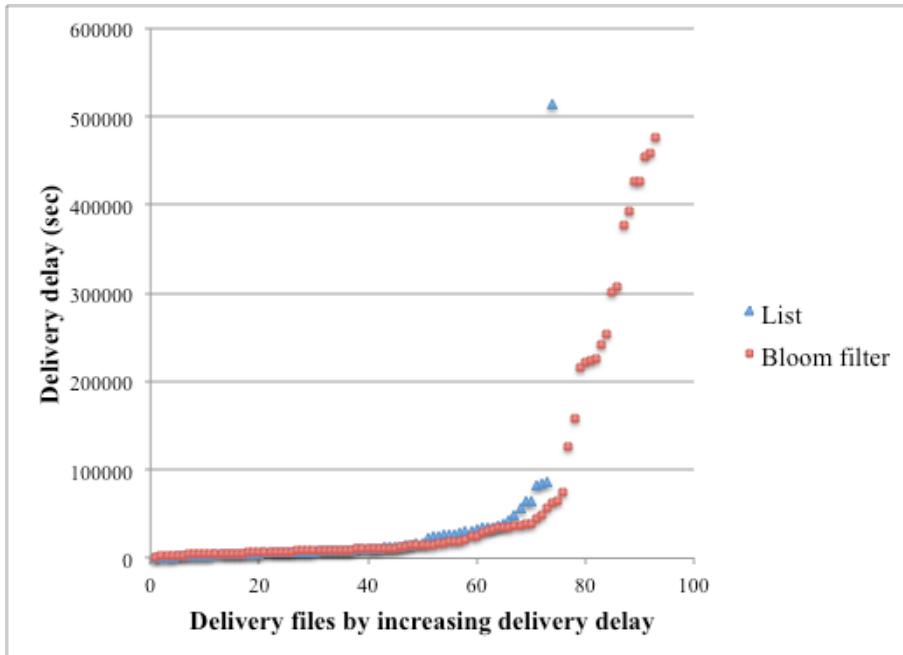


Figure 4.4. File delivery delays of the F2-2 scenario by increasing order

* **Remarks:** 1. The Bloom filter type of the summary vector slightly outperforms the list type when the network is highly congested in the ferry mobility scenarios.

2. When a network consisting of a simple topology and low node density is under moderate or light traffic, the Bloom filter performs worse than the list if nodes encounter other nodes less frequently.

4.3.2 NYC scenario

Like the ferry scenario, the NYC scenario shows similar results. If the data traffic is light, there is little difference in the performance of the ER when using the Bloom filter summary vector compared to using the list type. However, if the data traffic is high, the Bloom filter outperforms the list slightly.

Figure 4.5 shows the ADDs of the delivered files for the ER scheme with both types of summary vector under the third settings of transmission rate and range in Table 3.2. The file generation scenario of the left plot is the first one in Table 3.2 generating total of 32 100MB files according to a Poisson process with the rate of 0.00005 file/s throughout the whole simulation, while for the right plot we use the second scenario that creates a total of 330 10MB files with the rate of 0.0005 file/s. Table 4.1 displays both FDR and ADD for each case.

Note unlike the ferry or the Istanbul scenario, in the NYC scenario, the source and destination of a file are also randomly picked among nodes, and the source and destination are moving. In spite of these distinctions, the plots and table show that the Bloom filter does not have any significant performance gain over the list when the traffic is light. The slight gap between them is within the margin of error with the 90 % of the confidence level.

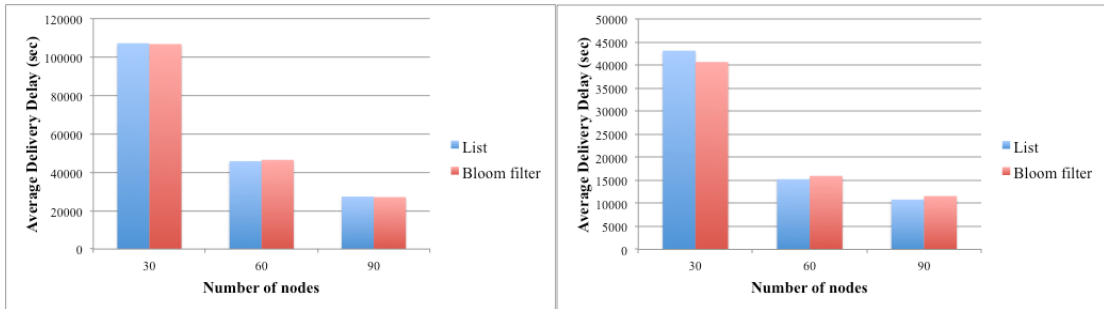


Figure 4.5. ADD of ER scheme with Bloom filter and list summary vector under the NYC scenario

100MB	List		Bloom filter	
# of nodes	ADD	FDR	ADD	FDR
30	107165.17	0.933	106741.27	0.933
60	45761.67	0.941	46440.81	0.941
90	27343.95	0.968	27061.22	0.969

10MB	List		Bloom filter	
# of nodes	ADD	FDR	ADD	FDR
30	43116.17	0.906	40660.12	0.919
60	15239.15	0.970	15954.60	0.970
90	10812.81	0.985	11576.01	0.988

Table 4.1. ADD and FDR of ER scheme with Bloom filter and list summary vector under the NYC scenario with the transmission range of 100m

In the network setting where the traffic is higher, the NYC scenarios also experience the slight performance gain from the Bloom filter implementation of the summary vector over the list type. Table 4.2 shows the results when the transmission range is decreased to 50m from 100m (i.e., the second setting of transmission rate and range in Table 3.2). Other parameters remain the same. While the average contact time between nodes in the previous network scenario where the transmission range is 100m is about 155 seconds for all of three node density settings, it drops to about 76 seconds with the smaller transmission range. Due to the shorter contact time, the transmission time saved by using the Bloom filter type of summary vector would give more positive impact on the performance.

In particular, with a 30-node network, the FDR is very low much like with the list type of the summary vector. Over the 10 runs, either only one file or none out of 30 files is delivered to the destination. However, with using the Bloom filter, every run successfully delivers more files to the destination resulting in better performance

in terms of FDR. With the smaller files (i.e., lower rate of file generation), it also shows similar results.

100MB	List		Bloom filter	
# of nodes	ADD	FDR	ADD	FDR
30	471029.19	0.023	457028.30	0.033
60	189201.98	0.782	190359.60	0.824
90	71210.18	0.921	70007.44	0.938

10MB	List		Bloom filter	
# of nodes	ADD	FDR	ADD	FDR
30	210001.71	0.089	227703.38	0.106
60	99810.66	0.801	105991.64	0.843
90	21312.87	0.935	28422.26	0.953

Table 4.2. ADD and FDR of ER scheme with Bloom filter and list summary vector under the NYC scenario with the transmission range of 50m

* **Remarks:** 1. The Bloom filter type of the summary vector slightly outperforms the list type one when the network is highly congested, regardless of the node density of the network.

Chapter 5: Encoding Based Routing Scheme

In this chapter, we study the functionality and performance of the EBR router. We study two different modes of the EBR scheme, the source coding and the network coding. We first explain each mode of the EBR scheme itself in detail, and then investigate the benefits of coding by examining the performance of both ER and EBR scheme. The rest of this chapter is organized as follows: In Section 5.1, we study EBR with source coding. Then, we examine the network coding mode of the EBR scheme in Section 5.2. Lastly, we investigate the performance of ER and both modes of the EBR in various network settings in Section 5.3.

5.1 Encoding Based Routing with Source Coding

In this section, we explain the EBR with source coding scheme focusing on three aspects, the encoding operation, the relay nodes' capability, and the control message. In each sub-section, we describe the implementation details of the routing protocol regarding that aspect, and examine the performance of EBR scheme to explore it thoroughly. In Section 5.1.1, we describe how the coding technique is employed when creating a file. We also include a study of the impact of the *coding weight*. That is the number of chunks used to generate an encoding. Then, we explain our implementation of the nodes' rank calculation capability in Section 5.1.2. Also, we investigate the performance of EBR with different configurations of relay nodes with respect to the rank calculation. Finally, we study the control message implementation of the EBR scheme in Section 5.1.3. In particular we examine how

the different implementations of the summary vector affect the performance of the EBR scheme as we did for the ER scheme in Chapter 4.

5.1.1 File Creation

5.1.1.1 Implementation Detail

As explained in Chapter 4, when a source has a large file to deliver, which is too big to transfer in a single contact, we fragment it into smaller pieces, called chunks. Each of these chunks is transported in a separate bundle in the ER scheme. In the EBR scheme, the source generates encodings from the chunks and forwards these encodings in bundles.

Encodings can be generated using different coding schemes. In EBR, we use a random binary linear coding scheme. Suppose a large file is divided into a large number of equally sized chunks N ($N \gg 1$). To generate a single encoding, the source randomly chooses at most K ($1 \leq K \leq N$) chunks out of the N chunks, takes a binary sum of these K chunks, and puts the binary sum in the payload of the bundle along with the associated encoding vector in the extension block. We call K the *encoding weight* since it is the maximum number of distinct chunks in an encoding. Algorithm 1 shows how to create a new encoding vector with a given K .

Algorithm 1. Creating a new encoding vector
N = number of chunks K = encoding weight V = N -dimensional zero vector for i in $[0, \dots, K]$ j = random(0, $N-1$); $V[j] := 1$; end for

When N is large and $K > 1$, the number of distinct encodings we can generate (i.e., the number of different nonempty sets of at most K chunks we can choose) is given by, $P = \sum_{1 \leq i \leq K} \binom{N}{i} = \Theta(N^K)$, which is in general much larger than N . However, in order to prevent flooding the network with too many encodings from a single file, we limit the number of encodings generated by the source to $m \cdot N$, where $m \geq 1$. For our simulations in this dissertation, we select $m = 3$. Note that there is a chance some of these $m \cdot N$ encodings we produce are identical since we do not check the redundancy of encoding vectors at the source.

We also limit the maximum number of encodings held in the buffer of the source at any one time to B , in order for the source nodes to avoid buffer overflow caused by encodings belonging to a single file. In this dissertation we use $B = 20$. Thus, initially, the source generates only 20 encodings when a file arrives, and, once the number of encodings in the buffer drops below $\frac{B}{2}$, it generates additional encodings as encodings are transferred to other nodes. In addition, when it loses the contact with a neighbor, the source replenishes the B encodings as long as the total number of encodings generated does not exceed the limit. This implementation detail is obviously different from the ER scheme where the source nodes generate all of chunks when a file arrives. Moreover, each new encoding created by the source is forwarded only to a single neighbor and is removed from the buffer after it is successfully transferred to another node. Hence, no two nodes receive the same encoding from the source directly.

Setting a proper value of K is important for the performance of the routing scheme. If the encodings generated by the source miss any chunk (i.e., a specific

chunk is not chosen at all while the source generates encodings), there will be no chance to collect all of chunks at the destination. The probability that the file is not delivered depends on the value of K especially when K is small. Hence, we compute the proper weight for encodings to guarantee that the probability Prob[every chunk is used in at least one of $N + \varepsilon$ encodings] is at least some threshold

Let us say E_i is the event that the chunk i is missing in the first N encodings.

Then, the probability q that at least one chunk is missing in the first N encodings can be calculated by the following expressions.

$$q = P(\cup_{i=1}^N E_i) \leq \sum_{i=1}^N P(E_i) = N \times P(E_1) \quad (5.1)$$

$P(E_1)$ is the probability that the first chunk is missing in the first N encodings. Given the encoding weight K , when N is very large, $P(E_1)$ is given by

$$P(E_1) = \left(1 - \frac{K}{N}\right)^N \approx e^{-K} \quad (5.2)$$

From (5.1) and (5.2), the value of K that keeps the probability that the first N encodings miss any chunk below q is derived as follows.

$$N \times e^{-K} \leq q$$

$$e^{-K} \leq \frac{q}{N}$$

$$\therefore K \geq \log_e N - \log_e q$$

Therefore K should be larger than $\log_e N - \log_e q$. In the next section, we examine the performance of the EBR scheme as we vary the encoding weight. We will show the

optimal value of K to create innovative encodings (i.e., encodings with independent encoding vectors) is as we calculated above.

5.1.1.2 Simulation Results

Figure 5.1 shows the performance of EBR scheme with varying encoding weight in the ferry scenario with the second setting for file size and generation rate. In this scenario, 100MB of files are fragmented into 1000 chunks. Figure 5.2 shows the same with the first setting, where the file size is 10MB, which is split in 100 chunks. The name of each line is the mobility scenario name, which is introduced in Table 3.1. For example, the graph named ‘F2-1’ shows the result where two ferries are moving on the assigned segments $\{1, 5\}$, $\{3, 5\}$ respectively. In addition, Figures 5.3 and 5.4 are the results of the NYC scenarios with the second setting for the transmission rate and range and the second and the first setting for the file size respectively.

According to the calculation in the previous section, in order to keep the probability that the first N encodings miss any chunk at or below 5%, K should be larger than 9.903 when $N = 1000$, and it should be larger than 7.601 when $N = 100$. Figures 5.1 and 5.3 show that when the coding weight is equal to or larger than 10, which is very close to our calculation, the performance of EBR does not increase noticeably as the coding weight increases. Also, Figures 5.2 and 5.4 show that according to the simulation results, the optimal value of K is about 8 when $N = 100$, which is similar to our calculation.

In addition, it is observed that there is a tendency for the optimal value of K to be smaller in the denser networks (when the number of nodes is larger).

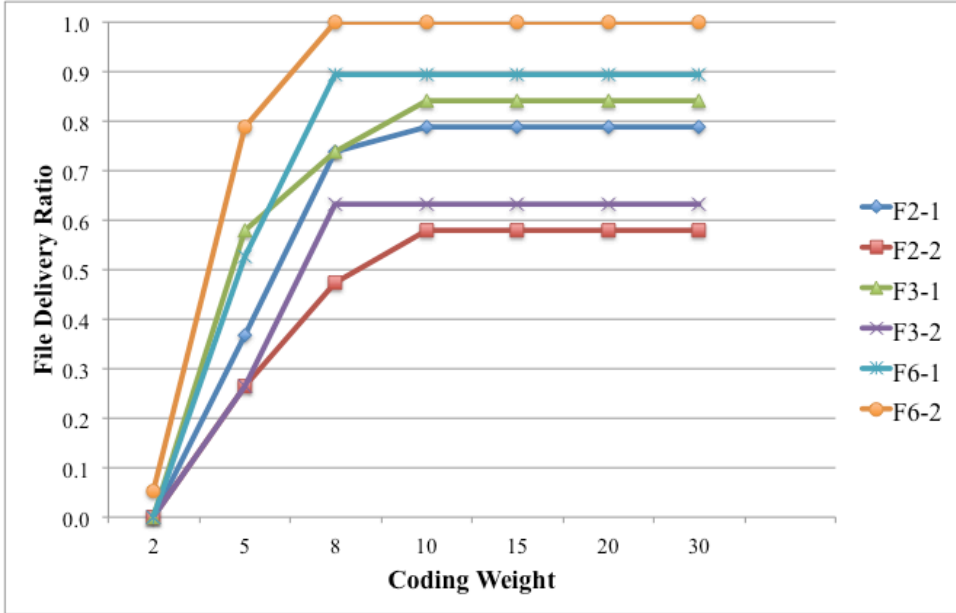


Figure 5.1. FDR of the EBR scheme varying the encoding weight in the ferry scenario when $N = 1000$

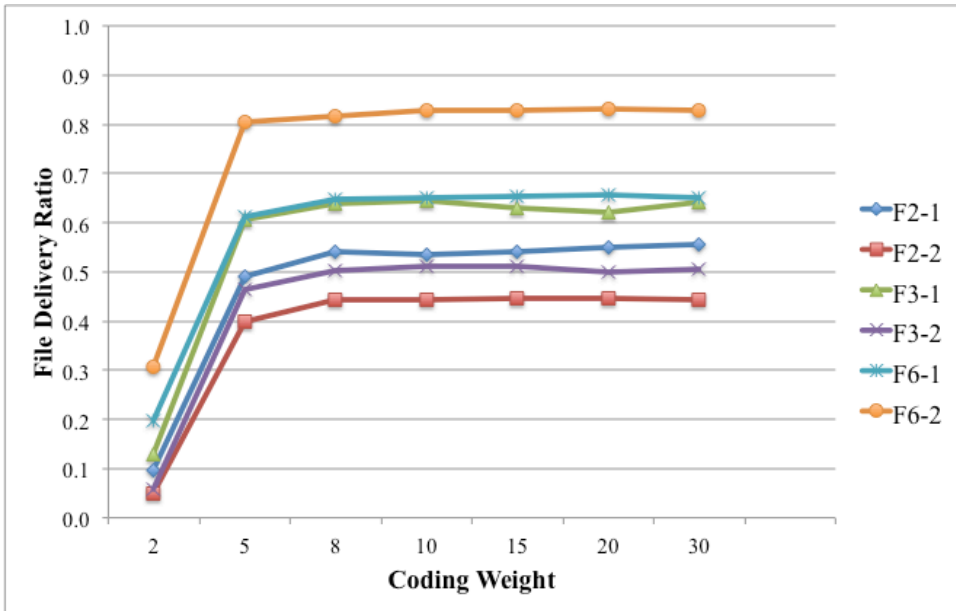


Figure 5.2. FDR of the EBR scheme varying the encoding weight in the ferry scenario when $N = 100$

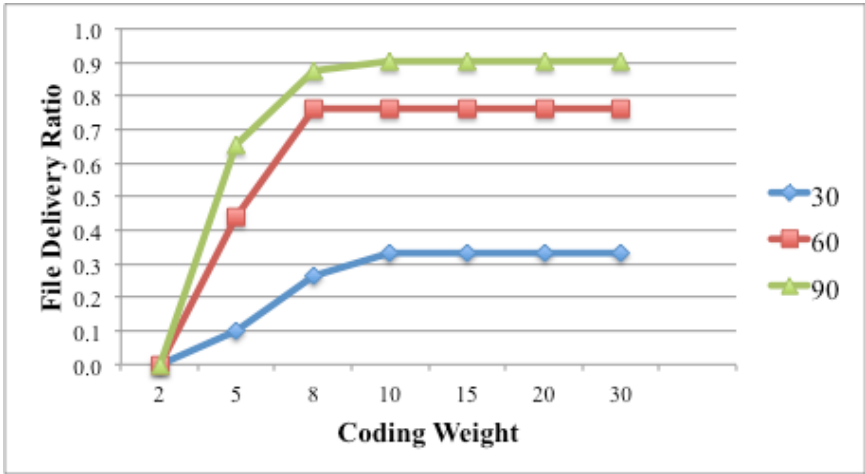


Figure 5.3. FDR of the EBR scheme varying the encoding weight in the NYC scenario when $N = 1000$

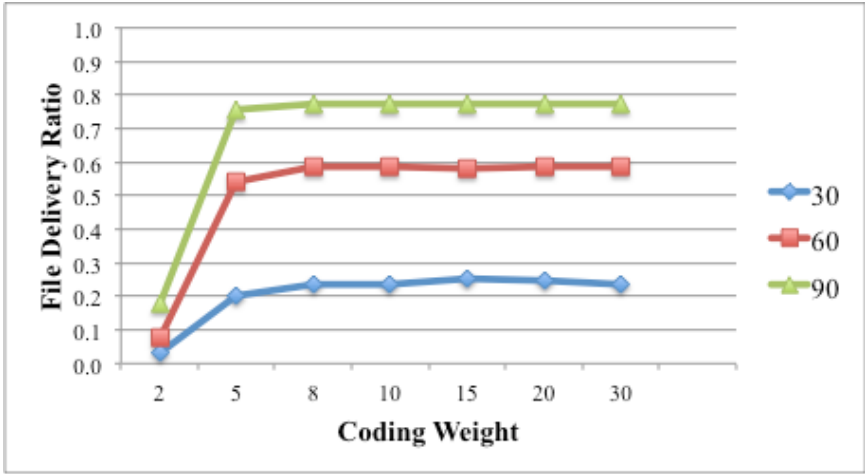


Figure 5.4. FDR of the EBR scheme varying the encoding weight in the NYC scenario when $N = 100$

5.1.2 Rank Calculation

5.1.2.1 Implementation Detail

In order for the destination to be able to recover the file, it needs to collect N encodings with linearly independent encoding vectors, where N is the number of chunks of the file. In other words, if we denote by e_i the encoding vector of the i^{th}

encoding that the destination has and A is the total number of encodings that the destination possesses, a necessary and sufficient condition for recovery is:

$$\text{rank}([e_1 \ e_2 \ \dots \ e_A]^T) = N \quad (5.3)$$

where $[e_1 \ e_2 \ \dots \ e_A]^T$ is an $A \times N$ matrix which row i is e_i . We call this matrix containing the encoding vectors the **encoding matrix**. When the condition in (5.3) is satisfied by the encodings held at the destination, we say that the destination has reached full rank. Once the destination reaches the full rank, it starts to decode the original chunks through matrix inversion (e.g., Gaussian elimination [5]) and reconstruct the file. In the simulation, we do not consider the cost of reconstructing a file, which would depend on the capability of (destination) nodes. Instead, we only calculate the rank, and when the destination reaches full rank, we say the file is delivered. Therefore, the time at which the last encoding (i.e., the N^{th} innovative encoding) arrives at the destination is the time of delivery for the file. Algorithm 2 shows how to calculate the rank of the encoding matrix by keeping it in row echelon form [13].

