An Efficient Approach to Coding-Aware Routing

by

Harveer Singh

A thesis submitted to

School of Graduate Studies and Research

in partial fulfillment of requirements for the degree of

Master of Applied Science

Master Program in Electrical and Computer Engineering

School of Electrical and Computer Engineering

Faculty of Engineering

University of Ottawa

August 17, 2016

Abstract

Network coding is an emerging technology that intelligently exploits the store/forward nature of routers to increase the efficiency of the network. Though the concept works in theory, the segregation of coding and routing decisions makes them inapplicable in almost any practical environment. Coding-aware routing takes the network coding a step further to lessen its disadvantages by allowing interlayer communication while making routing decisions. However, most of the existing work exploits coding benefits only for fixed wireless networks, making them dependent on the types of network medium, topology and mobility and thus inapplicable for wired and mobile Ad Hoc networks. The aim of this thesis is to present a generalized algorithm that can detect any possible coding opportunity in a network of any medium, topology and mobility while making routing decisions. We have tested and evaluated our algorithm in six different network topology settings i.e. small wired, big wired, small Ad Hoc network with regular trajectories, big Ad Hoc network with regular trajectories, small Ad Hoc with random trajectories and big Ad Hoc with random trajectories. Improved performance in network throughput, mean queue size and mean end-to-end delay confirms the validity of our algorithm.

Acknowledgement

First and foremost, I would like to show an expression of gratitude towards my research supervisor Dr. Oliver Yang for his assistance and dedicated involvement in every step throughout the process. Without him, this research would have never been accomplished. I also thank my peers and professors for providing guidance and encouraging me throughout my life.

I would also like to thank my family, especially my mother, for their unconditional love and support. I also appreciate the support of my friends in every nook, especially with whom I have shared my apartment in these past 2 years.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

		Section of 1st Reference
ACK	Acknowledgement	1.2.4
AODV	Ad Hoc On-demand Distance Vector	1.4
CAR	Coding-Aware Routing	1.1
ETX	Expected Transmission Count	2.3
ETE	End-to-End delay	1.1
FIFO	First In First Out	2.6
IoE	Internet of Everything	1.1
IoT	Internet of Things	1.1
IP	Internet Protocol	2.1
LAN	Local Area Network	1.4
MAC	Media Access Control	1.2.2
MANET	Mobile Ad Hoc Network	1.4
OPNET	Optimized Network Engineering Tools	1.4
PDR	Packet Delivery Ratio	2.4.4
ROCX	Routing with opportunistically coded exchanges	1.2.1
RREQ	Route Request	2.3
RREP	Route Reply	2.3
TCP	Transmission Control Protocol	1.2.4
XOR	Exclusive OR	1.2.1

List of Notations and Symbols

		Section of 1st Reference
ber	bit error rate	2.4.4
{*F*}	set of links in network	2.1
{*n*}	set of neighbour nodes of node 'n'	2.1
*n*i	*i*th neighbour node of 'n'	2.1
m	total number of flows in network	2.1
{*N*}	set of nodes in the network	2.1
p	packet loss rate	2.4.4
pd	Number of dropped un-decoded packets	2.4.4
pep	packet error rate	2.4.4
Pr	Packet-reception threshold	4.1
*S*o	set of source nodes	2.1
*S*i	set of sink nodes	2.1
w	waiting time for packets at coding node	2.3.2
x	number of seconds an intermediate nodes hold temporary packets for	2.3.2

Chapter 1
Introduction

1.1 Overview

Computer communication conceives to provide more than just connecting people, sharing blogs and Information exchange. With the global trend towards the Internet of Things (IoT) and Internet of Everything (IoE), billions of devices will connect to internet which requires tremendous amount of data flow. Academia has been actively trying every possible way to improve the efficiency of the networks. Many traffic engineering protocols have been proposed just in the past decade itself.

Network coding (NC) is becoming an emerging communication paradigm that can provide performance improvements in throughput, energy consumption, security and End-to-End (ETE) delay. Though it was first proposed for the wired networks, there is a growing interest to apply network coding onto wireless networks since the broadcast nature of wireless channel makes it particularly advantageous in terms of bandwidth efficiency and enables opportunistic encoding and decoding. NC scheme refines the precise flow of data in a network by transmitting combined digital message from many sources. Prior to NC, the only job of intermediate nodes (i.e. switches and routers) within a network is to forward data packets towards the destination. NC terminology encourages that, in addition to packet forwarding, intelligent mixing (from different sources before forwarding) increases the network performance. Therefore, the logical operation of routers and switches can be replaced by coders that allows the intermediate nodes to encode packets before transmitting them out. This process of encoding and forwarding increases effective capacity of the networks by minimizing number of bottlenecks. The only condition to encode packets is to make sure that destination should be able to decode it. Otherwise the packet would be marked as a “bad packet” which will eventually be dropped. The concept of network coding does not guarantee this and thus limits its usage in two-hop scenarios only as described in [KaRa08].

CAR (Coding-Aware-Routing) is proposed on basis of network coding and can validate whether destination would be able to decode the encoded packets or not during route-discovery.

Recently, many coding-aware algorithms have been proposed that takes active part during the path discovery and tries to capture the degree of coding benefit from all possible paths and then choosing the one which can yield maximum benefits.

1.2 Literature Review

We shall review the existing work done in the topics related to our work in NC and CAR protocols which often have dependence on the types of network topology, network size, and/or their physical medium.

1.2.1 Network Coding

The basic characteristic of network coding is the intelligent processing of data packets reaching a common intermediate node. Processing packets involves either one or more of the following operations: coding, decoding, listening or relaying. The concept of network coding was first proposed by R. Ahlswede et. al. [AhCa00], who showed that having the routers mix information from different messages allows the communication to achieve multicast capacity. But, there wasn't any practical model until the authors in [KaRa08] proposed COPE which embraces the radio characteristics of the wireless mesh networks and tries to a-bridge the gap between the theory of network coding and practical network design for unicast traffic. Our CCNR lab has also confirmed performance gain from network coding under different regular topologies and under-water networks [Ding14].

In general, the coding algorithm should guarantee that intended nodes can decode the original packets after they received a certain set of coded and single packets. This requires overhearing the packets by intermediate or by intended nodes in network. The coding algorithm can be divided into three categories: linear coding, algebraic coding, and random coding.

A linear coding construction was proposed for its simplicity and practicability [LiYe03]. They formulated a multicast network and proved that the max-flow bound is possible to be reached through a linear coding multicast. The XOR (Exclusive-OR) coding works on the principle of bit arithmetic (particularly, it XORs information section of a packet from (n) different sources to form a new packet). Hence, it requires knowledge of (n-1) original packets to decode the original packet. Many protocols including COPE, and ROCX (Routing with Opportunistically Coded eXchanges) [NiSa06] have been employed with XOR-based coding.

The algebraic framework-base coding strategy [SaEg03] showed how a polynomial algorithm can be used to solve network problems and an algebraic tool was provided to the research of network coding. A randomized network coding for multiple source multicast networks was introduced in [Home03], the success probabilities of the randomized coding in networks with unreliable and delay links was demonstrated.

There is no deny that network coding can yield performance improvements as compared to traditional networks, but its limited knowledge on neighbours (i.e. just up to 2 hops) can miss some potential coding chances (i.e. a chance to obtain coding benefit) in network. This is a good reason to review coding-aware routing which can yield better results than network coding.

1.2.2 Topology

Most of the research on network coding so far relies on the nature of network topology. Early implementation of the network coding algorithms as discussed in [TaQi11] or more requires a particular network infrastructure or the network would not be able to facilitate coding/decoding. We shall categorize and summarize below some common topologies used in literature:

1) Linear Topology

In linear topology, every node has a downstream and an upstream node to transfer or receive data. A coding node usually sits between them to forward packets downstream and upstream. The Alice/Bob model (e.g., [ZhZh09, SeRa10]) is a common instance of the linear topology.

2) X topology

Studies have shown that network coding can improve the coding gain in the X-topology [KaRa08]. A double decoding method was proposed to increase the network throughput in both loss-free and slightly lossy network [TaQi11].

3) Butterfly Topology

Butterfly topology is the most widely discussed topology in wireless networks since it facilitates the overhearing of the packets into more coding chances. Network coding has been shown to reduce queue length, and to increase throughput [PaCh08, Ding14] when compared they with classical routing. The theoretical model of the inter-flow coding-aware routing in butterfly networks has been studied [QiYa13] and improvements in delay performance has been reported but the proposed algorithm is limited to only wireless mediums.

4) Random Topology

Most of the early research on the Network coding has considered regular topologies, because it was much easier to say whether the next-hop would be able to decode the packets or not. Since regular topologies only represents hypothetical concept, which is contrary to the real world, in which the routing objects are, located at random places. Unlike traditional work on regular networks (e.g. the NC benefits on X and butterfly topologies [Ding14]), it would be interesting to see if there are NC benefits in random networks especially the recent Ad Hoc networks where topologies changes quite often.

There are few algorithms that discusses and generalizes the concept to any topology form. Protocol for applying CAR to irregular topologies has been studied [LeLu10], but it cannot be applicable to Ad Hoc and wired networks. The throughput performance of the coding-aware multipath routing and single-path routing based on a 15-node random wireless topology has been compared [SeRa10] and it has been observed that more coding gains can be obtained on applying multipath routing scheme. Furthermore, some research also emphasizes in integrating network coding with MAC (Medium Access Control) layer based on the CSMA/CA mechanism and conducted simulations on a topology with 50 randomly distributed nodes [CaGo13].

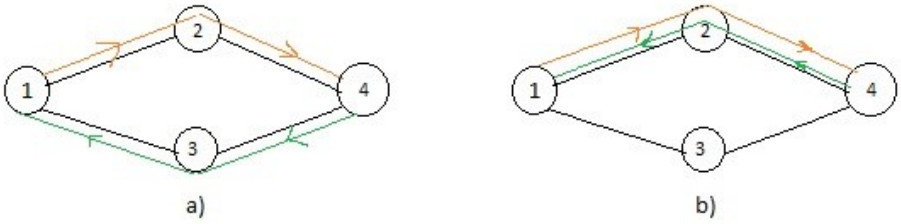

Figure 1: An example illustrating the Effects of Coding Aware Routing:
a) routing without coding consideration, and b) routing with coding consideration

1.2.3 Coding Aware Routing

Most of earlier network coding work (e.g., [AhCa00], [KaRa08], SaEg03], [TaQi11)] exploited the benefits in wired and wireless networks. With network coding protocols, a node can only have neighbor information up to 2-hops. Thus coding chances more than 2-hops away cannot be exploited by these algorithms. CAR improves this 2-hop limitation by advancing the route-discovery phase of routing protocols [SeRa10], and thus capturing any coding chances in network before sending actual data packets. The CAR operation can best be illustrated by the example in Figure 1.

Fig. 1a below shows the disjoint paths commonly found by a general routing protocol that does not account for coding-opportunities when making routing decisions (finding routes). On the other hand, using a coding-aware routing decision as shown in Figure 1b, node 2 has the opportunity to perform coding, and hence obtain coding benefits which will result in higher end-to-end performance.

Coding benefits can be derived on certain types of media and topologies. It has been identified that coding benefits can yield best results when applied to wireless networks [ChCh12, DiZh09, CHAG13, GuJi14, NiSa06], but not in fixed nor in completely random networks (e.g., Ad Hoc networks where nodes move randomly). Performance gain has been demonstrated in wireless Ad Hoc networks, [e.g., LiRa07, WaZh14] using CAR. Since these algorithms rely on the broadcast nature of wireless medium to overhear decoding packets (discussed in Section 2.3), they are not applicable to wired networks. Coding gain has been observed when applying CAR to any networks independent of their topologies [e.g. LeLu10, SeRa10, and CaGo13], but these algorithms are specific to wireless static networks but not applicable to wired networks. Also, since they cannot evaluate coding paths dynamically, they are inapplicable to random Ad Hoc networks as well.

Although COPE [KaRa08] can detect coding opportunities along a selected path, it often overlooks some other potential coding opportunities because its coding and routing operations are independent. Even if there are more than one coding routes possible in the network, the passive nature of COPE in seeking coding opportunities does not allow it to change the route selection dynamically. It has been shown that more performance gain can be obtained by capturing system state during path evaluation time, by monitoring it until the end of a flow lifetime [e.g., CHAG13, GuJi14, LeLu10, NiSa06] and by changing dynamically when required. But as discussed above, these algorithms are only applicable to wireless networks and regular topologies. Thus, makes them infeasible for wired and Ad Hoc networks.

Note that coding-aware routing exploited in our work refers to network coding which takes place between network and MAC layers as explained later in Chapter 2. This is different from other forms of coding (like channel coding) and their coding-aware operations which is usually done at the physical layer.

1.2.4 Network Coding in MAC and TCP

Some researchers have also proposed a way to implement network coding in other protocol layers such as MAC and TCP (Transmission Control Protocol) layers.

BEND is a typical MAC layer coding method in multi-hop networks that also happens to be the first exploration of broadcast nature of wireless channels [ZhZh07]. BEND provides the scenario of opportunistic coding in which all the neighbours are allowed to overhear the packets but only one can forward it. Various topologies were used to evaluate BEND and to compare IEEE 802.11 with COPE. The results show that BEND can achieve a higher coding ratio and throughput.

In order to make network coding compatible with the retransmission and sliding window mechanisms of TCP, a new scheme is proposed to incorporate network coding into TCP layer [SuSh09]. In their scheme, the source transmits a random but linear combination of packets in the sliding window. Instead of sending an ACK (Acknowledgement) for each packet decoded successfully, the sink will send an ACK indicating the number of coded packets it has already received.

1.3 Motivation and Objectives

As discussed in Sections 1.2.1, 1.2.2 and 1.2.3, the reliance of existing NC and CAR algorithms on type of topology, size, and physical medium makes them infeasible in real-world scenarios. Thus, it would be desirable to have an algorithm that promises to generalize (i.e. exploit coding chances in network irrespective of particular type of topology, size and physical medium) the CAR decisions by capturing the system state during path evaluation and monitoring them dynamically in the lifetime of a flow without its dependence on the type of topology and physical medium.

The general objective of this work is to propose a CAR algorithm that tries to evaluate any possible coding benefit in the network and is general in the sense that it could be applied to a network of any size, topology and physical medium. Specifically, we would like to

1) Propose a CAR protocol that exploits the coding benefits in mobile Ad Hoc networks.
2) Implement and verify its operation in a simulation testbed.
3) Evaluate its performance under different topology and mediums (wired and wireless) settings.

1.4 Methodology and Approaches

In order to achieve our objectives, we need to first understand the operation details of coding algorithm so that we can incorporate routing and coding seamlessly. We shall provide models and algorithms used in this thesis along with assumptions used throughout our work.

Unlike the regular wireless networks used by many researchers to show the benefits of coding-aware routing, we shall study its effect on irregular wireless and MANETs (Mobile Ad Hoc NETworks) while using the fixed wired network as a reference. For every type of network under our consideration, we have used two different sizes of networks in terms of the number of nodes. We have chosen 20 nodes as an instance of a small network, and consider doubling that (40 nodes) to be a big network in comparison. We could consider big network to be of exponential size as compared to small network, but exponentially increasing number of nodes also leads to exponential increase in simulation time. In order to save that time, we consider using 40 mobile nodes as our big network because it reflects well about the algorithm scalability with reasonable simulation time. Both the small and big topologies we use for the wired networks are two-dimensional that can be used in LAN (Local Area Network) and Internet backbone infrastructures. We shall extend our study to a wireless Ad Hoc networks in order to conduct more stringent test on the generalized CAR algorithm and to see the potential benefits of Ad Hoc infrastructure. MANETs with regular trajectories, both small and big, are first studied before we venture into completely random trajectories. We will also see that how our algorithms work seamlessly regardless of the types of networks. The commonly used performance measures of throughput, mean ETE delay, packet delivery ratio and mean queue size are used to evaluate our algorithm under different scenarios.

For evaluating network performance, we shall rely on simulations instead of live testbeds because of the expenses involved in real-life implementation. With its more robust, flexible and efficient system for simulation, we have chosen OPNET (OPtimized NETwork tools) [YaLu12] to be our simulation tool among other simulation tools. The OPNET expertise of our CCNR Lab would also facilitate an easy learning process.

We want to use the popular AODV (Ad Hoc on Demand Vector) as our routing protocol for the Ad Hoc networks. Since it is time-consuming to implement AODV from scratch, we have decided to modify the OPNET's built-in AODV model, and use it to evaluate coding opportunities during route discovery time. On the other hand, it is easy for us to implement everything (except

the static routing) from scratch for the wired networks. Much time has been spent in designing algorithms, and then programming in OPNET for all network topologies and conducting dynamic performance evaluation like queuing, delay, throughput and mean queue size. With the help of OPNET node model, we have programmed a general node rather than designing a task specific node. Any general node is capable of doing all the tasks a node may have to do in a network i.e. relaying, coding, decoding, packet generation, etc. which will be discussed in Ch.2.

1.5 Contributions

The contributions of this thesis are the following:

1) Proposing a generalized coding aware routing algorithm that exploit any coding possibility in a network of any topology, size and physical medium.
2) Studying and verifying the performance gain of our coding-aware routing over traditional network coding (where no CAR is used)
3) Using our algorithm to exploit coding benefit in dynamic MANET. To our best knowledge, there has been no work on the exploitation of coding opportunities in random MANETs.

1.6 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 gives the general network layout and model under study, and describes the network operation and the assumptions used throughout the thesis. It also describes the possible functions of a generalized node. Chapter 3 first evaluates the network performance and queuing behaviour of small wired network in order to verify the benefits in terms of throughput, delay and reliability. Then a bigger and more complicated network is simulated and its performance is analyzed. Chapters 4 and 5 analyze the performance of our algorithm in regular and in random MANETs (i.e. MANETs using regular and random trajectories) respectively. Chapter 6 discusses design issues and guidelines based on our experience from this research. We conclude our findings and observations in Chapter 7, and an Appendix containing some useful but supplemental information is provided at the end.

Chapter 2
Network Operations, Models and Assumptions

In this chapter, we shall provide the operation details of the network coding and generalized coding-aware routing. We have achieved this by designing proper OPNET Node and Process models. Any general assumptions used in our performance evaluation and analysis is provided at the end.

2.1 Network Layout and Organization

We consider a general IP (Internet Protocol) network of *N* nodes and *F* links. In a fixed network, the number of links *F* is usually fixed. A link exists if there is a reliable transmission medium for data transmission. In the case of a MANET, each node is equipped with antenna for transmitting and receiving from its neighbors. A link is formed if two nodes are within the communication range of each other. As the nodes are mobile, links can be formed or broken depending on the distance and the transmitting and receiving power of the antenna. Hence there is usually no fixed topology for a MANET as there is no fixed infra-structure. In other words, *F* is a number in the range of [0, *N*(*N*-1)]. Likewise, there can be different sources and sinks in a network at any one time.

For all the networks, let {*S*o} and {*S*i} be the set of sources and sinks, respectively. Let {*a*} be the set of neighbor nodes of node a, and *Nai* be the i^{th} neighbour of node a. The traffic between a source and a sink constitutes a flow. Let *m* be the number of flows passing through the node *a*, and *F*ai be the i^{th} flow. Network coding can be exploited when *m*>1. For example, the sets of sources and destination in the Fig. 2.2 example later can be {1, 6} and {7, 11} respectively.

The basic IP protocol allows a packet to be routed to the destination using a connectionless operation [IP81]. Variations are used depending on the nature of the networks used. For mobile network wireless (or Ad Hoc networks) the routing protocol is AODV (Ad Hoc On-demand Distance Vector). For the wired networks (Ch.3), we have assumed the route discovery has been already done (e.g. using a simple shortest path routing algorithm like Dijkstra's algorithm) and the paths are identified (and fixed) so that one can focus on coding

performance evaluation.

In addition to the general functions expected of an IP node, each node in a network is capable of the functions of coding, decoding, relaying or listening (in wireless mobile networks) if needed. Therefore, a node can be designated as a coding node when requested to actively perform network coding on incoming traffic flows. Obviously, there is a designated decoding node to decode the packets before handling them to a destination. Decoding nodes need decoding packets to recover an original packet from a coded packet. Generally, the decoding packets can be overheard by the decoding nodes directly, but if this cannot be done, innovative nodes have to supply them.

An innovative node is a one-hop neighbour of the decoding node. Accordingly, the decoding packets provided by innovative nodes are called innovative packets. Note that an innovative node is just used as a backup node to provide decoding packets in case when decoding nodes cannot over hear them directly. Packets are lost if a decoding node cannot receive decoding packets directly and an innovative node cannot be found. For example, there is no way for decoding nodes 5 and 10 in Fig. 2.2 to have decoding packets from flow 1 (node 1 -> node 6) and flow 2 (node 7 -> node 11) because it is a wired network and the packet transmission between nodes 8 and 4, and nodes 2 and 3 cannot be overheard/listened by nodes 5 and 10. But if nodes 8 and 2 can be designated as innovative node to provide decoding nodes 5 and 10 with decoding packets through links 8-5 and 2-10 respectively, coding benefits can be realized. This means that for wired networks, innovative nodes has to be on the path of actual flow as for network in Fig. 2.2. However, in case of wireless networks, since nodes can overhear the packets from neighbours, innovative nodes for wireless networks can be up to one-hop away from the nodes in actual path of the flow. The simplest of all node actions is the relay node, it just relays the packet to the next hop similar to nodes in traditional routing.

As mentioned in Section 1.2.1, a coding chance reflects the possibility of an intermediate node to become a coding-possible node. Thus, the number of coding chances is equal to the number of coding-possible nodes exists along the path. Accordingly, a coding degree can be defined as the number of coding chances (coding-possible nodes) along the path. For multiple paths to a destination, it would be natural for a source to pick a path with a higher coding degree. If there are two or more paths with the highest coding degree, our algorithm will choose the one from which the RREP (Route REPly) packet is received first.

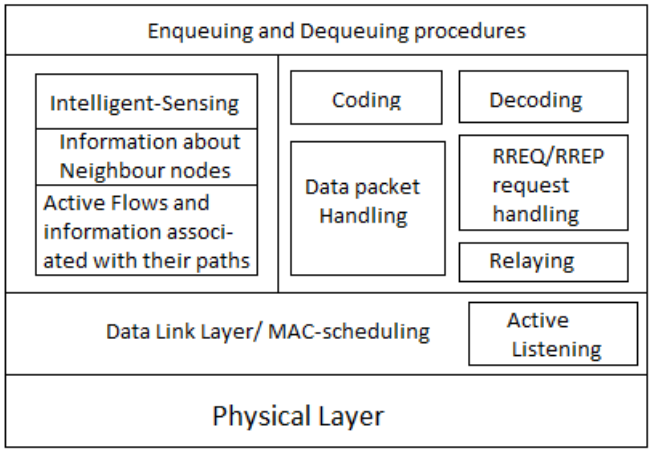

Figure 2.1: Functional Model of a Node

2.2 Functional Model, Operation and Algorithms

The function of coding-aware routing algorithm in general can foster inter-layer communication between the MAC and the Transport layers. Fig. 2.1 gives the communication model which illustrate the steps a node has to take to process information in order to truly explore coding chances in a network.

Packets not intended for a particular node are discarded at the MAC layer, but those packets that can be used to decode current and future flows would need to be buffered. Active listening would copy all packets receiving at MAC layer for later processing by the Intelligent Sensing (Intelli-sense) module. To decode a coded packet, management of overheard (innovative/decoding) packets is important. The Intelli-Sense module would temporarily hold packets for x seconds for various purposes. For example, the Source node in the next section may hold RREP packets to explore different route, or for an innovative node to buffer overheard packets which can be needed by the decoding nodes later. They all need same amount of time *x* to accomplish their purpose.

Choosing an optimal value of *x* can effect network performance because this is the time a source node has to wait before it can send any data packet. This is not very important in static networks where topology is formed only once. Even 3s for the first packet becomes insignificant when averaged over millions of packets. But for random Ad Hoc networks as we will see in later chapters (Chapter5), a topology might need to be re-configured number of times and hence, every time a source node would need to wait 3s before sending any data packet. Thus, it is pivotal to find an optimal value of *x* that would allow enough time for a source node to get back all RREP

packets from destinations and at the same time, not effecting average performance of the network. From our observation/evaluation of small random Ad Hoc network (Chapter 5), we found x = 3s to be an optimal value of *x* and hence, have used that value throughout our simulations.

Information about each one-hop neighbor nodes is collected at route discovery time and is kept until the flow lifetime. This is done in order to increase coding chances in the network, and can be best explained by following algorithm steps as mentioned below.

2.3 The Generalized Coding-Aware Routing Algorithm

The generalized coding-aware routing algorithm allows a node to capture any coding chances in network irrespective of its size, topology and physical medium and hence, to yield maximum coding benefit. Actions of a node are different depending on whether it is a source node receiving a new packet from application layer or a non-source node (intermediate or destination node) receiving a packet from physical layer. Algorithm further allows source node to designate roles to all other nodes involved in a flow i.e. whether a node is performing coding, decoding or just relaying packets. For clarity, the following node actions are provided for a particular flow. Their views/actions to other flows are commented later.

Source Node Actions

A source node can send two types of packets (data and RREQ) and receive one type of packet (RREP). The following are their corresponding actions:

A) Sending an RREQ packet during the Route Discovery Phase

A1) Generate a RREQ packet with format given in Fig. 2.13.

A2) Append its node-id and one-hop neighbors into the packet.

A3) Broadcast the packet to all downstream links.

B) Receiving RREP packets during Route Discovery Phase

B1) Wait for x seconds.

B2) From all RREP packets received in the wait, evaluate the coding degree/chances for all possible routes to the destination.

B3) Choose the path with highest coding degree for data packet transmissions.

B4) If no path yields coding degree greater than 0 (i.e. no coding chances exists), choose path based on the ETX (Expected Transmission count).

C) Sending data packet during Data Transmission Phase

C1) Create packet with the format shown in Fig. 2.14.

C2a) If this is the first data packet,

- append a data structure (similar to Table 2.2) containing information about the role designations and the node list for all the nodes along the path into the packet.
- forward the packet to next hop along the path chosen in Step B.

C2b) If this is not the first data packet, send the data packet to next hop on the chosen path.

Comments:

1. If there are multiple paths that has same coding degree, then the path from which the first RREP was received will be chosen over the others.
2. In practice, ETX is calculated based on different parameters i.e. available bandwidth, number of hops, average delay, etc. But ETX in our implementation is just based on number of hops.

<u>Actions of Non-Designated Intermediate Node</u>

The actions are different according to the types of packet received.

A) On receiving an RREQ packet:

A1) check its packet id to find out whether it has been received before.

A1a) If received before. Discard it.

A1b) If a new packet, proceed further.

A2) Store the packet for a duration of x seconds.

A3) Append its one-hop neighbour information and its node-id in the received packet

A4) Send it to the downstream node along the path to destination.

B) On receiving a RREP during the Route Discovery Phase:

B1) Scan through the list of current/active flows stored locally in the node.

B2) Find a match with one such current/existing flow with which coding can be possible.

B2a) If no match, mark the present node as not coding-possible by changing the 'coding-possible' flag in structure as represented by Table 2.2 to 0. Then go to B3.

B2b) If there is a match,

B2b-i) Mark the present node as coding-possible for this new flow by changing the flag to 1.

B2b-ii) Designate decoding and innovative nodes for this new and the existing flow.

B2c) Append the node designation information from B2b into the RREP packet.

B3) Send the RREP packet to the source.

C) On receiving the First Data Packet during Data Transmission Phase

C1) Extract the role designation information.
C2) Identify its role designated by the source.
C3) Obtain the node-list for the flow (which contains node-ids of all nodes in the path and of their one-hop neighbours).
C3a) Stores the node-list locally until the lifetime of the flow.
C3b) Forward the packet to next hop.
C4) Carry out its role function according to the following action list for different designated node function.

Actions of Intermediate node designated as a Coding Node (CC):
On receiving a data packet during the Data Transmission Phase
CC1) Checks whether there are packets available from other equivalent streams/flows.
CC1a) If no other stream is available, it just relays the packet to next hop.
CC1b) If other streams are available, it codes packets obtained from those streams together by following coding operation discussed in Section 2.3.

Actions of Intermediate node designated as a Decoding Node (DC):
On receiving a data packet during the Data Transmission Phase
DC1) Check the coding flag in 'list_struct' field of data packet as shown in Fig. 2.14.
DC2a) If packet is not coded, just relay the packet to next-hop.
DC2b) If packet is coded,
DC2b-i) obtain equivalent packets from other streams.
DC2b-ii) decode to obtain the original packet by following decoding operation in Section 2.4.
DC3b-iii) If packet is not successfully decoded, discard the packet.
DC4b-iv) If packet is successfully decoded (i.e. it is able to obtain all decoding packets required to decode), send the packet to destination.

Actions of Intermediate node designated as an Innovative Node (IC):
On receiving a data packet during the Data Transmission Phase
IC1) Send decoding packets to decoding node.

Actions of Intermediate node designated as Relay node (RC):
On receiving a data packet during the Data Transmission Phase
RC1) Relay the packets to next-hop.

Comments:
1) Any node in the network contains an array/list of current flows passing through it. An entry in the list corresponding to a particular flow contains information about all upstream and

downstream nodes i.e. their node-ids, one-hop neighbors and role designations. And obviously, an entry from the list is deleted once a flow ceases to exist.
2) A match means there exists one downstream node from a new flow that can overhear packet transmission from an upstream node of an existing flow before it gets coded (i.e. either directly or via innovative nodes) and vice-versa (i.e. there exists a downstream node from the existing flow that can overhear transmission from upstream node of new flow). Such downstream nodes would be regarded as decoding node for the flow.
3) The delay of *x* seconds is required to allow source node to collect all RREP packets from destination so that coding-chances can be evaluated by exploiting all possible paths to destination. This would require a space complexity of O(*N*) and then becomes stale information that needs to be removed regularly. We use *x* = 3 seconds in our simulation.
3) Node-list extracted for the first data packet is stored locally and can be used to find future coding-opportunities.
4) An intermediate node knows a data packet is the first data packet of a flow by looking at the packet id.
5) As mentioned, the node actions in this section are provided with respect to the type of packets a node is *expected* to receive from the particular flow it is serving. In general, a node can receive any types of packets from any flows at any times of the network operation. For example, although a node expects to receive RREQ/RREP packets during the route-discovery phase of a flow according to the above node actions, it can also receive RREQ/RREP packets from other new flows during its lifetime of serving its own flow. In this case, a node recognizes a new flow from the flow-id of the RREQ/RREP packet it received and registers this new flow in its cache which stores all flows (new or on-going) separately from each other.
6) From another perspective, a new flow for which job designations are not decided yet. A new flow in a network would treat all nodes as intermediate nodes irrespective of their previous designated roles for other flows. In summary, all designated roles are flow based, e.g. node A can be a coding node for flow 1 and can just be a relay node for some other flow 2.