Algorithm 2. Calculating the rank of a file
<p>N is number of chunks M is an $N \times N$ matrix L is a vector of size N L_i is the position of the leading 1 in row i of M. r is the rank of M</p> <p>Initialize $r = 0$;</p> <p>New vector v arrives, put it into row r of M for i in $[0, \dots, N - 1]$ do $M_{r,i} = v_i$; end for</p> <p>for i in $[0, \dots, r - 1]$ do if $M_{r,L_i} \neq 0$ then,</p>

```

        for j in [0, ..., N - 1] do
             $M_{r,j} = M_{r,j} \oplus M_{i,j}$ ;
        end for
    end if
end for

flag = false;
i = 0;

while flag is false do
    if i = N then flag = true;
    else if  $M_{r,i} \neq 0$  then,
        flag = true;
         $L_r = i$ ;
         $r = r + 1$ ;
    end if
    i = i + 1;
end while

```

Suppose that each new encoding that arrives at the destination is equally likely to be any of P possible encodings, where $P = \sum_{1 \leq i \leq K} \binom{N}{i}$ is the number of distinct encodings we can generate. Assuming that K is not too small, the expected number of distinct encodings the destination needs before it can recover the file is approximately $N + \varepsilon$, where ε is a constant number that is much smaller than N .

Relay nodes may get N linearly independent encodings with high probability when they receive $N + \varepsilon$ distinct encodings from others. Because receiving and carrying redundant encodings results in performance debasement, we restrict the number of encodings from a single file that a relay node may carry. We implement this restriction in two different ways. First, we set the maximum number of encodings belonging a single file to $N + \varepsilon$. Second, we allow the relay nodes to process the encoding vectors, so that they can drop any redundant encoding as they receive it. In other words, every relay node is able to calculate the rank of files when a new

encoding arrives, and if the newly received encoding does not increase the rank (i.e., it is not linearly independent with other encodings the node already has), it drops the encoding. We call this capability *rank check*. Once a relay node has reached full rank of a file, it no longer receives encodings for the file from other nodes.

There is a trade-off between these two implementations. Allowing extra encodings is cost-efficient in terms of calculation. In this way, nodes need to neither maintain the encoding matrices nor calculate the rank. They only count the number of encodings belonging to each file when they decide to receive an encoding or not. Yet, once they collect $N + \varepsilon$ encodings for a specific file, even though the number of innovative encodings is less than N (i.e., not full rank), they stop receiving new encodings for the file. Furthermore, there could be an opposite case as well; a node keeps receiving more encodings for a file even though it already reaches the full rank.

5.1.2.2 Simulation Results

In this section, we study how the rank check feature described above affects on the performance of the EBR with source coding scheme. For all of scenarios, we use $\varepsilon = 50$ for the configuration when relay nodes do not have the rank check capability. Also, we do not consider the computational cost of the rank check operations at the relay nodes. In order to explore both of simple networks and complex ones, we conducted simulations using the ferry scenarios and the NYC mobility model. Figures 5.5 and 5.6 show the performance of EBR in the ferry scenario, and Figures 5.7 and 5.8 show it under the NYC scenario. Each figure represents a different level of traffic congestion.

Figure 5.5 plots FDRs and ADDs of the EBR scheme in the source coding mode when the transmission rate and range is set to the second one for the ferry mobility in Table 3.2; 200kB/s and 50m. The file size and generation rate setting is the first one, 10MB and 0.0005 file/s. Every file is encoded with the coding weight of 15. This network setting generates the light to moderate traffic congestion under the ferry scenario. FDRs are from 0.4 to 0.95 depending on the scenario, where one can say the traffic is moderate or light.

According to the results from the plots, the rank check at the relay nodes always outperforms the configuration without rank check with respect to both of FDR and ADD.

When the network is more congested, the benefit from the rank check capability at relays is also significant. Figure 5.6 shows the results under the scenario where only the transmission rate is changed from the setting of Figure 5.5. When the transmission rate is reduced from 200kB/s to 100kB/s, the network ends up experiencing higher level of traffic congestion. But the relative performance of the EBR with rank check at relay nodes against the EBR with no rank check at the relay nodes remains almost same as the previous setting. With the rank check feature, the EBR outperforms the one without it under all of ferry scenarios.

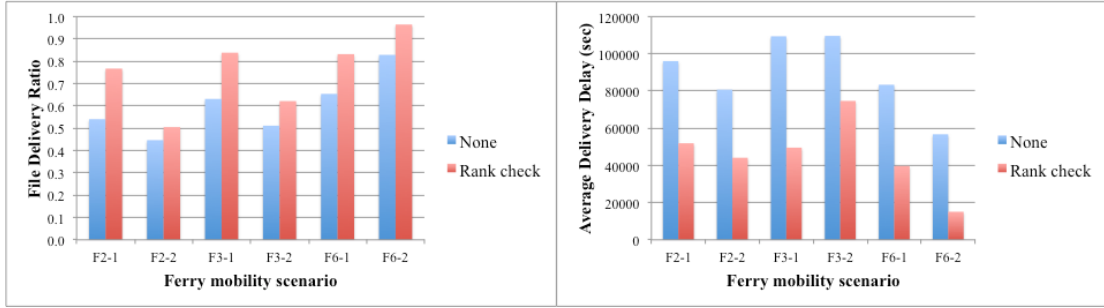


Figure 5.5. FDR and ADD of the EBR scheme with and without rank check at the relay nodes under the ferry mobility with light to moderate traffic congestion

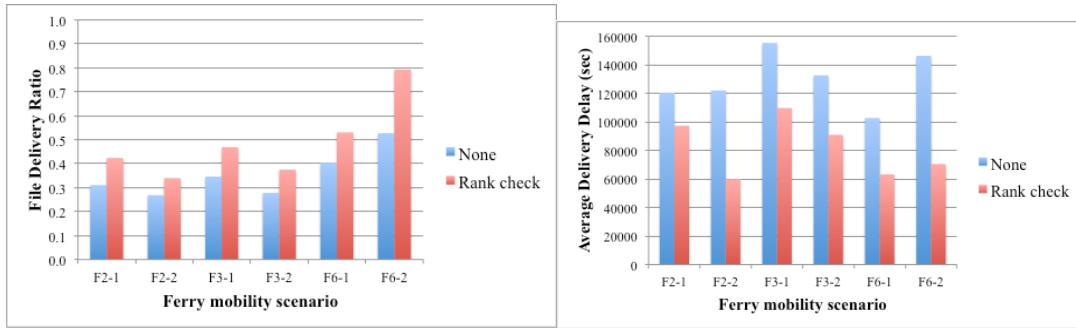


Figure 5.6. FDR of the EBR scheme with and without rank check at the relay nodes under the ferry mobility with moderate to high traffic congestion

The reason that the rank check feature works better than allowing extra encodings without any calculation seems the overhead of allowing extra encodings for each file. When a node collects N independent encodings before receiving $N + \epsilon$ encodings, if the node has rank check capability, it stops receiving more encoding from the file. However, without rank check, the node keeps receiving more encodings until it collects $N + \epsilon$ encodings, which is waste of resource such as transmission time. And, as the number of files is increased, the waste is cumulated resulting in the performance loss. We can see how the cumulated overhead hurts the performance of EBR by comparing the results between two different settings for file size and generation.

Figures 5.7, and 5.8 show the results under the NYC mobility scenarios. They are plots of FDR and ADD under the same setting for the transmission rate and range (200kB/s and 100m), but different settings for the file size and generation rate. Figure 5.7 shows the result with the first setting where 330 10MB files are created throughout the simulation, and Figure 5.8 shows the one with the second setting where 32 100MB files are generated.

Even though there is little difference in the total number of chunks generated between these two settings, it seems that the difference is significant in the overhead resulted by allowing extra encodings between the two settings. As shown in the plots, the benefit from the rank check feature is much bigger when the rate of file generation is larger.

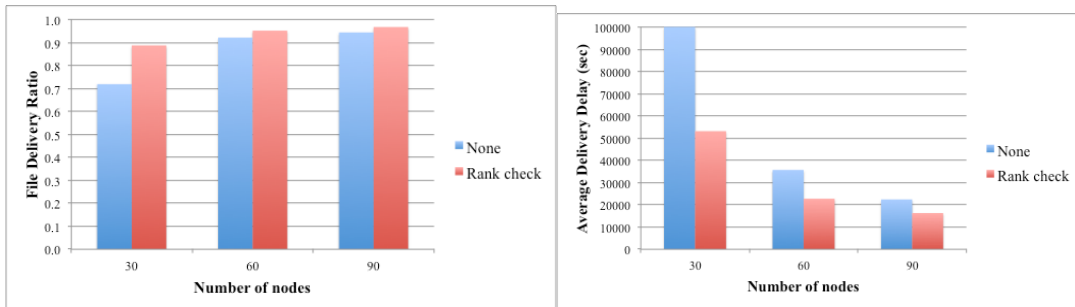


Figure 5.7. FDR and ADD of the EBR scheme with and without rank check at the relay nodes under the NYC mobility with file generation rate of 0.0005 file/s

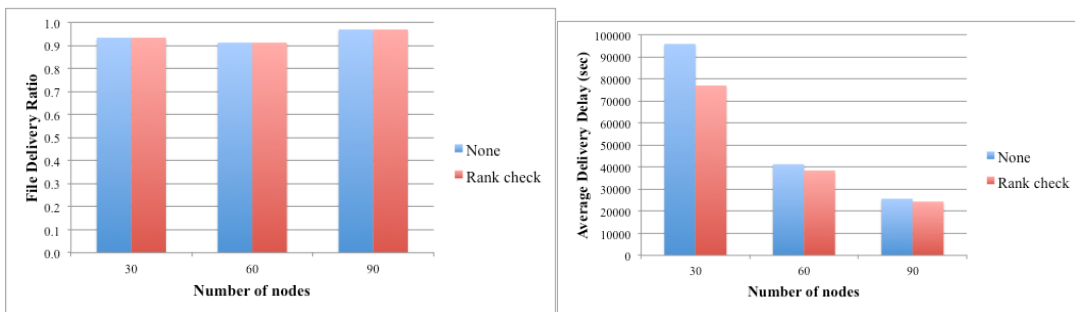


Figure 5.8. FDR and ADD of the EBR scheme with and without rank check at the relay nodes under the NYC mobility with file generation rate of 0.00005 file/s

5.1.3 Control Message

5.1.3.1 Implementation Detail

The EBR with source coding is based on the ER scheme. Other than encoding at the sources and decoding at destinations, the routing algorithm itself is very similar to the ER scheme. In particular, nodes make a decision on selecting which bundle to send to the other node by referencing the summary vector, exactly same as in the ER scheme. Creating the summary vector with the bundles in the buffer, exchanging it at the node encounter, and forwarding only those bundles that are not sitting in the buffer of the other node are also the same as the ER scheme, which described in Chapter 4. In the next section, we will investigate the performance of the EBR scheme with different summary vector implementations, the list of bundle IDs and the Bloom filter keyed by bundle IDs.

Unlike ER, in the EBR scheme nodes exchange another control message conveying the rank information of files the node carries. It is needed to limit the number of encodings carried by relay nodes. This control message simply conveys the number of encodings for each file and the UUID of the file. It is exchanged along with other control messages such as the summary vector. After exchanging these control messages, nodes figure out how many encodings for each file is needed for the other node to get the full rank as following. Assume that the other node has L encodings for a file, and the number of chunks for the file is N . Then, it limits on the number of encodings for the file to $N - L + \epsilon$, where ϵ ($0 < \epsilon \ll N$) is a configurable number. When $L = N$, the routing protocol forks off in two ways. If the rank check configuration is on, the node does not send more encodings. However, without the

rank check capability, it sends ε more encodings as long as it has enough encodings to send.

Also, the scheduling strategy is different from the ER scheme described in Chapter 4. In the EBR scheme, bundle scheduling is related to the files that encoding bundles are associated with. Every node maintains a queue for each file, which it carries. Once it receives a new encoding, it checks the UUID stored in the extension block of the bundle, which indicates which file the encoding comes from. If the node already has a queue for the file, meaning that it has at least one encoding for the file, it puts the received encoding into that queue. However, if it fails to find a queue associated with the UUID of the file, it sets up a new queue for the file, and stores the encoding in the queue just created.

When a node meets another node, assuming that the node has multiple queues for files (i.e., it carries encodings from different files) and the contact time is enough to send out more than one encoding, it needs to schedule the encodings for transferring. In order to decide which encoding (bundle) to transmit next, the node processes two steps of scheduling. First of all, it picks a queue of files in the round-robin fashion. It does not consider the number of encodings in a queue. Then, the node selects an encoding from the chosen queue in a round-robin fashion. If the selected encoding is not in the summary vector of the other node, it starts transferring the bundle. If not, it tries next encoding until either the chosen encoding is not in the summary vector or there is no more encoding to choose.

5.1.3.2 Simulation Results

In this section, we examine the benefit of the Bloom filter implementation for the summary vector against the list of bundle IDs. For this study, we carried out simulations with the ferry and NYC scenarios as we did for the ER scheme. We use the EBR scheme without the rank check implementation at the relay nodes. And, the file generation is exactly same as the simulation of Section 5.1.2.

In the ferry scenario, Figure 5.5 is the result from the network with the third setting for transmission rate and range; 200kB/s and 100m, and the second setting for file size and generation rate; 100MB and 0.00005 file/s. And, Figure 5.6 plots the result from the network with the same setting for the transmission rate and range and the first setting for file size and generation rate; 10MB and 0.0005 file/s, where the higher traffic congestion is demonstrated.

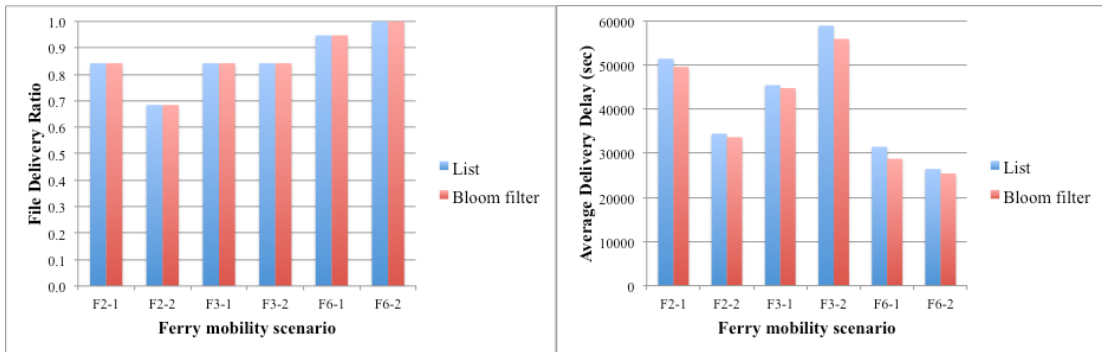


Figure 5.5. FDR and ADD of the EBR scheme with the Bloom filter and the list summary vector under the ferry mobility with light traffic congestion

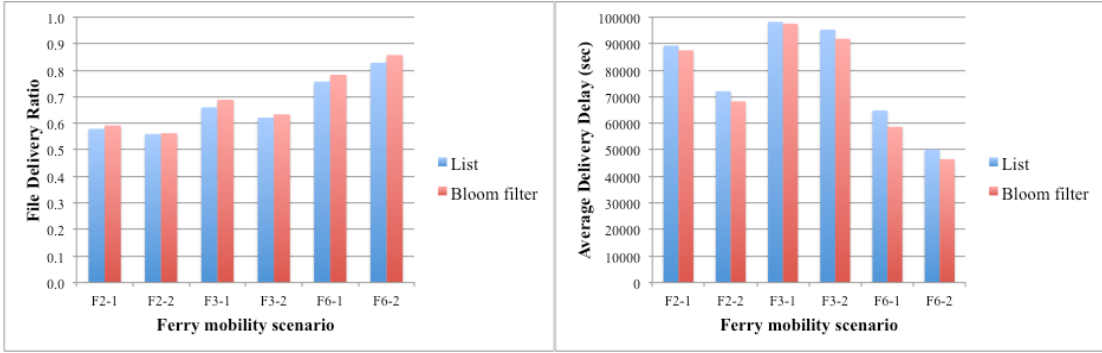


Figure 5.6. FDR and ADD of the EBR scheme with the Bloom filter and the list summary vector under the ferry mobility with moderate traffic congestion

According to the results in Figure 5.6, with respect to both of FDR and ADD, Bloom filter brings benefit to the performance of EBR scheme against the list type. Moreover, unlike the ER scheme, Bloom filter summary vector outperforms the list type in terms of ADD even in the network settings of light traffic (refer to Figure 5.5). This result shows that the false positive cases do not have much of an impact on the encodings. Regardless of which encodings are delivered to the destination, it will be able to reconstruct the file once it collects enough encodings. Hence, missing encodings due to false positives do not hurt the performance. In addition, it is observed that the performance gain from the Bloom filter implementation increases as the level of traffic congestion becomes higher in Figures 5.5 and 5.6.

For the NYC mobility scenario, Figure 5.7 demonstrates results from three different node density settings, each of which results in three different levels of traffic congestion. We use the second setting for transmission rate and range; 200kB/s and 50m for every scenario, and the second setting for file size and generation rate; 100MB and 0.00005 file/s.

With the NYC mobility scenarios, the performance benefit of the Bloom filter implementation for the summary vector is more noticeable than the ferry scenarios, even under the light traffic. This is demonstrated with high node density such as 90 nodes, the performance gain is experienced with respect to both of FDR and the ADD.

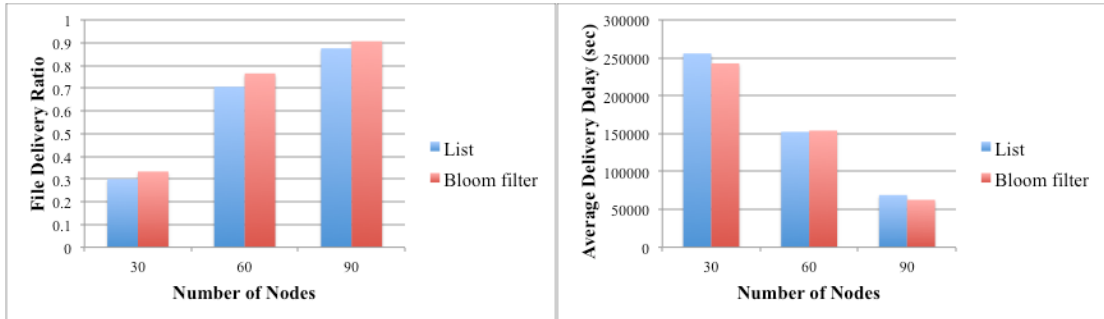


Figure 5.7. FDR and ADD of the EBR scheme with the Bloom filter and the list summary vector under the NYC mobility

5.2 Encoding Based Routing with Network Coding

In this section, we study the EBR with network coding scheme. We explain the implementation details of the routing scheme, and examine the performance of the EBR with network coding with varying the re-coding weight.

5.2.1 Implementation Details

EBR with network coding mode and EBR with the source coding mode have many common implementation details. Especially, file creation at sources and the reconstructing a file at destinations are exactly same in both of modes. However, the relay nodes behave distinctly in each mode. In the source coding mode, relay nodes make a copy of encodings, which received from the source or other relay nodes, and

just forward them to neighboring nodes. They never create a new encoding. Yet, the network coding mode of the EBR scheme allows the relay nodes to perform coding operations to generate a new encoding. Since the bases of the coding operation at the relay nodes are not chunks, but the encodings, we say that relay nodes perform *re-encoding*.

Obviously, re-encoding is only allowed for encodings associated with the same original file (i.e., with the same UUID). Therefore, nodes maintain a queue for each file, and encodings are stored in the queue of their associated file. When a node meets another node, in a round-robin fashion, it selects a file to generate a new encoding to send. Once a queue is selected, it picks encodings in the queue. The number of encodings that are used in re-encoding is *re-encoding weight*, which is configurable. Then, the node performs re-encoding with them (i.e., linear combination of them), and sends it to the other node. Once the newly created encoding is forwarded successfully, it deletes the encoding.

Like the source coding mode, nodes exchange the rank information of the files they are carrying before they start re-encoding and transmitting the results of it. After exchanging these control messages, nodes figure out how many encodings for each file are needed for the other node to reach the full rank. Note that in EBR with network coding mode, every node has the rank check capability. We set the stopping condition of disseminating the re-encoded bundles as follows: let us say that the node has R encodings for a file, where the number of chunks for the file is N . Since all of encodings are linearly independent, the rank of the encoding matrix for the file is also R . Theoretically, the maximum number of innovative encodings generated from an

encoding set is the rank of the encoding matrix of the set. Therefore, we limit the number of re-encoded bundles to R . In the case the other node needs less than R encodings to reach the full rank, it only generates as needed.

Let us explain the receiver side. In the network coding mode, there is a new system architecture called a *temporary buffer*. In order to avoid selecting encodings that just forwarded from the other node when generating a new encoding, nodes keep the newly received encodings in the temporary buffer during the session. When the link is disconnected, nodes start processing each encoding bundle in the temporary buffer to figure out whether it is innovative or not. While processing, nodes push the encoding vector of each encoding into the encoding matrix of the associated file. If the encoding is not innovative, they just drop it. Otherwise, they store the encoding in the associated queue. This process is same as the rank check implementation of the source coding mode, which is presented in Section 5.1.2.

5.2.2 Simulation Results

We investigate how the re-encoding weight affects the performance of the EBR with network coding scheme. Unlike the source coding, re-encoding can use existing encodings rather than only the original chunks to generate a new encoding. Since each encoding already has information from multiple chunks, the optimal re-encoding weight of our intuition was very small such as less than 5. In order to check if the small re-encoding weight engenders enough number of innovative encodings, we carried out simulations varying the re-encoding weight.

Figures 5.8 and 5.9 show the FDR of EBR with network coding under the NYC mobility scenario with the second setting for transmission rate and range; 200kB/s and 50m. We used the set of $\{2, 5, 10, 15, 20, \ln r\}$ as the re-encoding weight, where r is the rank of the encoding matrix for the file at the moment re-encoding is conducted. The idea of using $\ln r$ as the re-encoding weight comes from [21]. Each figure shows the result from different setting for file size and generation rate. Result from the first setting is shown in Figure 5.8, while the one from the second setting is shown in Figure 5.9.

As shown in the plots below, the results are very different from our intuition. According to the simulation results, if the re-encoding weight is set to very small number such as less than 5, it hurts the performance of EBR significantly. At least 10 encodings should be used for re-encoding. Especially, when the traffic is light, the performance loss due to the small re-encoding weight is critical. Refer to the result of 90 nodes in Figure 5.8. When the re-encoding weight is 2, the FDR is smaller than 0.4 while it is larger than 0.9 when the re-encoding weight is larger than 5.

In addition, the plot shows that $\ln r$ is not optimal for the re-encoding weight. Constant value brings better performance than $\ln r$ for all of three different node density settings when it is large enough (e.g., 10).

Figures 5.8 and 5.9 show that the number of chunks in a file does not affect significantly the performance depending on the re-encoding weight. With both settings for file size and generation rate, when the re-encoding weight is larger than 10, the performance increase is not noticeable as the re-encoding weight increases.

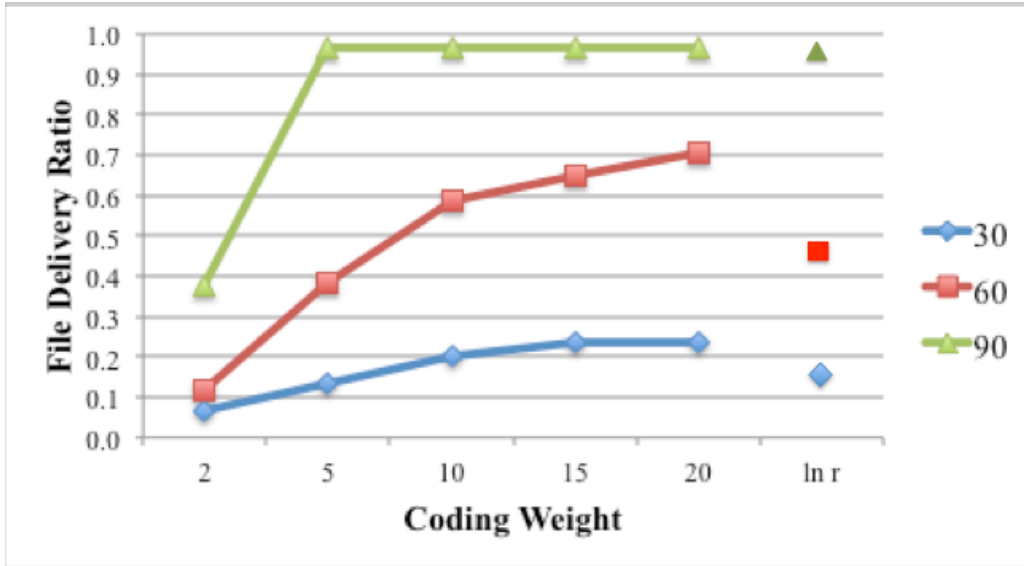


Figure 5.8. FDR of the EBR with network coding scheme varying the re-encoding weight under the NYC scenario with 100MB files

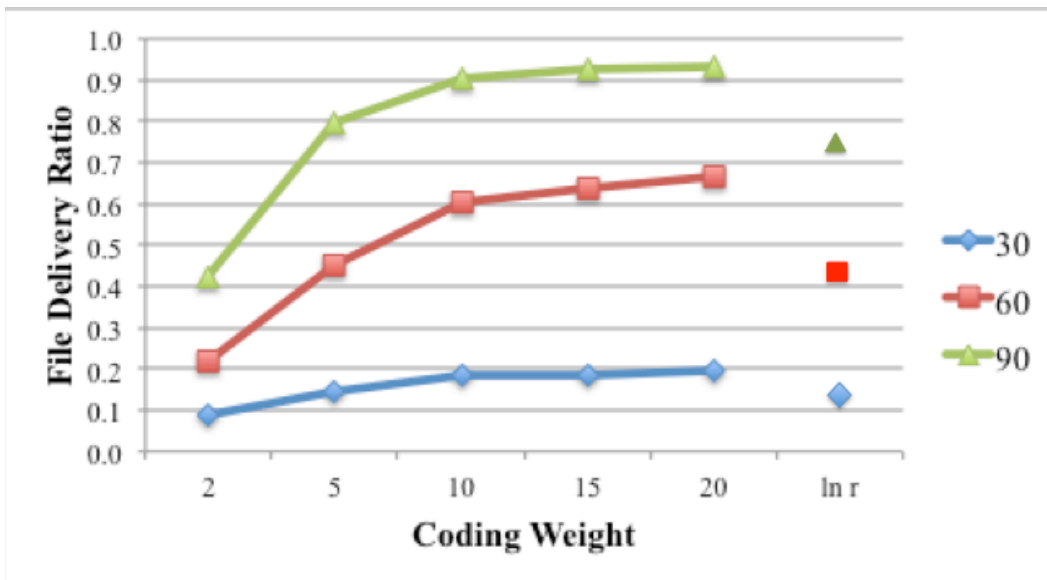


Figure 5.9. FDR of the EBR with network coding scheme varying the re-encoding weight under the NYC scenario with 10MB files

5.3 Benefit of Coding Technique

In this section, we evaluate the performance of the ER scheme and the EBR scheme with both source coding and network coding in the various network settings.

For the EBR scheme, in every scenario we set both of encoding and re-encoding weight as 15. Also, the EBR scheme with source coding mode does not allow the relay nodes to check the rank. We use $\epsilon = 50$ in the stopping condition of receiving the encodings for each file at relays. In order to investigate the benefit from coding under the various network environments, we carried out the extensive simulations with all of four different mobility scenarios that are described in Chapter 3.

Let us see the results of the ferry scenario first. Figure 5.10 shows the FDR results under the network with the second setting for the transmission rate and range; 200kB/s and 50m. We use the second setting for file size and generation rate. As shown in figures, the performance largely depends on the mobility scenario. Even under the scenarios with same number of the ferries, the relative performance between three routing schemes is different. However, there are still trends observed in these results.

First, in lightly congested networks, which are demonstrated in Figure 5.10, there is no benefit from the coding scheme observed with respect to FDR. For every mobility scenario, ER outperforms both modes of EBR. Especially, the source coding scheme demonstrates worse performance compare to the ER scheme.

Second, network coding scheme results in very low FDRs against other two routing schemes in the scenarios F2-2 and F3-2. In order to investigate if there is any specific network setting that incurs this result, we analyze the mobility scenarios. Table 5.1 shows the statistics of the ferry mobility scenarios with the transmission range of 100m. Each number is the average out of ten same scenarios with different random seed for the movement, all of which we used for the simulations. As this table

indicates, the scenario F2-2 and F3-2 has relatively short average contact time between nodes. Therefore, network coding could hurt the performance under this network environment where the contact time between nodes is short and the traffic is relatively light.

Third, when the traffic congestion level is higher, the benefit from using coding scheme is clearly demonstrated. Figure 5.11 shows the FDRs of ER and both of EBR schemes when we use the network settings with the first setting for the transmission rate and range; 100kB/s, 50m. The setting for file size and generation rate is same as the one used in Figure 5.10. With these settings, the network experiences higher congestion, which results in lower FDRs. We can see the EBR with source coding outperforms two other routing schemes, no coding scheme (i.e., ER) and network coding scheme.

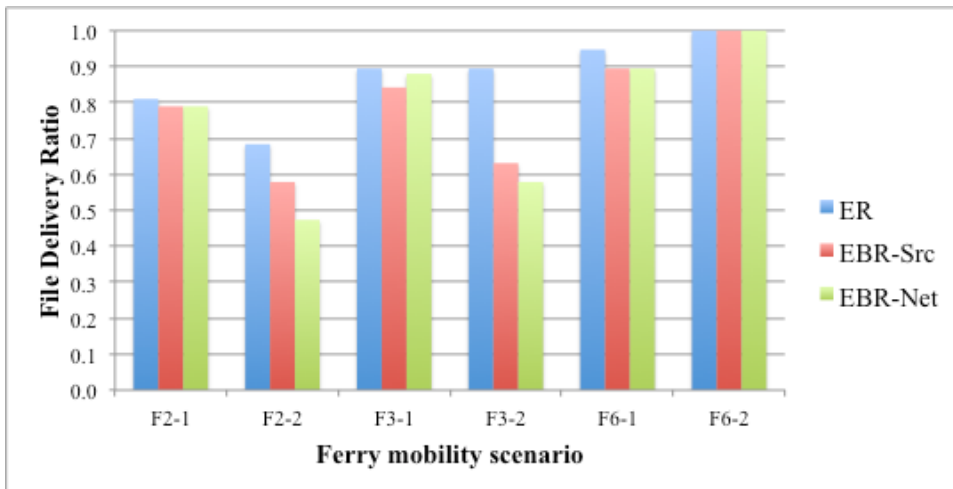


Figure 5.10. FDRs of ER and two modes of EBR scheme under the Ferry scenario with light traffic

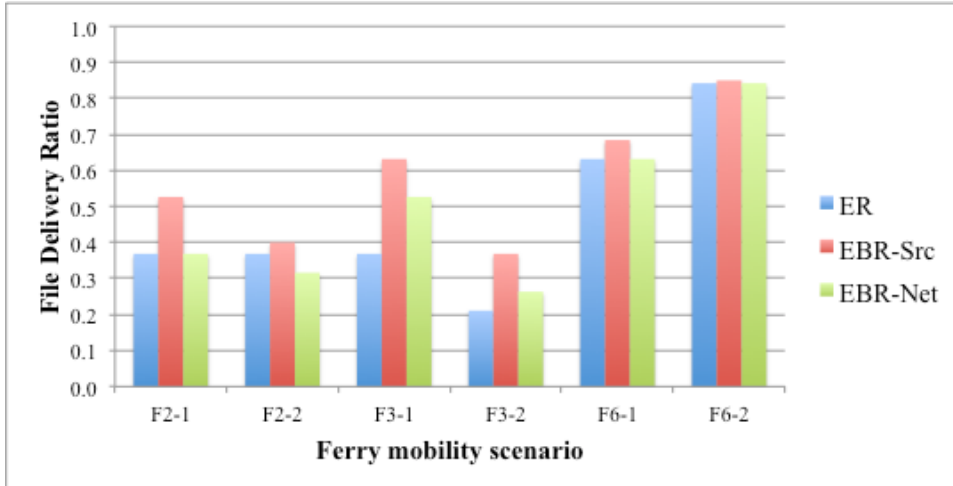


Figure 5.11. FDRs of ER and two modes of EBR scheme under the Ferry scenario with moderate traffic

Scenario	Average contact time	Median contact time	Total contacts
F2-1	169.26	168.32	767.12
F2-2	92.97	92.09	1342.22
F3-1	150.01	138.99	1077.14
F3-2	95.33	94.12	1640.87
F6-1	102.47	85.76	3993.98
F6-2	132.69	113.18	3090.01

Table 5.1. Statistics of the ferry mobility scenarios with the transmission range of 100m

In the NYC scenarios, the map where nodes are moving is much more complicated compared to the ferry scenarios. Also, there is no stationary node, and every node can be a source and a destination, therefore, sources and destinations are also moving in this scenario. Under this homogeneous node setting, the node density affects the network traffic congestion level significantly. With low node density, the traffic is highly congested, while the light traffic is demonstrated with higher node density such as 90 nodes.

Figures 5.12 and 5.13 are results under the second setting for the transmission rate and range; 200kB/s and 50m. We use the first setting for the file size and generation rate in the scenarios of Figure 5.13, and the second setting in Figure 5.12. Figures 5.14 and 5.15 show the results under the third setting for the transmission rate and range; 200kB/s and 100m. For the file size and generation rate, we use the first setting in Figure 5.15, and the second one in Figure 5.14.

In this type of network setting, the coding scheme outperforms the ER scheme when the network is highly congested. It is observed in the Figures 5.12 and 5.13 when number of nodes is 30. The benefit from both of source coding and network coding is significant with respect to both of FDR and ADD. When comparing the performance between source coding and network coding, the results show that source coding works better than network coding in terms of FDR.

When the network traffic is light and the size of file is large such as 100MB, the benefit from coding schemes is observed with respect to the ADD. Figure 5.14 shows that with all of three node density settings, both of coding schemes demonstrate better ADD than the ER while the differences in FDR between the three routing schemes are negligible.

When the network traffic is light and the size of file is small, there is no benefit from the coding technique observed. The ER scheme shows always the best performance regardless the node density and file generation setting. Especially, it is not advisable to use the EBR with source coding when the traffic is light and the size of files is small. Figure 5.15 shows that the source coding scheme demonstrates

the worst performance in all of three node density settings with respect to both of the performance metrics.