<u>Destination Node Actions</u>
There can be two types of packets received by a destination
A) On receiving a RREQ packet:
A1) Generate a RREP packet.
A2) Reply back to with RREP packet using the reverse path obtained from RREQ packet.

B) On receiving Data packet:
B1) If it's the first data packet, obtain the node-list and role designation.
B2) For all other data packets:

If the destination node has not been assigned any decoding capability (DS)

DS1) Buffer packet in application layer.

If the destination node has been assigned to be a decoding node (DD) as well.
DD1) Perform step DC1.
DD2a) If packet is not coded, perform step DS1.
DD2b) If packet is coded, perform step DC1b.
DD2b-i) If packet is successfully decoded, perform step DS1.
DD2b-ii) If packet is not successfully decoded, dump the packet to sink.

A pseudo-code for this algorithm is also provided in Appendix A. It is part of the explanation for the logic that helps a node in deciding whether it would operate in listening, relaying, coding or decoding mode for a particular flow.

The above CAR algorithm can be used in both the wired and the MANET networks. Chapters 3, 4 and 5 compare the performance measures of a network using CAR and without CAR. A network without CAR is capable of performing traditional NC as depicted in our literature review and coding node is decided in a distributed manner i.e. any intermediate node can be coding-node if the previous hop of a packet from one flow is same as the next hop of packet from another flow and vice-versa (i.e. if coding and decoding nodes are just 1-hop away).

Unlike stationary nodes in wired and static wireless networks, a mobile node (as ones shown in Fig. 2.17) can move with a random speed and trajectory. Thus, the network topology is dynamic due to changing node positions, and there can be a number of intermediate hops for each source-destination pair. Like most modern routing protocols, the next hop is chosen depending upon many factors of which proximity is one. As the relative proximity of nodes changes (i.e. when one or more nodes becomes un-reachable), the “next hop” also needs to be re-evaluated. In another word, the network topology needs to be “re-configured”. Due to its popularity as already discussed in Section 1.4, we have used AODV (Ad Hoc on Demand Vector) for our Ad Hoc network. Each node broadcasts AODV “Hello messages” after a pre-defined time interval called the “Hello interval” to inform neighbor nodes about their presence (proximity) so that source nodes can decide when to re-initialize route discovery. Only after running AODV to list all possible paths to the destination that our generalized CAR algorithm can be executed to exploit coding opportunities in them. As mentioned in Section 1.4, we need to modify the built-in AODV algorithm in OPNET so that it can invoke our generalized CAR algorithm each time the

topology changes. In order to overhear the packet transmissions in the network, some modifications in the MAC layer has also made.

Coding-decoding procedures used to perform coding/decoding for all networks under consideration are given in Section 2.3.2 and 2.3.3, it explains how a node code and decode packets.

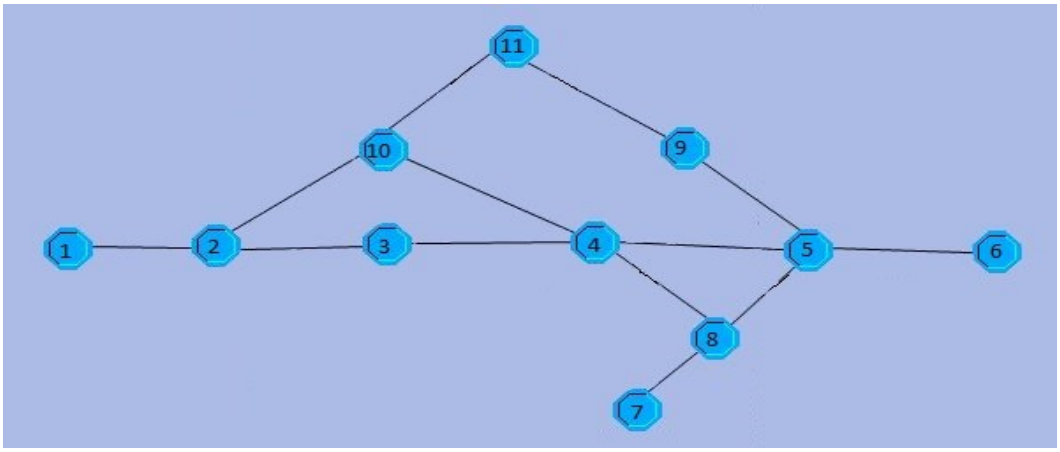

Figure 2.2: Network Layout

Table 2.1: Immediate Neighbour Information

Node/NN	1	2	3	4	5	6
NN1	2	1	2	3	4	5
NN2	-	3	4	10	8	-
NN3	-	10	-	8	6	-
NN4	-	-	-	5	9	-

2.3.1 Example

We provide an example here to illustrate the node behaviours discussed above. Figure 2.2 represents the one of packet switching networks under our investigation. It consists of *N*=11 nodes (IP nodes). Suppose at any given time, there is an existing flow (call it Flow 1) between source node 1 and destination node 6 and the path happens to be 1-2-3-4-5-6.

Table 2.1 identifies the immediate neighbors of each node along the path. The top row are the nodes along the path and the NNx in left column gives the possible innovative nodes (one-hop neighbours as defined in Section 2.1) of each node. Since there is a maximum of 4 neighbors according to Fig. 2.2, we only use NN1 to NN4 for the neighbor possibilities of each node.

Now assume at some point Node 7 wants to initiate a new flow (call it Flow 2) in order to transmit some data to Node 11. The following are the actions of the nodes along the path according to the Generalized Code-Aware algorithm:

1. According to Action-A of the Source Action, Node 7 would send an RREQ packet containing information about its one-hop neighbour nodes list.
2. According to Action-A of the Intermediate Node Action, Node 8 would store the RREQ packet temporarily for x seconds (x=3 used in our simulation). Node 8 then broadcast the RREQ packet after appending its neighbor list {4, 5, and 7} in the packet.
3. According to Action-A of the Destination Node Actions, upon receiving the RREQ packet, Node 11 sends back an RREP packet with the node-list of the entire path as well as the one-hop neighbours.
4. According to Action-B of the Intermediate Node Action, on receiving a RREP packet, Node 4 tries to determine if it can be the coding-possible node.
    - Since Flow-1 is the existing flow, by iterating through the node-ids list of Flow 1 and Flow 2 according to Step B2, node 4 detects node 5 in Flow 1 can be a decoding node since it is just one-hop away from node-8 which will transmit packets from flow-2 before they get coded. Likewise, node 10 along Flow 2 is just one-hop away from node 2 which has packets from Flow 1 before coded.
    - Since there is a match found in the iteration process, Node 4 realizes he is the coding-possible node.
5. According to Step B2b, being a coding-possible node, Node-4 would like Node 10 and Node 2 to be the decoding and innovative nodes respectively for Flow 2 while Node 5 and Node 8 too be the decoding and innovative nodes respectively for Flow 1. At this point, Node 4 would set the coding-possible flag, embeds the information on the possible decoding and innovative nodes into the RREP packet and sends it to source node.

We have discussed in Steps 4 and 5 to show how Node 4 determines itself to be the coding-possible node along with other nodes to support the decoding operation. This is shown in row-3 of Table 2.2 below. Obviously, other nodes along the path are carrying out the same operation and their findings that are sent back to the source node are shown in rows 1, 2, 4 and 5.

6. The notation "-" indicates that decoding and innovative nodes for the corresponding nodes (i.e. nodes along the path) are not applicable since there is no coding possibility.

Nodes along Path	Coding Possibility	Decoding Node1	Decoding Node2	Innovative Node1	Innovative Node2
11	0	-	-	-	-
10	0	-	-	-	-
4	1	5	10	8	2
8	0	-	-	-	-
7	0	-	-	-	-

Table 2.2: Final Next-Hop Information

7. According to Action-B of the Source Action, upon receiving all RREP packets, Node 7 would go through steps B2 and B3 to evaluate the best path based on coding-degree and hop-count. Since node-4 comes out to be the only coding node in the network and it is on the only path that yields a coding degree greater than one, the source node will pick the path 7-8-4-10-11over others.
8. According to Action-C of the Source Node Actions, Step C2a requires Node 7 to append the role designations for all the nodes along the chosen path in its first data packet. All data packets are sent out along the designated path as in Step C3.
9. According to Action-C of the Intermediate Node Actions, an intermediate node would realize its role designated by the source upon receiving the first data packet of a new flow. It will follow the corresponding action list until the end of a data flow.

As discussed in Section 2.3, NC (network without CAR) can only exploit coding chances if at any intermediate node, the previous hop of a packet from one flow is same as the next hop of packet from another flow (i.e. if coding and decoding nodes are just 1-hop away), and vice-versa. At the coding node 4 in the above example, the next hop of flow 1 (i.e. node 5) is not same as the previous hop of flow 2 (i.e. node 8) and the next hop of flow 2 (node 10) is not same as previous hop of flow 1 (i.e. node 3). Thus, the optimal path can only be obtained in terms of hop-count which can either be 7-8-5-9-11 or 7-8-4-10-11. Even if the path 7-8-4-10-11 is chosen, there is no way for nodes to know about innovative nodes that would allow the network to take advantage of the coding benefit.

For simulations in Chapters 3, 4, and 5 we shall use “network using CAR” for networks using our generalized CAR algorithm and “network without CAR” for networks with traditional NC.

2.3.2 Coding Node Behaviour

As discussed in the node behaviours/actions in Section 2.3, one sees that a coding node can be identified by following general coding aware routing algorithm. Source node informs the coding node for its role by embedding this information into first data packet. From our discussion in Section 2.3, coding-possible nodes is used to represent possible coding nodes (i.e. for which decoding and/or innovative nodes have been identified) on a path while source is in route-discovery process. After an optimal path based on coding-degree is chosen, all coding-possible nodes on the chosen path will be regarded as coding nodes.

When a data packet reaches coding node, it might want to wait for packet(s) from other streams by which they can be coded and sent together. Non-deterministic packet arrival time does not guarantee about this waiting period and thus, the amount of time wasted in this can lead to disastrous results even as compared without traditional networks. That is why all networks under our consideration are not waiting for packet from other streams and the packet will be relayed un-coded.

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.3: XOR Truth Table

For coding operation, we have used XOR (^) bit operation as shown in Table 2.3. The data portion of each packet must have the same length (i.e. same number of bits) so that coding computation in bits can be carried out. Shorter packets are padded with zero before coding. To obtain a coded packet p from n packets, where $1 < n <= m$.

$p = p1 \text{ ^ } p2 \text{ ^ } p3 \text{ ^ } \ldots \text{ ^ } pn$

The above equation also suggests that packet can be coded from packets belonging to a subset of m streams. We shall refer the coding from $n < m$ streams as “defective coding” for future reference because a coding node does not have the chance to obtain maximum coding benefit. However, we have come across this kind of situations in our investigation later.

Example: If a packet p1 from flow 1 has '01100100' in its data-field and a packet p2 from flow 2 has '11001000'. Then, by following Table 2.3, the new coded packet p3 has its data field obtained from p1 ^ p2 = 01100100 ^ 11001000 = 10101100.

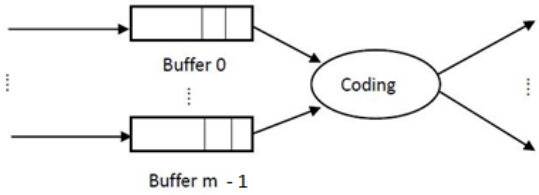

Figure 2.3: Coding-Node Behaviour

Figure 2.3 shows the scenario where a coding node wants to perform network coding on data traffic streams from *m* different logical links (incoming data flows). It depicts the model of a single server queue with *m* infinite buffers (i.e. sub-queues), one corresponding to each data flow. The information about the streams involved in coding is sent as an ICI (Interface Control Information, an interface provided by OPNET)[1] along the coded packet. Note that there can be more than one logical links (flows) per physical link although our network examples later all have one logical link per physical link.

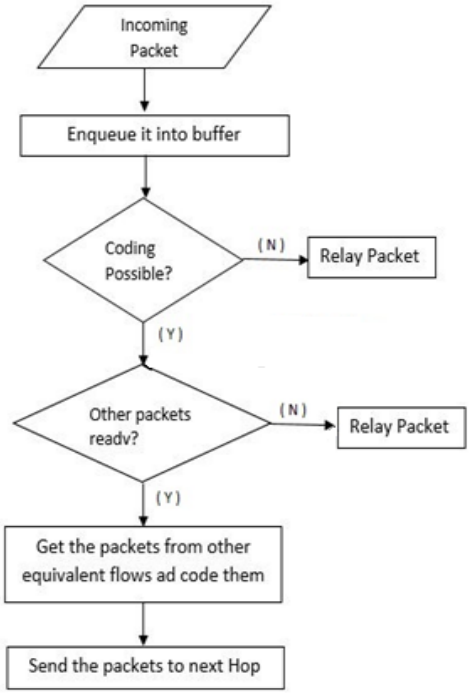

Figure 2.4: Coding-Node Flowchart

1 One could have put this information into the control field of the data packet itself. At the moment our data packet does not have a field (no space) for this purpose.

Figure 2.4 is the flow chart summarizing the coding operation. The coding possibility is assessed against the conditions explained in next section. Locally, once the flow is assigned with coding-possible or coding-impossible flags, it would stay with them until the coding conditions change in the network or until the end of the lifetime of the flow. It would save the efforts and computational time in checking the coding conditions every time when a packet arrives. Note that it is possible that not every buffer is non-empty when a packet needs its counter part to code together. In general, this packet is made to wait a time in the range of [0, *w*] seconds. After *w* seconds, it is just coded with whatever available and sent out. Fig. 2.4 indicates the case of *w*=0 implemented in our algorithm. Note that even without waiting, a packet is still required to go through queueing (if present) in its own buffer before the coding process.

2.3.3 Decoding Node Behaviour

A decoding node can be an intermediate node or destination node. Figure 2.5 below represents the operation of decoding-node, in that if *n* packets are coded together then, it would need n-1 packets to obtain orginal decoded packet. i.e. an orginal packet p1 can be obtained by XOR-ing coded packet p with n – 1 other original packets using the following formula:

p1 = p ^ p2 ^ p3 ^ … ^ pn.

Like Section 2.3.2, one may have to handle the situation of "defective decoding" when n<m, and remedies are proposed to make the decoding node aware of this later on.

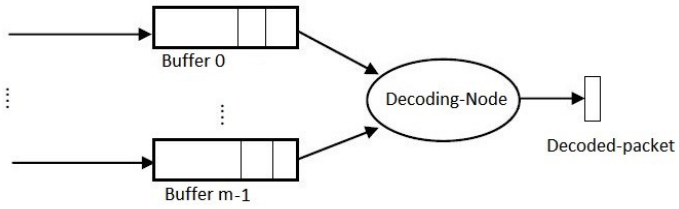

Figure 2.5: Decoding-Node Behaviour

For example, from Section 2.3.2, contents of the coded packet '10101100' can be XOR with any of the original packets '11001000' from Flow 2 to obtain original packet '01100100' of Flow 1.

Figure 2.6 below is the operation flowchart of a decoding node. When it receives a data packet, it would check the status of coding-bit as defined in the packet structure to see if it is set or not. If the Not-Set bit is set, it indicates the "packet is not coded", and would just be treated

as a normal packet. If it is a coded packet, the node would have to gather other equivalent packets from *n*-1 streams so that it can again XOR it to obtain the original packet. The streams involved in coding can be obtained from ICI which has been sent along the coded packet by coding node. We say decoding is successful if all *n*-1 decoding packets are found. Otherwise, the decoding fails to obtain the original packet. The coded packet would be marked as a bad packet and discarded.

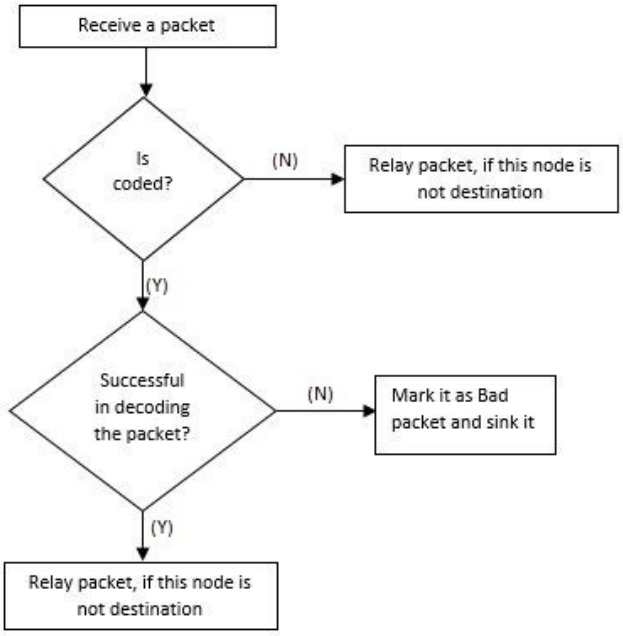

Figure 2.6: Decoding-Node Flowchart

2.4 OPNET Models

OPNET has a sophisticated workstation-based environment for the modeling and simulation of communication systems, protocols and networks in order to simulate communication networks with detailed protocol modeling and performance analysis. On the other hand, it is relative easy to use once its design concepts are understood. OPNET consists of four major components: Project Editor, Node Editor, Process Editor and Packet Editor [YaLu12]. Project Editor is used to build the topology of a network and provide the basic analysis capabilities and simulation. In the node editor, we can construct a node with different objects that define various interfaces. The process editor consists of several states connected with transaction conditions; the behaviour of the process is specified using C/C++ language features. The Packet editor is used to define the internal structure of a packet and obviously can contain many different fields mimicking real packets.

As stated in Ch.1, we have used OPNET Modeler for all our simulations. Here, we present the network, node, process, link and packet structures/models that we have used to implement our alogrithms.

2.4.1 Network Model

The network model gives the layout and communication among the components in the network. The components can be subnets, nodes, links, etc. Examples can be found in Fig. 2.2 and Fig. 4.5.

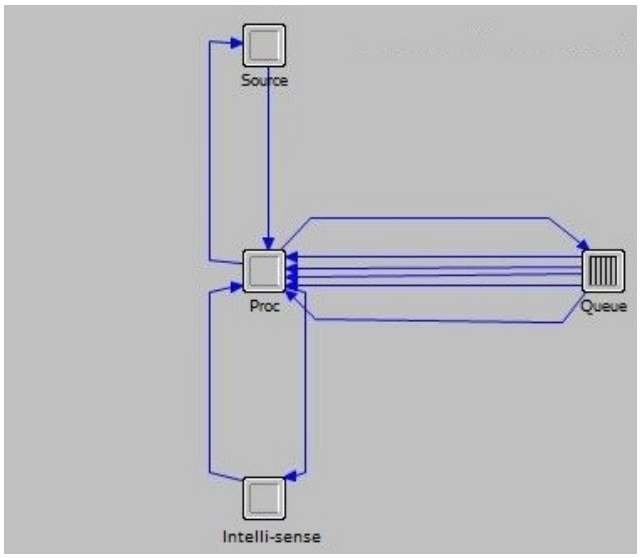

Figure 2.7: OPNET Node Model

2.4.2 Node Model

The node model gives information of each node under a network model. It gives the communication relationship with the neighboring nodes as well as its internal components. Each node in a network is captured by a node model such as Fig. 2.7 for a source node of the network in Fig. 2.2. Fig. 2.7 shows the communication between a module “Proc” in the third layer with a module “Source” in the fourth layer. It also communicates with the module “Intelli-sense” to gather information and update the tables about neighbour nodes along the path. Hence “Proc” is the central component for all the activitites of a node. Furthermore, all of the actions mentioned in Section 2.3 are implemented in “proc” module. The module “Queue” is used to buffer the overheard packets (i.e. in wireless network, nodes generally communicate over one channel, thus, the packets exchanged between any two nodes can also be heard by any other node with in the tramission range) and to emulate the store/forward operation of data packets.

Note that there is no communication with a process in the wireless Physical layer. This lower layer is modeled through the link models with configuration settings like bit error rate, interference and propagation delay to be presented in Section 2.4.4.

As discussed in Section 1.4, we have created a general node instead of task-specific one. Thus, the same node model in Fig. 2.7 can be used to model a node in any of four possible capabilities (i.e. coding, decoding, relaying, or innovative) depending upon its designated role.

2.4.3 Process Model

Each process module in a node model is captured through a process model. A process model consists of states representing different operations of a process. It performs logical operations on data and the conditions that caused these operations. Execution control can be only one of these states depending upon the type of event occurs.

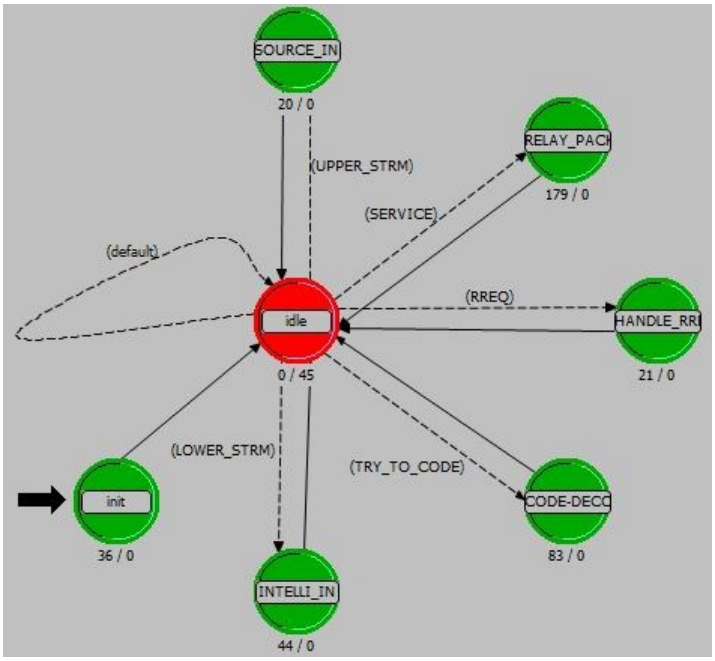

Figure 2.8: "proc" Process Model

Figure 2.8 shows the model for “proc” process in Fig. 2.7 which is used in both the destination and the intermediate nodes (whether it is later designated as coding, decoding, innovative, relay node.)

The “init” state is entered (executed) at the start of simulation. On packet arrival from upper layer, source node triggers the control to be executed in “the SOURCE_IN” state. If it is the first packet, it would perform Action-A (from source node actions) of algorithm in Section 2.3.

Likewise, “INTELL_IN” is responsible for handling packet arriving from lower layer, and it checks whether a packet is data packet or RREQ/RREP packet. On receiving RREQ/RREP packet, an event for “HANDLE_RR” state occurs and the control would start executing in this state.

Intermediate and destination nodes would behave differently depending upon whether a RREQ or RREP has been received. Intermediate nodes would perform Action-A or B (from intermediate node actions). Whereas destination nodes would perform Action-A (from destination node actions). Source node would process RREP packet following Action-B (from source node actions).

On the other hand, on receiving first data packet intermediate nodes would perform Action-C (from Intermediate Node Actions) and depending upon its designated role, it would either code/decode packets in “CODE-DECODE” state or just relay packets from “RELAY_PACKET” state. Destination nodes would perform Action-B (from destination node actions). The actual number of packets used (<=m) to produce a coded packet is also recorded and sent to the decoding node using the ICI (Interface Communication Information) construct in OPNET.

Figure 2.9 below shows the process model for source module in Fig. 2.7. This state is executed in a source node (for generating data and RREQ packets) and destination node (for generating RREP packets). The “init” state is the same as that in the “proc” process and gets executed at simulation start.

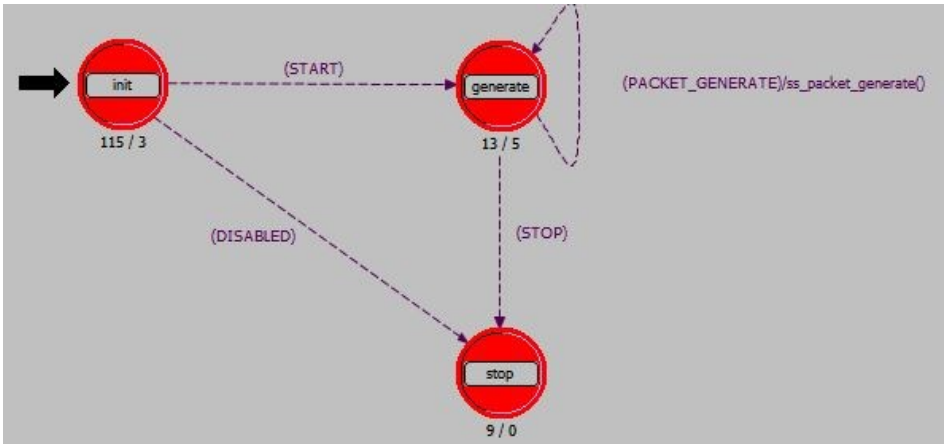

Figure 2.9: Source Process Model

The state “generate” is responsible for generating packets and “stop” gets executed at the end of simulation to write statistics.

Figure 2.10 below represents the process model for “queue” module in Fig. 2.7. It consists of 4 states, “SEND_HEAD”, “BRANCH” and “INS_TAIL” are used to manage packets in a queue

fashion and gets executed for all packets reaching at all nodes (source, Intermediate or destination). However, “GET_INNOV” only gets executed for intermediate nodes if their designated role is to be an innovative node. It can retrieve innovative packets from queue.

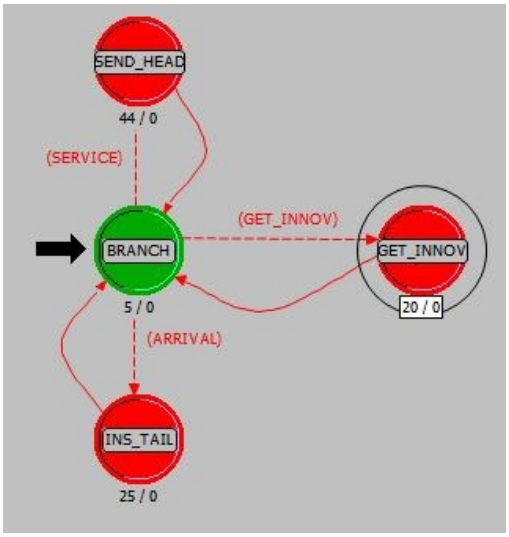

Figure 2.10: Queue Process Model

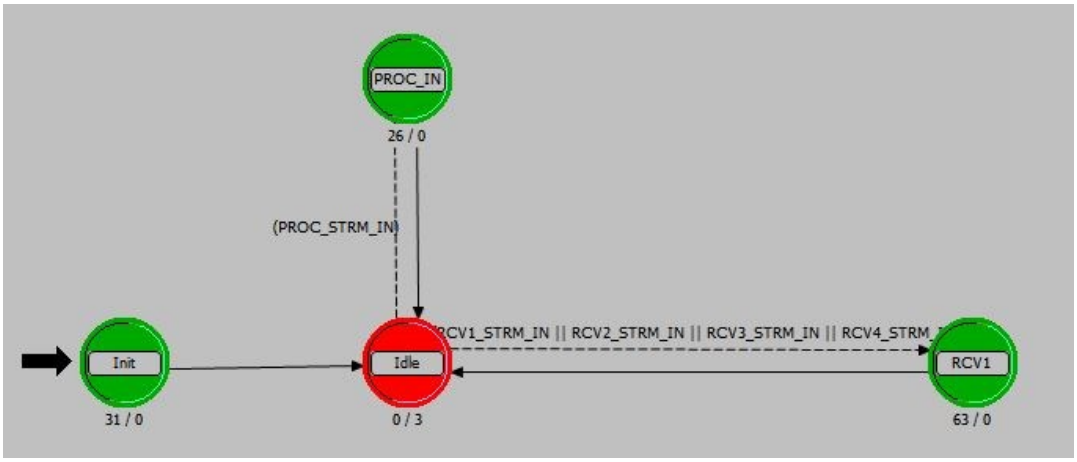

Figure 2.11: Intelli-sense Process Model

Figure 2.11 shows the process model of “INTELLI_IN” module in Fig. 2.7 and is the simplest of all. It only checks if a packet is coming from upper or lower layer. Packets from upper layer will be routed to next hop via interface information obtained from routing table and whereas, lower layer packets will be handed over to “proc” module for processing based on Section 2.3 algorithm. The “init” state is the same as that in the “proc” process and gets executed at simulation start.

In summary, the discussion above has described the process models that are commonly used among all node models to carry out all 4 different node operations (coding, decoding,

innovative or relaying).

2.4.4 Link Models

The Link model provides the physical layer properties of the physical communication link/medium such as bandwidth, frequency, transceiver, channel, noise, etc. In OPNET, operation in the physical layer is modeled through a number of pipeline stages and in each stage there are some parameters need to be set to define the operation of that stage. Each of such parameters can be set to one of many pre-defined values accessible through a drop-down menu option or OPNET calculates them dynamically based on instantaneous values for other parameters. Out of all such parameters, noise, signal power and *ber* (bit error rate) are the most important to us because they are important contributors to packet error probability (*pep*) used in our performance evaluation throughout.

OPNET models two different types of network noise: Interference and background noise. Interference noise occurs because of concurrent packet transmissions. Thus, higher the number of transmissions higher is the probability for interference noise. On the other hand, background noise accounts for noise sources other than explicitly modeled interferers such as galactic, thermal, or urban noise depending upon the environment in which the receiver is operating. Noise accounts for SNR (Signal to Noise ratio) and *ber* on incoming packets at receiver. Noise can be modeled through OPNET program by editing links in link editor for the wired networks and by changing the signal power for wireless networks [YaLu12].

Packet loss probability *p* is the percentage of packets lost on its way from source to destination. In this work, it is the same as packet drop probability (or packet drop rate). It is given as the sum of packet error probability (or packet error rate '*pep*') and number of non-decoded coded packets dropped at the decoding nodes. i.e.

$p = pep + pd$,

where *pd* is the percentage of coded packets dropped at decoding nodes because they cannot be decoded to yield useful information. They are marked as bad packets and dropped. On the other hand, packet error captures the packets in error due to bit errors. Packet error probability is related to bit error probability (or commonly called the bit error rate *ber*) by,

$pep = 1 - (1 - ber)^L$, where '*L*' is the length of a packet in bits.

In OPNET, *ber* is determined in one pipeline state according to the SNR (Signal to Noise

Ratio which is the ratio of signal power to network noise) of a signal modulation scheme. Two major sources of noise power are interference and background noises. Interference noise occurs during concurrent packet transmissions. The higher the number of transmissions, higher is the probability for interference noise. In our work, we use the default OPNET options of Gaussian noise model (capturing the two major types of noises) and the BPSK (Binary Phase Shift Keying) signal modulation scheme.

For evaluating one of our performance measures "PDR vs Packet Error Rate" as you will see in later chapters, we need to obtain different values of *pep*. From above relationship between *pep* and *ber*, it can be seen that *pep* is directly proportional to *ber* i.e. *pep* increases as *ber* increases. In OPNET, *ber* can be modified by regulating transmitter power of the node which in effect, changes *pep*. For example, transmitter power of 0.01 corresponds to 10 % *pep* and transmitter power of 0.005 corresponds to 33 % *pep*. In reference to later chapters, it can also be observed that network performance gets significantly affected by SNR and node speed.