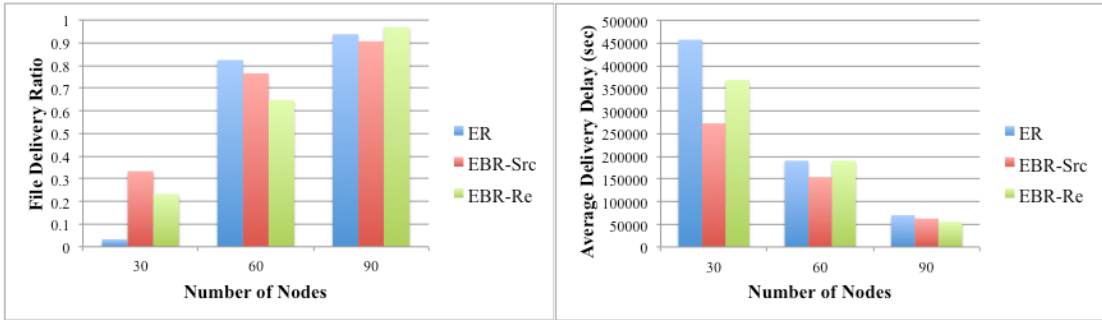


Figure 5.12. FDRs and ADDs of ER and two modes of EBR scheme under the NYC scenario with 100MB files

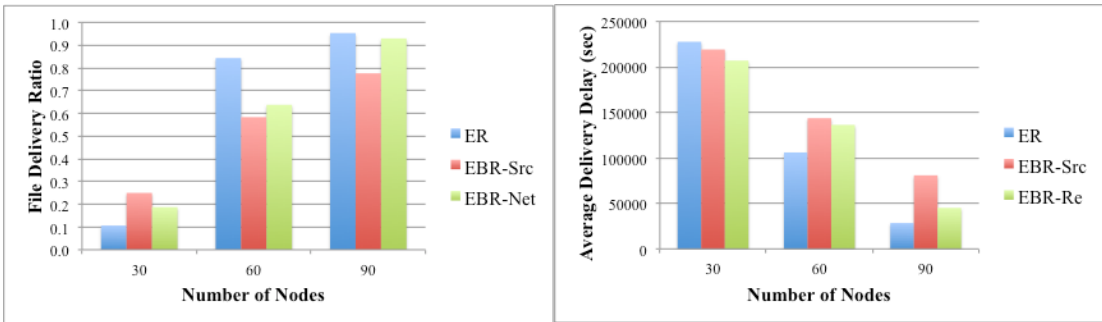


Figure 5.13. FDRs and ADDs of ER and two modes of EBR scheme under the NYC scenario with 10MB files

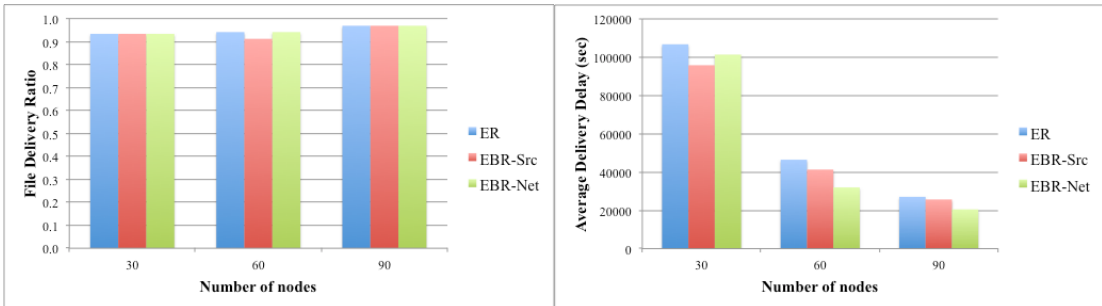


Figure 5.14. FDRs and ADDs of ER and two modes of EBR scheme under the NYC scenario with 100MB files

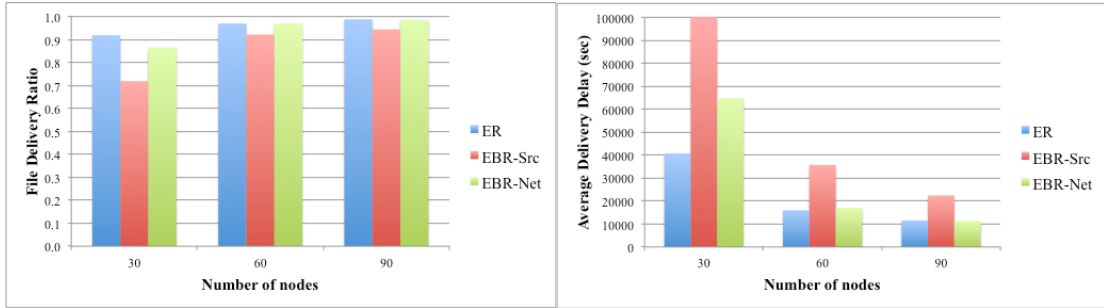


Figure 5.15. FDRs and ADDs of ER and two modes of EBR scheme under the NYC scenario with 10MB files

The Istanbul scenario is an island hopping mobility model on map of the city of Istanbul. There is no specific traveler node between islands, but we set three sub-areas that are overlapped each other, so nodes move around in each sub-area meet another group of nodes in the overlapped area. Also, the source and destination are picked among the three stationary nodes located in each sub-area. Figure 3.3 shows the location of each stationary node. Since the locations of destinations do not fall into the overlapped areas, the only way they receive the bundles destined to them is via the moving nodes in the same sub-area when the transmission range is small enough. With our settings in Table 3.2, each destination can communicate only with nodes moving in the same sub-area. Thus, the moving nodes meet other moving nodes from another sub-area, and convey the bundles to the stationary destination in their sub-area.

Figure 5.16 plots FDRs and ADDs of delivered files of three different schemes under the Istanbul scenario with the first setting for the transmission rate and range; 200kB/s and 100m, and the second setting for the file size and generation rate; 100MB files are generated according to a Poisson process with the rate of 0.0005 file/s. Total number of files generated is 24 during the whole simulation time of a

week (i.e., 604800 sec). The setting for Figure 5.17 is same as Figure 5.16 except the traffic generation. We use the first setting; 10MB files are created with 0.0005 file/s and total number of files generated is 325. Figures 5.18 and 5.19 show the results under different transmission rate setting, 500kB/s.

In the Istanbul scenarios, depending on the level of traffic congestion, the recommended routing scheme is different. First of all, with the high traffic congestion, the EBR routing with source coding delivers the files best. When we see the FDRs in Figure 5.16 when the number of nodes is 30 or FDRs in Figure 5.17 when the node density is set to 30 and 60, the EBR with source coding works best over other two routing scheme. Also, it seems that the ER scheme is not suitable when the network is highly congested with the low transmission rate and large number of files. Figure 5.17 shows that both of EBR schemes outperform the ER scheme significantly.

Second, when the traffic congestion is moderate, network coding enabled EBR scheme performs best among these three routing schemes. It is observed from the FDR results in Figure 5.16 when the number of nodes is 60 and 90, the same ones in Figure 5.17 when the node density is set to 90, and Figures 5.18 and 5.19 when the number of nodes is 30 and 60.

Lastly, when the network is lightly congested and the node density is high, no benefit from coding scheme is demonstrated. The ER scheme shows better performance than both of EBR schemes in Figures 5.18 and 5.19 when the node density is set to 90.

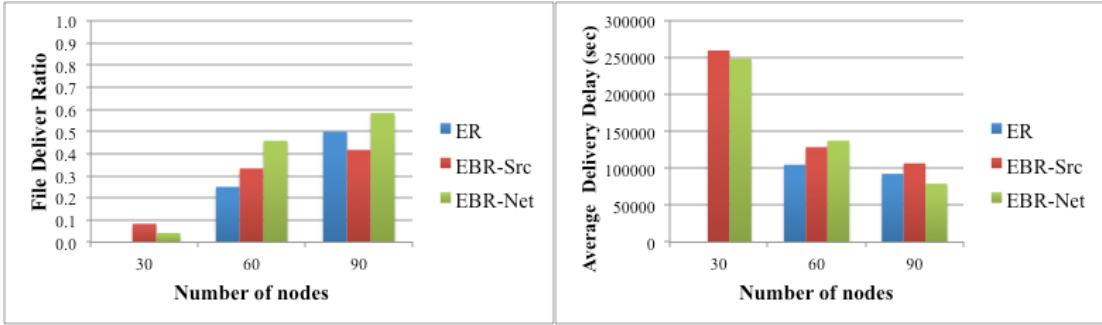


Figure 5.16. FDRs and ADDs of ER and two modes of EBR scheme under the Istanbul scenario with 24 100MB files

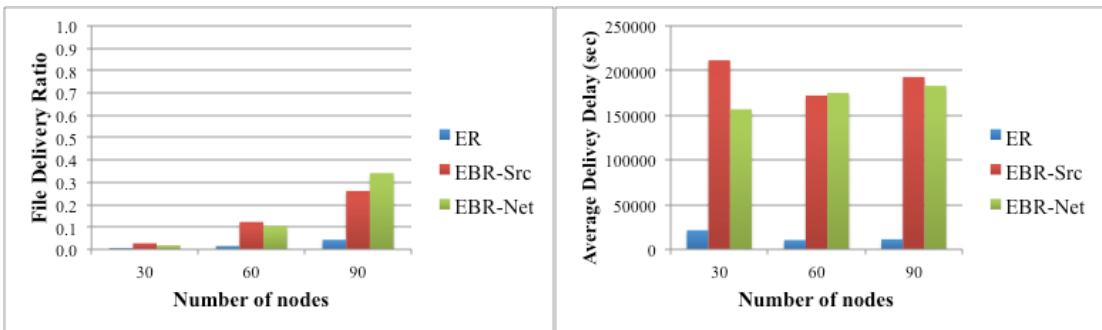


Figure 5.17. FDRs and ADDs of ER and two modes of EBR scheme under the Istanbul scenario with 325 10MB files

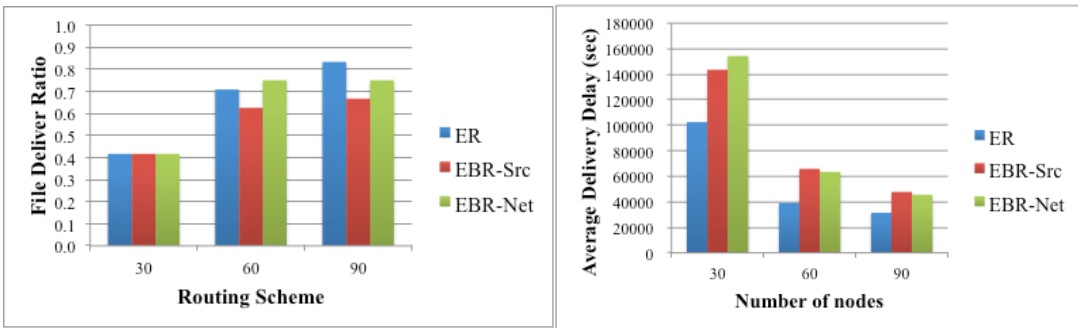


Figure 5.18. FDRs and ADDs of ER and two modes of EBR scheme under the Istanbul scenario with 24 100MB files

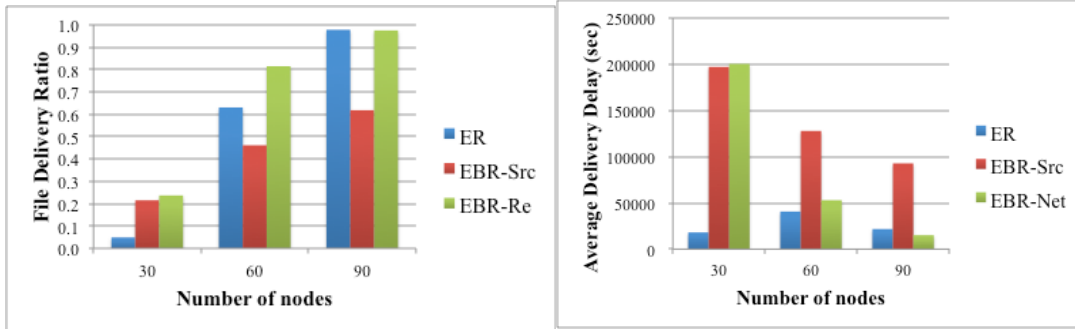


Figure 5.19. FDRs and ADDs of ER and two modes of EBR scheme under the Istanbul scenario with 325 10MB files

The taxi trace in San Francisco is a real life mobility trace collected in 2008. Table 5.2 shows some statistics of the trace mobility when we set the transmission range to 50m, 100m, and 150m. As explained in Section 3.4.4, we process the trace data by selecting only a part of each taxi’s trace and interpolating the intervening location data. We generate the numbers in Table 5.2 with this processed data by checking when a link is up and down between nodes. One remarkable characteristic is that there is a huge gap between the mean and median contact time. Also, note that each simulation is conducted with identical node mobility since we use the same real trace data for each run. The traffic generation scenario used for the taxi trace is very similar to the NYC case, where nodes are homogeneous. Every node can be a source and a destination.

Transmission range	Mean contact time	Median contact time	Total contacts
150m	793.79	28.00	241002
100m	722.88	24.00	194482
50m	426.06	18.00	141455

Table 5.2. Statistics of the taxi trace mobility scenarios

Figure 5.20 shows the results when we use the second setting for the file size and generation rate, and Figure 5.21 shows the results when using the first setting. The trends in the relative performance between the three routing schemes in the taxi mobility are very similar to those of the Istanbul mobility scenario. First, when the file size is big, so the number of chunks in a file is relatively large such as 1000, the source coding scheme is the best with high traffic, and the network coding scheme is better than others in less congested networks. Regardless of the traffic congestion level, we observe that there are benefits from using the coding technique. Moreover, the result of small files in the taxi trace is very similar to the result in the Istanbul scenario with moderate traffic congestion, which is shown in Figure 5.19. When the size of files is relatively small, the source coding even hurts the performance. However, the network coding improves the performance with respect to both of FDR and ADD.

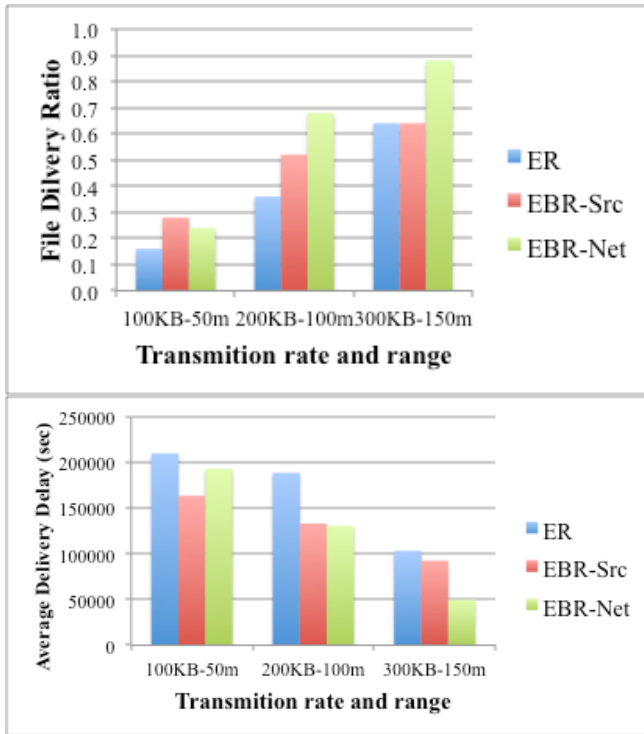


Figure 5.20. FDRs and ADDs of ER and two modes of EBR scheme under the taxi trace scenario with 100MB files

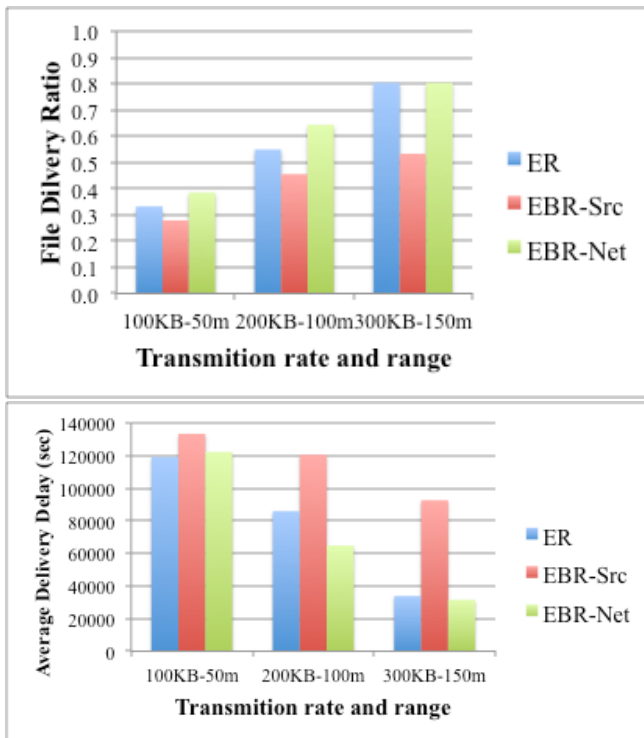


Figure 5.21. FDRs and ADDs of ER and two modes of EBR scheme under the taxi trace scenario with 10MB files

Chapter 6: Immunity Mechanism

ER scheme is a replication-based routing scheme. Since EBR with source coding is also built on the ER, it relies on flooding the network with multiple copies of the same bundles. Even the EBR with network coding scheme also keeps generating encodings belonging to same file. Once these copies of bundles or new encodings are generated, following contacts between nodes, they stay in the network until they are dropped due to (i) buffer overflow at the nodes or (ii) expiration of time-to-live (TTL). Because we do not consider both of the buffer constraint and the file expiration in this dissertation, the number of bundles in the network keeps increasing as time goes unless there is a mechanism that deletes the bundles.

For the nodes may not be aware of the delivery of certain bundles or files, they cannot use such information to determine which bundles to safely remove from their buffer. And, when they choose bundles to transfer to other node, they may pick bundles that have already been delivered even though they are carrying bundles that have not been delivered to the destinations and need to be spread out quickly. Such a bundle transfer is the waste of transmission time, and it could be a critical issue to the performance especially when the contact time is very limited.

It is clear that if the nodes were made aware of the list of files and/or bundles delivered to their destinations, such information could be exploited to help nodes make better use of limited contact times between nodes. This is the basic intuition behind the two immunity mechanisms we describe in this chapter. The first immunity mechanism, BBI, is based on the unique bundle IDs. The second immunity

mechanism, UBI, uses the UUIDs that identify the file from which the bundles were constructed.

We investigate the benefit of the immunity mechanisms in various network scenarios. We employ the immunity mechanisms on not only the ER scheme but also two different modes of the EBR scheme when applicable. In Section 6.1, we study the BBI in detail. The performance gain from the compression technique over the BBI is also discussed in the same section. In Section 6.2, we investigate the UBI.

6.1 Bundle Based Immunity Mechanism

6.1.1 Implementation Details

In the ER and the EBR with source coding, the communication between two nodes at their encounter is as follows: first, nodes exchange summary vectors. Then, by comparing the summary vectors, each node determines the bundles to transfer, which the other node does not have. The BBI is implemented by modifying this simple protocol. Similar to the summary vector, nodes maintain an immunity list, which consists of the bundle IDs that are successfully delivered to their destination. And, they exchange the immunity list as a part of the control message like the summary vector before starting the real data communication.

When a BBI-enabled destination receives a bundle, it generates a bundle immunity message (BIM) with the bundle ID and adds the BIM to its immunity list. In the immunity list, we append ‘|’ as a delimiter between BIMs. Once a BIM is generated, it is propagated throughout the network using flooding. After the exchange of immunity lists between a pair of BBI-enabled nodes, they first scan their buffer to

see if there is any bundle to purge. If any bundle matches a bundle ID in the immunity list of the other node, they eliminate it. This ends up purging the unnecessary copies of delivered bundles from the network. Then, they update their immunity list by merging two immunity lists together.

After processing this immunity mechanism, nodes follow the original routing schemes such as the ER and the EBR with source coding. Note that BBI is not applicable to the EBR with network coding because it does not replicate bundles (i.e., there is no bundle that has the same bundle ID with another one).

6.1.2 Simulation Results

We examine the benefit of the BBI when it is employed onto both of the ER and the EBR with source coding scheme in the following sub sections. To investigate the performance gain due to the BBI on each routing scheme in various network settings, we carried out simulations using the ferry and NYC mobility scenarios. Since the BBI generates a large number of BIMs as the simulation is going on and bundles are delivered to their destination, it is a challenge to run a simulation compared to other scenarios. First of all, it needs a large size of virtual memory to run the simulator, which is developed in java. Second, it takes very long time to finish a simulation, especially with a setting of the light traffic and high node density.

6.1.2.1 BBI mechanism with ER Scheme

Let us check the ferry scenario first. The results are shown in Figures 6.1 to 6.4. We adjust the traffic congestion level by changing the transmission rate and the range. Figures 6.1 and 6.2 show results from the simulation with the first setting for

the transmission rate and range; 100kB/s and 50m. Figures 6.3 and 6.4 are the result of the third setting; 200kB/s and 100m. We use the first setting for the file size and generation rate for Figures 6.2 and 6.4, and the second setting for Figures 6.1 and 6.2.

In the ferry mobility scenario, when the number of chunks in a file is large such as 1000, the BBI helps the ER scheme to improve the performance in overall network environments with respect to FDR. See Figures 6.1 and 6.3. In particular, when the network is more congested, the performance gain due to the BBI is more significant. However, if the files are small, so the number of chunks in a set is also small such as 100, the BBI is not beneficial to the ER scheme in some network settings. Referring to Figure 6.2, in the scenarios with two or three ferries, the BBI even hurts the performance of ER. There results show that the BBI is not suitable for the ER when the number of chunks is relatively small and the traffic congestion is high. Figure 6.4 shows the results in the lighter traffic congestion, which brings the noticeable performance gain due to the BBI.