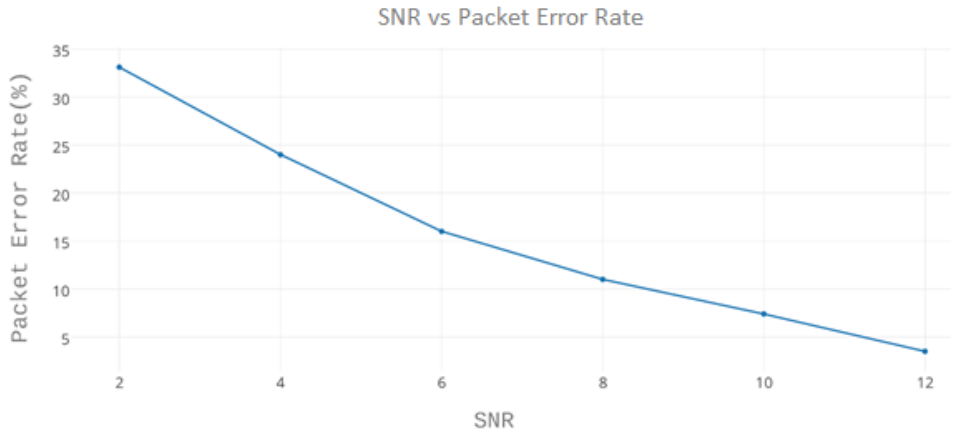

Figure 2.12: Packet Error Rate (*pep*) vs SNR

Fig. 2.12 above shows the relationship between SNR and *pep* for a small random Ad Hoc network, with a default reception sensitivity and a node speed of -95dBm and 5 m/s respectively. As expected for *pep*, it is inversely proportional to SNR i.e. with increasing SNR *pep* decreases or vice-versa. For example: *pep* is at 33% when SNR is 2 but as SNR is increased to 12, *pep* decreases to reach 4%.

Figures 2.13 below shows the comparison between small and big random mobile Ad Hoc networks for the *ber* performance as a function of node speed when reception sensitivity and node speed are set to their default values i.e. -95 dBm and 5 m/s respectively. The *ber* curve for

the small network increases with node speed and deteriorates (increases) more rapidly as speed increases. For example, *ber* for the small network is 25µ ($25*10^{-6}$) when nodes are idle (i.e. speed = 0 m/s) and reaches 90µ ($90*10^{-6}$) at 10 m/s. This is because SNR is decreasing with node speed but more at high speed due to more noise accumulated at higher speeds. The *ber* performance of the big network is similar but better (lower). This is because there are more nodes within same area. Hence the stronger signal power from a neighboring node decreases SNR and provides better connectivity.

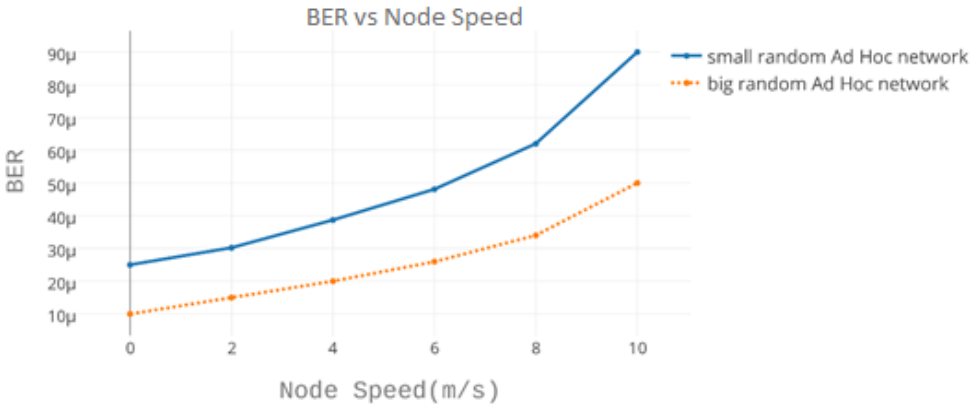

Figure 2.13: Bit Error Rate (*ber*) vs Node Speed

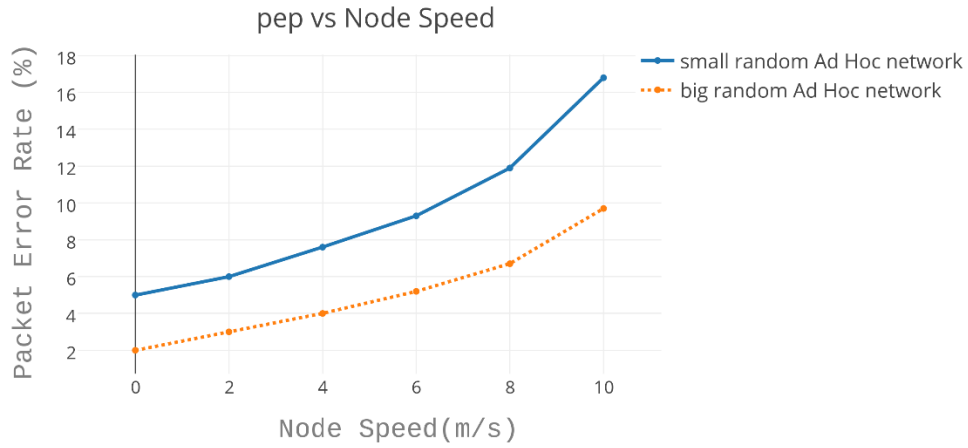

Figure 2.14: Packet Error Rate (*pep)* vs Node speed

Almost similar observations can be made for the *pep* performance from Fig. 2.14 above i.e. increasing with node speed and is always higher for small network because of relatively worse connectivity and hence, lower SNR. For example: *pep* for small random network at speed = 0 m/s is 5% and that for big random network is 2% whereas at node speed = 10 m/s, *pep* for small random network is 17% and that for big random network is 10% respectively.

Note that the equation pep = 1 – (1 – ber) ^ L in Section 2.4.4 can be shown to be pep ~ L*ber as an approximation when ber is very small (<<1). This linear approximation is in fact observed when comparing 2.14 (per) to Fig. 2.14 (ber) as a function of node speed.

2.4.5 Packet Model

The packet model captures the structural format of a packet. A packet consists of many different types of fields required for containig control and data infromation. There are only two different types of packet format we use throughout our work, namely, the RREQ/RREP and the data packet format. They differ in size and format as follows.

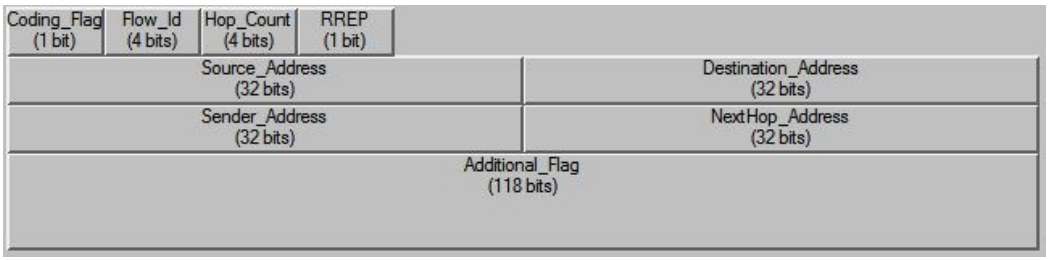

Figure 2.15: RREQ/ RREP Packet Structure

Figure 2.15 shows the packet structure for RREQ/RREP. We only need one packet structure for both because the "RREP" field indicates whether a packet is RREP or RREQ. The “Coding_Flag” filed indicates whether a packet is coded or not. “Flow_Id” is used to store flow id, “Hop_Count” stores number of hops have been traversed by a packet. Address information is stored inside a packet to identify source, destination, last and next hop addresses. All above discussed fields constitute control information about the packet. As explained in intermediate node actions in Section 2.3 algorithm, RREQ on its journey to the destination stores a list about the nodes that a RREQ/RREP packet has been traversed through and their one-hop neighbour nodes. The list gets stored into the “Additional_Flag” field in the similar tabular format as shown in Table 2.1.

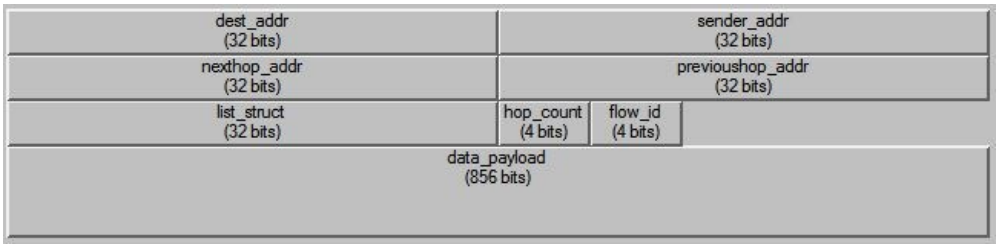

Figure 2.16: Data Packet Structure

Figure 2.16 presents the structural model of data packets. It consists of control and data payload fields. The fields of “dets_addr”, “sender_addr”, “nexthop_addr”, “previoushop_addr”, “list_struct”, “hop_count” and “flow_id” constitute the control information of a packet whereas the data_payload field is used to hold data. The fields of “sender_addr”, “dest_id”, “flow_id” and “data_payload” are updated by source node and stays constant for the lifetime of a packet (i.e. until it reaches destination). A source would store the packet sequence number and the role designations for different nodes in the form of binary data in “data_payload” field. The rest of the control fields are updated at each hop that a packet traverses.

As discussed in Section 2.3, an intermediate node would see if the flow is registered as coding-possible from the “flow_id” field. It then performs coding with equivalent packet from other flow by folowing procedures mentioned in Section 2.3.2 and 2.3.3. Since there can be only a subset of the *m* packets involved in coding, packet-id of those packets will be sent along coded packet using ICI (Interrupt Control Information, an OPNET’s interface with which some meta-data can be sent along the packet). In real life situations (where there is no ICI), the packet-ids can be included into the coded packet itself from which a decoding node can retrieve to identify the packets for decoding.

The performance evaluation in Chapters 3, 4, and 5 later use data packet of different sizes. This is done by varying the length of data_payload while keeping the length of control information the same.

2.5 Example of Coding Nodes Arising from Mobility

In a mobile network, as discussed in Section 2.3, both topology and coding chances need to be re-evaluated quite often. The OPNET built-in AODV model is modified so that the route re-evaluation is done whenever source re-initializes route discovery. The best path depending upon the algorithm (given in Section 2.3) has to be chosen again.

This can be better illustrated by explaining an example. We have simulated two different types of Ad Hoc networks as given in Chapters 4 and 5. In Ad Hoc networks with regular trajectories (or regular Ad Hoc networks) as discussed later in Chapter 3, all nodes move at same speed and trajectory such that the proximity of the nodes stays the same throughout the simulation. Therefore, the selected paths can remain the same and route discovery need not be re-evaluated; this is a less interesting case to us. In the case of Ad Hoc networks with random

trajectories (or random Ad Hoc networks) like one in Fig. 5.7, nodes are moving with different speeds and random trajectories. Relative proximity of the nodes changes quite frequently and route-discovery is often invoked in this case.

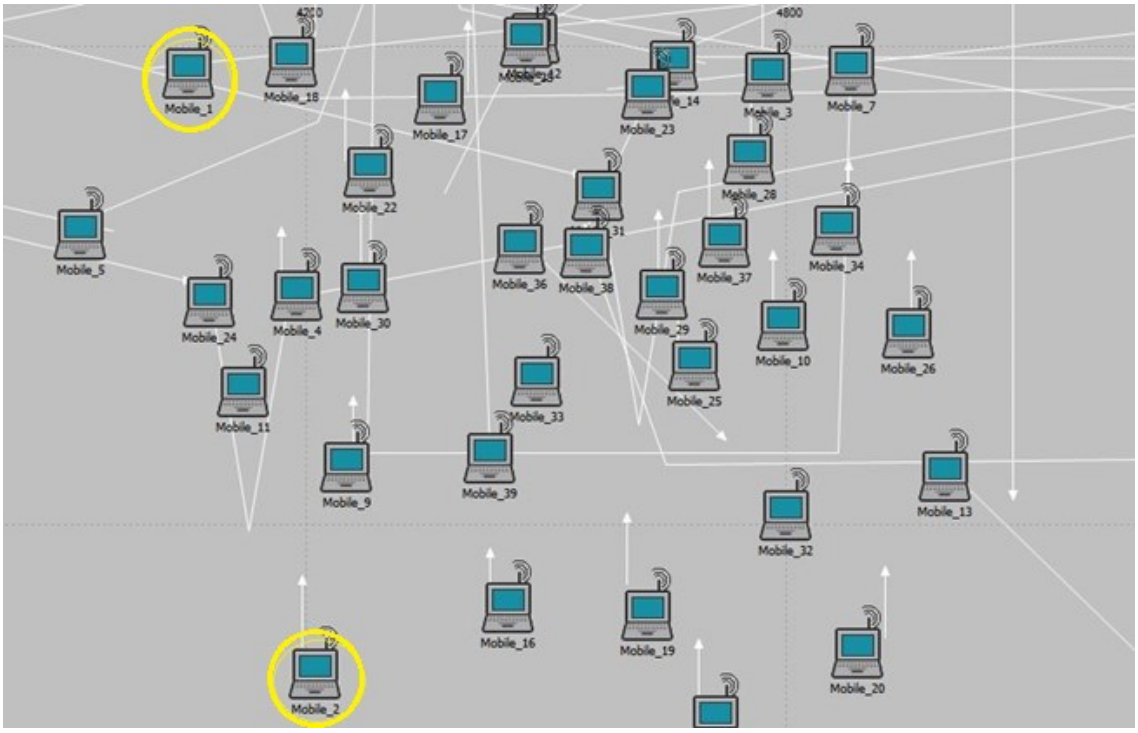

Figure 2.17: Random Ad Hoc Network snapshot at t = 100 s

Figure 2.17 gives a snapshot of big random Ad Hoc network as given in Fig. 5.7 taken at t = 100s (i.e. after 100 simulated seconds). It can be seen that all nodes are moving on random trajectories marked with a line and arrow in white, thus changing their positions randomly. Node 1 is the source node marked by the top yellow circle and Node 2 is the destination node marked by the bottom yellow circle.

Since the source and destination are too far apart, the packets can be relayed through intermediate nodes only. As discussed in Section 2.3 that the optimal path can be obtained by serializing the operations of AODV and generalized CAR. Specifically, when source does not have a valid route to destination, AODV starts route discovery. With our modifications in the built-in model, AODV algorithm at the source node (Node 1) informs the “Intelli-sense” process about the topology change and all possible paths to destination, which in response invokes the “proc” process to identify coding chances in the network.

By following Action- A of the Source Node Actions in Section 2.3, the “proc” process at the source node would request the “source” module to generate an RREQ packet in order to

capture coding-chances in the network. By following Actions A1 and A2, the source node (Node 1) further appends its one-hop neighbor information into the RREQ packet (by using the list maintained in the "Additional_flag" field) and broadcasts it. Likewise, Intermediate nodes also process the RREQ packets by following Action-A of the Intermediate Node Actions in the "proc" process in order to append its one-hop information into the list maintained at "Additional_flag" field. Then the intermediate node broadcasts it further for it to find its way to the destination. The destination node (node 2) on receiving an RREQ packet, invokes Action-A1 (from destination node actions) and generates an RREP packet (it also embeds the information about the node-list of all the nodes along the path and their one-hop neighbors into the "Additional_flag" field of RREP packet as shown in Fig. 2.15) in the "proc" module and routes it back to the source.

Intermediate nodes 11 and 5 on receiving an RREP packet would follow Action-B of the Intermediate Node Actions to find coding chances in the network. Since this is the only flow at the moment and there exists no other coding chances yet, according to Action-B2a, Nodes 5 and 11 would mark themselves as not coding-possible nodes (i.e. by setting the "Coding-Flag" field to "0") into the RREP packet (this is done through the "CODING_DECODING" process state in Fig. 2.8). By following Action-B4, RREP is routed back to node 1 (source node).

Node 1 on receiving the RREP packet would follow Action-B (from source node actions) and evaluate the coding chances in the state "CODING_DECODING". This state also mark all nodes as coding-impossible in the RREP since flow 1 (1 -> 2) is the only flow in the network. Thus, the path chosen by node 1 is based on ETX (hop-count) and it comes out to be 1-5-11-2. Node 2 start sending packets to node 1 after (flow 2, 2 -> 1), but there is an existing flow in the network this time. Thus, intermediate nodes 24 and 11 on receiving RREP packet from destination node 'node 1' would follow Action-B of the intermediate node actions to check if they can be coding-possible nodes. From Action-B1, since there is no existing flow at node 24, it will mark itself as coding-impossible in RREP packet and send it to node 11. On following Action-B2, node 11 analyzes and determines that it can be coding-possible node since it is able to find decoding nodes for flow 1 and 2 i.e. node 5 for flow 2 (since it can overhear transmissions from node '24') and node 2 for flow 1 since it has actual packets from flow 2). It will mark itself as coding-possible and send RREP to the source node 'node 2'. Node 2 on receiving RREP packet follows Action-B1 from source node actions and analyzes that path 2-11-24-1 is the only path that coding degree greater than zero. Thus, by following step B2, Node 2 choses path 2-11-24-1 over others and thus,

start sending data packet by embedding role designations and node-ids into the first data packet sent (as mentioned in Action-C1). On receiving the first data packet, intermediate nodes 11 and 24 by following Action-C gets their role designation as coding node and relay node respectively. Thereafter, node 11 will start coding flows 1 and 2.

Figure 2.18: Random Ad Hoc Network snapshot, t = 200 s

Since nodes move around randomly and continutiouly, we use Figure 2.18 to show another snapshot taken at t = 200s to illustrate how the the coding node is changed. On comparing with t=100s, it can be seen that most nodes have moved from their last position (at t = 100s). The set of neighbor nodes have changed too. Thus it is required to run route-discovery again to destination. Th coding-opportunities need to be evaluated again as the topology has been changed. The path between node 1 and 2 is now 1-38-31-33-2 and 2-33-38-31-1 respectively, and node 33 becomes the new coding node in the manner like node 11 in t=100s by involving the same OPNET processes.

As suggested, there is no fixed route between the source and the destination when the nodes are constantly moving randomly. The coding nodes are changed as well. All these contributes to lower performance in random Ad Hoc networks when compared to their non-random counterparts as expected and will be demonstrated in future chapters. Note that when CAR fails to find any coding possible nodes in the network, the path chosen would be based on ETX. This is just the ordinary shortest path routing algorithm and there would be no coding

benefits for the flow.

2.6 Assumptions

Unless otherwise stated, the following assumptions applies to the remainder of this thesis:

1) Infinite data buffer B: Although this is not true in a real-life system, it can be approximately true because a large amount of memory can be implemented (at a low price) nowadays.
2) The node buffer is sub-divided into a number of sub-queues, one for each data flow passing through the node.
3) All data packets in a network have the same size to allow for XOR-ing in the coding operation. Extra zero bits are used to pad shorter packets.
4) Packet arrival is constant at a rate λ as it presents a worst case scenario as there is always traffic flows.
5) Queuing discipline at a node is FIFO and all sub-queues are served in round-robin fashion.
6) A one-hop neighbor can always be determined if a node can overhear its packets from near-by communication.
7) Links are bi-directional in half-duplex manner. That is a transmission can be started from either end of the link but only one at a time. This is because the same bandwidth is shared between the upstream and the downstream flows.
8) Packet arrival rates are the same for all source nodes. This allows us to have less factors/variable to be kept constant during systematic evaluation of different performance measures.
9) Shortest path routing algorithm is used for network without CAR and the distance metric is hop.
10) For wired networks, route discovery has already been done and each node has valid route to any other node in the network.
11) For mobile Ad Hoc networks, each node is moving at 5 m/s unless otherwise specified.

Chapter 3
Wired Networks

In this chapter, we shall investigate the benefits of using coding-aware routing in comparison to traditional network coding, in both small and big networks. The performance can be used as a bench mark (reference) for comparison with wireless networks in later chapter. Before we do that, we shall first provide information on our simulation and performance evaluations.

3.1 Performance Evaluation using Simulation

To evaluate the performance of our algorithm, we shall use the OPNET simulation testbed whose network, node, process, link and packet models have been discussed Section 2.4. Some parameters of the small and the big networks are the same in their performance evaluation. Both study the effect of 3 different packet sizes of 256 bytes, 512 bytes and 1024 bytes. Contrary to bits/sec commonly used in circuit switched networks, we are using packets/sec as the information processing (e.g. in throughput) as commonly done in packet switching network where packet is the level of abstraction. This can be easily converted to bits/sec once the packet size is known. Here, the link/server rate is arbitrarily fixed at 16,384 bits/second (and is same for all links and nodes in a network) which gives rise to different packet rate when the packet size is changed. For example, using a packet size of 256 bytes, the link/server would be able to process 8 packets/sec[2]. Buffer size is infinite as previously assumed. Unless otherwise stated in some performance comparisons, packet error rate is 0 and packet size is 256 bytes.

In all performance evaluation, we have included the performance measures of network-coding for comparison with coding-aware routing. The following performance measures are used in our simulation and are defined as follows:

(1) Throughput: This is the average number of packets per unit time received at all destination nodes in a network. In our simulation, we find the ratio of total number of packets successfully accumulated at sink node to the total simulation time.

[2] Based on packet size of 256 bytes, server/link can process 16384/(256*8) = 8 packets/sec.

(2) ETE (End-to-End) Delay: This is the average time spent by a packet from its arrival at source until it is received at its destination node. The packets that are lost before reaching destination due to error do not contribute to average delay. For multiple flows in network, ETE delay is an average of ETE's at all sink nodes. Un-decoded packets receiving at destination are not included in this average either.

(3) Mean Queue Size: This is the time average number of packets waiting for service seen by an arrival from all the sub-queues among all coding nodes in network when system achieves steady state. For multiple coding nodes in the network, mean queue size is given by an overall average of all mean queue sizes of all coding-nodes.

(4) Packet Delivery Ratio (PDR): This is the percentage of packets from the source that are successfully received at the intended receiver. Mathematically, this is $1 - p$.

All simulations are run in a computer system using Intel Core2 T6600 processor at 2.20 GHz with 4 GB of memory. The operating system is Windows 7. We have determined that a typical simulation would collect about 36000 packets to reach the steady state of a statistics (e.g. mean queue length). A typical simulation would take about 80s to complete. OPNET also provides 95% confidence interval at the end of the simulation. An example is provided in Fig. 3.1b which indicate the intervals are very small. Similar observations are made for many other performance curves. For clarity purpose, we have omitted 95% confidence intervals for them. For later performance figures, 2 decimal points are used for computed values as opposed to integer values that are read directly from graph.

3.2 Small Network

We use the small network in Fig. 2.2 with N= 11 nodes and F=13 links for our study here. Two source-destination communication pairs (and their paths) are used to validate the performance gains of the coding-aware routing over without coding-aware routing. Nodes 1 and 7 are the source nodes with Nodes 6 and 11 being their corresponding destinations, respectively. Packet arrival rates to both sources are the same as assumed in Section 2.7. Packets arriving at any intermediate node would be buffered into their respective sub-queue while waiting in order to get service. Since there are only two flows evaluated here, we need only m=2 buffers for the coding node in our simulation (although a node can have more than that if needed).

Nodes 1 and 7 start sending their packets at t=10 s and 20 s, respectively. At *t*=10s, since packets at node 1 would be the only flow in the network, it will pick the path n1-n2-n3-n4-n5-n6 because the ETX /Hop count of this path has the lowest value of 6. At this point, there are no coding benefits. After the arrival of second flow at node 7, node 4 will serve as the coding node according to Step CC* of Intermediate Node Actions of our generalized CAR algorithm in Section 2.3. Nodes 10 and 5 will serve as decoding nodes because they can over-hear the packet transmissions of Flow 1 and Flow 2 before they get coded (i.e. original packets) from innovative nodes 2 and 8 respectively. Coding Node 4 can now code the packets from the two flows that are buffered at the two sub-queues and broadcast them. As discussed in Section 2.3, the first data packet received at intermediate nodes carries the information about their role designations. From this, the decoding nodes 10 and 5 will get to know that innovative nodes 2 and 8 are assigned in order for them to decode the packets successfully.

The benefits arising from this coding-decoding operation will be illustrated in the performance evaluations below.

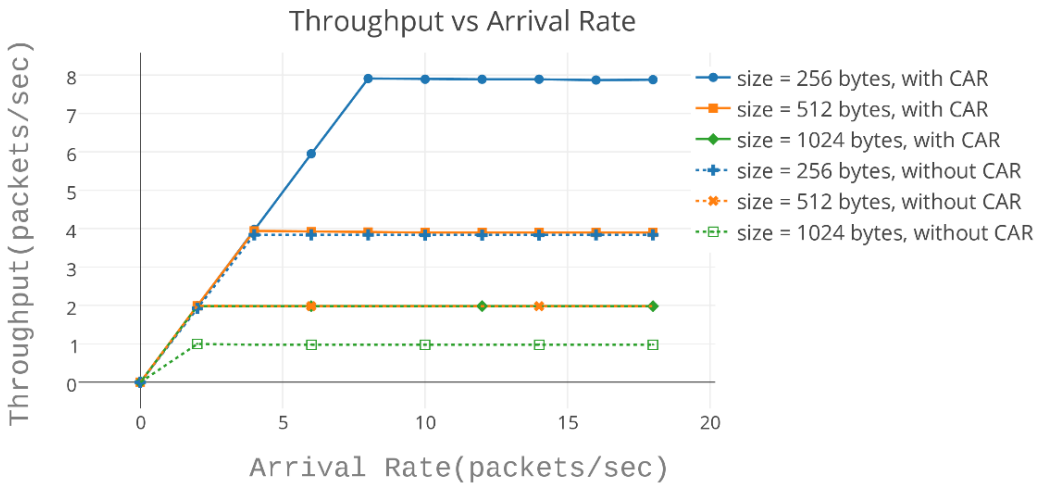

Figure 3.1a: Throughput vs Arrival Rate for Small Wired Network

3.2.1 Throughput

Figure 3.1a gives the throughput at the sink as a function of the arrival rate at the source for different packet sizes. For a size of 256 bytes using CAR (Coding Aware Rate), the throughput increases linearly w.r.t. the arrival rate and levels off beyond 7.8 packets/second. This is because the throughput depends on the service/link rate. Since the link rate is fixed at 16384 bps i.e. 8 packets/sec for a packet size of 256 bytes, the throughput is capped at 8 packets/sec. Until packet arrival rate is lower than link rate, every curve rises linearly up because system can service

incoming packet right away. The observations for different packet sizes using CAR are similar except that maximum throughputs (levelling off levels) are progressively decreasing (4 and 2 packets/sec) and the arrival rates beyond which maximum levels are reached are also decreasing (5 and 2.5 packets/s) as the packet size is increased from 256 bytes to 512 and 1024 packets, respectively. As exemplified above, the maximum packet/s throughput has to decrease when packet size increases for a fixed link rate in bps.

As a comparison to show the advantage of CAR, we have also provided the results when no CAR is used. As one can see, due to the coding benefit of the coding-aware routing the system is able to achieve almost double the performance. For example, for a packet size of 256 bytes, we have 7.95 packets/sec using CAR and 3.97 packets/sec when no CAR is used. This is because of coding benefit using CAR, per unit time Node 4 would be able to send almost twice as packets as network without CAR.

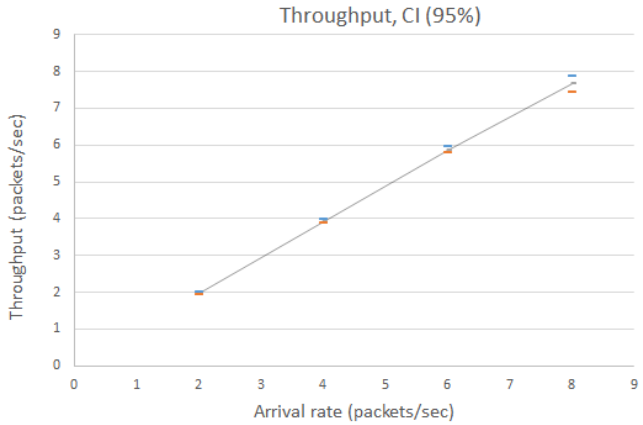

Figure 3.1b: Throughput vs Arrival rate for Small Wired Network

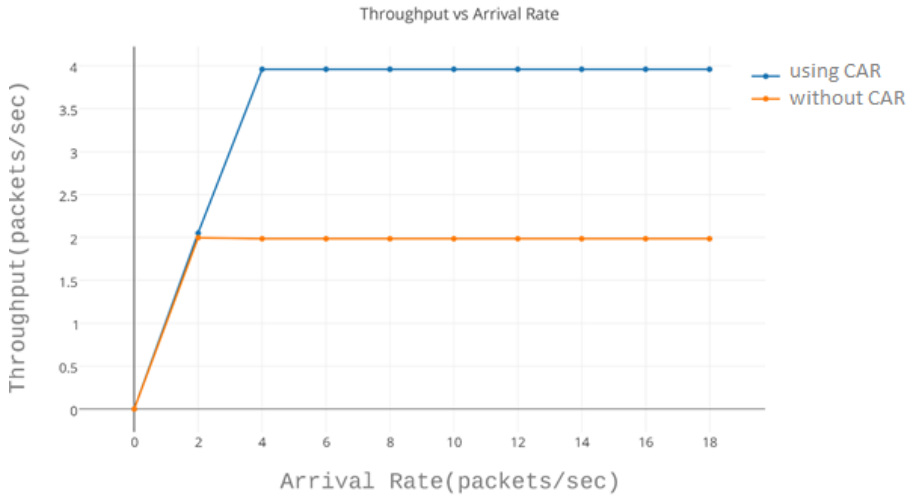

Figure 3.1c: Throughput vs Arrival Rate for Small Wired Network

Figure 3.1b shows the 95% confidence interval (CI) of the throughput for packet size of 256 bytes. The interval is defined by the blue and orange dots with the mean shown in solid grey line. The interval for each data point is obtained from 3 simulation runs, and each with different seed. It can be seen from the Fig. 3.1b that the upper and lower bound of intervals are very close to the mean throughput.

Figure 3.1c shows the individual throughput at node 6 for packet size of 512 bytes. It can be seen that throughput at node 6 for both CAR and network without CAR is almost the same as the average throughput in Fig. 3.1a. This is because both flows 1 and 2 have very similar characteristics i.e. same link/server rate, packet arrival rate, coding-degree, buffer size, etc. and thus, they yield similar performance.

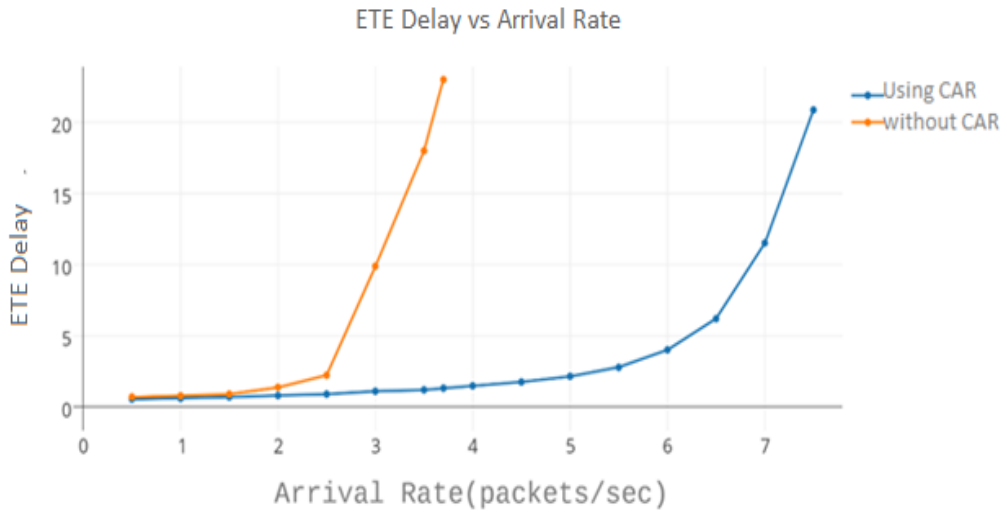

Figure 3.2: ETE Delay vs Arrival Rate for Small Wired Network

3.2.2 ETE Delay

Figure 3.2 compares the ETE delay performance of using CAR with that without using CAR (i.e. where traditional NC is used). Packet size is set to 256 bytes. For network without CAR, the ETE delay is low and then shoots up beyond arrival rate of 2.5 packets/second. The system becomes unstable beyond arrival rate = 4 packets/sec which is the service limit of the server. Same observation can be made for network using CAR except that the ETE delay remains low for higher arrival rates and takes off only when 8 packets/sec is approached. This is because the CAR algorithm has found and designated node 4 as the coding node from the network information during route discovery. In network without CAR as explained in Section 2.3.1, node 4 will not recognize such opportunity (or does not has the capability) to setup node 4 for the coding operation and node 10 for decoding even it has the capability to perform network coding. Similar

to the throughput performance, we have obtained ETE delay performance for Flow 1 at node 6. The observations are again very close to that of average which can be explained similarly.

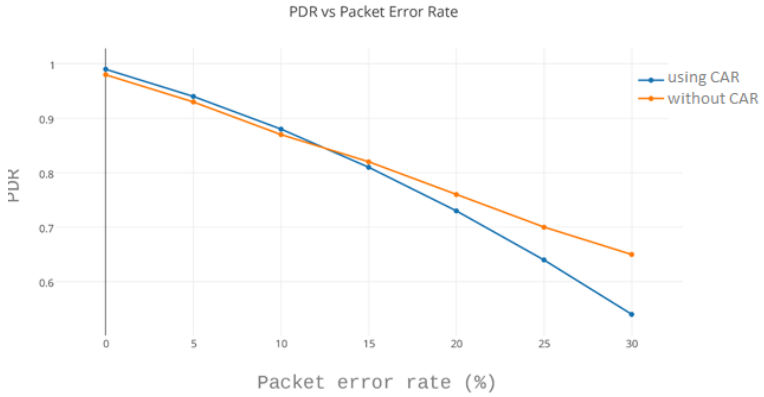

Figure 3.3: PDR vs Packet Error Rate for Small Wired Network

3.2.3 Packet Delivery Rate (PDR)

Figure 3.3 shows that Packet Delivery Rate performance at the sink node as a function of packet error rate (*pep*) when the packet size is 128 bytes and the packet arrival rate is set to 2 packets/sec. For network using CAR and zero *pep*, the PDR is at the highest level as expected but still at 0.98 only. The PDR performance decreases more with increasing *pep*, reaching approximately 0.54 at *pep*=30%. This can be explained by the fact that packets need to be decoded (at nodes 10 and 5) before delivering to their destinations at Node 11 and Node 6. Since it is less probable to find decoding packets with increasing *pep*, un-decoded packets were dropped. This is why the curve is not linear with *pep* since packet loss rate is going up faster with both *pep* and *pd* contributing to it.

The PDR performance when no CAR is used starts at ~0.97 when *pep*=0 which is smaller than the performance when CAR is used. However, it has a linear decrease down to ~0.675 at *pep*=30% which is much better than the case of using CAR. This is because there is no reliance on decoding packet to deliver packets. So the PDR is a function of *pep* only.

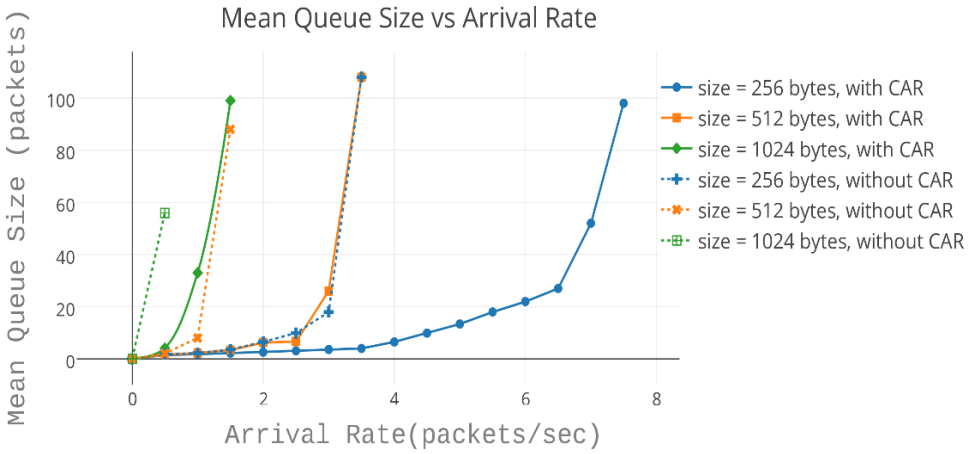

Figure 3.4: Queue Size vs Arrival rate for Small Wired Network

3.2.4 Mean Queue Size at the Coding Node

Figure 3.4 shows the mean queue size of the coding node for different packet sizes. For a size of 256 bytes using CAR, the mean queue size increases linearly w.r.t the arrival rate but, beyond 7.5 packets/sec it grows to infinite because server limit has been reached and node server cannot cope up with arriving packets. The observations for different packet sizes are similar except that for bigger packets, curve grows to infinite at lower arrival rates (i.e. at 3.5 packets/sec for packet size of 512 bytes and 2.5 packets/sec for packet size of 1024 bytes) as expected because when more bits are packed in each packet, the system would find it difficult to process and cope up with arriving data rates (in bps). So the stability limit for arrival rate (in packets/s) has to be lower.

As a comparison to show the advantage of CAR, we have also provided the results when no CAR is used. As one can see, due to the coding benefit of the coding-aware routing the system is able to achieve almost double the performance. For example, for a packet size of 256 bytes, mean queue size surges to infinite beyond 3.5 packets/sec arrival rate when no CAR is used and beyond 7.5 packets/sec when CAR is used. The same explanation for the ETE delay performance above can be used for the observation that the coding-node 4 is able to process almost twice number of packets as compared to network without CAR.

As a summary for this Section 3.2, our performance evaluation on a small network shows that by sensing coding-opportunities while finding route, code-aware routing can provide coding-benefits even while traditional network coding may fail. It would be interesting to see how CAR would perform in a big network to be investigated in the next section.

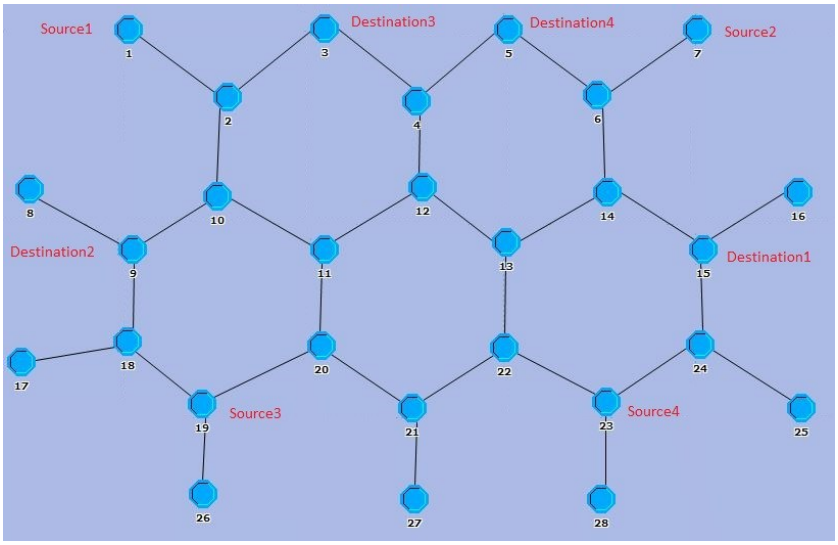

Figure 3.5: Big Wired Network

3.3 Big Network

We have confirmed significant savings on applying coding aware routing to small network in Section 3.2. Now, we shall see how it scales with bigger networks. Figure 3.5 shows a network consisting of *N* = 28 nodes and F = 32 links. Four source-destination pairs are used to evaluate the performance measure of CAR network against network without CAR. Nodes 1, 7, 19 and 23 are the source nodes with their corresponding destination nodes at 15, 9, 3 and 5, respectively. There is one data flows per source-destination connection. Therefore, we need at most m = 4 buffers at the coding nodes in this network. For notation convenience, we shall denote the flow for the source-destination pair of 1-15 as Flow 1; likewise Flow 2 for the node pair of 7-9, Flow 3 for node pair 19-3, Flow 4 for node pair 23-5. These flows are started at t=10, 20, 30 and 40s respectively.

At t=10s, when flow 1 is the only flow in the network, path chosen in case of both with and network without CAR scenarios is same i.e. {1-2-3-4-12-13-14-15} because the ETX /Hop count of this path has the lowest value of 7. After the arrival of flow 2 at t=20s, using the general coding-aware algorithm (from Section 2.3) allow the source node 7 to recognize the path {7-6-5-4-12-11-10-9} has a coding chance because node 4 can be a coding-possible node. Furthermore, nodes 10 and 14 can be the decoding nodes because they can overhear from nodes 2 and 6 the transmission of the original packets (i.e. un-decoded packets) from Flow 1 and Flow 2 respectively. Therefore, nodes 2 and 6 will be treated as innovative nodes for flow 2 and 1 respectively. Without CAR in comparison, there is a no-way for the network to have the above

knowledge to obtain the coding-possible path. Instead it just picks the path based on the lower hop count of 7 i.e. {7-6-14-13-12-11-10-9}.

After the arrival of flows 3 and 4 at t=30 and 40s respectively, the CAR network is able to pick the coding-possible routes {19-18-9-10-2-3} and {23-24-15-14-6-5} respectively for these two flows. Node 4 is an innovative node, and nodes 10 and 4 are the coding nodes for flows 3 and 4 respectively. Due to its inability to capture the system state (coding possibility, innovative nodes etc.) while sources in the network without CAR fail to find the coding-possible paths when using the ordinary shortest path routing algorithm.

In summary, there are 3 coding nodes identified by the CAR; node 4 for the flows 1 → 15 and 7→ 9, coding node 10 for the flows from node 7-9 and node 19→3 and coding node 14 for the flows from node 1→15 and node 23→5. Without using the CAR, we have the following paths; {1-2-3-4-12-13-14-15} for 1 → 15, {7-6-14-13-12-11-10-9} for 7→9, {23-22-13-14-6-5} for 23→5 and {19-20-11-10-2-3} for 19→3. Clearly, nodes 4, 10, and 14 can be the coding nodes even for the network without CAR. But since it has no way of telling that nodes in the downstream can perform coding, the coding-capable paths are not chosen instead a route based on hop-count is chosen. The benefits arising from this coding-decoding operation will be illustrated in the performance below in terms of throughput, ETE delay and PDR.

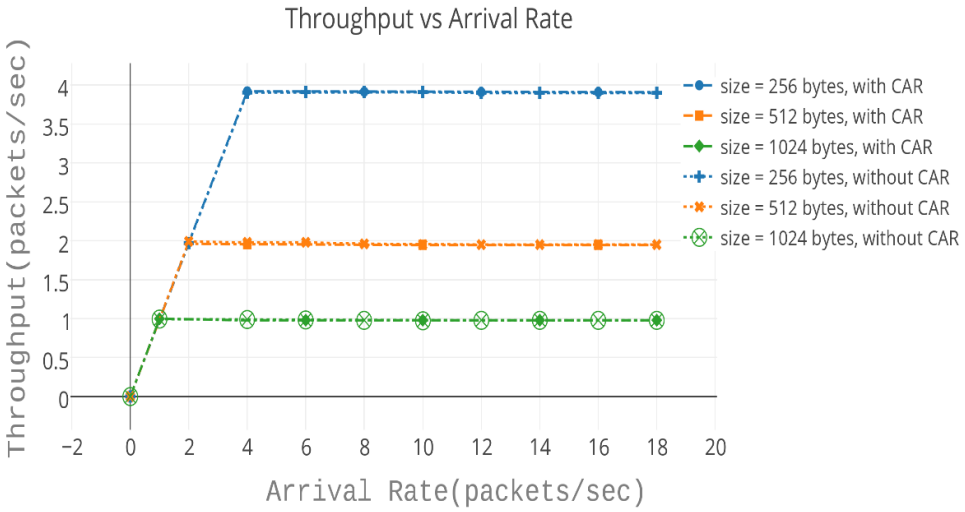

Figure 3.6a: Throughput vs Arrival Rate for Big Wired Network

3.3.1 Throughput

Figure 3.6a shows the throughput at the sink as a function of arrival rate at the source for different packet sizes. For a packet size of 256 bytes, the throughput increases almost linearly

with packet arrival rate and levels off beyond 4 packets/sec. The observations for different packet sizes using CAR are similar except that maximum throughputs (levelling off levels) are progressively decreasing (2 and 1 packets/sec) and the arrival rates beyond which maximum levels are reached are also decreasing (2 and 1 packets/s) as the packet size is increased from 256 bytes to 512 and 1024 packets, respectively.

In comparison to network without CAR, the throughput behaviour is similar to what we have obtained for CAR network. This is because flows 3 and 4 are still sharing the same bottleneck links[3] 9-10 and 14-15, and their bandwidth consumption remain the same even with packets coded from nodes 10 and 14. Surely, there are savings in terms of transmission time (since coded packets can be sent simultaneously to their respective streams), but as a network there is no improvement in terms of performance.

Unlike the comparison in Fig. 3.1 for a small network, the big network here has shown no CAR improvement over the network without CAR because of the bottleneck links (9-10 and 14-15). Similar to Chapter 3, throughput of an individual flow (e.g. flow 1 at node 15) is the same as the average of all flows because all the nodes along the path has similar characteristics and thus, yields very closer performance.

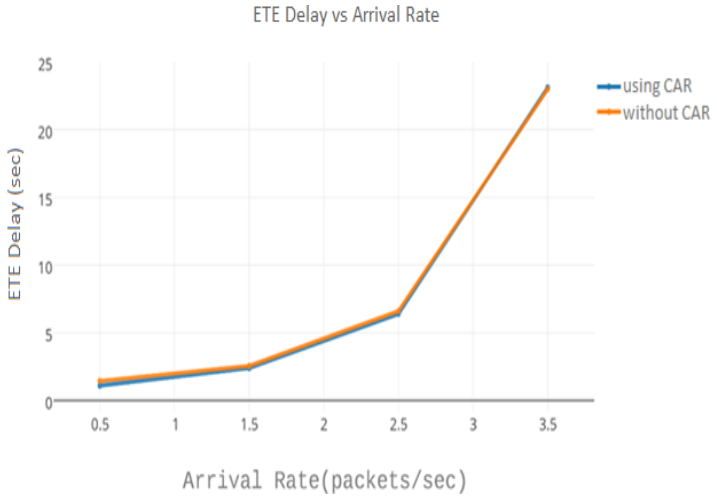

Figure 3.7: ETE Delay vs Arrival Rate for a Big Wired Network

3.3.2 ETE Delay

Figure 3.7 compares the ETE delay performance of CAR network against network without CAR. Packet size is set to 256 bytes. For network without CAR, the ETE delay is low and then shoots up

[3] More discussion of bottleneck links can be found in Section 6.1.

beyond arrival rate of 2.5 packets/second. The system becomes unstable beyond arrival rate = 4 packets/sec which is the service limit of the server. Same observation can be made for network using CAR because even with the presence of coding-possible node, this network cannot yield any performance improvements due to the bottleneck links 9-10 and 14-15 as discussed in Section 3.3.1.

As compared to small network, the ETE delay for the big network is little higher because of its size. As the packets needs to go through more number of intermediate nodes and hence, more delay is incurred before packet reaches its destination. Similar to throughput, we have obtained ETE delay for individual flow and we have observed performance very close to that of average.

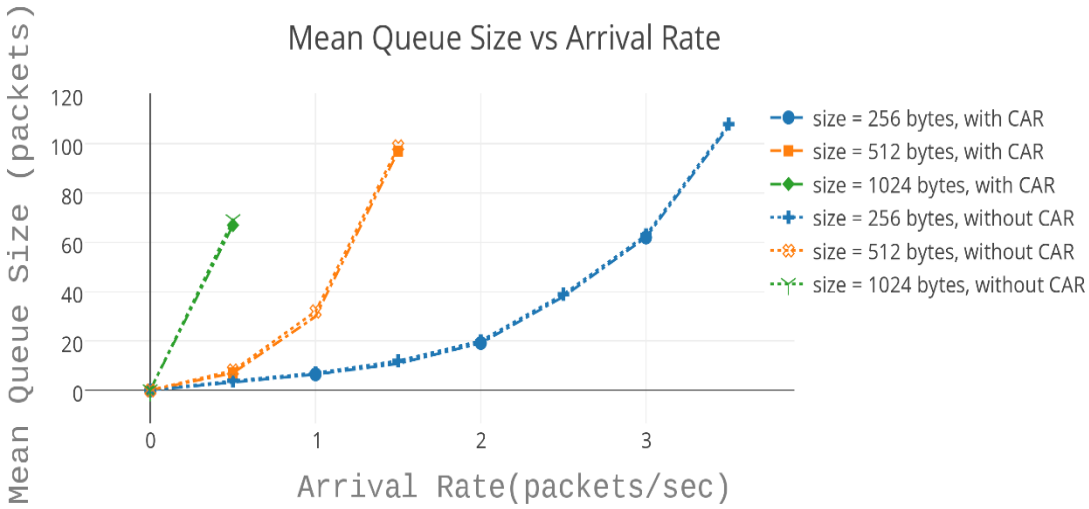

Figure 3.8: Mean Queue Size vs Arrival Rate for a Big Wired Network

3.3.3 Mean Queue Size vs Arrival Rate

Figure 3.8 shows the mean queue size of a coding node as a function of arrival rate when packet size is fixed to 256 bytes. Since in this network there are three coding nodes, thus the mean queue size is obtained by taking an average of mean queue size of coding-node 4, 10, and 14. For network using CAR and packet size of 256 bytes, the curve first increases slowly w.r.t arrival rate and then, more rapidly beyond 2.5 packets/sec. Beyond 4 packets/sec, the system limit has been reached and the curve grows to infinity. Similar observations can be made for different packet sizes, but since bigger the packet, more time it would take in transmission and results into faster queueing up of packets. Thus, system limit for bigger sized packets is reached earlier which can be confirmed from the Fig. 3.8 i.e. system limit for packet size of 512 bytes has been reached at 2 packets/sec as compared to 1 packet/sec when packet size is 1024 bytes.

Almost the same observations can be seen for the network without CAR. This is because CAR network besides able to be on the coding-possible path, failed to obtain coding benefits due to bottleneck links 9-10 and 14-15as discussed in Section 3.3.1.

3.3.4 Packet Delivery Ratio (PDR):

Figure 3.9 below shows that Packet Delivery Rate performance at the sink node as a function of *pep* (packet error rate) when the packet size is 128 bytes and the packet arrival rate is set to 2 packets/sec. For network using CAR and zero *pep*, the PDR is at the highest level as expected but still at 0.97 only. The PDR performance has a slightly concave decrease w.r.t increasing *pep*. It reaches 0.52 when *pep*=30%. This is because packet loss rate is the function of *pep* + *pd*, and there are more packets that are unable to get decoded (and thus dropped) at decoding nodes 10 and 14 when *pep* increases.

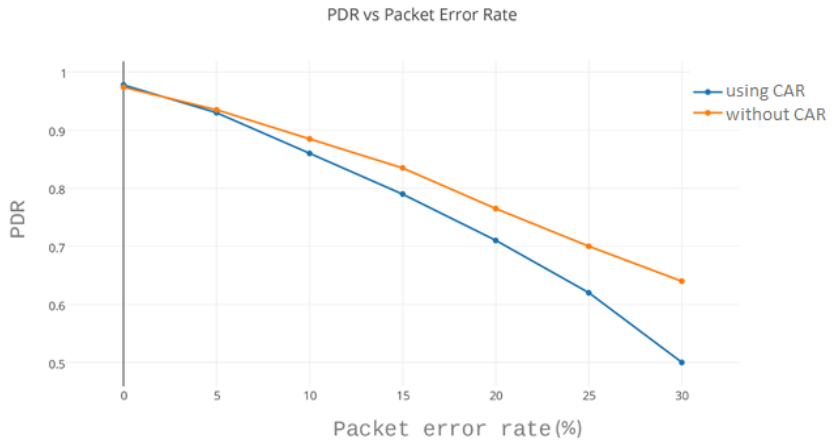

Figure 3.9: Packet Delivery Ratio vs Packet Error Rate for Big Wired Network

The PDR performance for the no CAR scenario is slightly worse by starting at ~0.96 when per=0 and decreases approximately linearly to ~0.68 at per=30% which is much better than the case of using CAR. This is because there is no reliance on decoding packet to deliver packets. So the PDR is a function of *pep* only.

As compared to the small network, PDR in this big network is slightly less because on average a packet needs to go over more number of links than small network and thus, more probable for a packet to get lost due to error.

3.4 Concluding Remarks

From the comparisons in both small and big networks, coding-aware routing can identify any

coding-possible nodes in the network to take advantage of the coding benefits where the traditional NC would fail to do. This is illustrated in the performance evaluation of throughput, mean ETE delay and mean queue length (at the coding node). The traditional NC has lower performance in general because when it fails to find coding-possible routes/nodes, then we are seeing the performance of normal store/forward routing.

As discussed in Section 3.3.1 because of the nature of topology we have not seen any performance improvements in big network when using CAR. But this anomalous behaviour is specific to this topology. For some other topologies, we might see performance similar to or better than smaller network in Fig. 2.2 such as those in the next couple chapters for big networks with regular or random Ad Hoc networks.

Chapter 4
Mobile Ad Hoc Network with Regular Trajectories

We have demonstrated in the previous chapter that CAR (Coding-Aware Routing) can identify any coding-possible nodes in the network in order to take advantage of the coding benefits. Traditional NC has lower performance in general because when it fails to identify and utilize these coding nodes and routes, and therefore can achieve the performance of normal store-and-forward routing. We would like to find out if CAR can achieve better performance in Ad Hoc networks. Due to the complexity, we propose to take an intermediate step of studying mobile Ad Hoc network with regular trajectories (as shown later) as opposed to totally random trajectories. As mentioned earlier, we use regular Ad Hoc network terminology to represent Ad Hoc networks with regular trajectories and random Ad Hoc network for Ad Hoc network with random trajectories.

Like previous chapter, we shall first introduce/establish some common parameters and general understanding of the network environments required for in-depth performance evaluation. Then we shall study both the small and big networks to see the effect of scalability. Again like last chapter, this is done through different performance measures, i.e. throughput, ETE delay, PDR and mean queue size.

4.1 Performance Evaluation using Simulation

The networks we shall study are created within an area of 2.5 km by 2.5 km. Each mobile node follows an octagonal trajectory. Each octagon has a corner-to-corner diagonal of 600 m, and is randomly laid down by OPNET before simulation starts so that the distances w.r.t. its octagon neighbors remain fixed. Nodes are moving constantly in a clock-wise manner at a default speed of 5 m/s unless otherwise specified. Like Chapter 3, we apply our coding-aware routing algorithm on top of the AODV as discussed in Section 2.3. Due to the size and proximity of the octagons, the path of each flow in our Ad Hoc networks (Figs. 4.1 and 4.8) can be considered fixed for its lifetime once established because the relative distances among the octagons never change.

Our link/server rate is 1 Mbps so that a node can transmit at 128 packets/s using a packet

size of 1024 bytes[4]. Like the previous chapter, we also study the effect of 3 different packet sizes of 256 buts, 512 bytes and 1024 byte which would give different effective link service rate in packets per second. Buffer is set to a very large size of 1024MB to emulate the situation of an infinite buffer. We need antenna to support wireless communication. The transmitting power of an antenna is Pt=0.0020W by default unless otherwise stated, while its reception sensitivity (packet-reception threshold) is set at *Pr*=-95 dBm. The *ber* (bit error rate) of a link depends on the SNR as discussed in Section 2.5 and the default Gaussian noise model is used.

We shall run the same set of performance measures as Chapter 3. Being a mobile network here, it would be interesting to see some performance as a function of node speed that can affect SNR. As done in Ch. 3, the implementation of our algorithm has been done in-between MAC and Transport layer and all nodes conforms to the notion of a general node in order to take on any one of the four types of node function it may be designated to.

As discussed in Section 2.4, OPNET allows us to modify its built-in Ad Hoc model to adapt to the CAR. Along with the network, node, process, link and packet models discussed Section 2.4, we can evaluate the performance of networks of different sizes. For example, the parameters (such as the default Gaussian noise model) for SNR and BER models can be modified through the dra_SNR and dra_BER functions. The system configuration used to run simulations is same as Chapter 3, i.e. using an Intel Core2 T6600 processor at 2.20 GHz with 4 GB of memory. Since the networks in this chapter have mobile wireless nodes, it takes quite a long time for a simulation to reach steady-state. On an average, each simulation takes about 10 mins to complete and about 600,000 packet would be collected from across all the sinks.

4.2 Small Ad Hoc Network with Regular Trajectories

Figure 4.1 below shows a small regular MANET under our consideration. It consists of *N*= 20 mobile nodes and four source-destination pairs chosen arbitrarily (but defined before simulation starts). Nodes (1, 3, 11 and, 16) and (2, 7, 12 and 17) are the sources and the corresponding destinations respectively such that node 1 is a source for node 2; node 3 is a source for 7 and so on. There is maximum of 4 flows in this network, so we need *m* = 4 buffers to exploit coding in

[4] The 1Mbps is a generic rate. Actually for 1024 bytes per packet, and 128 packets/sec, we have 1024*128*8/sec = 1.048576 Mbps

this network. Packet arrival rates at all four sources are same as assumed in Section 2.7.

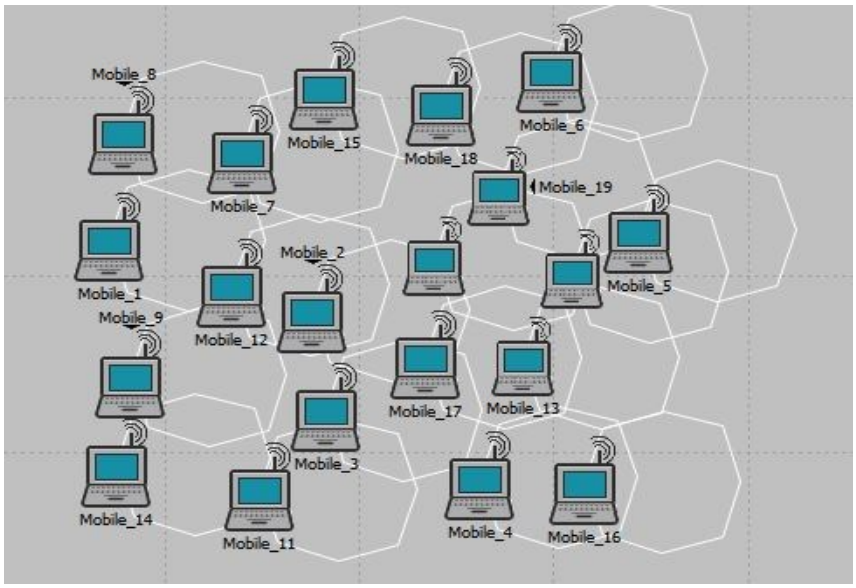

Figure 4.1: AODV-Based Small Ad Hoc Network with Regular Trajectories

Node 1, 3, 11 and 16 starts sending their packets at *t* = 10, 20, 30 and 40s respectively. At t = 10s, since packets at node 1 is the only flow in the network, it will pick the path 1-12-2 because the ETX/hop-count for this path has the lowest value of 2 but at this point, there would be no coding benefits. After the arrival of flow 2 at t = 20 s, the source node 3 (by following algorithm steps given in Section 2.3) determine that node 12 can be the coding-node for flows 1 and 2, if flow 2 takes the path 3-12-7. It can gather that Node 7 can overhear the packet transmission between 1 and 12 before they are coded. Likewise, node 2 can overhear packet communication between 3 and 12, and both nodes 2 and 7 can have decoding packets and thus, behave as decoding-nodes for coding node 12. Node 12 would start coding packets from flow 1 and flow 2.

At t = 30s, node 11 would start generating packets for node 12 (and it constitutes flow 3), but there exists no coding-chances for this flow. Thus, it chooses the shortest path (based on hop count) 11-3-12. Similarly, the new flow 4 between nodes 16 and 17 at t= 40s would not be able to find any coding nodes, because are none, and the shortest path 6-18-15-7 is chosen.

In summary, there is only one coding node (node 12) which will code flow 1 and flow 2, flow 3 and flow 4 were not able to find any coding nodes in the network and hence, will be sent un-coded. On the other hand, network without coding-aware routing would not be able to capture coding-chances exists at the only coding-node 12 because traditional NC is not able to

see coding chances 1 hop or more away (as discussed in Section 2.3.1.). When no coding-node can be found, network using CAR would just pick the shortest path as network without CAR do.

We shall now compare the performance benefits obtained from coding-aware routing against without coding-aware routing network.