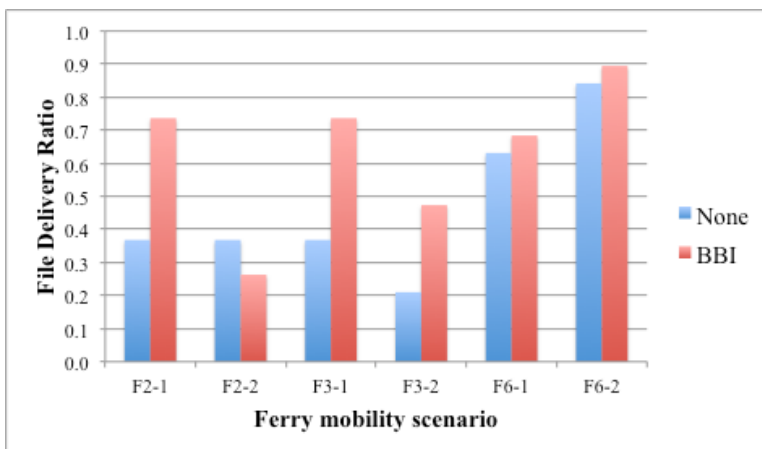


Figure 6.1. FDRs of ER scheme with and without BBI under the ferry scenario when the network is moderately congested with large files

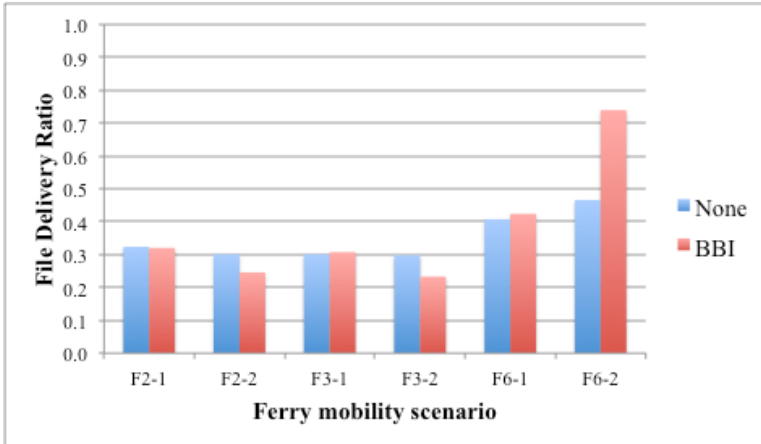


Figure 6.2. FDRs of ER scheme with and without BBI under the ferry scenario when the network is moderately congested with small files

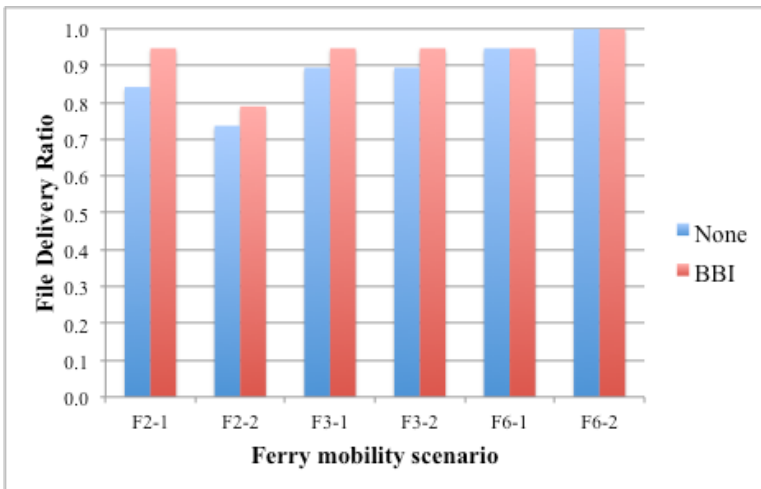


Figure 6.3. FDRs and ADDs of ER scheme with and without BBI under the ferry scenario when the network is lightly congested with large files

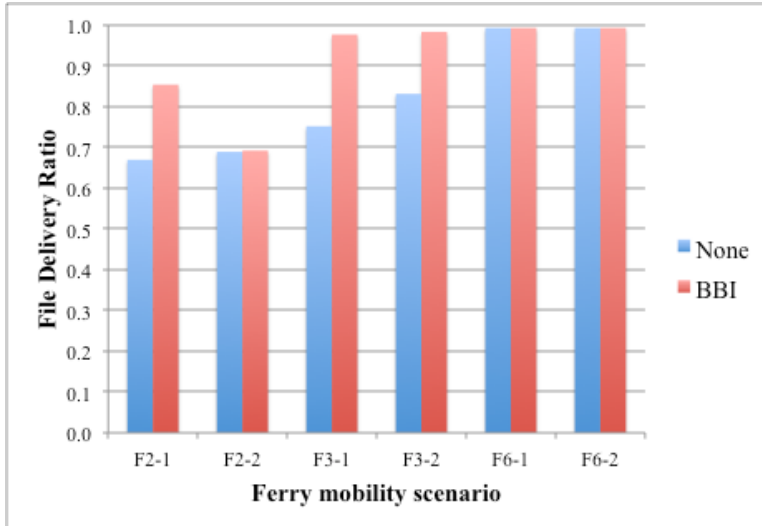


Figure 6.4. FDRs and ADDs of ER scheme with and without BBI under the ferry scenario when the network is lightly congested with small files

Now let us study the more crowded network setting, the NYC scenario.

Figures 6.5 and 6.6 plot the results from the setting where the transmission rate is 200KB/S and the range is 50m. The first setting for the file size and generation setting is used. Depending on the node density, they show three different levels of network congestion scenarios.

In the complex mobility with higher node density such as the NYC scenario, with respect to the FDR, the benefit from BBI to the ER scheme is significant only when the node density is relatively low. Figure 6.5 shows that the network traffic is light with higher node density like 60 or 90 nodes (i.e., the FDR is higher when the node density is high). In such network settings, FDRs of the ER scheme with and without BBI fall into the confidence interval of each other, which are calculated with 90% of confidence level. However, when the node density is lower like 30 nodes, the BBI increase the FDR of the ER significantly.

With respect to the ADD, the performance gain due to the BBI is noticeable unless the node density is very high. Figure 6.6 shows that the BBI improves the performance of ER in the network with 30 or 60 nodes, while it fails to improve in the setting of 90 nodes. These results show that the BBI is not beneficial to the ER when the network traffic is too light so most of bundles are delivered to the destination very quickly even before the BBI starts taking effect.

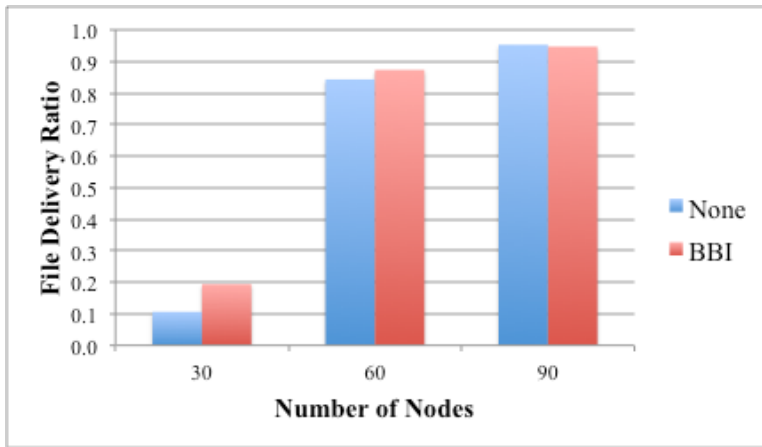


Figure 6.5. FDRs of ER scheme with and without BBI under the NYC scenario

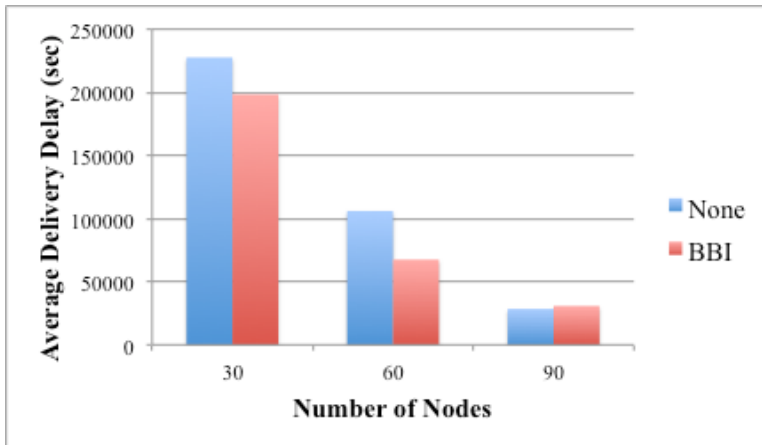


Figure 6.6. ADDs of ER scheme with and without BBI under the NYC scenario

6.1.2.2 BBI Mechanism with EBR with Source Coding Scheme

The simulation results show that the BBI is not beneficial, but rather harmful to the EBR scheme with respect to both of FDR and ADD in overall network settings. This result contradicts claims in literature, which say that the BBI improves the performance of every replication-based routing scheme.

As we study the performance gain due to the BBI on the ER scheme in the previous section, we also use the ferry and NYC mobility to examine the benefit of BBI to the EBR. We are very confident that the similar results will be demonstrated in other network settings. We use the third setting for the transmission rate and range; 200kB/s and 100m, in the ferry scenario. In the NYC scenario, we select the second one, which is 200kB/s and 50m. Figures 6.7 and 7.8 show the simulation results run in the second setting for the file size and generation rate; 100MB and 0.00005 file/s.

In Figures 6.7 and 6.8, it is shown that the BBI hurt the performance significantly in regards both of FDR and ADD. In particular, under the NYC scenario, the amount of performance debasement in FDR due to the BBI is significant. These results clearly tell us that the use of BBI is not suitable for the EBR scheme.

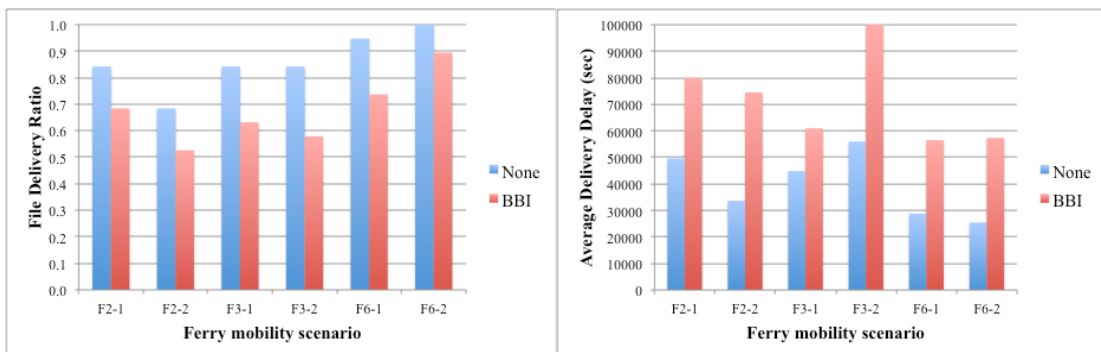


Figure 6.7. FDR and ADD of the EBR scheme with and without the BBI in the ferry scenario with transmission rate of 200kB/s, range of 100m, file size of 100MB, and file generation 0.00005 file/s

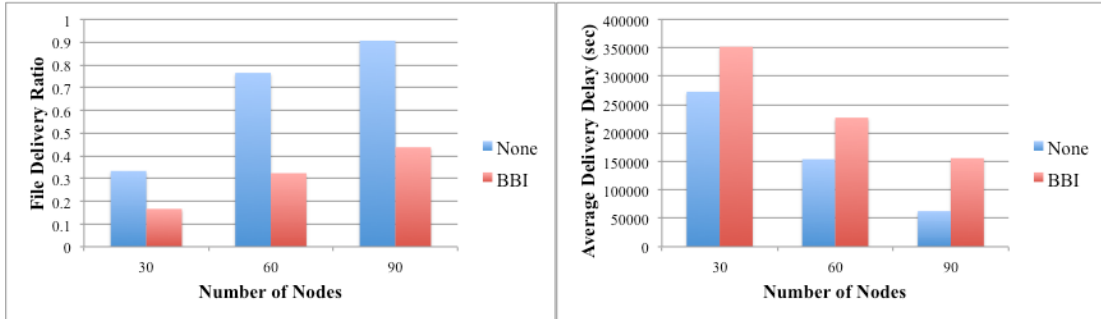


Figure 6.8. FDR and ADD of the EBR scheme with and without the BBI in the NYC scenario with transmission rate of 200kB/s, range of 50m, file size of 100MB, and file generation 0.00005 file/s

Let us explain the rationale behind these results. Purging the encodings from the nodes by the BBI induces unwanted situations. It is because nodes are aware of only the delivery of bundles, but not of the delivery of files. When a file is delivered, at least N encodings are delivered to the destination, where N is the number of chunks of the file. Then, nodes immunized for the delivered bundles remove them from the queue associated with the delivered file. However, because the nodes do not know the file is already delivered, they keep filling the queue with the new encodings for the file until having $N + \epsilon$ encodings in the queue. These new encodings are totally unwanted copies in the network that result in the waste of resource, especially the contact time. This behavior of nodes hurts the performance of the EBR scheme.

6.1.3 Compression over the Immunity List

As the bundles are delivered to the immunity mechanism(s) enabled destination, immunity messages keep generated in the network. Even though the immunity list consists of only the bundle IDs, the size of immunity list keeps increasing, and at some point, it would take significant amount of the contact time to

just transmit the immunity list between nodes. Especially, when the bundles have very long TTL like our setting (no expiration), there is no chance to remove the immunity messages in the network.

It is our motivation of applying the compression technique on the immunity list to save the transmission time for immunity lists. Unlike the summary vector, the Bloom filter, which is a space-efficient data structure, is not suitable for the purpose of the immunity list because the immunity lists need to be reconstructed without any loss. In order to reduce the size of the immunity list, we chose Lempel-Ziv-Welch (LZW) [31], which is a universal lossless data compression algorithm. We investigate how much this compression algorithm could reduce the immunity list resulting in saving the contact time and how this affects the performance of the ER scheme. Because the BBI hurts the performance of the EBR scheme, we are not interested in studying the impact of the compression technique on the immunity lists of the EBR scheme.

6.1.3.1 Simulation Results

To study the benefit from the compression technique on the BBI, we investigate the performance of the ER scheme that is enabled the BBI with and without the compression employed. We use the ferry and NYC mobility and change the network settings in order to investigate three different levels of traffic congestion. Figure 6.9 shows the result under the ferry mobility scenarios with the first setting for the transmission rate and range; 100kB/s and 50m, and the second setting for the file size and generation rate; 100MB and 0.00005 file/s. Figure 6.10 displays the results of simulations that use the NYC mobility with the second setting for the transmission

rate and range; 200kB/s and 50m, and the second setting for the file size and generation rate; 100MB and 0.00005 file/s.

As shown in Figure 6.9, compression technique does not improve the performance significantly with respect to FDR in the ferry scenarios. However, in terms of ADD, using compression brings performance improvement in every ferry mobility scenarios.

Figure 6.10 shows that benefit from compression is demonstrated more noticeably in the NYC settings. Not only the ADDs but also the FDRs are improved when compression technique is employed. Especially, when the network is congested highly with 30 nodes, the benefit is enlarged with respect to both of FDR and ADD.

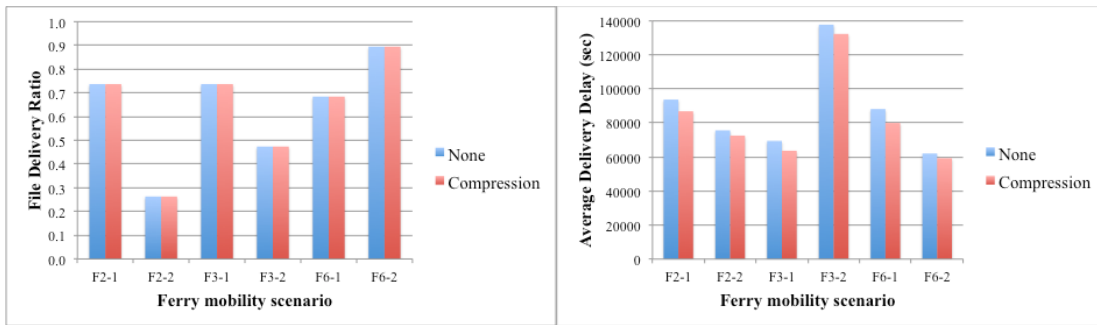


Figure 6.9. FDR and ADD of the ER scheme with BBI that the compression technique is employed and not employed in the ferry scenario

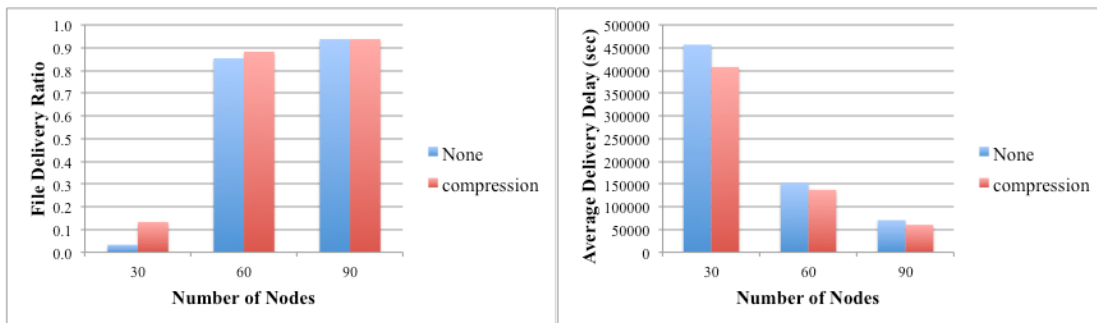


Figure 6.10. FDR and ADD of the ER scheme with BBI that the compression technique is employed and not employed in the NYC scenario

6.2 UUID Based Immunity Mechanism

6.2.1 Implementation Details

The UBI is similar to the BBI in principle. Instead of a bundle, it works based on a file. When a large file is broken up into chunks and either chunks or encodings are forwarded in separate bundles, we insert the UUID associated with the file in the extension block of each bundle generated from that file. When the destination of the file successfully reconstructs the file and has UBI employed (i.e., the UBI mechanism is working at the destination), it generates a UIM that contains the UUID of the file, and adds the UIM into the UUID immunity list maintained by the node. In our EBR implementation, a node maintains separate immunity lists for the UBI from the BBI.

In the UBI, nodes purge bundles in a different way from the BBI. When a node receives the immunity list from its neighbor, it removes all the bundles with a UUID that matches any new UIM received from the neighbor. And then, it updates its own UUID immunity list by merging two UUID immunity lists together. Therefore, exchange of UIMs allows the nodes to safely remove all unwanted bundles containing the chunks or encodings that came from files, which have been already delivered to their destinations.

Unlike the BBI, the UBI is applicable to any routing scheme when the fragmentation is employed in the network. Therefore, UBI is suitable not only to the ER and the EBR with source coding scheme, but also to the EBR with network coding scheme.

6.2.2 Simulation Results

In this section, we will investigate the performance gain due to the UBI on three different routing schemes, the ER and EBR with source coding and network coding scheme. In addition, we study the relative performance between these three routing schemes when the UBI is employed on them in the last section. In order to explore more diverse network environment, we use four different mobility scenarios, the ferry, NYC, Istanbul, and the taxi trace. Even for each mobility scenario, we conduct simulations with different transmission rate and range settings as well as a couple of traffic generations.

6.2.2.1 Benefit of the UBI to ER scheme

Overall, the UBI improves the performance of the ER scheme with respect to both of FDR and ADD. However, the trend is not consistent for every network setting. Let us explain the observed trends at each of mobility scenarios in detail.