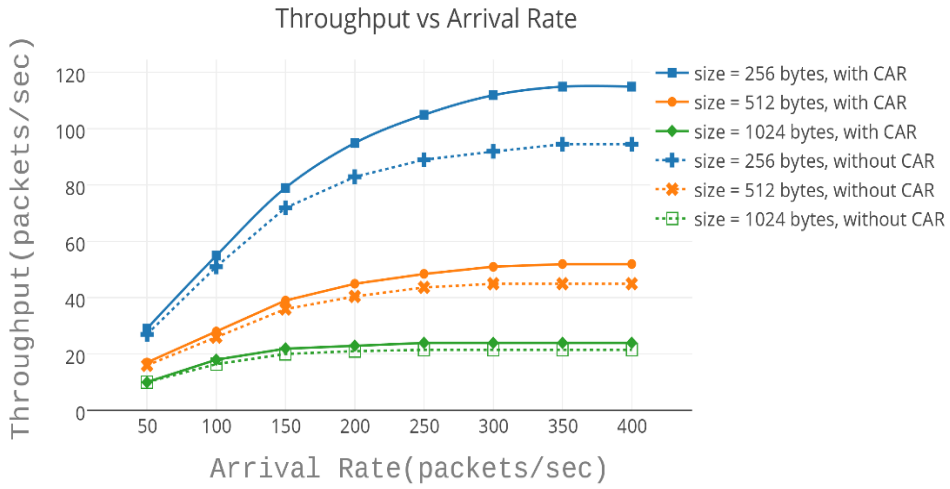

Figure 4.2a: Throughput vs Arrival Rate for a Small Regular Ad Hoc Network

4.2.1 Throughput

Figure 4.2a represents the network throughput as a function of arrival rate when node speed is set to 5 m/s. As defined before, this is the average of the throughput at all the 4 sink nodes.

For a packet size of 256 bytes using CAR, the throughout increases linearly w.r.t arrival rate but only when arrival rate is low. As arrival rate increases, surge in throughput decreases exponentially and then, levels off at 115 packets/s beyond an arrival rate of 350 packets/sec. This can be explained by the fact that more overhead due to MAC layer and AODV communication are required to support higher data rates. Hence the throughput decreases as there is lesser and lesser bandwidth available for data packets. The observations for larger packet sizes are similar except that maximum throughputs (levelling off levels) are progressively decreasing (52 and 24 packets/sec respectively) as the packet size increases from 256 bytes to 512 and 1024 bytes. For the reason provided in Chapter 3, the maximum packet/s throughput has to decrease when packet size increases for a fixed link rate in bps.

The behaviour of the throughput performance when network without CAR is used is similar for different packet sizes but at lower levels. For example, for a packet size of 256 bytes, we have 115 packets/sec using CAR but only 94.54 packets/sec when no CAR is used. This is because of coding benefit using CAR, per unit time node 12 would be able to send almost twice

as packets as network without CAR. Note that the throughput for network using CAR is not doubled as observed in Section 3.2.1 because there are 4 flows and only one coding node. Also, there are some packets that are not even able to get coded because they cannot find packets from other equivalent streams on their arrival and they have to be routed without coding. The comparisons at other packet sizes are similar which shows clearly the advantages of using CAR.

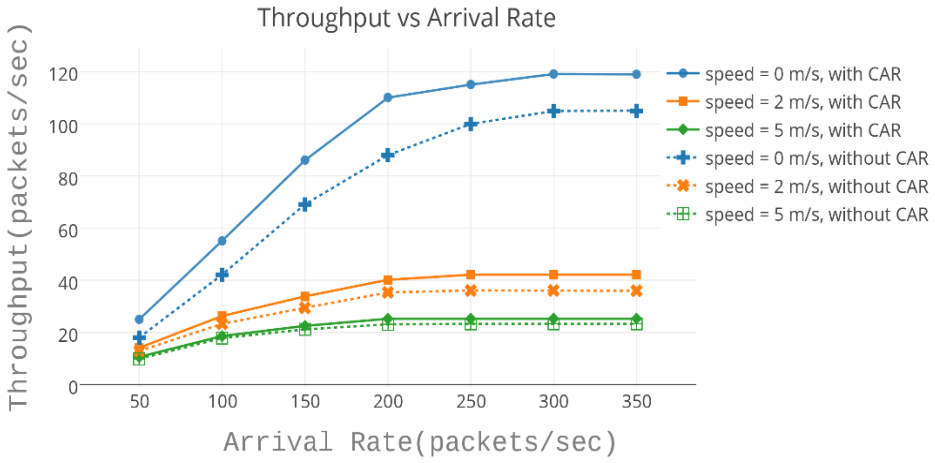

Figure 4.2b: Effect of Node Speed on Throughput for a Small Regular Ad Hoc Network

Figure 4.2b shows the performance of throughput as a function of packet arrival rate. Here, the size of packet is fixed to 1024 bytes and network is simulated for different node speeds (0, 2 and 5 m/s). At speed = 0 m/s for network using CAR, the throughput increases linearly w.r.t arrival rate at low arrival rates and then, slows down exponentially thereafter until system limit is reached at 350 packets/sec. The throughput levels off at 122.2 packets/sec because server utilization has reached at 100 % and more packets cannot be processed. The performance at higher speeds is similar but throughput decreases w.r.t speed because the SNR decreases. For example, throughput levels-off at 24.1 packets/sec when speed = 5 m/s as compared to 122.2 packets/sec at zero speed.

When using network without CAR, observations are same as speed increases except throughput is at a lower level (~15 % less). The throughput obtained in CAR network is higher because of the coding-gains obtained at node 1.

As noted, the SNR for any packet transmission always deteriorates w.r.t. node speed. This is probably due to the additional noise components/power a node collects along its path. Examples are different environmental factors such as signals intercepted from other sources

(considered as noise) and due to multipath interference. Thus, for the same signal strength, the net SNR decreases.

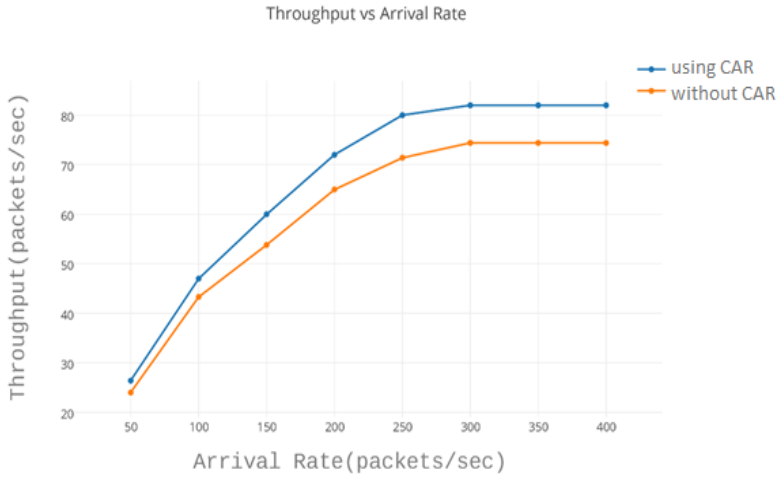

Figure 4.2c: Throughput vs Arrival rate for a Small Regular Ad Hoc Network

Figure 4.2c shows the throughput of flow 1 as a function of packet arrival rate when packet size is set to 512 bytes and node speed at 2m/s. It can be seen that throughput for flow 1 is little higher than the average throughput. This can be attributed by the fact that the source and the destination (nodes 1 and 2 respectively) are only 2 hops apart. For this reason, the SNR for packet transmission is higher and therefore the packet error probability is lower. Hence a higher throughput. Similar observations have been made for other performance measures that flow 1 has better performance than others because of the physical proximity between source and destination nodes.

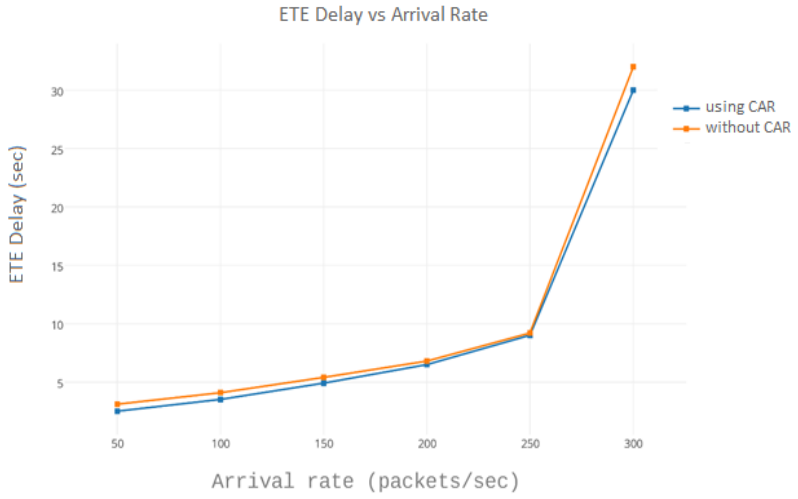

Figure 4.3a: ETE Delay vs Arrival Rate for a Small Regular Ad Hoc Network

4.2.2 ETE Delay vs Arrival Rate

Figure 4.3a compares the ETE delay performance of using CAR with performance of network without CAR (i.e. where traditional NC is used). Packet size and node speed is set to 256 bytes and 5 m/s respectively. For network without CAR, the ETE delay is low and increases approximately linearly before shooting up beyond an arrival rate of 250 packets/second. The system becomes unstable beyond the arrival rate of 350 packets/sec when it is reaching the service limit of the server. Although the server has a capacity of 512 packets/sec, but due to AODV communication in the background and the lower SNR usually sustained in a mobile network, the effective service rate is lower. We can estimate the AODV communication overhead for this network to be about 5.8 % of the server capacity.

Similar observations can be made for network using CAR except that the ETE delay is slightly lower because the CAR algorithm has found and designated node 12 as the coding node which was able to process two packets at a time. Since there are 4 flows in the network but only two flows are able to get coded, the average coding gain for this network was not as much as in case of small wired network where we had two flows and both were able to get coded. Therefore, the mean delay across all the destination nodes was only reduced by a small amount with coding.

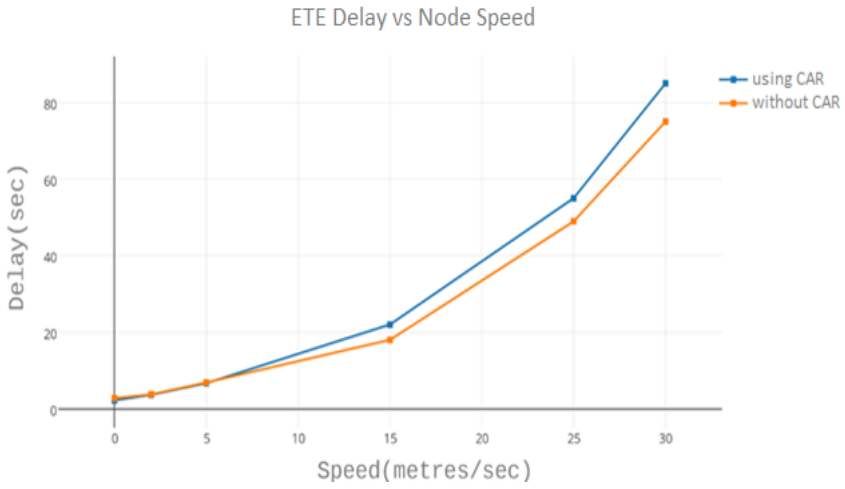

Figure 4.3b: Effect of Node Speed on ETE Delay for a Small Regular Ad Hoc Network

Figure 4.3b shows the behaviour of a network using and without CAR when node is moving at different speed. We have fixed the packet arrival rate and packet size to 100 packets/sec and 1024 bytes respectively. For a network without CAR, the ETE delay increases linearly at low node speed but increases more rapidly as node speed becomes very high. This is because SNR decreases with an increasing speed which increases the *ber* (and hence, the per).

As more packets are dropped, more retransmissions are required. These retransmissions would eventually cause waiting packets in a queue to wait longer. For example, ETE delay is 2.9 s when node speed = 0 m/s as compared to 78 s at node speed = 30m/s.

Similar observations can be made for network using CAR except the ETE delay is lower because of the coding gains obtained at node 12; node-12 was able to send two packets at once. Coding gains can be confirmed from Fig. 4.3b, when node speed = 0 m/s, the ETE delay for CAR network is 2.15 s as compared to 2.9 s for network without CAR. But as the node speed increases, delay for CAR network increases more rapidly than network without CAR which can be confirmed from the figure i.e. ETE delay for network using CAR at speed = 30 m/s is 85.2 s as compared to 78 s for network without CAR.

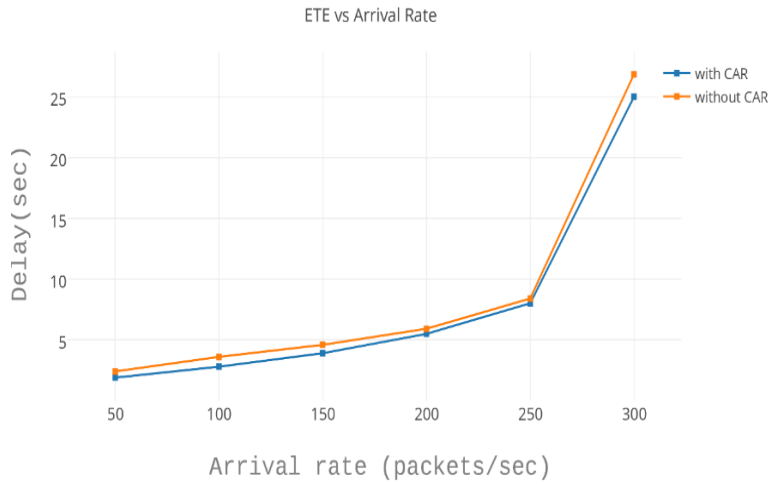

Figure 4.3c: ETE Delay vs Arrival Rate for a Small Regular Ad Hoc Network

Figure 4.3c shows the performance of ETE delay for a single flow (i.e. flow 1 at node 2) when packet size and node speed is fixed to 256 bytes and 5 m/s respectively. It can be seen that ETE delay for both network using CAR and without CAR is lower than that of average ETE delay. This is because node 1 and 2 (i.e. source and destination for flow 1) are close to each other due to which SNR for their packet transmissions is quite high and hence, has better performance than average.

4.2.3 PDR

Figure 4.4a below shows packet delivery ratio performance as a function of packet error rate (*pep*) when packet arrival rate and size is 100 packets/sec and 512 bytes. At node speed = 0 m/s for network using CAR and *pep* = 0%, PDR is at its highest value but still at 0.62. PDR decreases

slowly thereafter, but faster as *pep* increases and reaching a value of 0.13 when *pep*=30%. Same observations can be made at higher node speeds, but the PDR is even lower. For example, at *pep*=30%, PDR is 0.08 and 0.06 at 2m/s and 5 m/s respectively. Lower SNR as explained before can be attributed to the lower PDR as speed increases. The faster decrease in PDR at higher *pep* is due to more un-decodable packets that are forcibly dropped at decoding nodes 2 and 7.

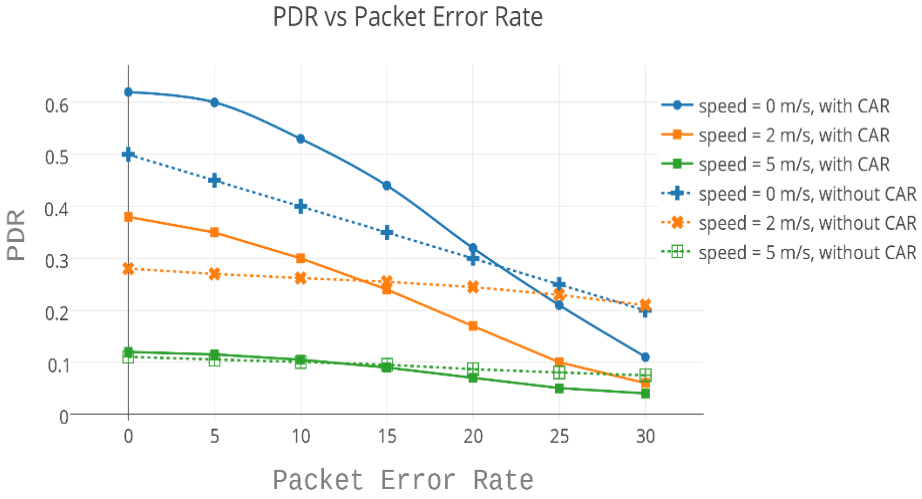

Figure 4.4a: PDR vs Packet Error Rate for a Small Regular Ad Hoc Network

Network without CAR on the other hand has a lower PDR but only when *pep* is low. As the *pep* increases, its PDR performance actually becomes better than network using CAR. This is because CAR network is able to get coding gains at node 12. But since the packet loss rate for CAR is a function of *pep* and *pd*, more packets are likely to be lost as *pep* increases. For example, network with CAR has better PDR than network without CAR due to its coding gain for *pep*<22.4% at node speed=0 m/s, but this has diminished because performance decline at decoding nodes due to dropped un-decodable overcomes coding gains obtained at node 12. For *pep*>22.4 5 network without CAR has a higher PDR because it has no reliance on decoding packet (i.e. *pd*=0).

Similar observations can be made at speed = 2 m/s and 5 m/s. However, it is interesting to see that crossover point (i.e. the point when PDR for network without CAR overtakes that of CAR) happens earlier as node speed increases. For example, *pep* corresponding to crossover point decreases from 22.4 % to 15% to 12.5% as speed increases from 0 to 2 to 5m/s. This can be explained by the fact that coded packets are more vulnerable to be lost with increasing SNR due to increased speed.

To push the performance to the limit, Figure 4.4b below shows the PDR performance at much higher speeds of 15m/s and 30 m/s (which is equivalent to an automobile speed of ~110

km/hr). The packet arrival rate and packet size is still fixed at 100 packets/sec and 512 bytes respectively. The performance trends for networks with and network without CAR are similar to those at low speeds. The PDR performance is much worse for any use. For example, at node speed = 30m/s, PDR = 0.072 at *pep*=0% and decreases to a meagre PDR=0.001 at *pep*=30%. It is interesting, however, to see that the PDR performance using CAR here is always lower than network without CAR. This is because the *pd* is quite high even at 15 m/s, and since network without CAR does not have any reliance on decoding packet i.e. PDR is a decreasing function of *pep* only unlike CAR network, for which PDR is effected by *pep* and PDR. There is no cross-over point as observed in Fig. 4.4a because PDR for CAR network is already lower than network without CAR even at *pep* = 0%. The SNR is too high here and the PDR performance continues to decrease with increasing *pep* as observed before.

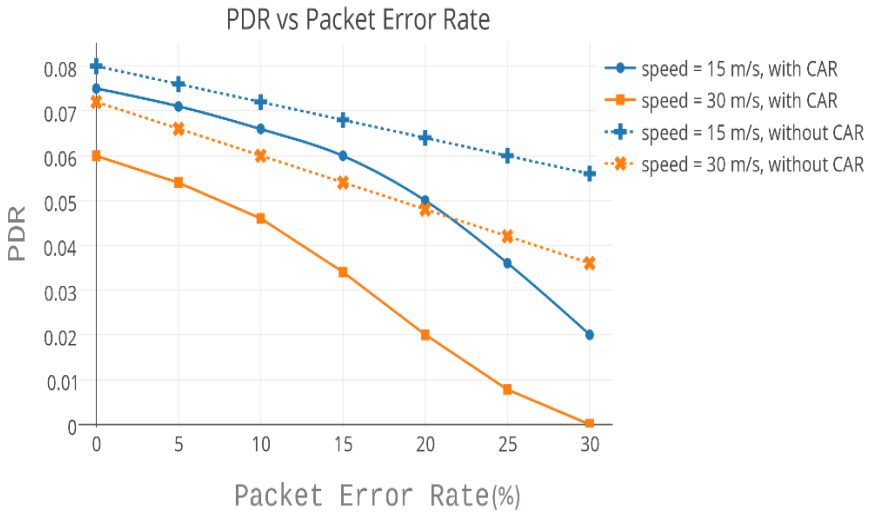

Figure 4.4b: PDR vs Packet Error Rate for a Small Regular Ad Hoc Network

4.2.4 Mean Queue Size at the Coding Nodes

Figure 4.5 below compares the behavior of mean queue size at the only coding node 12 as a function of packet arrival rate. The node speed and packet size are set to 5 m/s and 512 bytes respectively. For a speed of 30 m/s in a network without CAR, the queue size increases more or less linearly but only up to 150 packets/sec. Then it grows exponentially when reaching the service limit of the server (estimated to be around 250 packets/sec). Similar observation can be made for lower node speeds which result in lower mean queue size. For example, at 50 packets/sec, the mean queue size decreases from 50 to 28 and to 26 packets as speed decreases from 30 to 15 and to 5 m/s. As discussed earlier, higher SNR can be obtained at lower node speed which result in lower packet errors and therefore less retransmissions to cause queue build up.

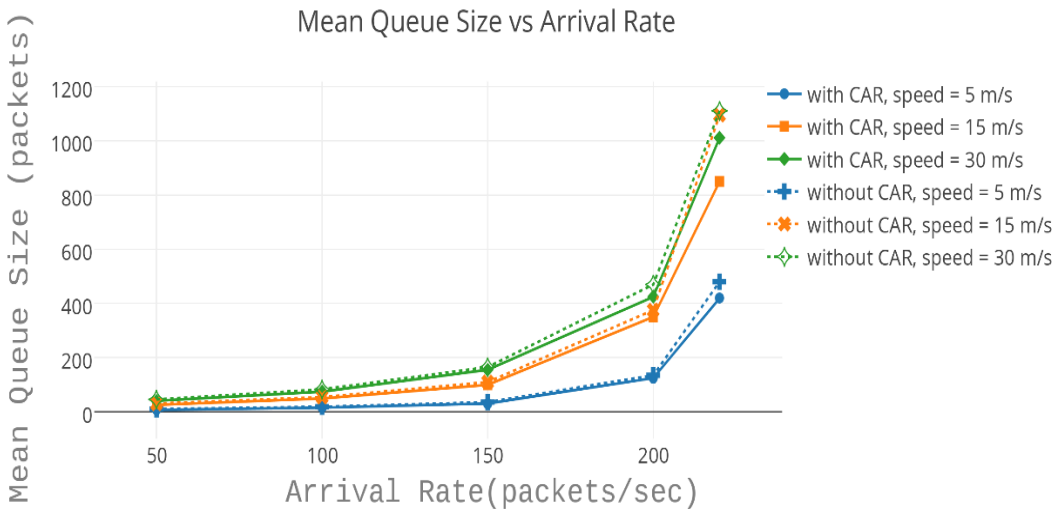

Figure 4.5: Mean Queue Size vs Arrival Rate for a Small Regular Ad Hoc Network

Same observation can be made for network using CAR but due to the fact that CAR algorithm has found and designated node 12 as the coding node which was able to process two packets per unit time, the queue size is always lower than that of network without CAR. For example, at packet arrival rate = 200 packets/sec, the mean queue size for CAR network is 430 packets as compared to 470 packets when network without CAR has been used.

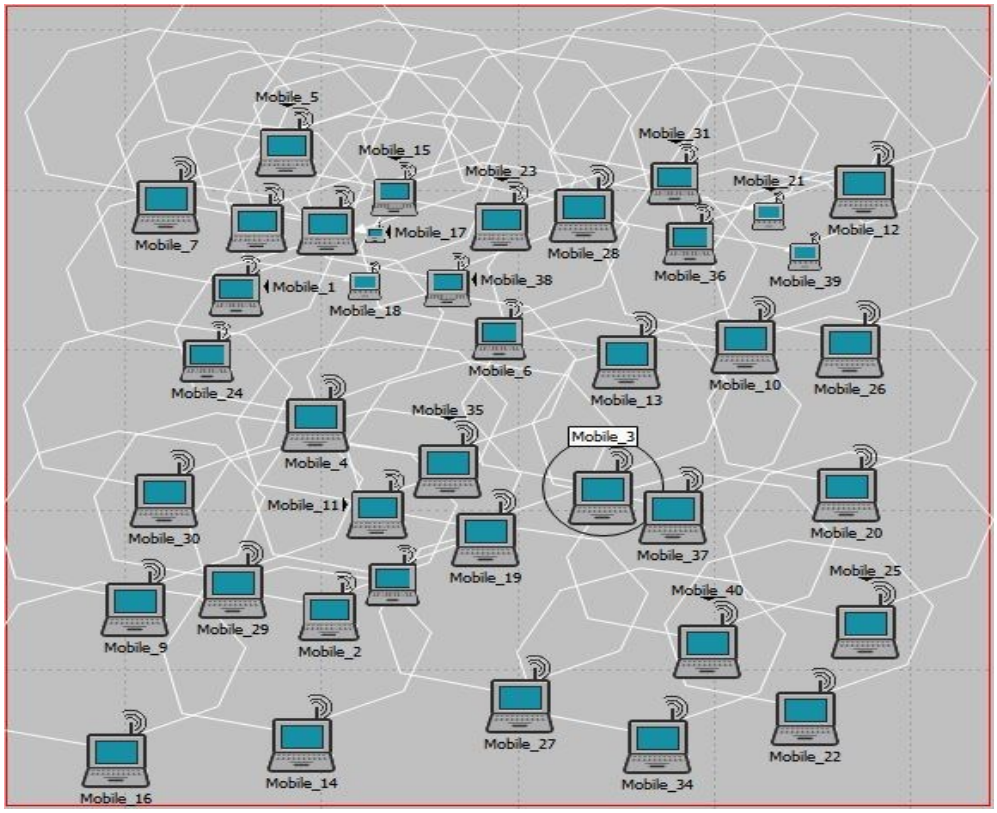

Figure 4.6: AODV based Big Ad Hoc Network with Regular Trajectories

4.3 Big Ad Hoc Network with Regular Trajectories

The performance evaluation of a small Ad Hoc network in the previous section shows that using CAR on MANETs can have better performance in general except at high speeds. It would be interesting to see if how the performance would scale in a big network.

Figure 4.6 shows consists big network consisting of *N* = 40 nodes with five sources and destination pairs. Node 1, 7, 15, 22 and 6 are the sources and their corresponding destinations are nodes 11, 17, 25, 32 and 30 respectively. To support the 5 flows in the network, a coding node would need, m = 5 buffers to exploit coding benefits. The same default parameters in Section 3.1 are used here.

At t = 10s, node 1 would start Flow 1 by sending packets for destination node 11. Since this is the only flow at that time, it will pick the shortest path of 1-24-4-11 with ETX/hop-count of 3. Even after the arrival of flow 2 and flow 3 at nodes 7 and nodes 15 at t = 20s and t = 30s respectively, there exists no coding chances in the network because there does not exist any decoding or innovative nodes in the downstream. Therefore, the path 7-32-33-17 is chosen for flow 2 and 15-23-6-3-37-25 for flow 3 based on the lowest value of ETX/hop-count of 3 and 5 respectively. At t = 40s, node 22 starts flow 4 for destination node 32. During the route discovery phase, the source node 22 finds out that node 33 can be the coding-node if flow 4 takes the path 22-40-37-13-6-18-33-32 because node 17 can overhear the original packets (i.e. before it gets coded) of flow 4 from node 18, and thus can serve as decoding node for flow 2. Likewise, node 18 can overhear flow 2 transmissions from node 32 and thus can serve as decoding node for flow 4. There also exist coding chances for flow 5 after its arrival at t = 50s at node 6. Following steps in the Source Actions in Section 2.3, node 6 after receiving an RREP knows that by picking the path 6-35-4-30, node 4 can be a coding-node for flows 1 and 5. Since node 11 can overhear packet transmissions for flow 5 from node 35, it can serve as the decoding node for flow 1. Likewise, node 30 can overhear packet transmissions for flow 1 from node 24 and thus can be the decoding node for flow 5. In summary, we have 2 coding-nodes (nodes 4 and 33), 4 decoding nodes (nodes 11, 17, 18, 30) and no innovative nodes after running the generalized CAR to network in Fig. 4.6.

On the other hand, network without CAR would not be able to exploit any of these coding-chances because there is no way for source node to analyze the existence of decoding nodes which are more than 1-hop away. Fortunately, routes chosen by network using CAR happen to be the shortest paths too from the RREP packet received earlier. Therefore, routes

chosen by network without CAR (based on hop-count) are the same as that of network using CAR but with no coding gains. Note that this is specific for this network topology only; other topologies usually have different routes. The benefits arising from this coding-decoding operation will be illustrated in the performance evaluations below.

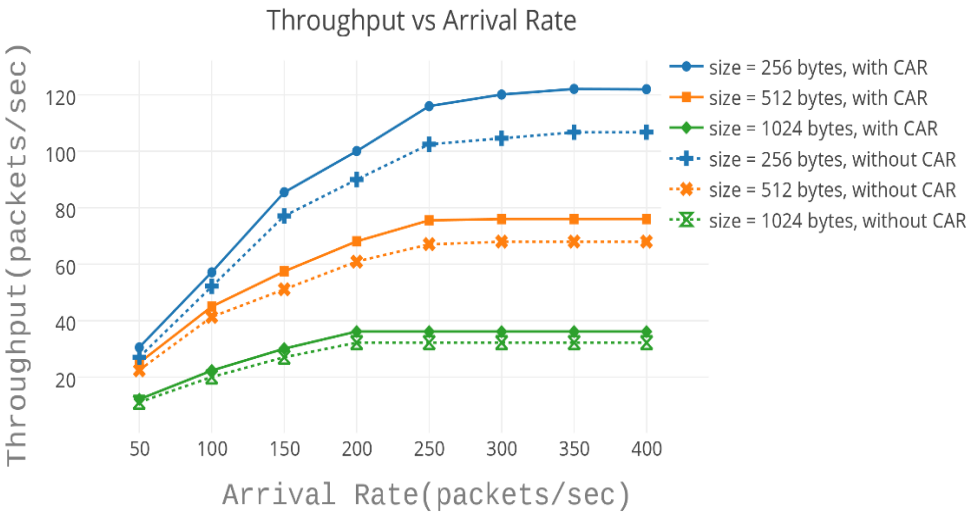

Figure 4.7a: Throughput vs Arrival Rate for a Big Regular Ad Hoc Network

4.3.1 Throughput

Figure 4.7a shows the behaviour of network throughput as a function of packet arrival rate for different packet sizes when node speed = 5 m/s. For a packet size of 256 bytes using CAR, the throughput continues to rise up w.r.t arrival rate and levels off at 122 packets/sec when the arrival rate increases beyond 350 packets/sec. This is because the system limit has been reached and node is not able to transmit all the arriving packets. The observations for larger packet sizes (512 bytes and 1024 bytes) are similar except they level off more quickly (beyond arrival rates of 250 packets/sec and 200 packets/sec respectively) and at lower throughput levels of 77 packets/sec and 38 packets/sec respectively.

The throughput of networks without CAR behaves similar to network using CAR where larger packet size has lower levelling throughput. On comparison, using network without CAR provides lower throughput for all the packet sizes. This is because a network without CAR was not able obtain coding gains from coding nodes 4 and 33. For example, for a packet size of 256 bytes, we have maximum throughput of only 106.89 packets/sec compared with 122 packets/sec using CAR

The higher throughput in big network using CAR may come from more path choices and

therefore, better connectivity than small networks because there are more nodes within the same area. There is also a higher probability of finding decoding nodes and innovative nodes. Therefore, less coded packets would be dropped due to non-decodeability of the nodes (i.e. when the decoding nodes are not able to decode coded packets).

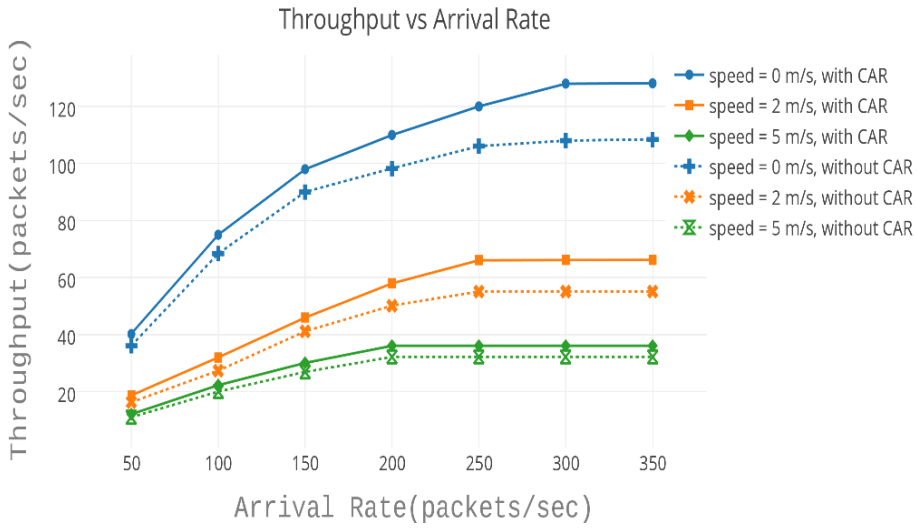

Figure 4.7b: Effect of Node Speed on Throughput for a Big Regular Ad Hoc Network

We have also investigated the effect of speeds when packet size = 1024 bytes. Figure 4.7b shows the throughput performance at node speeds of 0, 2 and 5 m/s. For network without CAR at 5m/s, the throughput increases almost linearly w.r.t the arrival rate before leveling off at ~ 36.2 packets/s when arrival rate goes beyond 200 packets/s. The max throughput increases to 56 packet/s and then to 108 packets/s with node speed decreasing to 2 m/s and then to 0 m/s. This is because with the decrease in node speed, the SNR increases which results into higher throughput.

In comparison, the throughput performance for CAR network is similar except higher because of the coding gains provided by nodes 33 and 4. For example, at node speed = 2 m/s and packet arrival rate = 150 m/s, throughput obtained for CAR network is 45.5 packets/sec as compared to 41.2 packets/sec for network without CAR.

Figure 4.7c below shows the throughput of flow 2 between 7 and 17 as a function of packet arrival rate. Packet size is fixed to 512 bytes and node speed = 5 m/s. When compared to the average throughput (of all the flows), it can be seen that flow 2 has a higher throughput because the source and destination nodes are just 2 hops away, which in turn gives higher SNR and lower *pep* for packet transmissions.

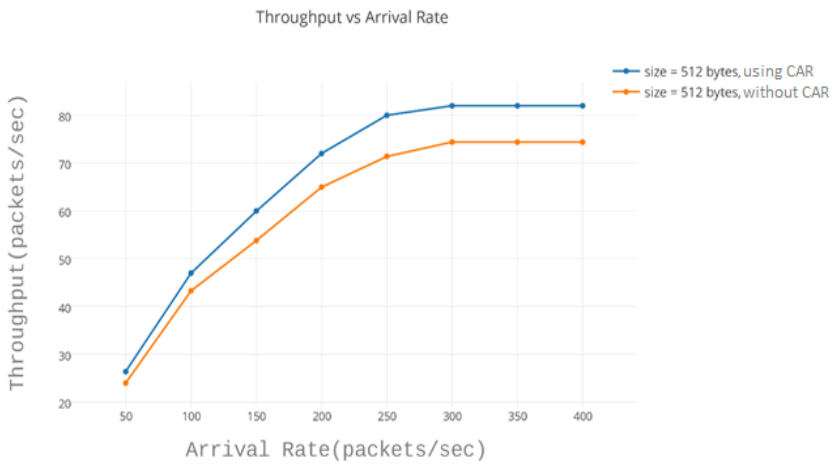

Figure 4.7c: Throughput vs Arrival Rate for a Big Regular Ad Hoc Network

Similar to throughput, we have observed better ETE delay performance for flow 2 as compared to the average performance to be given in Fig. 4.8c later.

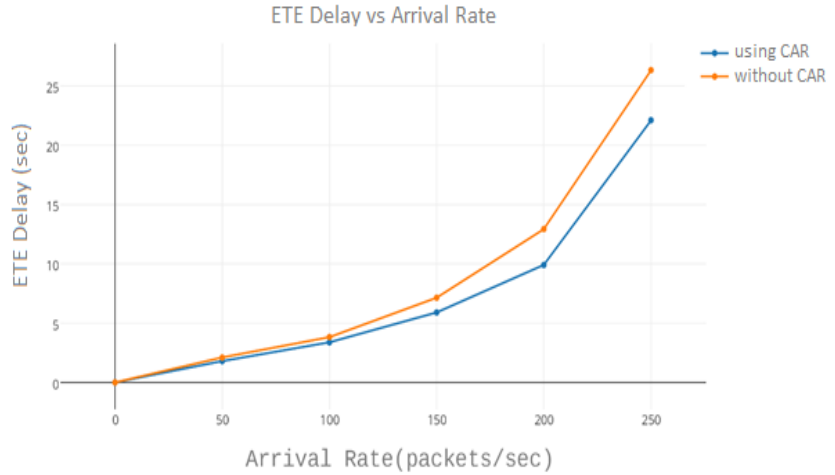

Figure 4.8a: ETE Delay vs Arrival Rate for a Big Regular Ad Hoc Network

4.3.2 ETE Delay vs Arrival Rate

Figure 4.8a shows the behaviour of average ETE delay curve comparison between network using CAR and network without CAR. Packet size and node speed are fixed at 1024 bytes and 5 m/s respectively and the curve response is graphed as a function of packet arrival rate. It can be seen from Fig. 4.8a that for network without CAR, ETE delay is quite low (i.e. 2.45 seconds) when packet arrival rate is 50 packets/sec and then, shoots up beyond arrival rate of 200 packets/sec to 13s. The system becomes unstable beyond packet arrival rate of 250 packets/sec because the service limit of the server has been reached.

Same observations can be made for network using CAR but since nodes 4 and 32 has some coding gain, the ETE delay is relatively smaller for all packet arrival rates. For example, at

packet arrival rate = 150 packets/se, ETE delay for CAR network is 5.82 s as compared to 7.1 s for network without CAR.

As compared to ETE delay performance of small network in Fig. 4.3a, the big network has a lower delay due to more nodes within the same area. This has resulted in a smaller average distance among the nodes which in turn gives a higher SNR and lower *pep* and thus lower ETE delay.

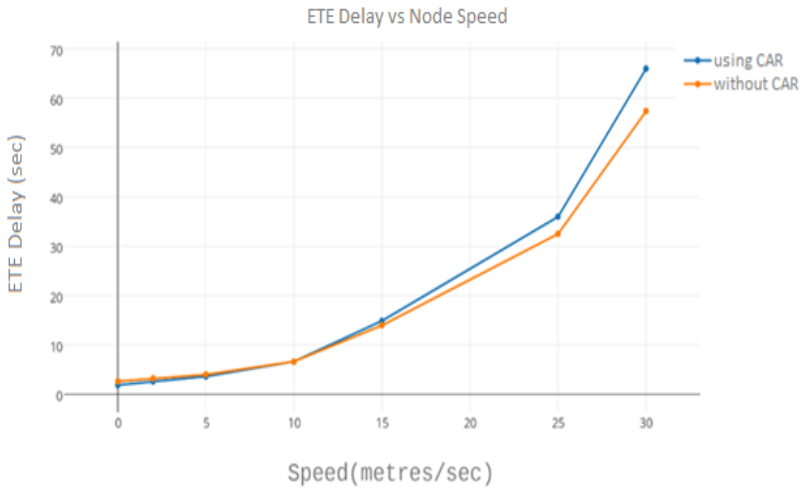

Figure 4.8b: Effect of node speed on ETE Delay for a Big Regular Ad Hoc Network

We have also investigated ETE delay performance as a function of node speed while keeping packet size and arrival rate at 1024 bytes and 100 packets/s respectively. Figure 4.8b compares network using CAR and network without CAR for different node speeds. For a network without CAR, ETE delay increases linearly up to node speed of 10 m/s, and then more rapidly after. The higher delay at higher node speed is due to decreased SNR which results in higher packet error rate and therefore more dropping of the packets before reaching destination. These dropped packets would need retransmissions which would eventually cause the waiting packets in the queue to wait longer.

CAR networks have very similar performance for the same reason. However, delay is slightly lower at low speed because queue size is relatively smaller due to higher SNR and less packet loss that requires less re-transmissions. Unlike the absence of decoding nodes in network without CAR, the reliance of decoding nodes in network using CAR results in more dropping of non-decoded packets (i.e. *pd* increases with pep as confirmed in Section 4.3.3) and increased ETE delay. Hence the ETE delay becomes higher as node speed increases. This can be confirmed by comparing performance curve when speed = 0 m/s and 30 m/s. At 0 m/s, CAR network has lower

delay (at 1.85 s) as compared to 2.6 s of without CAR network. The delay becomes higher beyond 10m/s, e.g., at 30 m/s, CAR network has higher delay (at 66 s) as compared to 57.4s of without CAR network.

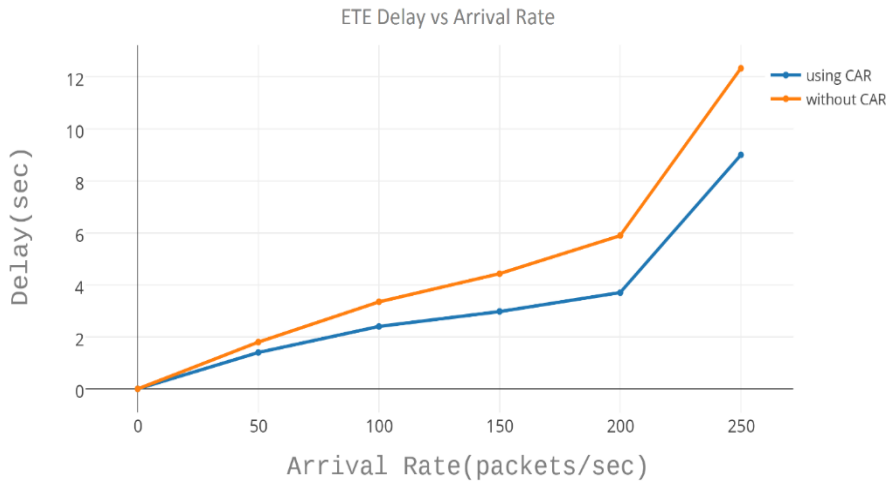

Figure 4.8c: ETE Delay vs Arrival Rate for a Big Regular Ad Hoc Network

Figure 4.8c shows the ETE delay performance of a single flow (i.e. flow 2) between node 7 and 17. As discussed in Fig. 4.7c that nodes 7 and 17 are physically closer to each other because of which they have higher SNR and lower *pep* for their packet transmissions, the ETE delay for flow is lower than that of average.

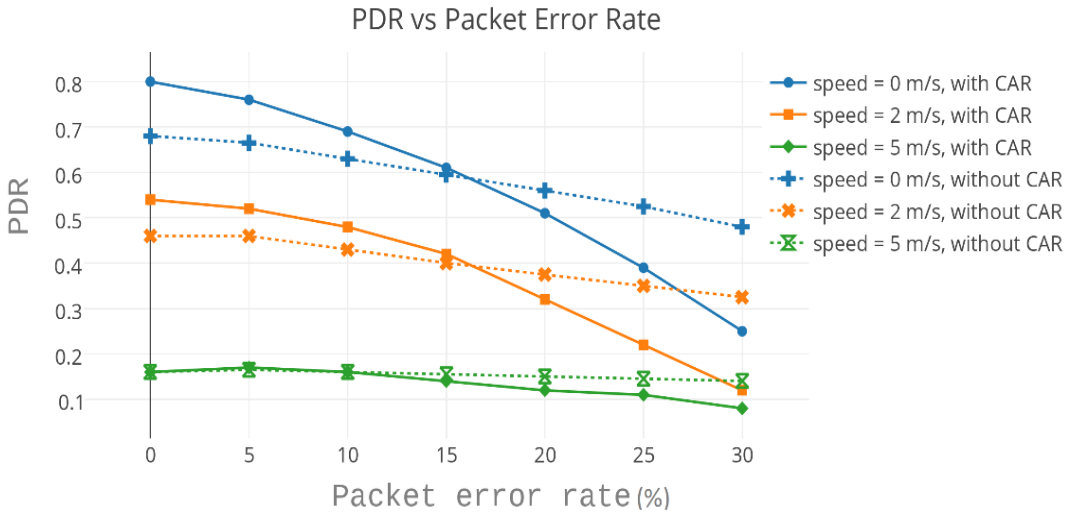

Figure 4.9a: PDR vs Packet Error Rate for a Big Regular Ad Hoc Network

4.3.3 PDR vs Packet Error Rate

Figure 4.9a shows the behaviour of PDR as a function of packet error rate for different node speeds. The arrival rate and packet size is set to 100 packets/sec and 512 bytes respectively. For CAR network at 0 m/s, PDR is quite high (i.e. 0.82) when packet error rate is 0%, but decreases

significantly w.r.t increasing packet error rate and reaching 0.28 at pep = 30%. This can be explained by the fact that packets need to be decoded (at nodes 17 and 18) before delivering out to the destination nodes (i.e. node 17 and node 25) and since it is less probable to find decoding packets at decoding nodes with increasing pep, the un-decoded packets are dropped before reaching destination. Note that *pd* at *pep* = 0% is measured to be 0.18 as compared to 0.42 when pep is increased to 30%. Almost similar performance is observed at higher node speeds except PDR is even lower because SNR decreases and hence PDR w.r.t. speed. This can be confirmed from Fig. 4.9a that PDR decreases from 0.28 to 0.12 to 0.06 when node speed increases from 0 m/s to 2 m/s to 5 m/s respectively at pep = 30%. Similar observations can be made for nodes moving at different speeds, but due to lower SNR at high speeds, PDR decreases further. For example: at pep = 15%, PDR decreases from 0.62 to 0.42 to 0.14 with increasing node speed corresponding to 0, 2, and 5 m/s respectively. Compared with the network without CAR, the rapid drop in PDR performance (especially in 0 and 2 m/s) are due to the larger *pd* at high pep to be discussed later. In comparison, network without CAR have a lower PDR of 0.7 when node speed = 0 m/s and *pep* = 0%. This is because network without CAR was not able to identify and designate nodes 4 and 33 as the coding nodes. Similar to our discussion in Section 4.2.3, as speed increases PDR for network without CAR decreases in a more linear manner as compared to CAR because packet loss rate is a function of *pep* only. As discussed in Section 4.2.3, the *pep* at which the crossover happens also decreases with increasing node speed i.e. the crossover point decreases from 17 % to 15.7% to 10.4% as the node speed increases from 0 to 2 to 5 m/s because of decreasing SNR and increasing *pep*.

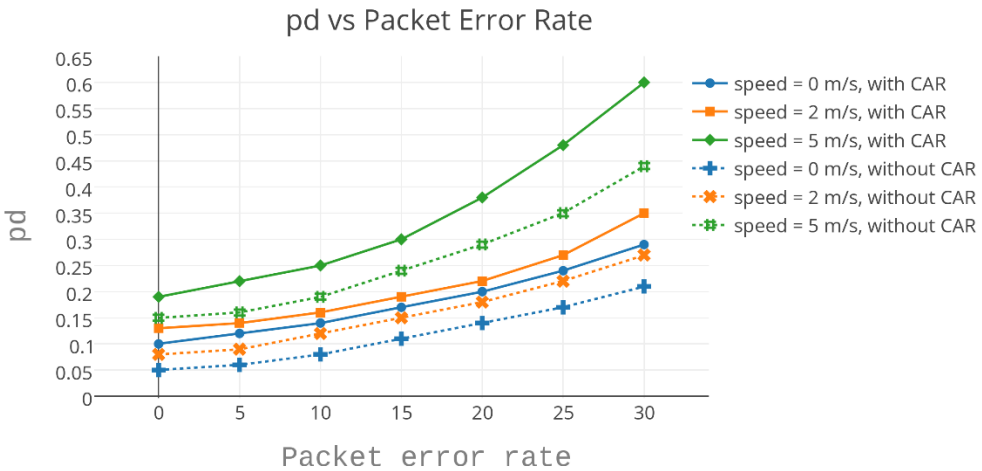

Figure 4.9b: pd vs Packet Error Rate for a Big Regular Ad Hoc Network

Like *PDR*, *pd* is also affected from *pep* i.e. it increases with *pep* as observed confirmed

in Fig. 4.9b. For network using CAR and node speed = 5 m/s, *pd* increases more or less linearly up to *pep* = 10%, and beyond that it grows more rapidly and reaches 0.6 when pep = 30%. This is because at high packet error rate there are more chances that decoding packets would not be able to reach decoding nodes. Similar observations can be made at lower node speeds. Since SNR is relatively higher at lower moving nodes, *pd* grows more linearly. When *pep* = 30%, the maximum value of *pd* decreases from 0.6 to 0.35 to 0.2 when node speed decreases from 5 m/s to 2 m/s to 0 m/s.

Like networks using CAR, observations for network without CAR are very similar except pd is lower all the time. This is because networks without CAR have less dependency on decoding. For example, the (minimum, maximum) values of *pd* for network using CAR are (0.2, 0.6), (0.15, 0.35) and (0.1, 0.2) for node speeds = 5 m/s, 2 m/s and 0 m/s respectively, while the value pairs are (0.15, 0.45), (0.1, 0.25) and (0.05, 0.15) for networks without CAR. In general, it is observed that *pd* starts to escalate more after certain threshold of *pep* which can be estimated to 15% for this network.

Similar to our observations for throughput and ETE delay, the PDR for a big network is better as compared to a small Ad Hoc network due to better connectivity and hence higher SNR. For example: For *pep* = 0 % and node speed = 0 m/s, the PDR for big Ad Hoc network using CAR has PDR of 0.804 as compared to 0.616 for small Ad Hoc network using CAR.

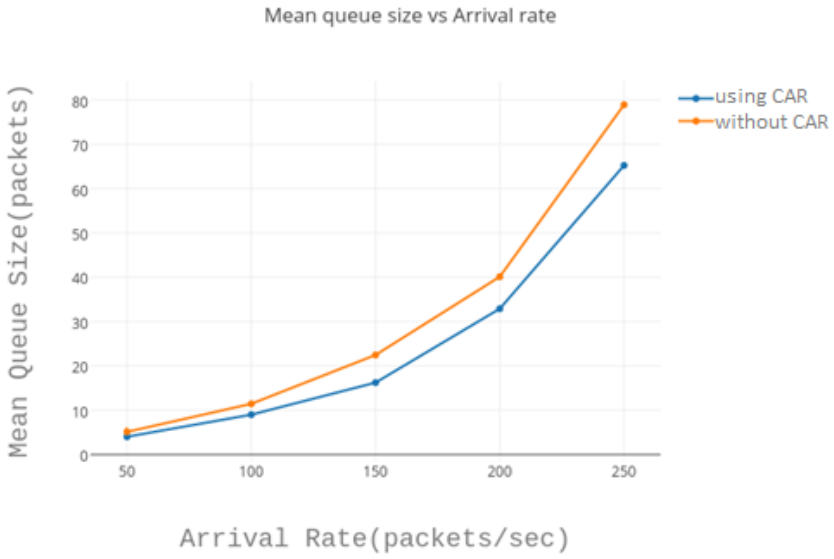

Figure 4.10: Mean queue size vs Arrival Rate for a Big Regular Ad Hoc Network

4.3.4 Mean Queue Size at Coding Nodes

Figure 4.10 shows mean queue size curve comparison between mean queue size and arrival rate

when the packet size is set to 256 bytes and node speed to 5 m/s. For networks without CAR, queue size is low (i.e. 5 packets) at 50 packets/sec but then shoots up to 80 packets at arrival rate of 250 packets/sec. Curve grows to infinity beyond 250 packets/sec because system limit (estimated to be around 480 packets/sec) has been reached and nodes would not be able to process incoming packets right away.

Similar observations can be made for network using CAR but which has a lower mean queue size. This is because a network using CAR is able to exploit coding chances at nodes 4 and 33 so that 2 flows can be processed together at a time.

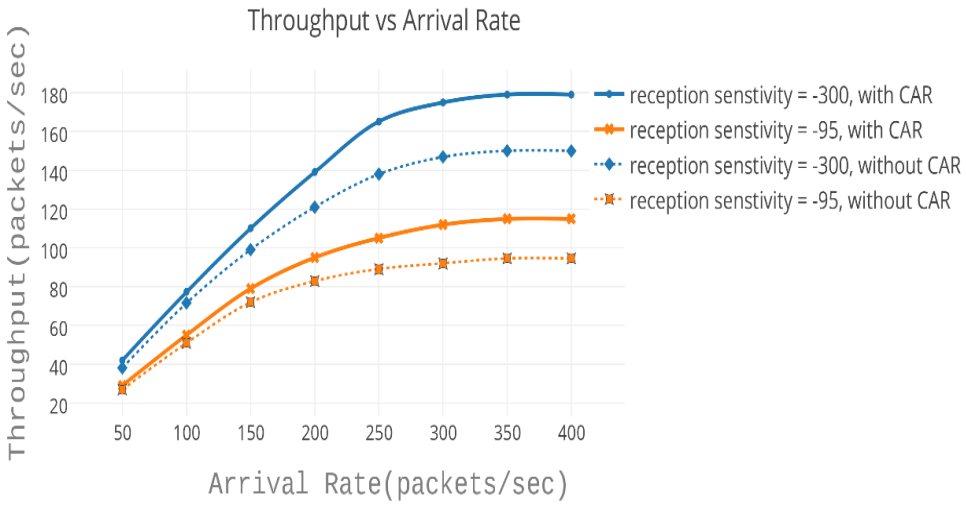

Figure 4.11: Effect of Receiver Sensitivity on Throughput for a Small Regular Ad Hoc Network

4.4 Effect of Receiver Sensitivity

It is also observed that network performance (in terms of throughput, ETE delay, PDR and mean queue size) can also be effected by reception sensitivity. Fig. 4.11 shows its effect on throughput for both network using CAR and network without using CAR when packet size and node speed is set to 256 bytes and 5 m/s respectively. For reception sensitivity of -300 dBm and network using CAR, throughput starts to increase more or less linearly with arrival rate until it levels off at 178 packets/sec. Almost similar observations can be seen at reception sensitivity = -95 dBm but throughput here levels off at relatively lower value i.e. 149 packets/sec. This is because in former case, nodes can sense packets with relatively lower power threshold with the help of which more number of packets can be successfully transmitted from source to destination.

Similar to Fig. 4.2a, network without using CAR has shown similar but lower performance than their networks using CAR counterparts. For example, for reception sensitivity of -300 dBm,

network without CAR levels off at 150 packets/sec as compared to 179 packets/sec for network using CAR.

Similar to throughput for this network, we found that performance for all Ad Hoc networks under our investigation improves when reception sensitivity is decreased to -300 dBm from -95 dBm (default).

4.5 Concluding Remarks

We have used simulations to evaluate the performance of both small and big Ad Hoc networks with nodes moving in regular trajectories. Parameters such as packet size, node speed, trajectory pattern, number of nodes, etc. can have significant effect on all network performances. In general, big networks have better performance than small networks in all performance measures because there are more nodes with in the same area, thus improving the connectivity and SNR. Networks using CAR have shown better performance than networks without CAR because they can find more coding-possible paths especially in a big network to exploit existing coding chances. However, at higher *pep*, network using CAR performs worse than network without CAR because CAR was able to identify more coding nodes and hence, more coded packets in the network. Since it is less probable to find decoding packets as *pep* increases, PDR performance for network using CAR decreases significantly. For the situation explained in Section 2.3.2, we have observed scenarios of defective coding for networks in this chapter.

Because of mobility, SNR of the network decreases considerably whose effect is evident from the much degraded performance (compared to wired/static networks) in throughput, ETE delay, PDR and mean queue size in networks using CAR or without CAR. For network using CAR, PDR performance decreases further because of its dependence on decoding packet which becomes less probable to find as SNR decreases. Also, unlike big wired network in Chapter 3 regular Ad Hoc network yields better performance for CAR network than network without CAR because here we find no bottleneck links.

It is also observed that SNR and hence, network performance for both regular and random Ad Hoc network also gets effected by reception sensitivity. From Fig. 4.2d, it can be confirmed that throughput is improved when reception sensitivity is increased (the more negative, the more sensitive) because packets even with relatively lower power threshold can be sensed by next-hop nodes.

Chapter 5
Mobile Ad Hoc Network with Random Trajectories

Knowing that CAR can provide better performance in a MANET with regular trajectories, we now investigate further the performance of a MANET with totally random trajectories (or, random Ad Hoc networks). After introducing the network parameters, we shall again study a small network first to validate CAR benefits before checking on how the big network scales in its performance. As done in previous chapters, the evaluation is performed on different performance measures of throughput, ETE delay, PDR and mean queue size for nodes moving at different speeds.

5.1 Performance Evaluation using Simulation

Like Chapter 4, both small and big networks are created in an area of 2.5 km by 2.5 km, where each node is moving randomly with a default speed of 5 m/s unless otherwise specified/investigated. For comparison with previous chapters, we shall again use N = 20 nodes for the small network and N = 40 nodes for the big network. Most parameters in the network or physical layers are the same as those in Chapter 4. For example, the node server/link rate is set to 1 Mbps, so a node can transmit at 128 packets/sec for packet size of 1024 bytes. Like Chapters 3 and 4, we shall study the effect of 3 different packet sizes of 256 bytes, 512 bytes and 1024 bytes. We use the default value of transmitter power Pt=0.0020W unless otherwise stated and receiver reception sensitivity Pr = -95 dBm.

Since nodes are moving randomly, the relative distance of the nodes to each other changes constantly as simulation progresses and the route-discovery needs to be initiated again once any node along the path goes out of transmission range. Thus, there is no fixed route between a source and a destination i.e. all intermediate, coding, decoding and innovative nodes are changing dynamically. Again, our CAR is overlaid on AODV as our routing algorithm. When a connection is lost, AODV ignore packets in the intermediate nodes along the path. These packets cannot go anywhere and are eventually dropped. In that case, source has to reinitiate the delivery of the lost packets after forming new route. Since nodes are moving in a random fashion, the relative proximity between any two nodes is changing all the time and from Section 2.5, the

nodes performing the particular roles based on their designations needs to be changed as well. For example: on an average in any typical simulation, the number of times the coding nodes appear and disappear in random Ad Hoc networks is 20. For each such instance, a source node would have to wait x = 3s as discussed in Section 2.2 and this can have significant effect on end-to-end performance as we will see in Sections 5.2 and 5.3.

The same system configuration as Chapters 3 and 4 is used to run our simulations, i.e. using Intel Core2 T6600 processor at 2.20 GHz with 4 GB of memory. It takes quite a long time for a simulation to reach steady-state for mobile wireless networks. On the average, a simulation takes about 15 mins to complete and there are ~1, 000, 000 packets. The performance measures similar to those of Chapter 3 and 4 (i.e. throughput, ETE delay, PDR and mean queue size) are used to evaluate CAR benefits for this chapter which is discussed as below:

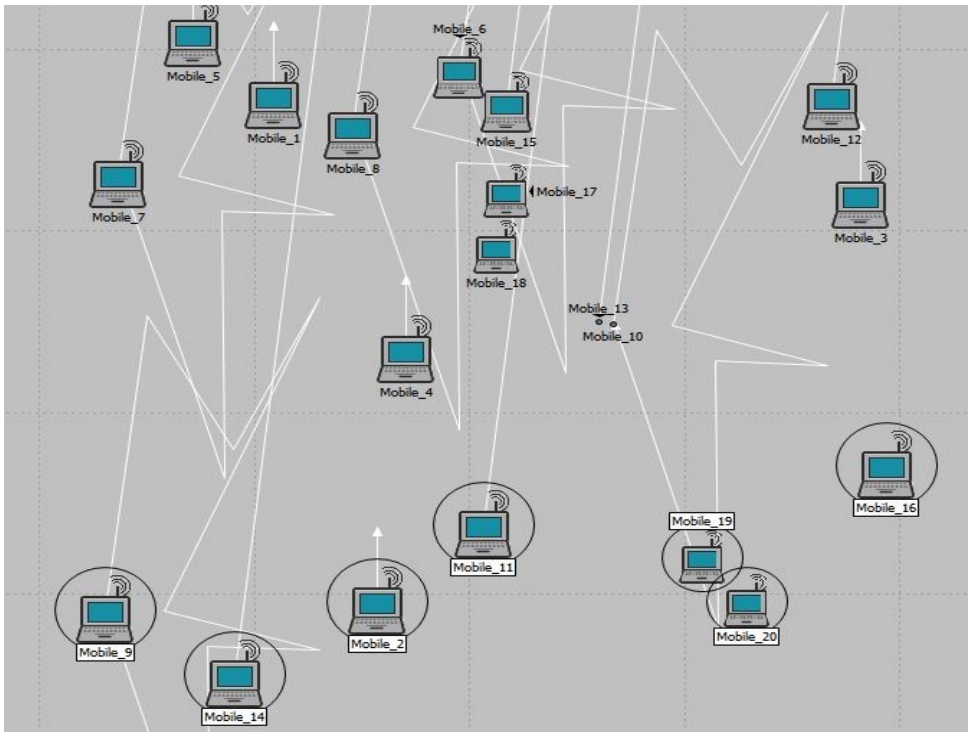

Figure 5.1: Network Model for a Small Random Ad Hoc Network

5.2 Small Ad Hoc Wireless Networks with Random Trajectories

Figure 5.1 shows the network model for a small random Ad Hoc network consisting of N=20 mobile nodes. To provide better comparison with Chapter 4, we have chosen the same set of source and destination pairs i.e. Nodes (1, 3, 11 and 16) constitutes source and nodes (2, 7, 12 and 17) are their corresponding destinations. Like Chapter 4, there are 4 flows corresponding to each source-destination pair in the network, i.e. m = 4 buffers are required in order to exploit

coding in this network. Similarly, we have sources start sending packets with gaps of 10s with node 1 starting at 10s, node 3 at 20s, node11 at 30s and node 16 at 40s. As discussed in Section 5.1, since there is no fixed route between source-destination it is very hard to present the paths taken by packets, and also about the coding, decoding, and innovative nodes. The benefits arising from this coding-decoding operation will be illustrated in the performance to be evaluated below.