In the ferry mobility scenarios, which represent the simple topology and low node density network, the relative performance between the ER scheme with and without the UBI depends on the node mobility considerably. Figure 6.11 shows that the plain ER outperforms the UBI enabled ER in the mobility scenario F2-2, while UBI is beneficial to the ER in every other scenario. The amount of benefit from the UBI is also different per mobility scenario. In Figure 6.11, the benefit due to the UBI is significant under the mobility F3-1. In addition, Figures 6.11 and 6.12 show that the fluctuation in the relative FDR between two routing schemes depending on the mobility scenario is larger when transferring larger files.

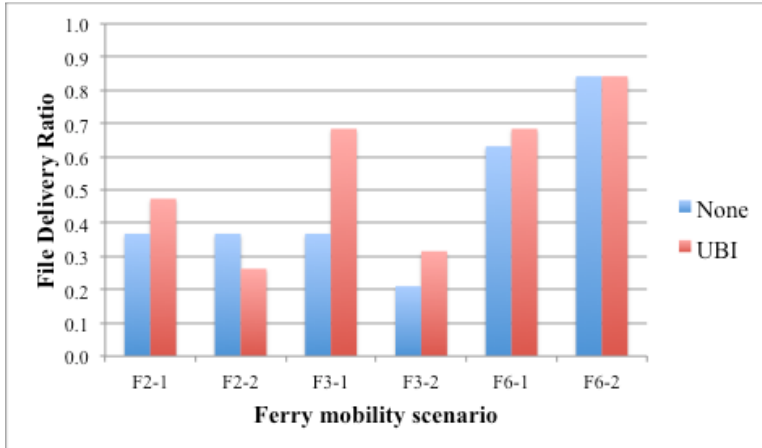


Figure 6.11. FDR of the ER scheme with and without UBI under the ferry mobility when transferring 100MB files

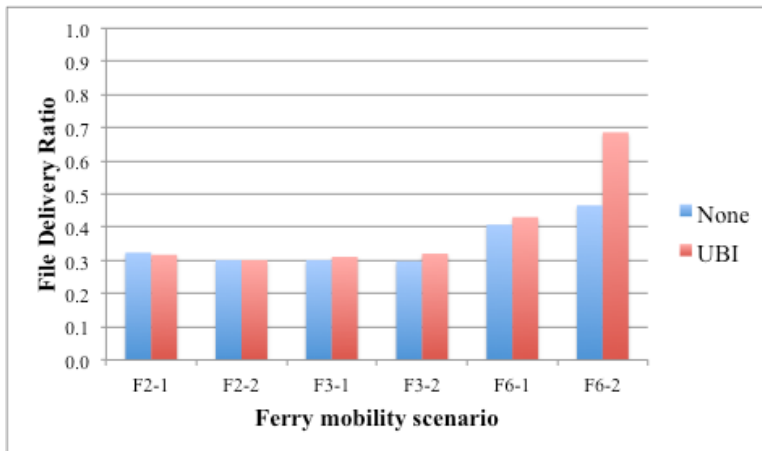


Figure 6.12. FDR of the ER scheme with and without UBI under the ferry mobility when transferring 10MB files

In contrast, the number of ferries (i.e., node density) in the network does not affect the performance. Even though the number of ferries is same, the results are very different between two network settings in Figure 6.11. Furthermore, there is no specific trend in the performance with respect to the level of traffic congestion.

In the NYC mobility scenarios, the UBI achieves a little bit performance improvement in overall. However, when the traffic congestion level is light, the benefit due to the UBI is negligible with respect to FDR as it is observed in Figures

6.13, 6.14, and 6.15. Also, under the light traffic as shown in Figure 6.15, if the node density is high such as 90 nodes, the benefit to the ADD of ER scheme is also negligible.

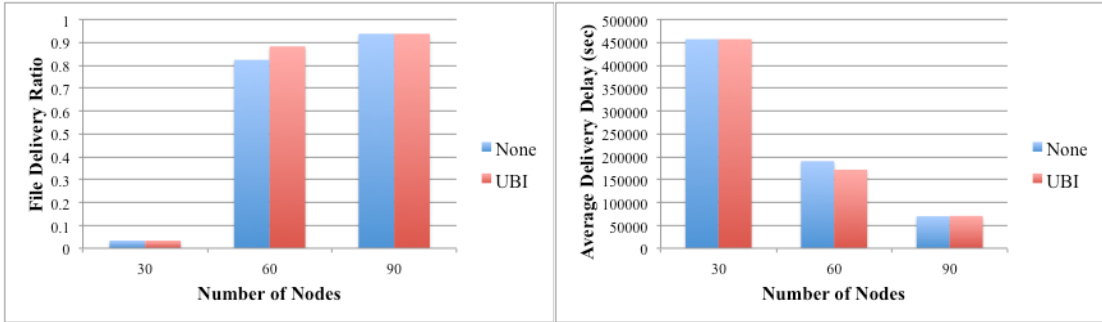


Figure 6.13. FDR and ADD of the ER scheme with and without UBI under the NYC mobility when transferring 100MB files

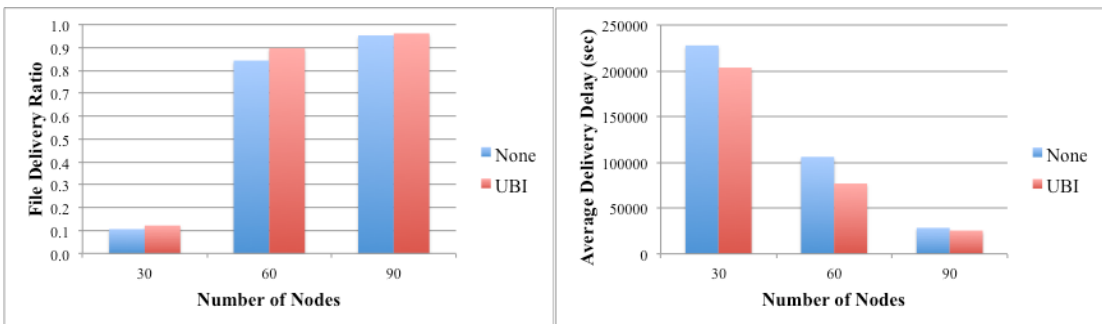


Figure 6.14. FDR and ADD of the ER scheme with and without UBI under the NYC mobility when transferring 10MB files

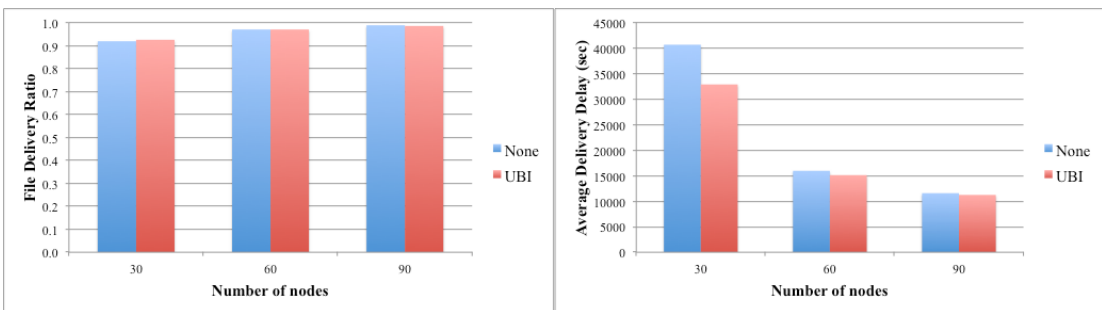


Figure 6.15. FDR and ADD of the ER scheme with and without UBI under the NYC mobility when transferring 10MB files in a lightly congested network

Under the island hopping mobility scenario in the Istanbul city, the UBI also leads the performance gain of the ER scheme, but not very much. Figure 6.16 shows

the FDR results of ER in two different transmission rates. Left graph displays the results when we use the first setting for the transmission rate and range; 200kB/s and 100m, while right one shows the results when we use the second setting; 500kB/s and 100m. Figure 6.17 is the ADD results that are collected from the same setting of the right graph in Figure 6.16.

When the level of traffic congestion is high and the node density is high such as 60 or 90 nodes, the benefit from the UBI is noticeable with respect to FDR as shown in Figure 6.16. However, if the transmission rate is higher resulting in less congested network environment, the benefit due to the UBI with respect to FDR is not demonstrated regardless of the node density. In this network setting, the UBI is beneficial in regard of ADD, which is shown in Figure 6.17. The amount of benefit is significant when the node density is not high like 30 nodes.

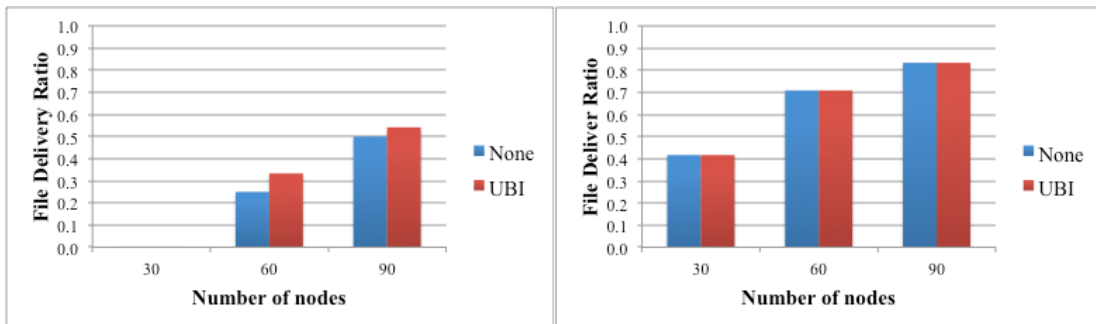


Figure 6.16. FDR of the ER scheme with and without UBI under the Istanbul mobility with two different transmission rates (Left: 200kB/s, Right: 500kB/s)

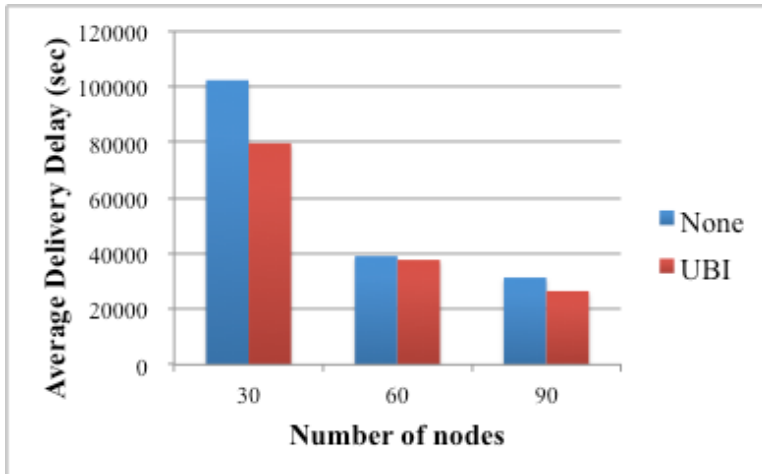


Figure 6.17. ADD of the ER scheme with and without UBI under the Istanbul mobility with light traffic

Now, let us see the results from the simulation when we use the real mobility trace that is collected by taxis in San Francisco. Figures 6.18 and 6.19 show the FDR and ADD of ER with and without UBI enabled when transferring 100MB files and 10MB files respectively. It is clear that, when the UBI is enabled, the ER substantially outperforms the plain ER with respect to both of FDR and ADD in overall network settings.

In Figures 6.18 and 6.19, the benefit from the UBI is most significant in the second setting for the transmission rate and range; 200kB/s and 100m with respect to both of FDR and ADD. It shows that the UBI brings the performance gain to the ER especially in the moderate traffic congestion setting. However, no specific trend is observed regarding the size of files or the number of chunks in a file.

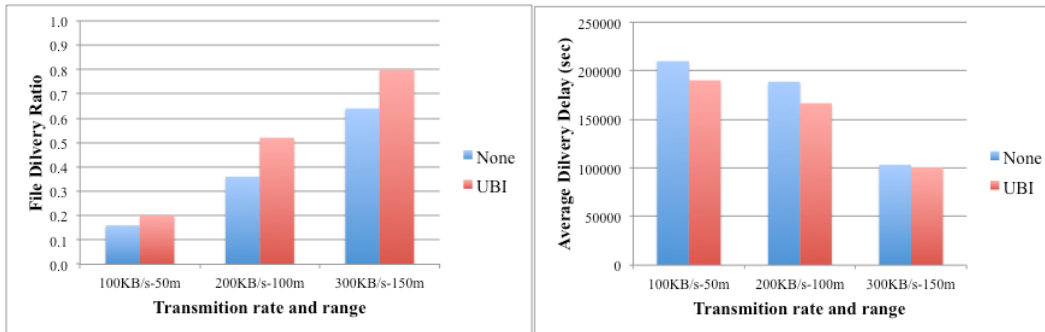


Figure 6.18. FDR and ADD of the ER scheme with and without UBI under the taxi trace when transferring 100MB files

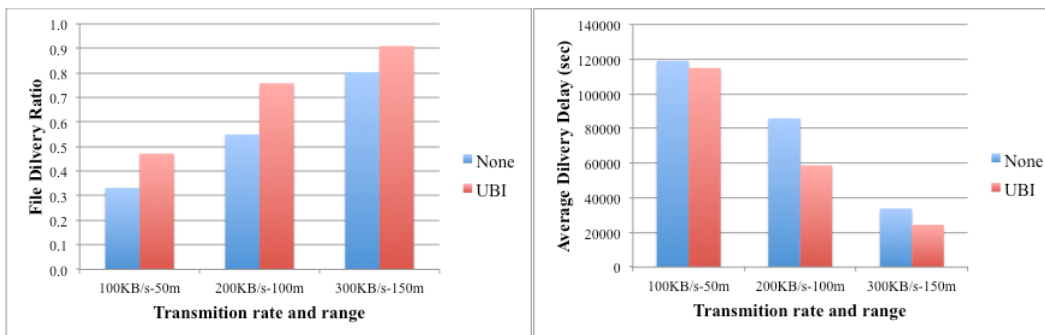


Figure 6.19. FDR and ADD of the ER scheme with and without UBI under the taxi trace when transferring 10MB files

6.2.2.2 Benefit of the UBI to the EBR with Source Coding

In this section, we will study the relative performance between the EBR scheme with and without the UBI enabled in different network settings. The benefit of the UBI is obviously demonstrated when it is employed on the EBR with source coding. Regardless of the mobility scenario, it leads the performance gain significantly.

Figures 6.20 and 6.21 plot the FDRs and ADDs of the EBR in source coding mode under the ferry mobility scenarios. Figure 6.20 shows the results from the scenario where we use the first setting for both of the transmission rate and range and

the file size and generation rate. Figure 6.21 displays ones from the setting we use the second setting for the transmission rate and range.

As shown in Figure 6.20, even though the EBR scheme experiences the performance gain due to the UBI in every mobility scenario, the relative amount of performance improvement largely relies on the mobility pattern. Also, Figure 6.21 shows that the UBI brings more improvement in FDR when transferring smaller files.

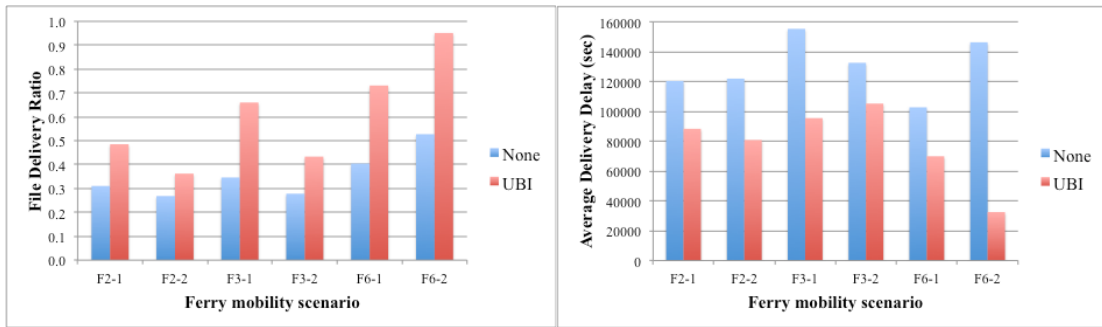


Figure 6.20. FDR and ADD of the source coding enabled EBR scheme with and without UBI under the ferry mobility

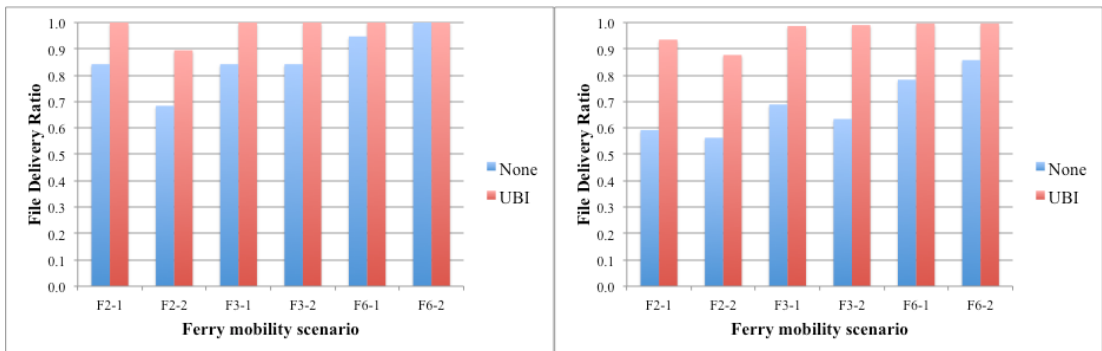


Figure 6.21. FDR of the source coding enabled EBR scheme with and without UBI under the ferry mobility when transferring different size of files (Left:100MB, right: 10MB)

In the NYC mobility scenario, the UBI enabled EBR also outperforms the plain EBR with source coding scheme. Moreover, the same trend we observed with the ER scheme is also observed in the Figures 6.22 and 6.23. The UBI induces the

largest performance gain with respect to both of FDR and ADD when the node density is moderate such as 60 nodes. When the network is very lightly congested like the simulations in Figure 6.24, the UBI is not beneficial with respect to the FDR. But, the benefit in regard of the ADD is clearly demonstrated. In particular, when the node density is relatively low like 30 nodes, the performance gain is significant.