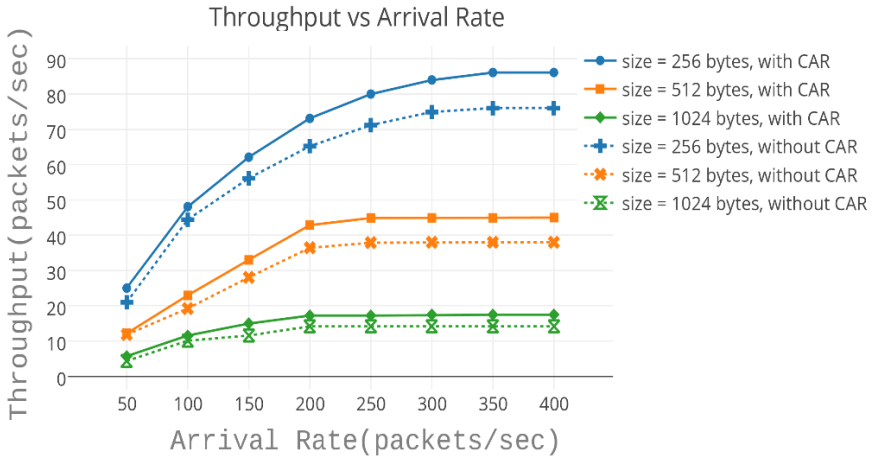

Figure 5.2a: Throughput vs Arrival Rate for a Small Random Ad Hoc Network

5.2.1 Throughput

Figure 5.2a shows the network throughput as a function of packet arrival rate for different packet sizes. For a packet size of 256 bytes using CAR network, the throughput increases more or less linearly at lower packet arrival rates. With increasing arrival rate (after 100 packets/sec), the rate of increase is decreasing until the throughput levels off at a level of 87 packets/sec when the arrival rate goes beyond 350 packets/s. Similar to regular Ad Hoc networks, this is because increasing data rate requires control messages to support their transmission which in turn take up more bandwidth from the server. The observations for different packet sizes are similar except that maximum throughputs (levelling off levels) are progressively decreasing (48 and 19 packets/sec) and the arrival rates beyond which maximum levels are reached are also decreasing (250 and 200 packets/s) as the packet size is increased from 256 bytes to 512 and 1024 packets, respectively.

Like CAR, network without CAR has shown similar behaviour but because CAR was able to find more coding chances in the network, throughput is higher for CAR. For example, for a packet size of 256 bytes, throughput levels off at 85 packets/sec for CAR network compared to 76 packets/sec for no CAR network.

In general, the throughput is less than the regular small network because the topology here needs to be re-evaluated quite often instead of only once for regular Ad Hoc networks. Each path evaluation needs to exchange control messages in the MAC and AODV layers. Thus, using generalized CAR has reduced data packet throughput.

As discussed for the regular Ad Hoc networks in Ch.4, node speed can have a considerable effect on network performance especially on throughput and ETE. The performance at different node speeds (i.e. 0, 2 and 5 m/s) are presented below.

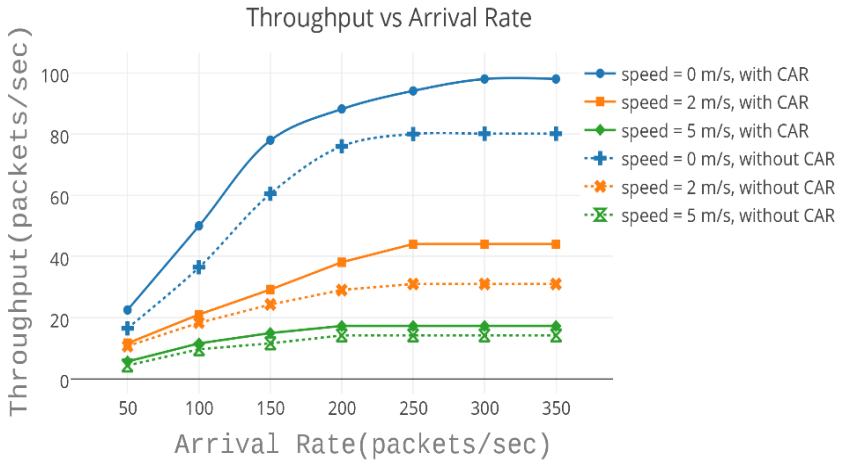

Figure 5.2b: Effect of Node Speed on Throughput for a Small Random Ad Hoc Network

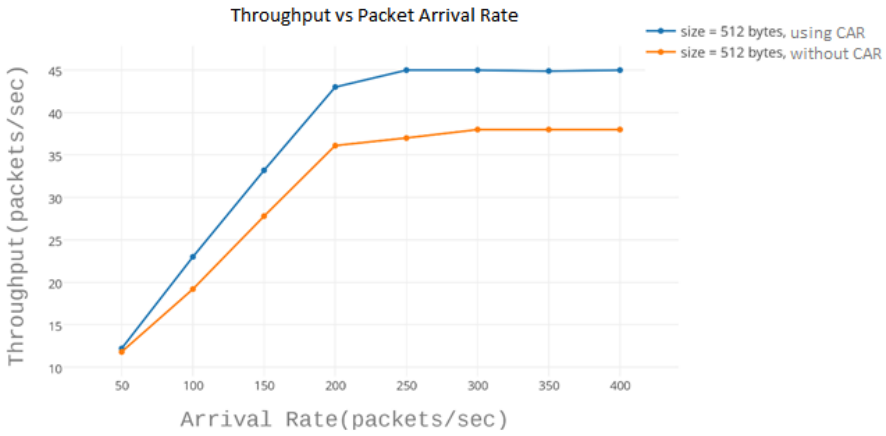

Figure 5.2c: Throughput vs Arrival Rate for a Small Random Ad Hoc Network

Figure 5.2b shows the behaviour of throughput as a function of packet arrival rate at different node speeds when packet size is fixed to 1024 bytes. At speed = 0 m/s for network using CAR, the throughput increases w.r.t arrival rate until the system limit is reached at 300 packets/sec and the throughput levels off at 98 packets/sec. The performance for networks when

nodes are moving faster are similar except throughput decreases with increasing speed. For example, throughput when node speed = 0 m/s levels off at 98 packets/sec as compared to 18 packets/sec. Again, because of more coding gain, CAR network was able to obtain more throughput than network without CAR.

Figure 5.2c shows the throughput of flow 1 at destination node 2 as a function of packet arrival rate. Packet size and speed are fixed to 512 bytes and 5 m/s respectively. It can be seen that for both CAR and without CAR networks, their throughputs are similar to the average throughput in Fig. 5.2a.

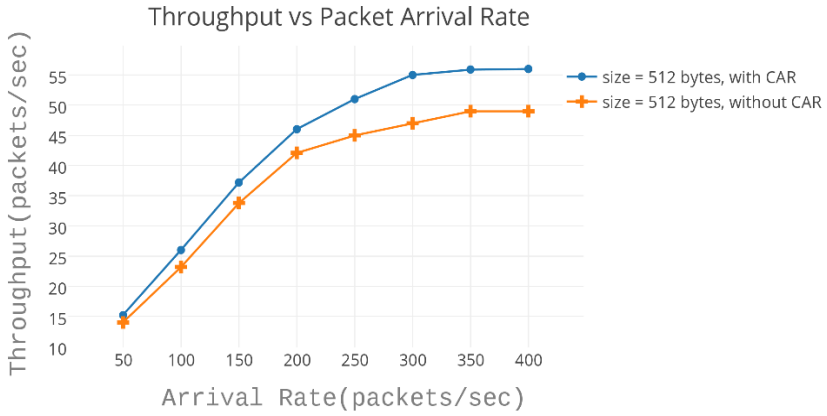

Figure 5.2d: Throughput vs Arrival rate for a Small Random Network

Since we were expecting to see quite different performance for different flows in a random Ad Hoc network, we check other flows by keeping above configuration i.e. packet and speed of 512 bytes and 5 m/s respectively. Figure 5.2d shows the throughput of flow 2 at destination node 7. It can be seen that throughput for flow 2 is higher than that of average i.e. it levels off at 55 packets/sec as compared to 45 packets/sec for average of flows. This can be explained by the fact that all intermediate nodes for flow 2 are moving in the same direction so that the topology (and therefore finding a new path) does not change frequently and hence, better performance in terms of data packets. Similar to flow 2, we also checked for flow 3 and 4, and they came out to be lower than that of average.

5.2.2 ETE Delay vs Arrival Rate

Figure 5.3a below explains the behaviour of average ETE delay as a function of packet arrival rate. Node speed and packet size is fixed to 5 m/s and 256 bytes respectively. For the network using CAR, ETE delay is relatively lower (i.e. 5.2 s) when packet arrival rate is less than 50 packets/sec.

Then it grows more rapidly towards infinity as the arrival rate is approaching the effective system service limit of 250 packets/sec. Like Chapter 4, the effective service rate due to background AODV and MAC communication can be much lower than the system capacity of 512 packets/sec. Similar observations and comments apply to the network without CAR, but since there is no exploitation of coding possibilities, its ETE delay is bit higher than the network without CAR. For example, when packet are arriving at 50 packets/sec, the ETE delay is 5.6 s for no CAR network and 5.2 s for network using CAR.

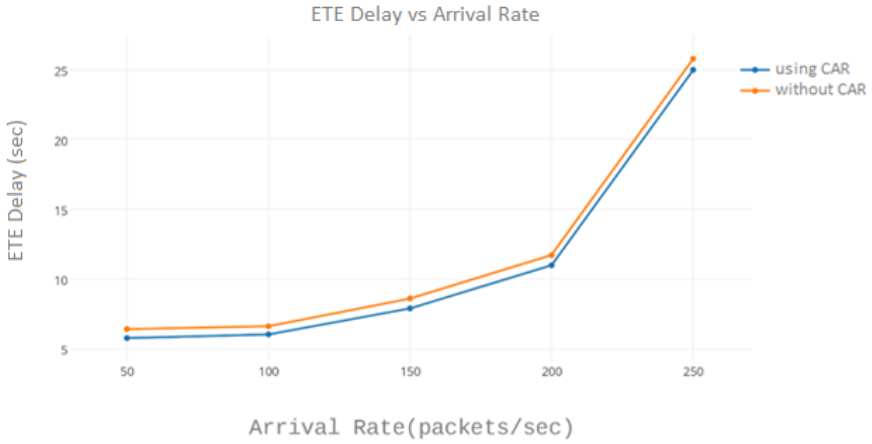

Figure 5.3a: ETE Delay vs Arrival Rate for a Small Random Ad Hoc Network

Being random in nature, the ETE delay for small random Ad Hoc network is bit higher than that of regular Ad Hoc network because the topology might need to be re-configured quite often. As discussed in source node actions in Section 2.3, a source has to wait for *x*=3 sec for evaluating the optimal path based on received RREPs; this extra wait contributes to higher average ETE delay for random networks.

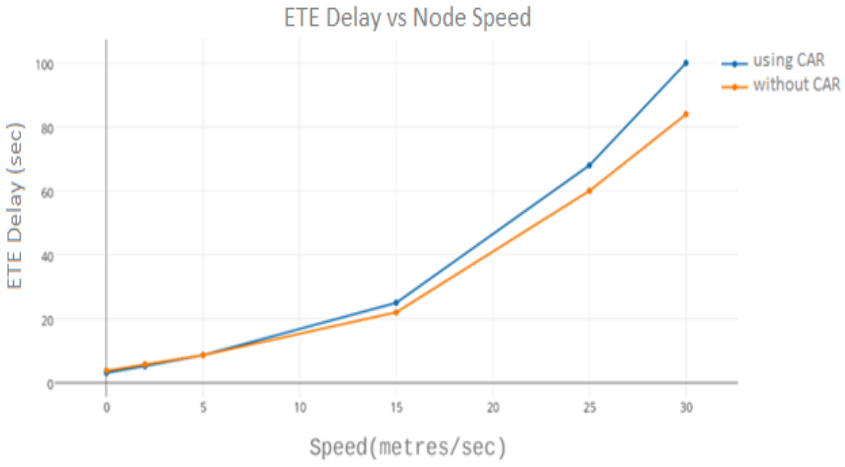

Figure 5.3b: Effect of Speed on ETE Delay for a Small random Ad Hoc network

Figure 5.3b shows the curve for ETE delay as a function of node speed when packet size is fixed to 1024 bytes and packet arrival to 100 packets/sec. It can be seen that ETE delay for network using CAR is lower than network without CAR up to node speed of 5 m/s, and beyond the delay using CAR actually increases more rapidly. This is because as speed increases network topology might need to be re-configured more often, and with each route-discovery, generalized CAR has to wait for *x*=3s in order to evaluate coding benefits for all possible paths to destination. This additional wait requires data packets to be queued longer before they can get serviced and hence, leads to higher average ETE delay for the flow. This can be confirmed from the performance obtained at 0 m/s and 30 m/s. At 0 m/s, CAR network has ETE delay of 3.1 s which is lower than that of network without CAR (at 3.9 s) but at 30 m/s, CAR network has more delay at 99 s as compared to 85 s for without CR network.

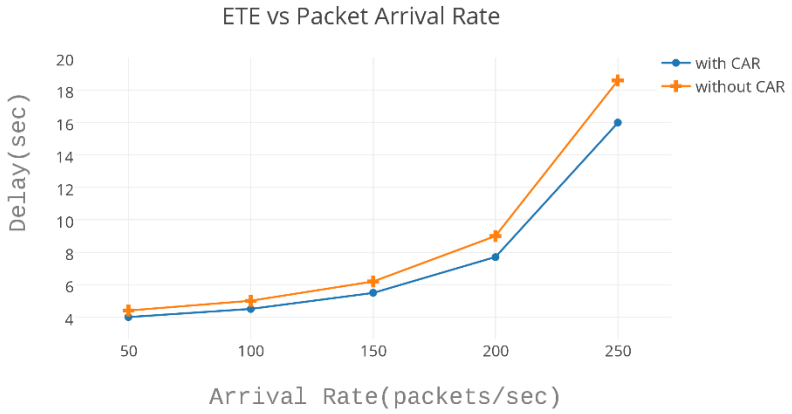

Figure 5.3c: ETE Delay vs Arrival Rate for a small Random Ad Hoc Network

Similar to the throughput in Fig. 5.2c, the ETE delay for flow 1 arbitrarily came close to average but ETE delay performance as shown in Fig. 5.3c is better for flow 2 because intermediate nodes between source and destination are moving in the same direction and hence, require less number of topology re-configurations. ETE delay for flow 3 and 4 is higher than that of average ETE delay because intermediate nodes for these flows are moving at different directions unlike for flow 2.

5.2.3 PDR vs Packet Error Rate

Figure 5.4 below shows the network behaviour in terms of PDR against packet error rate for different node speeds. Packet size is fixed at 512 bytes and packet arrival rate at 100 packets/sec.

For network using CAR at zero *per*, the PDR is at the highest level as expected but still only 0.44. The PDR performance decreases constantly reaching the lowest value of 0.07 when *pep* = 30% and speed = 0m/s. Similar observations can be made for curves corresponding to higher node speeds. For example, the PDR drops from 0.33 to 0.186 to 0.07 corresponding to increasing node speeds (i.e. 0, 2 and 5 m/s respectively) when *pep* = 15%. As discussed earlier, lower PDR can be attributed towards lower SNR at higher speed. PDR drops further at higher *pep* due to more un-decodable packets dropped at decoding nodes.

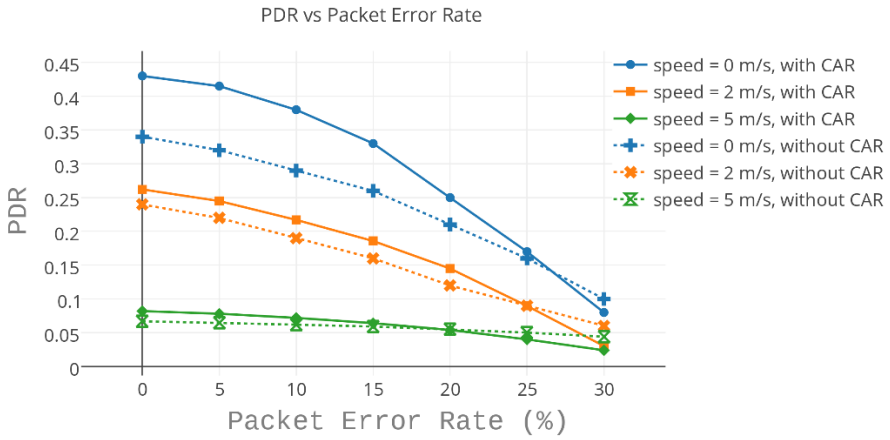

Figure 5.4: PDR vs Packet Error Rate for a Small Random Ad Hoc Network

For network without CAR, similar behaviour can be observed in Fig. 5.4 except PDR is relatively low as compared to CAR network when *pep* is low (i.e. 0.34 when *pep* = 0% and speed = 0 m/s) because network without CAR has obtained less coding gain than network using CAR. But unlike CAR network, PDR decreases slower because there are less number of packets that gets coded and therefore, less number of packets that are vulnerable to be lost at higher *pep* which can be confirmed from PDR of 0.1 for network without CAR and 0.08 for network using CAR when node speed = 0 m/s and *pep* = 30 %.

Similar to Fig. 4.4b, we have obtained PDR performance at node speed = 15 m/s and 30 m/s, but performance is much worse because of the random nature of this network.

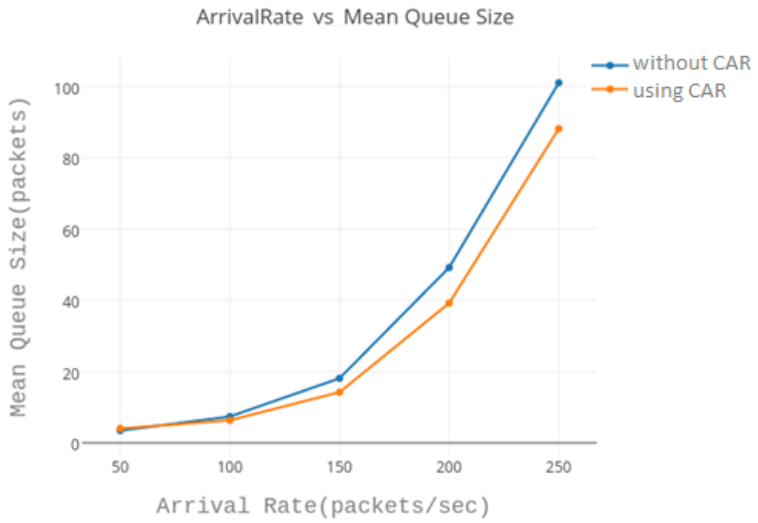

Figure 5.5: Mean queue size vs Arrival rate for a Small Random Ad Hoc Network

5.2.4 Mean Queue Size at Coding Nodes

Figure 5.5 depicts the average mean queue size of the coding nodes as a function of packet arrival rate when the node speed and packet size are fixed to 5 m/s and 512 bytes respectively. For network without CAR, the mean queue size is quite low and then increases exponentially beyond the arrival rate of 200 packets/sec. System becomes unstable after the arrival rate of 250 packets/sec as the number of packets arriving becomes higher than a node can process. As a result, queue size grows to infinity. So, the stability limit for arrival rate (in packets/sec) has to be lower.

Similar behaviour can be observed for network using CAR, but since network using CAR was able to be on more coding-possible paths as compared to network without CAR and due to coding gain, mean queue size is always lesser as compared to network without CAR. For example, at the arrival rate of 200 packets/sec, the mean queue size for CAR network is 40 packets as compared to 48 packet for network without CAR.

As a summary of Section 5.2, our performance evaluation has shown that despite lower performance in random networks as compared to regular networks, CAR is still able to get more performance gain by finding more coding-possible routes than network without CAR.

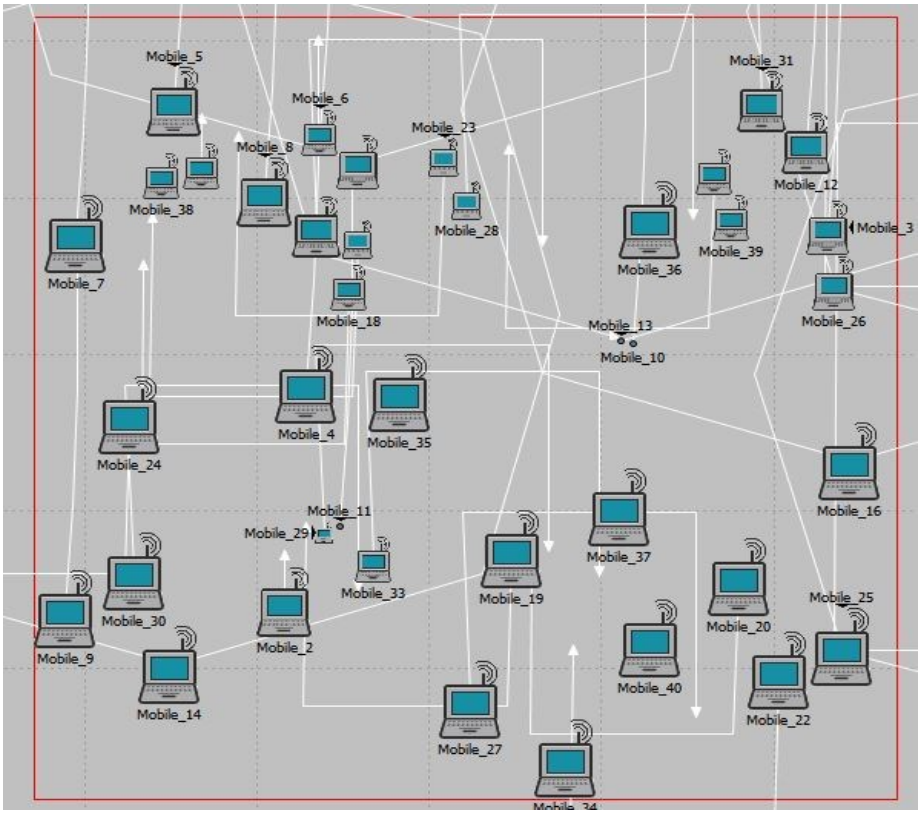

Figure 5.6: Network Model for a Big Random Ad Hoc Network

5.3 Big Ad Hoc Networks with Random Trajectories

We have confirmed performance gain in using CAR for small random networks. Now we shall see how does it scales in bigger networks. Figure 5.6 shows the network model of big random Ad Hoc network consisting of *N*= 40 mobile nodes, each node is moving at a speed of 5 m/s (default) in a random trajectory. To allow comparison with the regular Ad Hoc networks from before, we shall use the same set of source-destination pairs i.e. nodes (1, 7, 15, 22 and 6) and (11, 17, 25, 32 and 30) are the sources and corresponding destinations respectively. There are five flows in the network corresponding to each source-destination pair i.e. m = 5 buffers are needed in order to exploit coding opportunities in the network. Similarly, sources start their flows at intervals of 10s i.e. node 1 would start sending packets for node 11 at 10s, node 7 at t = 20s, node 15 at 30s, node 22 at 40 s and node 6 at 50s. As explained in Section 2.5, there are no fixed routes between the source and the destination when the nodes are moving randomly. All coding, decoding and innovative nodes need to be obtained dynamically mutiple times as the simulation progresses.

Similar to previous chapters we have evaluated this network to obtain Throughput, ETE delay, Mean Queue Size and PDR. The subsections below explain the results obtained.

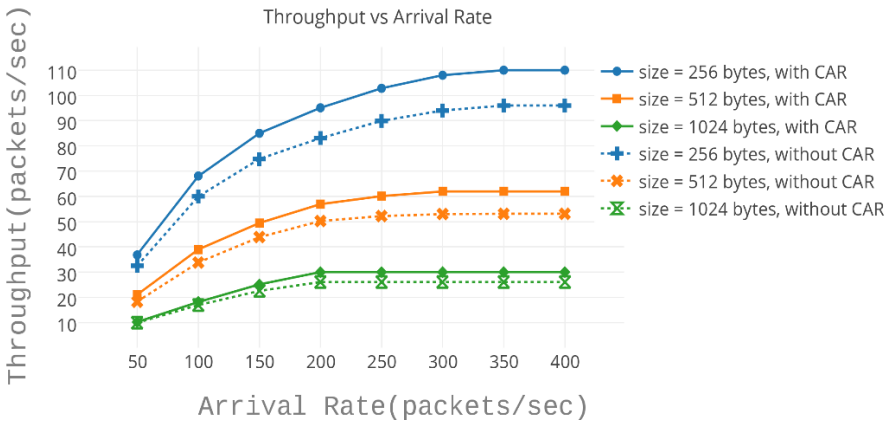

Figure 5.7a: Throughput vs Arrival Rate for a Big Random Ad Hoc Network

5.3.1 Throughput

Figure 5.7a shows the behaviour of throughput as a function of packet arrival rate for different packet sizes. Node speed is set to the default value of 5 m/s. For a packet size of 256 bytes using CAR, the throughput increases linearly for a packet arrival rate up to 100 packets/sec. After that it slows down progressively because with more arriving packets more control packets need to be exchanged between the nodes which in turn reduces further the bandwidth available for data packets. Beyond 350 packets/sec, the throughput curve levels off at 110 packets/sec when the system limit is reached and the node cannot cope up with the arriving packets. From Fig. 5.7a, similar observations can be made for other packet sizes (i.e. 512 and 1024 bytes), but as expected they all level off relatively earlier and at lower levels of throughput. For example, for a packet size of 512 bytes, the throughput levels off beyond arrival rate of 250 packet/sec and at maximum value of 62 packets/sec as compared to for packet size of 1024 bytes, the throughput levels off beyond arrival of 200 packets/sec and at maximum value of 25 packets/sec.

On the other hand, network without CAR has lower throughput than network using CAR for all packet sizes due to less coding nodes found and hence less coding gain. For example, for a packet arrival rate and a packet size of 100 packets/sec and 256 bytes respectively, the network using CAR was able to obtain throughput of 71.2 packets/sec as compared to 61.1 packets/sec for network without CAR. Similarly, for packet size of 1024 bytes, the network using CAR was able to obtain higher throughput (i.e. 25 packets/sec) than network without CAR (i.e. 22 packets/sec).

Similar to the observation and discussion in Section 4.3.1, the big random network has a better throughput performance than small random network because there are less control packets involved in re-transmissions, thus leaving more bandwidth available for data packets (as

explained before, SNR is higher when there are more nodes within the same area which reduce packet errors and therefore re-transmission). On the other hand, its performance is worse than the big regular Ad Hoc network in Fig. 4.7a because as explained for Fig. 5.2a, the random network needs more control packet communication which reduces the bandwidth available for data packets.

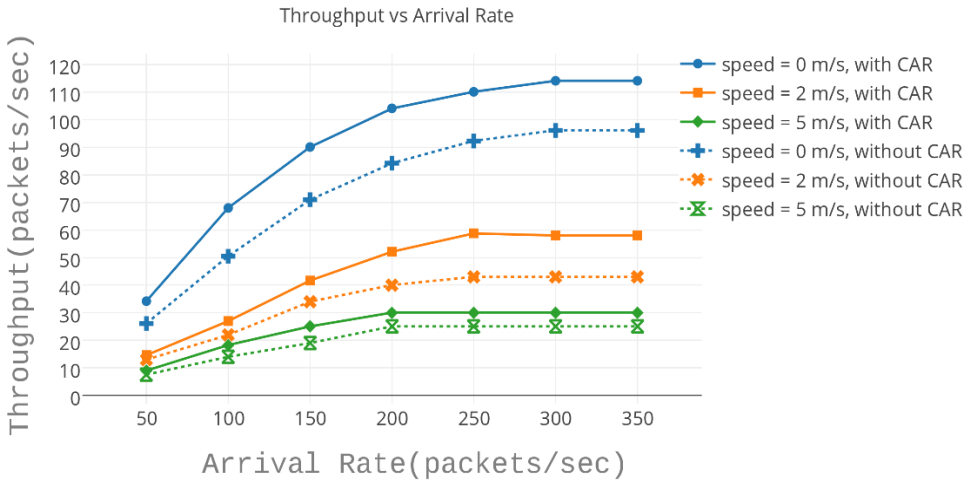

Figure 5.7b: Effect of Speed on Throughput for a Big Random Ad Hoc Network

As in the small network, it is interesting to see the effects of node speed on throughput. Figure 5.7b shows the network throughput as a function of packet arrival rate for different node speeds. Packet size is set to 1024 bytes. For network using CAR when the node speed is 0 m/s, the throughput increases more or less linearly w.r.t packet arrival rate but only up to 100 packets/sec. Beyond that, the increase slows down progressively. The explanation is similar to that of a small network where increased speed decreases SNR quite rapidly which further degrade network throughput. Beyond an arrival rate of 350 packets/sec, the throughput levels off at a maximum throughput of 116.7 packets/sec when the node cannot cope up with arriving packets. Similar observations can be made for curves corresponding to node speeds of 2 m/s and 5 m/s. Since the SNR decreases with increasing node speed, the throughput levels off relatively earlier and at lower levels of maximum throughput obtained. Again, because of more coding-benefits CAR network, throughput for network using CAR is always higher than no CAR network.

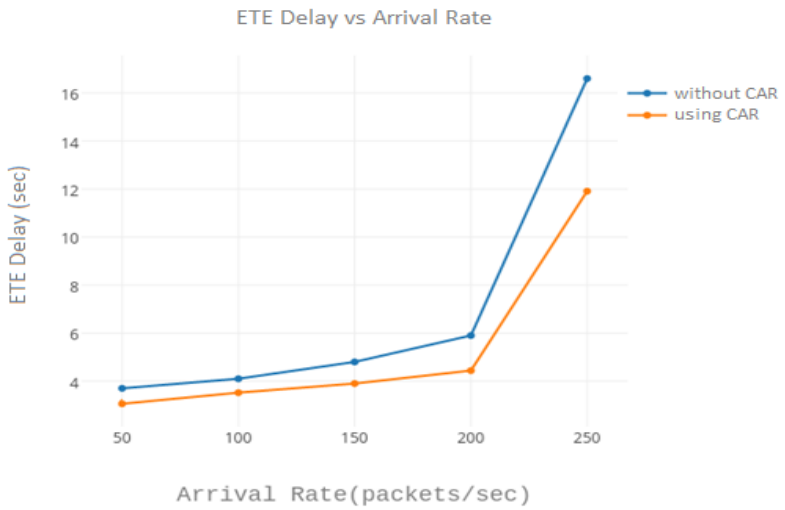

Figure 5.8a: ETE Delay vs Arrival Rate for a Big Random Ad Hoc Network

5.3.2 ETE Delay vs Arrival Rate

Figure 5.8a shows the performance of end-to-end delay against packet arrival rate when node speed is 5 m/s and packet size is 1024 bytes. For network without CAR, the ETE delay is quite low (i.e. 3.9 s) and then, shoots up beyond the arrival rate of 200 packet/sec. Beyond the packet arrival rate of 250 packets/sec, the ETE delay grows to infinity because the system limit has been reached and the queue builds up continuously.

Similar observations can be made for network using CAR but its ETE delay is lower. This is because CAR was able to find more coding-possible paths, and the more coding gains result into lower ETE delay. For example, at packet arrival rate of 150 packets/sec, ETE delay for network using CAR is 3.82 s as compared to ETE delay of 5.09 s for no CAR network.

As explained for Chapter 4, bigger networks has better connectivity and hence, better ETE delay performance as compared to their corresponding smaller networks. This network is no exception, the ETE delay for big random Ad Hoc network is always less than that of small random Ad Hoc network. For example: for packet arrival rate of 100 packets/sec, ETE delay for big network using CAR is 4.2 s as compared to 5.8 sec for network without CAR. As in previous chapters, we are interested in seeing how node speeds affect delay as investigated and discussed below.

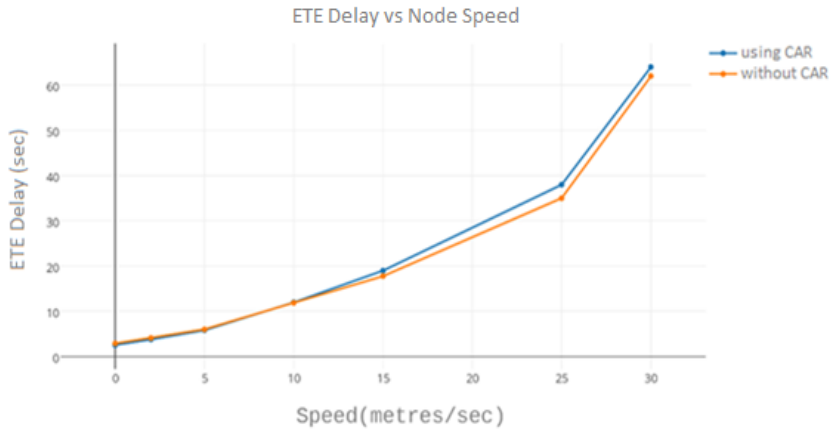

Figure 5.8b: Effect of Speed on ETE Delay for a Big Random Ad Hoc Network

Figure 5.8b shows the performance of ETE delay as a function of node speed when packet size and arrival rate is set to 1024 bytes and 100 packets/sec. The curve for network using CAR rises more or less linearly with respect to node speeds up to 15 m/s. It tends to increase more rapidly beyond 15m/s because higher speeds lower SNR significantly which results into increased delay.

Similar observations can be made for network without CAR but since CAR was able to exploit more coding chances in the network, the ETE delay is relatively lower for CAR but only up to 5 m/s. This is due to the same fact as we have discussed for small network that with increased node speed, the network topology and further, coding-chances for CAR network needs to be evaluated more frequently as compared to network without CAR which leads more rapid increase in ETE delay.

5.3.3 PDR vs Packet Error Rate

Figure 5.9a below shows the behaviour of packet delivery ratio as a function of packet error rate when the packet arrival rate and the packet size are set to 100 packets/sec and 512 bytes respectively. For the network using CAR and node speed = 0 m/s, PDR yields maximum output but still only 0.76 with 0.24 of *pd* when *pep* = 0 %. The PDR decreases further with increasing *pep* because it would be less likely to find decoding packets at decoding nodes. For example, PDR is at the lowest value of 0.28 with *pd* at the highest of 0.42 when *pep* = 30 % and node speed = 0 m/s. Similar observations can be made from curves corresponding to different node speeds but with increasing speed, the PDR decreases as SNR decreases with speed.

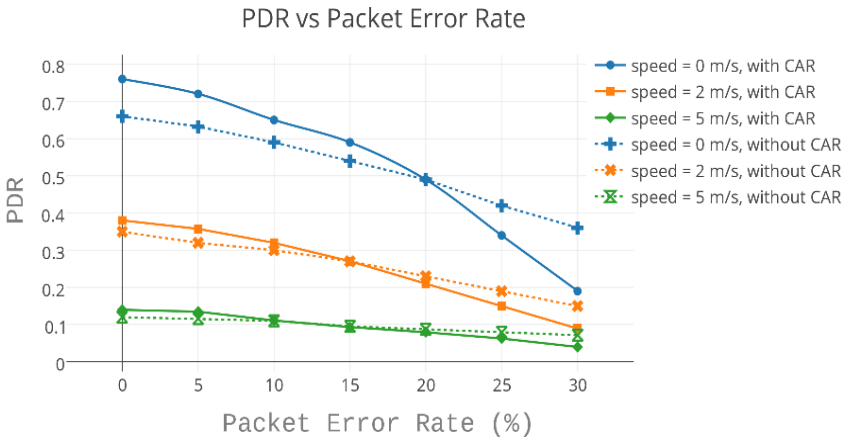

Figure 5.9a: PDR vs Packet Error Rate for a Big Random Ad Hoc Network

Similar observations can be made for network without CAR except PDR is lower but similar to previous chapter only at lower levels of *pep*. For example, the PDR for network using CAR is 0.77 and for network without CAR is 0.68 when *pep* = 0%. As *pep* increases, PDR for network without CAR drops more slowly than network using CAR. This is because a network without CAR locates less coding nodes, and therefore has less dependency on the decoding packets that are likely to be lost in higher *pep* and/or speed i.e. *pd* [using CAR] > *pd* [without CAR]. Similar to the small random Ad Hoc network, *pep* at which crossover point occurs decreases with increasing speed because of the lower SNR at high speed. For example: crossover points decrease from 20 % to 16 % to 11% as the node speed increases from 0 to 2 to 5 m/s respectively.

As discussed earlier because topology needs to be re-configured quite often for random networks, PDR for big random Ad Hoc network is lower than big regular Ad Hoc network as shown in Fig. 4.9a. The lower PDR can be attributed to more control communication in case of random networks which leaves less bandwidth available for the data packets. For example, the maximum throughput obtained for big random network is 0.768 as compared to 0.81 for big regular Ad Hoc network.

As compared to small random network as given in Fig. 5.4, significant improvements can be seen for this network (i.e. big random Ad Hoc). As already discussed, this is because of more nodes with in the same area compared to small network which increases network connectivity and in turn, increases SNR.

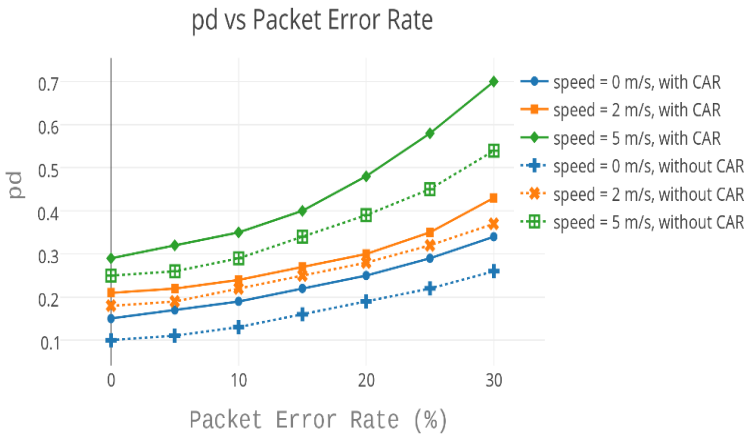

Figure 5.9b: pd vs Packet Error Rate for a Big Random Ad Hoc Network

Figure 5.9b shows the effect of *pep* on *pd* and *pep* when nodes are moving at different speeds. At a node speed = 5 m/s in networks using CAR, *pd* increases somewhat linearly up to *pep* = 10% and beyond that it starts to grow in exponential manner reaching 0.7 when pep = 30%. This is because at high packet error rate there are more chances that decoding packets would not be able to reach decoding nodes. Similar observations can be obtained for curve corresponding to lower node speeds but since SNR is relatively higher for slower moving nodes, *pd* grows more linear. When *pep* = 30%, the maximum value of *pd* decreases from 0.7 to 0.5 and to 0.3 when node speed is decreased from 5 m/s to 2 m/s and to 0 m/s, respectively.

Like networks using CAR, observations for network without CAR are similar but with a lower *pd* all the time. This is because networks without CAR have less dependency on decoding packets. For example, the (minimum, maximum) values of *pd* for networks using CAR are (0.3, 0.7), (0.2, 0.4) and (0.15, 0.3) for node speeds of 5 m/s, 2 m/s and 0 m/s respectively. For networks without CAR, these value pairs are (0.25, 0.55), (0.15, 0.3) and (0.1, 0.2) respectively.

It can also be observed that *pd* for regular Ad Hoc networks is generally lower because all nodes are moving in predefined circular octagonal manner as compared to random for random Ad Hoc networks and hence, SNR is relatively higher for regular Ad Hoc networks.

5.3.4 Mean Queue Size at Coding Nodes

Figure 5.10 below shows performance of mean queue size of the coding nodes as a function of packet arrival rate when packet size and node speed is fixed to 256 bytes and 5 m/s. For networks without CAR, their mean queue is quite low at low arrival rates (e.g., 5.2 packets when arrival rate is 50 packets/sec) but more rapidly up to 200 packets/sec. Beyond that, it starts shoot up.

Beyond 250 packets/sec, mean queue sizes appear to grow to infinite because the server is not able to cope with the incoming packets.

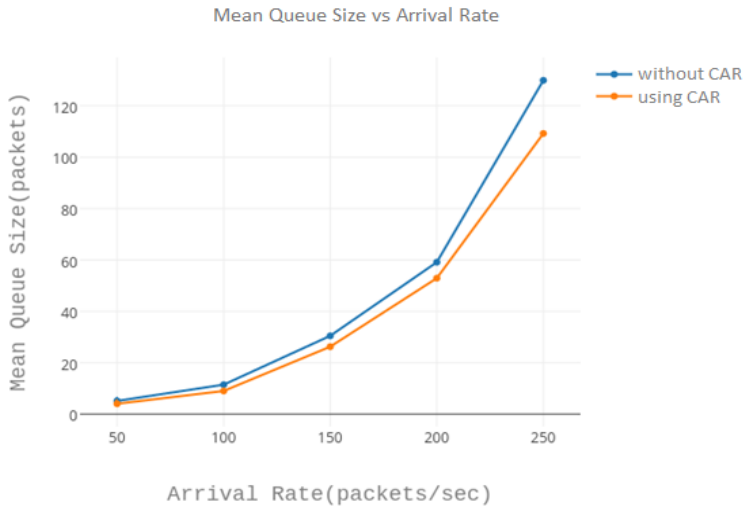

Figure 5.10: Mean Queue Size vs Arrival Rate for a Big Random Ad Hoc Network

Similar observations can be made for network using CAR, but since CAR network was able to find more coding nodes and send more than 1 packets together more often, the mean queue size for CAR network is relatively smaller as compared to no CAR network which has obtained lesser coding benefits.

5.3.5 Comparison with Other Networks

Since a big random network has more nodes within the same area as a small random network, the performances of throughput, ETE delay, PDR and mean queue size are better in general. The SNR is relatively higher than small random Ad Hoc networks because the connectivity among the nodes is better, thus leading to lower *ber* and *pep* which can also be confirmed from Figs. 2.13 and 2.14. On the other hand, all performance measures of big random Ad Hoc network are worse than big regular Ad Hoc network in general because more network bandwidth is wasted on control communication when topology undergoes constant changes, thus reducing the effective bandwidth to serve data traffic.

5.4 Effect of Receiver Sensitivity

Like Chapter 4, we have studied network performance when receiver sensitivity is reduced from -95 dBm to -300 dBm for big random Ad Hoc network. Fig. 5.11 shows its effect on throughput for both network using CAR and network without using CAR when packet size and node speed is

set to 256 bytes and 5 m/s respectively. For reception sensitivity of -300 dBm and network using CAR, throughput starts to increase more or less linearly with arrival rate until it levels off at 148 packets/sec. Almost similar observations can be seen at reception sensitivity = -95 dBm but throughput here levels off at relatively lower value i.e. 128 packets/sec. This is because in former case, nodes can sense packets with relatively lower power threshold with the help of which more number of packets can be successfully transmitted from source to destination.

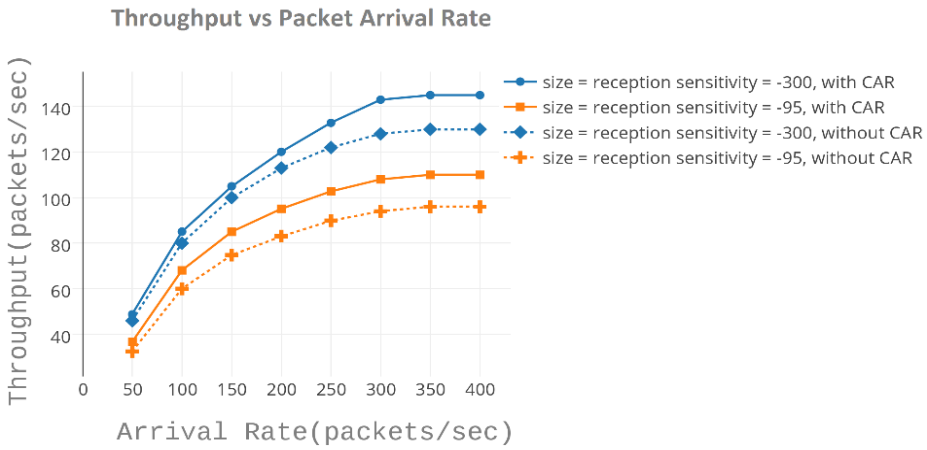

Figure 5.11: Effect of Receiver Sensitivity on Throughout for Big Random Ad Hoc Network

Similar to Fig. 5.2a, network without using CAR has shown similar but lower performance than their networks using CAR counterparts. For example, for reception sensitivity of -300 dBm, network without CAR levels off at 95 packets/sec as compared to 130 packets/sec for network using CAR.

Like Chapter 4, all performance measures (not shown here) are better when the receiver sensitivity is reduced to -300 dBm.

5.5 Concluding Remarks

From the performance measures obtained for both small and big random Ad Hoc networks, we can see their performance can significantly be affected by node and network parameters. In general, the throughput performance is increasing and then levelling off with respect to increasing arrival rate due to limited bandwidth available from the node. It is a decreasing function of node speeds due to decrease in SNR. The delay is an increasing function of arrival rate because the server capacity is limited. It is also an increasing function of speed because SNR is quite low at higher speeds because of which the packets that get lost needs to be

retransmitted. Packet loss rate is an increasing function of increasing *pep* and *pd* which in turn are increasing function of node speed because of increased SNR. Like Chapter 3 and 4, we have observed scenarios of defective coding in this chapter as well.

For comparison, networks using CAR has better performance than networks without CAR due to more coding gains obtained. Larger networks have better performance due to better connectivity between the intermediate nodes. Random networks here have shown lower performance because the topology needs to be re-configured quite often (i.e. once any node goes out of communication range) which leads to wastage of system resources and hence, has lower performance. Due to the fact that a route to the destination might need to be re-evaluated number of times for random trajectories as compared to one for regular ones, the performance for random Ad Hoc networks is generally lower than that of regular Ad Hoc networks. Also, like Chapter 4, here we find no bottleneck links which could hinder performance for networks using CAR.

Chapter 6
Design Issues and Guidelines

This chapter summarizes the design issues we have encountered in this research work and explains some guidelines that can be useful in exploiting network coding in network. From our experience on related literature review, building algorithms and on the finite set of simulation experiments in Chapters 3 to 5, we want to provide some recommendations that would give better network performance.

6.1 Good and Bad Placement of Coding Nodes for NC or CAR

Through our simulations, we have come to realize that placement of nodes in NC or CAR can be good or bad depending on the quite different performances they provide. Good placement improves performance over traditional routing whereas bad placement yields no performance gain. They are exemplified below.

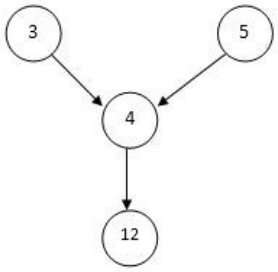

Figure 6.1: Good Placement Example

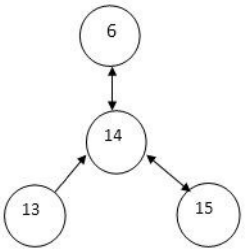

Figure 6.2: Bad Placement Example

Figure 6.1 provides an example of packet among transmission nodes 3, 4, 5, 12 from the big wired network as shown in Fig. 3.5. Packets from nodes 3 and 5 have to go through link between 4 and 12. Thus, the coding of packets from nodes 3 and 5 would allow node 4 to send just one packet instead of two, and this results in bandwidth savings in case of NC or CAR as compared to traditional routing.

Fig. 6.2 shows that packets between node 13 and 15 have to go through node 14. Likewise, node 14 has to relay packets between 15 and 6. Thus, node 14 is a common node for both flows and there is a coding-opportunity at this node. Coding between (13 -> 15) and (15->6) flows would not result in any bandwidth savings because links 14-6 and 14-15 would continue to

have the same transmission usage when compared to the traditional routing case. Thus, CAR algorithm should identify this type of scenarios in a network and try to avoid applying coding for those nodes in the networks as they do not yield any coding gains.

6.2 Use of Network Coding in Highly Lossy Networks

A network is lossy if its loss probability p as defined in Section 2.4.4 is greater than zero. In comparison to network without CAR, the performance evaluations in Chapters 3 to 5 show that PDR is worse in networks using CAR when *pep* is high (i.e. > 5%). This is because a coded packet has to rely on decoding packet to get decoded before reaching destination. This can be illustrated by the example below.

Let P(a|traditional routing) be the probability of a data packet 'a' reaching destination in traditional routing (without using NC). For NC or CAR networks, the coded packet depends on the decoding packet 'b' in order to be delivered to the destination. Let P(b) be the probability of packet 'b' reaching its decoding node. Assuming the events of packet 'a' reaching its destination and packet 'b' reaching the decoding node are independent. Thus the probability of packet 'a' reaching destination in networks using CAR, is P(a|NC or CAR]= P(a|traditional routing) * P(b) would always be lower than P(a|traditional routing) because P(b) is always less than 1 for lossy networks. Thus, one should avoid applying network coding in lossy environments as it can further decrease the probability of packet 'a' reaching destination.

6.3 Trade-offs

Some trade-offs in performance are apparent from the performance evaluations in Chapters 3 to 5. First, the normal delay-throughput trade-offs of an ordinary queueing system holds. That is, higher the throughput, higher the delay. Furthermore, the maximum throughput is capped by the server bandwidth. In our cases, the available bandwidth is further reduced by the overhead required for communication in the mobile wireless network to support data transmission and route discover. There are others specific to our systems.

1) Using CAR can improve the performance for throughput, ETE delay and mean queue size but PDR is generally lower when pep goes higher than 10%.
2) Using CAR with low *pep* is better than network without CAR but the reverse is true for high *pep*.
3) Using a big wired network (40 nodes) can have higher ETE delay than small wired network

(20 nodes) because packet might have to travel through more number of nodes (and therefore more queueing) before reaching destination.

4) Using a big Ad Hoc (network with 40 nodes) may yield better performance as they improve network connectivity and in turn, SNR.
5) Performance of network using CAR gets worse than network without CAR as node speed goes beyond 5 m/s.
6) The maximum arrival rate should not exceed maximum link/server or else, ETE delay and mean queue size will yield poor performance.
7) Packets of lower size (256 bytes) can yield better coding chances because smaller packets can be serviced faster and thus, is more probable for a node to have more than one packets to transmit simultaneously.

Chapter 7
Conclusions

We have read many recent research papers on network coding and coding-aware routing for this thesis work. Some major work are commented in our literature review from which we have proposed CAR (Coding Aware Routing) that have no requirement/dependency on particular medium, topology and size. Many simulations based on our CAR algorithm have been conducted to evaluate the performance and to compare with networks not using CAR for three types of networks (the wired networks and Ad Hoc networks with regular and random topologies). Results in Chapter 3, 4 and 5 have confirmed that our generalized CAR algorithm can provide coding-benefits independent of the types of topology, size and transmission medium. We have identified that NC in general can improve the performance of network throughput, end-to-end delay, and reliability when compared to the traditional routing approach.

Despite the significant performance gain in terms of throughput, ETE delay and mean queue size, we found that incorporating NC/CAR is not always the best solution because *pep* can be quite high in a lossy environment which results in lower PDR (because of the dependence of NC on decoding packets). As discussed in Chapter 4 and 5, the PDR in MANETs is generally very low because of lower SNR, which degrades further with increasing node speed.

We have spent a significant amount of time in learning how to model packet error rate (*pep*) in OPNET because it depends upon many different parameters especially under the mobile environment. We finally figured out that the best way to model *pep* is by varying the transmitter power; this amount of time could have been saved. Since OPNET was no longer supported by Riverbed, a lot of time was also spent on figuring out where to make the changes to allow overhearing. We eventually determined that it is easier to modify the built-in MAC layer code in OPNET. Initially, we were seeing some weird results in our simulations, which turns out to be the wrong buffer size we used. This is something one should be careful as it can effect packet queueing and in turn the network performance.

7.1 Future Work

Due to limited time, there are still issues/limitations that need to be investigated. The following is a summary for further investigation.

1) As discussed in Section 2.3.2, there can be situation of defective coding in a network when w = 0. However, if *w* /= 0, a packet would incur further queueing which could degrade the performance but at the same time might be able to lessen the probability of defective coding as discussed. Thus, we would like to see if there is some optimal value of w>0 that can yield optimal results.
2) Removing the reliance on decoding packet so that no NC is required when network is noisy. This is based on the PDR performance measures that CAR or NC has lower PDR performance than networks without CAR (traditional routing) when *pep* increases.
3) Implementing our algorithm with linear coding procedures to see if it can yield better performance.
4) Ways to identify bottleneck links in network (discussed in Section 6.1) to avoid applying NC in those networks because they yield no coding gains.
5) It can be observed from chs. 4 and 5 that we are using same node speed for all mobile nodes. However, we would also like to see how performance is effected when nodes are moving at different speeds.
6) Obviously, more performance evaluations on other types of networks (big or small) would help to ascertain certain statements we made in the design guidelines (Ch.6), concluding remarks (Chs. 3 to 5) as well as our Conclusions (Ch.7).

References:

[AhCa00] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung., "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, pp. 1204-1216, 2000.

[CaGo13] Chong Cao, Ping Gong and Li Chou, "Random Network Coding Based the Effective Wireless MAC Protocol", *Proceeding of 2013 4th IEEE International Conference on Software Engineering and Service Science (ICSESS),* pp.393-396, Beijing, May 2013.

[ChAg13] Yang Chi and Dharma P. Agarwal, "HyCare: Hybrid Coding-Aware Routing Metric in Multi-hop wireless", *Proceeding of 2013 IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems,* Hangzhou, China*,* 14 – 16 Oct. 2013, pp. 136-144.

[ChCh12] Kun-Cheng Chung, Hsin-Chun Chen and Wanjiun Liao. "CAOR: Coding-aware Opportunistic Routing in Wireless Ad Hoc Network", *Proceeding of 2012 IEEE International Conference on Communications,* Ottawa, pp. 136 – 140, 2012.

[Ding14] Xiake Ding, "Network Coding performance evaluation and an application to Underwater Sensor Networks", Master's program thesis, University of Ottawa*,* 2014.

[DiZH09] Xuyang Ding, Xue Zhang, Mingyu Fan, and Yaling Yang. "Coding-Aware Routing for Unicast Sessions in Wireless Networks", *Proceeding of 5th International Conference on Wireless Communications, Networking and Mobile Computing, 2009 (WiCom '09)* 24-26 Sept. 2009.

[GuJi14] Guanhua Guo, ZhenZhen Jiao, Zheng Yao, Baoxian Zhang and Cheng Li, "A Location-Based Friend-Assisted Coding-Aware Routing Protocol for Wireless Multi-hop Networks", *Proceeding of IEEE International Conference on Communications* Sydney*,* 10 -14 June 2014.

[HoMe03] Tracey Ho, Ben Leong, Ralf Koetter, Muriel Medard, Michelle Eflros, and D. R. Karger, "On randomized network coding", *Proceeding of 41st Ann. Allerton Conference of Communications, Control, and Computation*, vol.36, no.1, Urbana-Champaign, IL, Oct. 2003, pp. 63–68.

[LeLu10] Jilin Le, John Lui, Dah-Ming Chiu, "DCAR: Distributed Coding-Aware routing in Wireless Networks", *IEEE Transactions* on Mobile Computing, vol. 9, issue 4, pp.596-608, 2010.

[LiRa07] Li (Erran) Li, Ramachandran Ramjee, Milind Buddhikot, and Scott Miller, "Network-Coding Based Broadcast in Mobile Ad Hoc Networks*", Proceeding of IEEE INFOCOM 2007 – 26th IEEE International Conference On Computer Communications* Anchorage*,* 6 -12 May 2007.

[LiYe03] Shuo-Yen Robert Li, Raymond W. Yeung, "Linear Network Coding", *IEEE Transactions On Information Theory*, vol. 49, February 2003.

[KaRa08] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Medard and Jon Crowcraft, "XOR in the Air: Practical Wireless Network Coding", *IEEE/ACM Transactions on Networking*, Vol.16 (3), pp.497-510, June 2008.

[NiSa06] Bin Ni, Naveen Santhapuri, Zifei Zhong and Srihari Nelakuditi, "Routing with Opportunistically Coded Exchanges in Wireless Mesh Networks", *2006 2nd IEEE Workshop on Wireless Mesh Networks,* Reston, VA*,* pp.157-159, Reston, VA, 2006.

[PaCh08] Parimal Parag, Jean-Francois Chamberland, "Queueing Analysis of a Butterfly Network for Comparing Network Coding to Classical Routing", *Proceeding of IEEE*

International Symposium on Information Theory, Toronto, ON, Canada, 2008.

[QiYa13] Yang Qin, Lie-Liang Yang, "Delay analysis of network coding nodes and Butterfly network employing stop-and-wait automatic repeat request" *Proceeding of IET Communications,* Vol. 7, #5, pp. 490 – 499, 26 March 2013.

[SaEg03] Peter Sanders, Sebastian Egner, and Ludo Tolhuizen, "Polynomial time algorithms for network information flow", Proceeding *of the 15th Annual ACM Symposium on Parallel algorithms and architectures,* pp.286-294, 2003.

[SeRa10] Sudipta Sengupta, Sharavan Rayanchu and Suman Banerjee*,* "Network Coding-aware routing in Wireless networks", *Proceeding of IEEE/ACM Transactions on Networking*, pp.1158-1170, August 2010.

[SuSh09] Jay Kumar Sundararajan, Devavrat Shah, Muriel Medard, Szymon Jakubczak, Michael Mitzenmacher and Joao Barros, "Network Coding Meets TCP: Theory and Implementation", *Proceeding of the IEEE Conference on Computer Communication,* Brazil, 2009.

[WaZh14] Jin Wang, Cenzhe Zhu, Teck Yoong Chai, and Wai-Choong Wong, *"*SCAR: A Coding-Aware Routing Protocol with Self-Recommendation in Static Wireless Ad Hoc Networks*", Journal of Computer Networks and Communications*, 2014.

[TaQi11] TAN Xiaobin, QIN Guihong, CHENG Wenfei, "Loss-Aware Linear Network Coding for Wireless Networks", *Proceeding of the 30th Chinese Control Conference,* Yantai, China*,* 22 – 24 July 2011.

[YaLu12] Hongji Yang and Zheng Lu, "Unlocking the Power of OPNET Modeler", February 2012. Note that official documentation only comes with product and can be accessed from the Help section from within OPNET software. Since OPNET doesn't provide open and free documentation anymore, the above is the reference to a book for OPNET learning.

[YuZh12] Hao Yue, Xiaoyan Zhu, Chi Zhang, Yuguang Fang, "CPTT: A high throughput coding-aware routing metric for multi-hop wireless networks", Proceeding of *Global Communications Conference (Anaheim, CA)*, pp.5687-5692, 3 – 7 December 2012.

[ZhCh08] Jian Zhang, Yuanzhu Peter Chen and Ivan Marsic, "MAC-layer proactive mixing for network coding in multi-hop wireless networks*"*, *Proceeding of 9th ACM International Symposium on Mobile Ad Hoc Networking*, May 26-30, 2008.

[ZhZh09] Jin Zhang and Qian Zhang, "Cooperative Network-Coding aware routing for Multi-Rate Wireless Networks", *Proceeding of INFOCOM 2009,* Rio de Janeiro, Brazil*,* pp. 181-189, 19 - 25 April, 2009.

Appendix A: Pseudo-code for General Coding-Aware Routing Algorithm

Pseudo-code for our generalized CAR algorithm is given below. It defines the steps for source, intermediate and destination nodes and is implemented in "proc" module of OPNET for networks using CAR. It would be easier for someone to extend this work by this.

→Pick a packet 'x' at head of queue at node 'n'…..

if (x == new_packet) then //check to avoid looping of packets

//packet has been received at source node from application layer
→ if (n == {source}) then
  → generate(RREQ)
  → broadcast(RREQ)
  endif

→ if (x == RREQ) then

  // if node doesn't belong to destination node, then.
  if(n ∉ {destination})
  → x->neighbor_list += N {n} //Embeds the neighbors of n into the packet 'x'
  → broadcast x
  endif

  if (n ∈ {destination}) then
  → x->neighbor_list += N{n} //Embeds the neighbors of destination.
                              //previous hop information will be retrieved .
                              //from the packet itself

  // RREP packet is not actually a newly generated packet,
  // it's just setting the RREP bit in RREQ packet.
  → Generate(RREP)

  // RREQ packet has contained the information of all the previous nodes a
  // packet has traversed through, thus RREP can be unicast to previous node
  // obtained from RREQ packet.
  → Unicast_to_previous hop(RREP)
  endif

 if (x == RREP) then

  // if this node is not a destination node and satisfies the coding criteria
  // then calculate the coding benefit and embeds that into RREP packet.
  if (coding_criteria_satisfies && n ∉ {destination}) then
  → calculate (coding benefit) // say 'C'
  → x->coding_benefit += C //Embeds the coding benefit into RREP

```
        → associate(x_flow_id & y_flow_id)    // mark an association between this packet flow id
        // and other flow id 'y' with which the current flow satisfies the coding criteria.

        endif
        if (coding criteria doesn't satisfies) then
        → x->coding_benefit += 0
        endif
    → route(x towards destination)
    endif

    if (x == data_packet) then
    // if the packet received is a data packet and the node has been associated
    // to be a coding node for this flow id, then code the packet with an equivalent
    // packet from other stream and send it to next hop towards destination.
        if (coding_status == 0) then      //packet isn't coded yet
          if (n ∈ {coding_node}) then    //validate if coding can be possible
          → check the coding table if there is any association marked \
          \ corresponding to this flow

          //that have an association with this flow
          → get a packets from each of above flows
          → encode the packets together and forward it to the next hop.
          endif

          if (n ∉ {coding_node}) then
          → route the packet to next node
          endif

        if (coding_status == 1) then      //packet has been already coded, try to decode
          if (n ∈ {decoding_node}) then
          → get packets from other flows corresponding \
          to the coding_table
          → decode the packet and route it to the next hop
          endif

          if (n ∉ {decoding node}) then
          → route the packet to next hop
          endif
        endif
    endif
endif

if (x != new_packet) then //discard the packet to avoid looping
→ discard the packet
endif
```