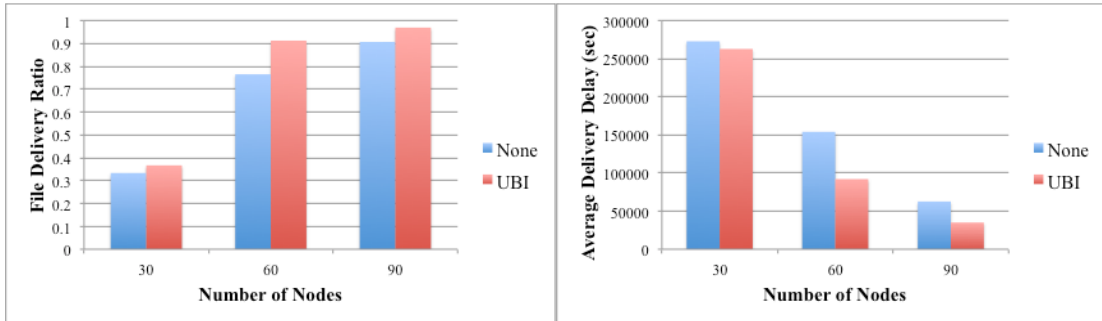


Figure 6.22. FDR and ADD of the source coding enabled EBR scheme with and without UBI under the NYC mobility when transferring 100MB files

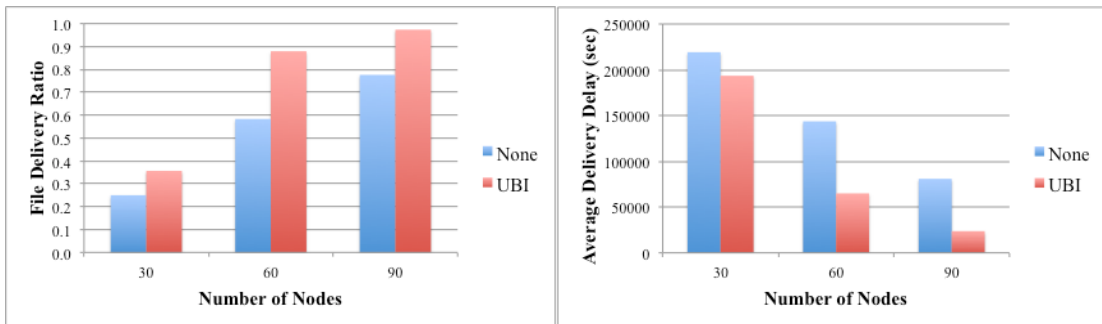


Figure 6.23. FDR and ADD of the source coding enabled EBR scheme with and without UBI under the NYC mobility when transferring 10MB files

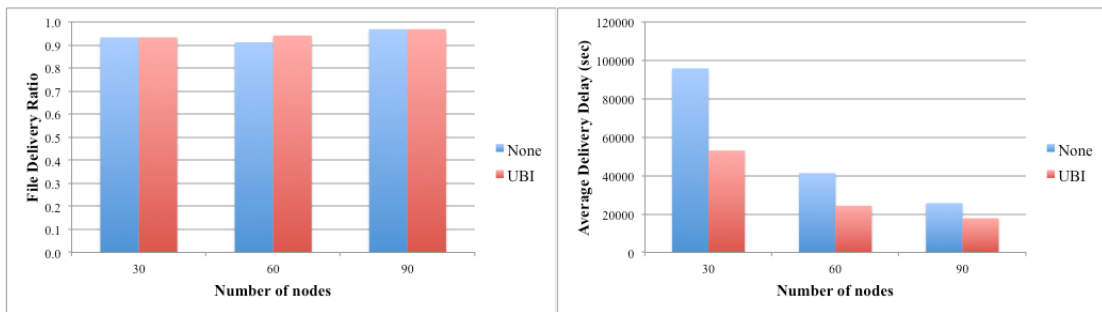


Figure 6.24. FDR and ADD of the source coding enabled EBR scheme with and without UBI under the NYC mobility and light traffic

In the Istanbul mobility, the benefit due to the UBI is clearly observed in every setting we simulate with respect to both of FDR and ADD. Figures 6.25 and 6.26 show the results from simulations run in the second setting for the transmission rate and range; 500kB/s and 100m. For the file size and generation rate, we use the first setting in Figure 6.26 and the second setting in Figure 6.25.

Like the NYC mobility settings, the UBI is most beneficial to the EBR with source coding scheme when the node density is moderate such as 60 nodes. In addition, as shown in Figures 6.25 and 6.26, the EBR with source coding experiences more benefit from the UBI when transferring small files rather than large files. Especially, with respect to the ADD, the performance improvement is significant.

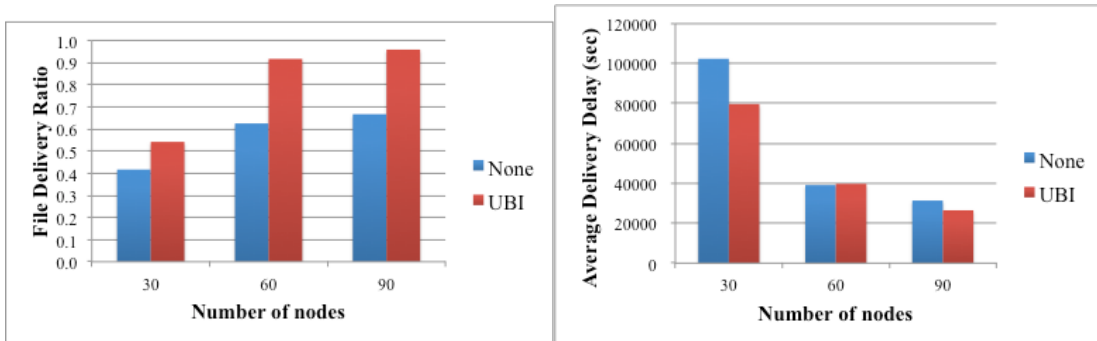


Figure 6.25. FDR and ADD of the source coding enabled EBR scheme with and without UBI under the Istanbul mobility when transferring 100MB files

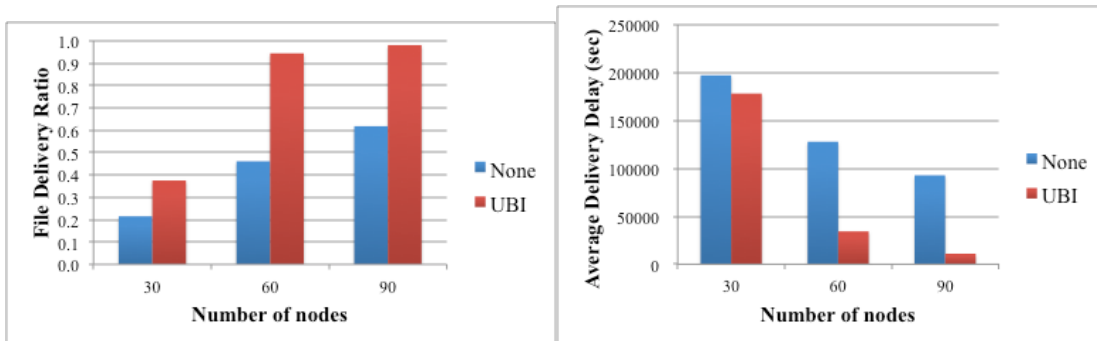


Figure 6.26. FDR and ADD of the source coding enabled EBR scheme with and without UBI under the Istanbul mobility when transferring 10MB files

Lastly, Figures 6.27 and 6.28 show the results from the simulations conducted with the real trace mobility. We vary the setting for the transmission rate and range in order to see the benefit from the UBI in the different traffic congestion levels. Like other mobility scenarios, the UBI is beneficial with respect to both of FDR and ADD in overall network settings regardless the traffic congestion level. Especially when the traffic congestion level is moderate, the benefit due to the UBI is most significant.

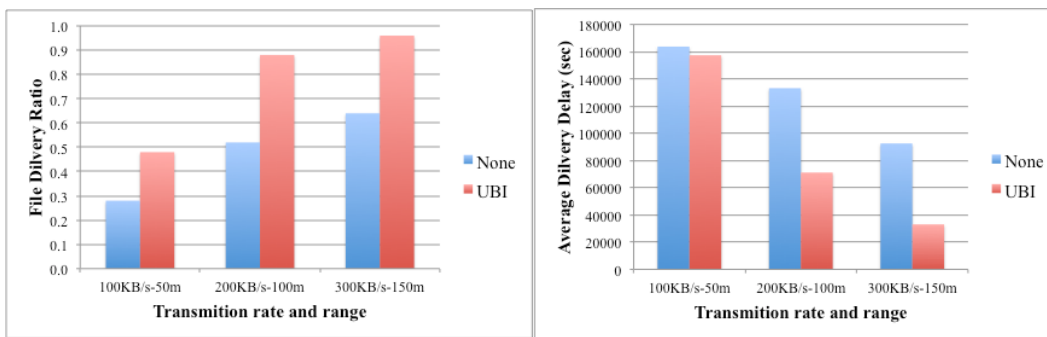


Figure 6.27. FDR and ADD of the source coding enabled EBR scheme with and without UBI under the taxi trace mobility when transferring 100MB files

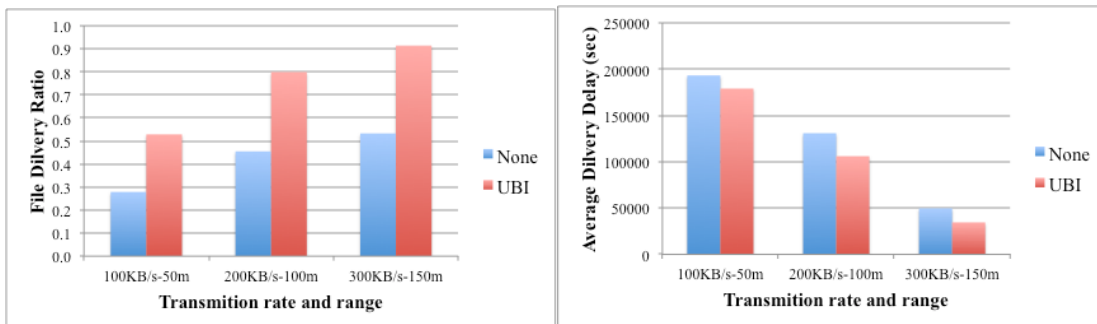


Figure 6.28. FDR and ADD of the source coding enabled EBR scheme with and without UBI under the taxi trace mobility when transferring 10MB files

6.2.2.3 Benefit of the UBI to the EBR with network coding scheme

We examine the benefit from the UBI to the EBR with network coding in this section. Figures 6.29, 6.30, 6.31, and 6.32 show the results from the simulation using different mobility scenarios; ferry, NYC, Istanbul, and taxi trace respectively. We use

the first setting for the transmission rate and range, and the first setting for the file size and generation in every simulation.

In summary, the UBI is beneficial to the network coding enabled EBR scheme as well. However the performance gain is not very significant as we observed in the source coding enabled EBR scheme. In particular, the benefit due to the UBI is not noticeable in the network with an urban map such as NYC and Istanbul mobility scenarios as shown in Figures 6.30 and 6.31. However, in the ferry mobility scenarios, which result is shown in Figure 6.29, the UBI improves notably performance of EBR with network coding with respect to both of FDR and ADD. In addition, the benefit from the UBI is most noticeable in the moderately congested network, where the FDR is between 0.3 and 0.7. The results from the taxi trace mobility in Figure 6.32 also show that the UBI is beneficial to the EBR with network coding with respect to both of FDR and ADD.

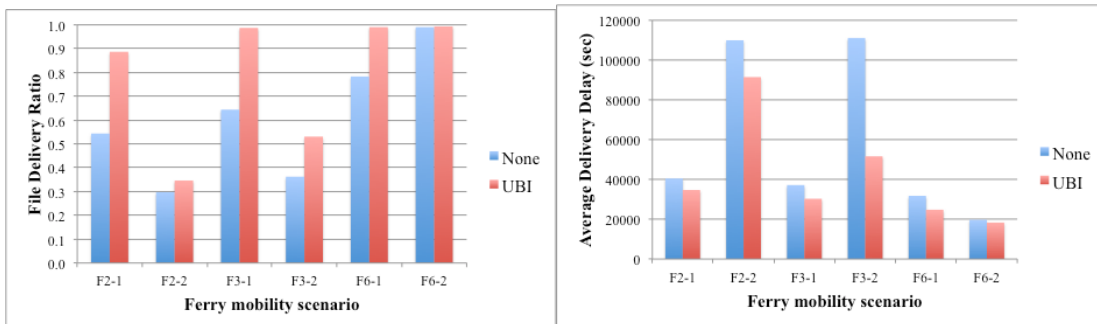


Figure 6.29. FDR and ADD of the network coding enabled EBR scheme with and without UBI under the ferry mobility

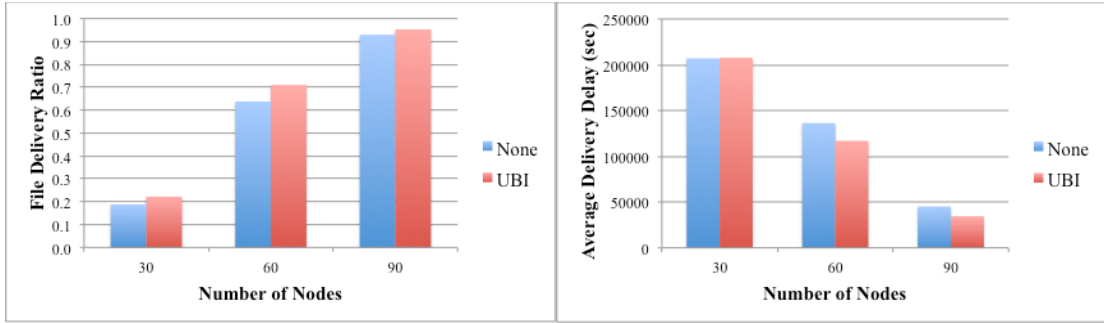


Figure 6.30. FDR and ADD of the network coding enabled EBR scheme with and without UBI under the NYC mobility

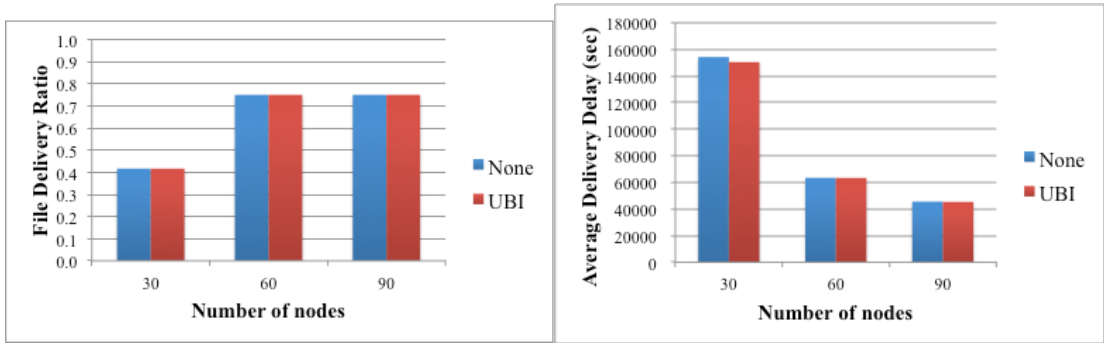


Figure 6.31. FDR and ADD of the network coding enabled EBR scheme with and without UBI under the Istanbul mobility

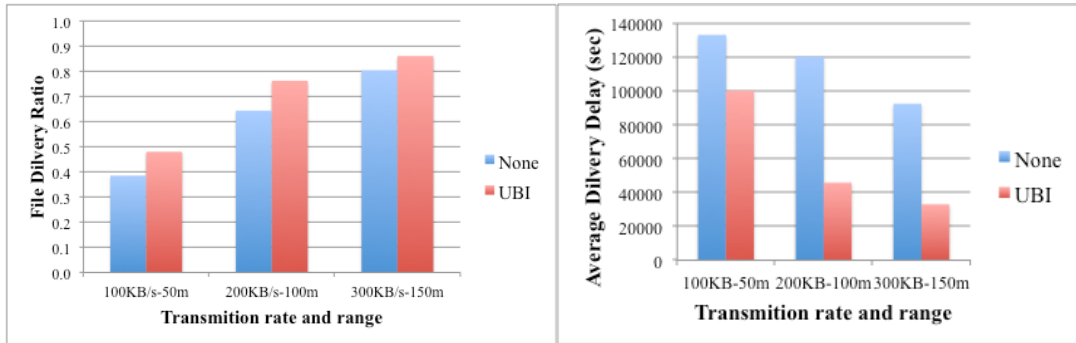


Figure 6.32. FDR and ADD of the network coding enabled EBR scheme with and without UBI under the taxi trace mobility

6.2.2.4 Performance comparison between UBI enabled routing schemes

In this section, we will examine the relative performance of three routing schemes when the UBI is enabled on them. Recall that we study the same when the UBI is not employed in the last section of Chapter 5. We will investigate how the UBI changes the trends.

Above all, the impressive change from the previous study is that the source coding enabled EBR scheme is the best scheme in overall network settings. According to our simulation results, the UBI and EBR with source coding is a very nice combination for transferring files in DTNs.

Let us examine the result of each mobility scenario. First, Figures 6.33 and 6.34 plot the FDR of three routing schemes under the ferry mobility. They show the results from the simulations that we use the first setting for the transmission rate and range. Regardless of the mobility pattern, traffic congestion level, and the size of file, the source coding enabled EBR scheme always shows the best performance. However the ER scheme and the network coding enabled EBR scheme beat each other in different network settings. When transferring large size files, the network coding scheme deliver more files than the ER scheme as shown in Figure 6.33 while the ER scheme works better than the network coding scheme when sending the smaller files as shown in Figure 6.34.

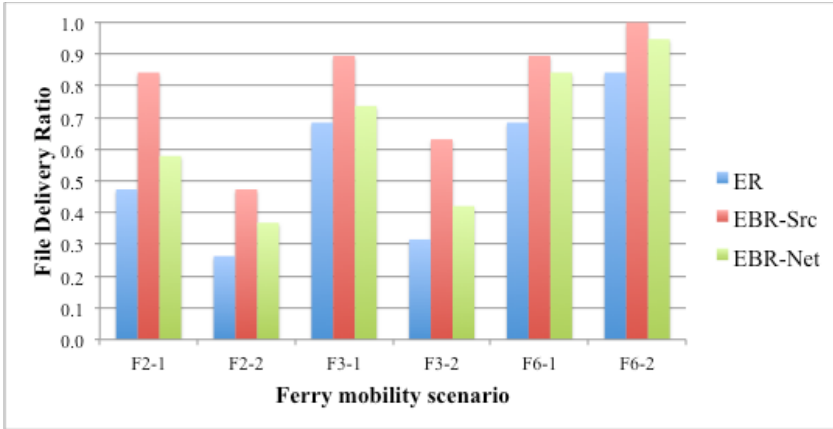


Figure 6.33. FDR of three routing schemes with UBI enabled under the ferry scenario when transferring 100MB files

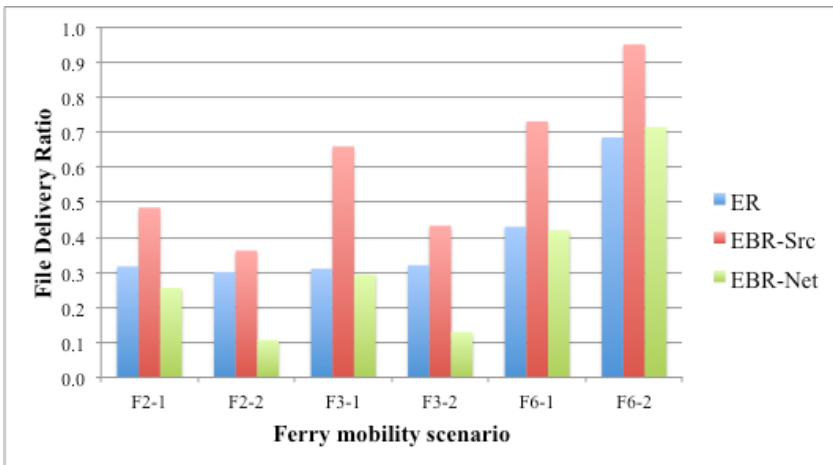


Figure 6.34. FDR of three routing schemes with UBI enabled under the ferry scenario when transferring 10MB files

In the NYC mobility scenario, the source coding scheme is also the best one with respect to both of the FDR and ADD. As shown in Figure 6.35, this trend is more clearly observed under the moderate or high traffic congestion. In Figure 6.36, when the network is lightly congested, the performance difference in terms of FDR is not very clear. But, with respect to the ADD, the source coding scheme demonstrates the best performance. Regarding the relative performance between the ER scheme

and the network coding enabled EBR scheme, it is observed that the network coding scheme brings better performance in terms of ADD than ER under the light traffic.

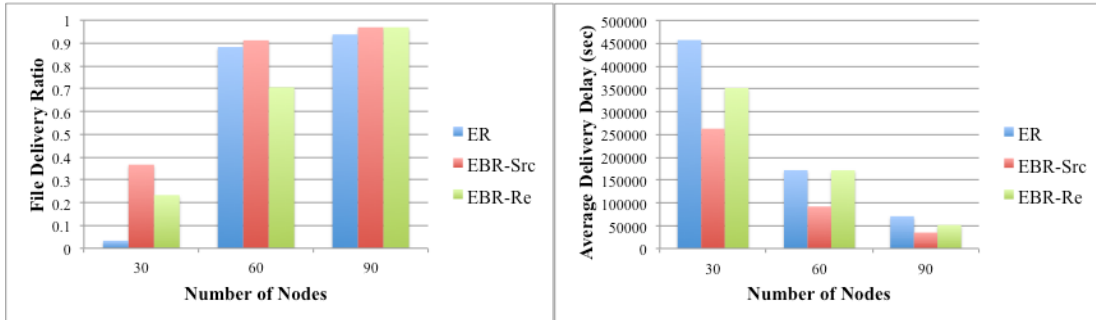


Figure 6.35. FDR and ADD of three routing schemes with UBI enabled under the NYC scenario with moderate traffic congestion

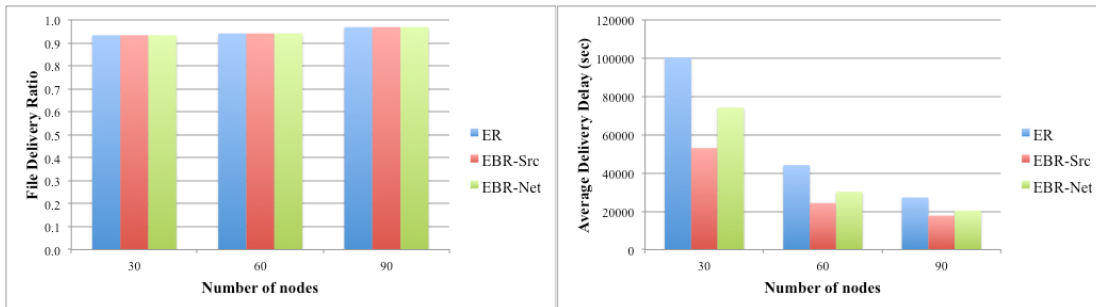


Figure 6.36. FDR and ADD of three routing schemes with UBI enabled under the NYC scenario with light traffic congestion

The simulation results from the Istanbul mobility scenario are displayed in Figures 6.37 and 6.38. We use the first setting for the transmission rate and range, and the second setting for the file size and generation rate in the simulation, which results shown in Figure 6.37. For the simulations shown in Figure 6.38, we use only change the transmission rate from 200kB/s to 500kB/s in order to set the less congested network. Again, EBR with source coding always performs best in this mobility scenario. And, the network coding scheme performs better than the ER scheme with respect to FDR unless the traffic congestion is light.

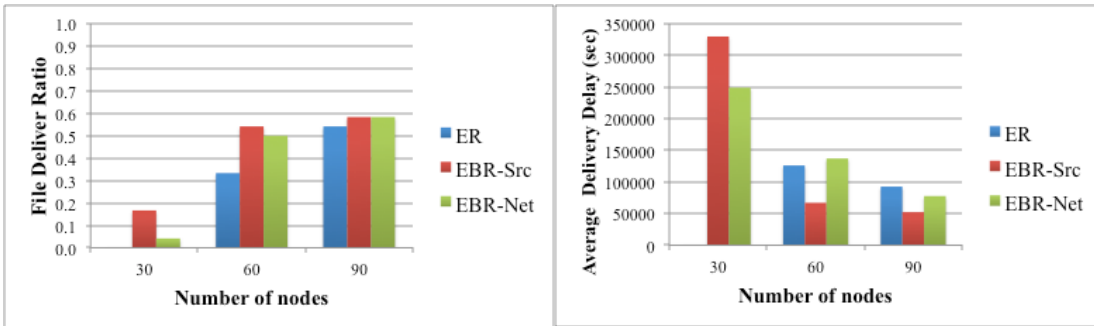


Figure 6.37. FDR and ADD of three routing schemes with UBI enabled under the Istanbul scenario with moderate to high traffic congestion

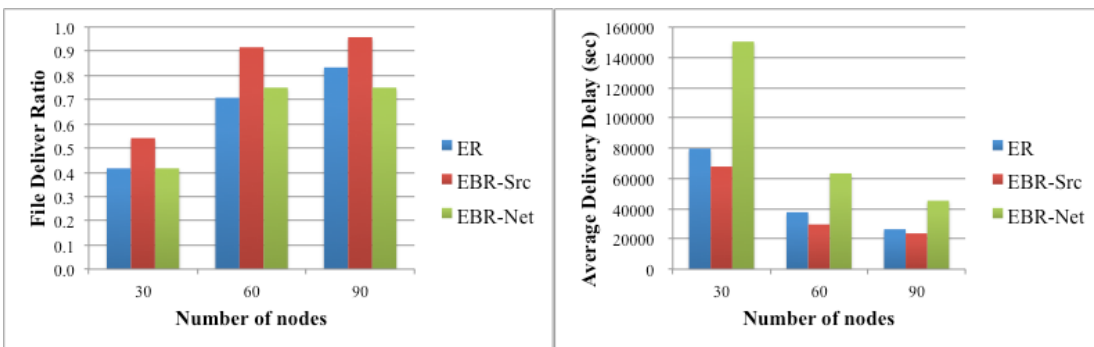


Figure 6.38. FDR and ADD of three routing schemes with UBI enabled under the Istanbul scenario with light to moderate traffic congestion

In the taxi trace scenario, the EBR with source coding still perform best among the three schemes. However, in this mobility setting, the network coding scheme is also good with respect to the FDR unless the network is highly congested. With the high traffic congestion, the source coding scheme obviously works best. Moreover, when the size of files is small such as 10MB, as Figure 6.40 shows, there is no big difference between the performances of three routing schemes with respect to the FDR.

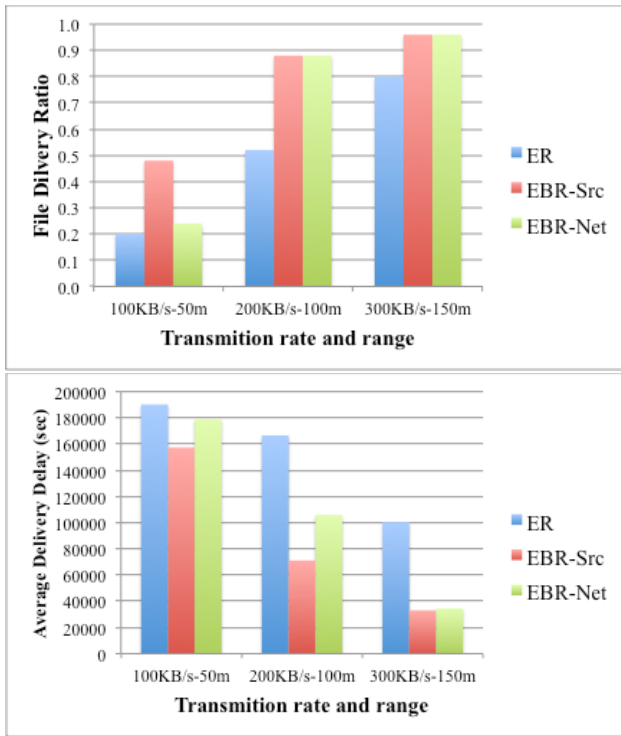


Figure 6.39. FDR and ADD of three routing schemes with UBI enabled under the taxi trace scenario when transferring 100MB files

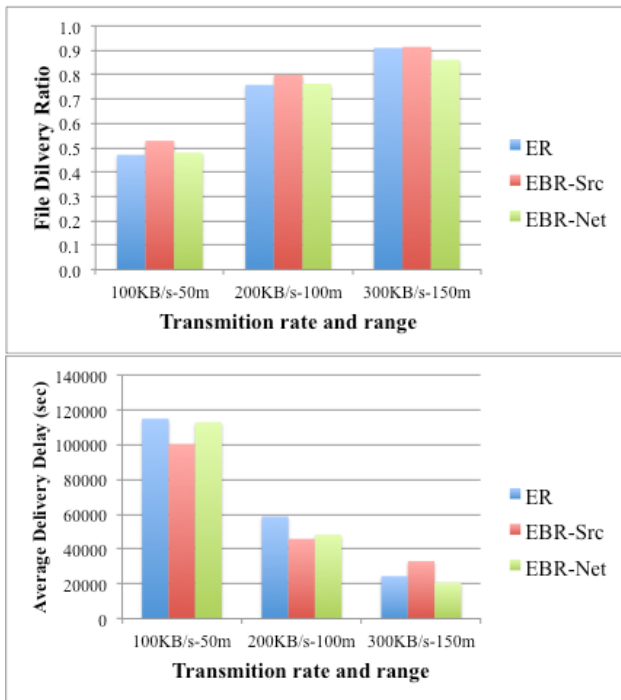


Figure 6.40. FDR and ADD of three routing schemes with UBI enabled under the taxi trace scenario when transferring 10MB files

Chapter 7: Conclusion

We have evaluated the ER and EBR DTN routing algorithms under a set of different configurations, and network scenarios. We first compared the performance of EBR in two different modes (i) source coding, where only the message's source is allowed to create new encodings, and (ii) network coding, where any relay node may create new encodings from existing ones. We also examined the performance of each of the three routing schemes with different configurations. For the ER scheme, we looked at the effects of using a Bloom filter vs. a simple list as the data structure of the summary vector. In addition, for the EBR with only the source coding scheme, we evaluated the effect of using different encoding weights, different forms of the summary vector, and the rank check capability at relay nodes.

We also investigated the two different immunity mechanisms, BBI and UBI, which can be used along with the ER and EBR schemes. Additionally, we examined the benefits of compressing the immunity messages in BBI. Through simulation we demonstrated that overall BBI is not suitable with the EBR scheme, but the UBI works best with the EBR with source coding scheme rather than the other two routing schemes (i.e., ER and EBR with network coding). Moreover, when the UBI is employed, the source coding-enabled EBR scheme outperforms other schemes under every network setting.

In evaluating the relative performance of the different routing schemes and configurations, the details of the network scenario, such as the node mobility, the node density, level of traffic congestion, etc affect the performance of routing

schemes and configurations considerably. In each section, we presented simulation results and provided our analysis of the performance trends in different network scenarios. We summarize our analysis in the following tables. These tables provide a concise guide that matches particular network characteristics to what we have determined to be the best configuration on the basis of our study. This could offer a guideline for choosing the suitable routing schemes for different network environments.

Tables 7.1 and 7.2 show the guideline for configuration of summary vector and rank check respectively. Regardless of mobility scenario, our recommendations are same for both configurations.

Traffic congestion level	High	Bloom filter	Bloom filter
	Moderate	List	Bloom filter
	Light	List	Bloom filter
		ER	EBR
Routing Scheme			

Table 7.1. Guidance for configuration of summary vector

Traffic congestion level	High	Rank check	Rank check		
	Moderate	Rank check	Rank check		
	Light	Rank check	Node density	Low	Rank check
		High		No rank check	
		<ul style="list-style-type: none"> • Simple topology • Low node density • Not all nodes meet each other 	<ul style="list-style-type: none"> • Urban map • High node density • Frequent contact • Long contact duration 		
Mobility					

Table 7.2. Guidance for configuration of rank check

Before providing a guide for choosing the suitable coding scheme, we first list up characteristics of the mobility scenarios used in our study in Table 7.3. Since we

provide a guideline for each mobility scenario, it is required to choose one mobility model that corresponds with the target network environment. Table 7.3 will help in selecting the appropriate mobility scenario to refer. From Table 7.4 to 7.7, we provide our guides for choosing the suitable coding scheme in different network settings.

Mobility Characteristic	Ferry	NYC	Istanbul	Taxi
Node density	Very low; $n \leq 10$	Low to high; $30 \leq n \leq 90$	Low to high; $30 \leq n \leq 90$	High; $n = 150$
Map	Simple topology	Urban map	Urban map	Urban map
Node behavior	- Node can not meet every other node - Nodes move on the pre-assigned paths only - Source and destination are stationary	- Nodes are homogeneous - Every node is moving	- Nodes have a moving boundary - Source and destination are stationary - Node can not meet every other node	- Nodes are homogeneous - Every node is moving - It is data collected from the real life
Node speed	Ferry; [10, 30] mph	Pedestrian; [1.5, 3.5] mph	Car in a city; [22.5, 36] mph	Car in an urban area

Table 7.3. Guidance for mobility scenario (n is the number of nodes)

Transmission range	High	Source coding (EBR)	Source coding (EBR)
	Moderate	Source coding (EBR) <i>Do not use network coding</i>	Source coding (EBR)
	Light	No coding (ER) <i>Do not use source coding</i>	No coding (ER)
		Short	Long
Average contact time			

Table 7.4. Guidance for coding scheme in the ferry scenario

Transmission range	High	Source coding (EBR)	Node density	High	Source coding (EBR) <i>Do not use ER</i>
	Medium	No coding (ER)		Medium	Source coding (EBR)
	Low	No coding (ER) <i>Do not use source coding</i>		Low	Network coding (EBR)
Small			Large		
File size					

Table 7.5. Guidance for coding scheme in the NYC scenario

Transmission rate / Node density	Low / Medium	Source coding / Network coding (EBR)	Source coding (EBR) <i>Do not use ER</i>
	High / High	No coding (ER) <i>Do not use source coding</i>	No coding (ER) <i>Do not use source coding</i>
	Otherwise	Network coding (ER)	Network coding (EBR)
		Small	Large
File size			

Table 7.6. Guidance for coding scheme in the Istanbul scenario

Traffic congestion level	High	Network coding (EBR)	Source coding (EBR)
	Medium	Network coding (EBR)	Network coding (EBR)
	Low	No coding (ER) / Network coding (EBR) <i>Do not use source coding</i>	Network coding (EBR)
		Small	Large
File size			

Table 7.7. Guidance for coding scheme in the Taxi scenario

Lastly, our guideline for the immunity mechanism enabled routing schemes is very simple. EBR with source coding with UBI is the best routing scheme in overall network settings according to our study.

Bibliography

- [1] R. Ahlswede, N. Cai, S. Y. R. Li, and R. W. Yeung. "Network information flow," IEEE Transaction on Information Theory, 2000.
- [2] F. Albini, A. Munaretto, and M. Fonseca, "Delay tolerant transport protocol – DTTP," Global Information Infrastructure Symposium, 2011.
- [3] E. Altman and F. Pellegrini, "Forward correction and fountain codes in delay-tolerant networks," IEEE/ACM Transactions on Networking, 2011.
- [4] J. Burgess, B. Levine, R. Mahajan, J. Zahorjan, A. Balasubramanian, A. Venkataramani, Y. Zhou, B. Croft, N. Banerjee, M. Corner, and D. Towsley, "CRAWDAD data set umass/diesel (v. 2008-09-14)," <http://www.crowdad.org/umass/diesel>, 2008.
- [5] T. Cormen, C. Leiserson, R. Rivest, and, C. Stein, "28.4: Inverting matrices," Introduction to Algorithms (2nd ed.), MIT Press and McGraw-Hill, 2001.
- [6] B. Dawkins, "Siobhan's problem: the coupon collector revisited", The American Statistician, 1991.
- [7] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets," Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '03), 2003.
- [8] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," IEEE Infocom, 2005.
- [9] Z. Haas and T. Small, "A new networking model for biological applications of ad hoc sensor networks," IEEE/ACM Transactions on Networking, 2006.

- [10] D. Hahn, G. Lee, B. Walker, M. Beecher, and P. Mundur, "Using Virtualization and Live Migration in a Scalable Mobile Wireless Testbed," HotMetrics, 2010.
- [11] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE Simulator for DTN Protocol Evaluation," Proceedings of the 2nd International Conference on Simulation Tools and Techniques, 2009.
- [12] P. Leach, M. Mealling, and R. Salz, "RFC 4122 A Universally Unique Identifier (UUID) URN Namespace," 2005
<https://www.ietf.org/rfc/rfc4122.txt>
- [13] S. Leon, "Linear Algebra with Applications (8th ed.)," Pearson, ISBN 978-0136009290, 2009
- [14] S. Li, R. Yeung, and N. Cai, "Linear network coding," IEEE Transactions on Information Theory, 2003.
- [15] Y. Lin, B. Liang, and B. Li, "Performance modeling of network coding in epidemic routing," Proceedings of the 1st international MobiSys workshop on Mobile opportunistic networking, 2007.
- [16] Y. Lin, B. Li, and B. Liang, "Stochastic analysis of network coding in epidemic routing," IEEE Journal on selected areas in communications, 2008.
- [17] D. Mackay, "Fountain codes," IEEE Proceedings Communications, 2005.
- [18] T. Matsuda and T. Takine, "(p, q)-Epidemic routing for sparsely populated mobile ad hoc networks," IEEE Journal on Selected Areas in Communications, 2008.
- [19] M. Mitzenmacher, "Digital Fountains: A Survey and Look Forward," IEEE Information Theory Workshop, 2004.

- [20] P. Mundur, M. Seligman, and G. Lee, "Epidemic routing with immunity in Delay Tolerant Networks," Proceedings of the Military Communications Conference, 2008.
- [21] A. Petz, C.-L. Fok, C. Julien, B. Walker, and C. Ardi, "Network coded routing in delay tolerant networks: An experience report," In Proc. of ExtremeCom, 2011
- [22] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "CRAWDAD data set epfl/mobility (v. 2009-02-24)," <http://www.crowdad.org/epfl/mobility/>, 2009
- [23] M.V. Ramakrishna, "Practical performance of Bloom filters and parallel free-text searching," Communications of the ACM, 32 (10). 1237-1239.
- [24] R. Rivest, "RFC 1321 The MD5 Message-Digest Algorithm", 1992
<https://tools.ietf.org/html/rfc1321>
- [25] N. Sarafijanovic-Djukic, M. Piorkowski, and M. Grossglauser, "Island hopping: Efficient mobility-assisted forwarding in partitioned networks," Sensor and Ad Hoc Communications and Networks, 2006
- [26] K. Scott and S. Burleigh, "RFC 5050 Bundle Protocol Specification," 2007
<https://tools.ietf.org/html/rfc5050>
- [27] D. Singh, N. Walde, R. Desai, "Flooding – An efficient routing algorithm," International Journal of Advanced Research in Computer and Communication Engineering Vol.2, Issue 10, 2013
- [28] T. Spyropoulos, K. Psounis, and C. Raghavendra, "Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks," Proceedings of ACM SIGCOMM Workshop on Delay Tolerant Networking, 2005.

- [29] A. Vahdat, and D. Becker, "Epidemic routing for partially-connected ad hoc networks," Duke University Tech. Report, 2000.
- [30] Y. Wang, S. Jain, M. Martonosi, and K. Fall, "Erasure-coding based routing for opportunistic networks," Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking, 2005.
- [31] T. Welch, "A Technique for High-Performance Data Compression". Computer 17 (6): 8–19, 1984.
- [32] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," Proceedings of the 5th symposium on Operating systems design and implementation, 2002.
- [33] J. Widmer and J-Y. Boudec, "Network coding for efficient communication in extreme networks," Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking, 2005.
- [34] E. Zhang, G. Neglia, J. Kurose, and D. Towsley, "Performance Modeling of Epidemic Routing," UMass Computer Science Technical Report, 2005.
- [35] X. Zhang, G. Neglia, J. Kurose, and D. Towsley, "On the Benefits of Random Linear Coding for Unicast Applications in Disruption Tolerant Networks," Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2006.
- [36] Java OpenStreetMap, <http://josm.openstreetmap.de/>
- [37] OpenJUMP, <http://openjump.org>