Chapter 1

INTRODUCTION

1.1 Introduction

There is an increasing need to exchange information easily without using the conventional wired communication. For example, participants may need to exchange contact information during a conference, students may want to download the presentation slides during a lecture, people in a disaster recovery team may need to retrieve and exchange information in order to manage the search and rescue operations, and travelers may wish to exchange data about the weather, and departure and arrival schedules in an airport. In such situations, a mobile ad hoc network (MANET) provides a means to set up a mode of communication easily and quickly.

In a MANET, the mobile nodes may continue to move while information exchange is taking place, therefore, the nodes must adapt to the continuous movement. Allowing mobile nodes to dynamically establish their own network on the fly without requiring any infrastructure-based communication, and supporting quick adaptation and ease of configuration involves a certain amount of overhead.

1.2 Background

Mobile ad hoc networks have been popular because they are very easy to implement without requiring base stations. The network allows nodes to seamlessly communicate in an area with no pre-existing infrastructure. The technology to support the formation of small networks already exists.

The mobility of nodes and the wireless medium have characteristics that are different from the traditional wired network. Therefore, the routing protocols for MANETs are based on different principles. The routing protocols for wired networks are designed to support a large number of static nodes and packets are transmitted over reliable links. On the contrary, the size of a MANET may be small with a few nodes, but the network topology may be very dynamic and changes constantly, and packets are transmitted over unreliable wireless links that are more prone to errors.

There are a number of challenges that must be overcome in order to deliver information efficiently in a MANET over unreliable links between a dynamic set of mobile nodes, such as limited wireless transmission range, limited bandwidth, battery power constraint, mobility-induced route changes, packet losses due to wireless transmission errors, and misinterpretation of congestion.

In a MANET, as the packet loss rate increases due to node mobility, the packet delivery fraction (i.e. how many packets a destination node successfully receives) and throughput decrease. Such packet losses and performance degradation occur continuously due to the inefficiency of the wireless medium and the weakness of the routing and transport protocols.

In MANETs, routing protocols perform differently depending on their core mechanisms (Liu and Kaiser, 2003). Proactive protocols discover all possible routes to the destination before the actual transmission. All nodes exchange route information periodically and maintain complete up-to-date network information.

On the other end of the spectrum are reactive protocols that discover routes on-demand, i.e. only when a route is needed to communicate. The nodes do not depend on periodic route exchanges. A route rediscovery is triggered whenever the source node receives a link failure signal.

A hybrid approach combines the proactive and reactive mechanisms, e.g. maintaining routing metrics to determine the best routes based on the distance vector and updating route information when topology changes occur.

Proactive routing protocols are not as efficient as reactive routing protocols in a large-scaled network because it carries all routing information together with data packets. On the other hand, reactive routing protocols cannot efficiently handle route rediscovery in a high mobility environment. Therefore, an efficient layer protocol (modification of each layer protocol) and cross-layer protocol (modification of layer to layer protocols) architectures (Jurdak, 2007) are required to optimize the throughput, delay, overhead and energy consumption of the network.

Reactive routing protocols that discover routes on-demand increase the routing overhead and delay. Because reactive routing protocols, e.g. Ad-hoc On-demand Distance Vector Routing Protocol (AODV) (Perkins and Das, 2003), discover a single path between a source and destination pair, it tends to increase route discovery frequency and overhead whenever a route failure occurs.

On the other hand, a multipath routing protocol, e.g. Ad-hoc On-demand Multipath Distance Vector (AOMDV) routing protocol (Marina and Das, 2006), that discovers multiple alternative paths between a source and destination pair incurs not only an increased processing overhead and memory usage, but also higher maintenance of unusable routes. Maintaining multiple routes also consumes scarce resources, such as bandwidth, memory usage and power consumption.

The transport layer protocols, such as Transmission Control Protocol (TCP)[1] (Postel, 1981) and User Datagram Protocol (UDP)[2] (Postel, 1980), do not perform well over

[1] TCP is a transport layer protocol that can be utilized in file transfer applications.
[2] UDP is a transport layer protocol that can be utilized in voice and audio data transmission.

the wireless medium, especially TCP. TCP’s assumption that packet losses are caused by a congestion tends to decrease performance because most packet losses in a MANET are due to route failures. Moreover, TCP uses the end-to-end acknowledgement (ACK) scheme to ensure the reliability of packets, which tends to decrease throughput when the end-to-end ACK packet losses happen due to the error-prone wireless medium. On the other hand, if an acknowledgement is sent by every intermediate node, as done in hop-by-hop transport protocols (Scofield, 2007) and (Heimlicher et al, 2007), it would consume the limited bandwidth and transmission power.

Finding intermediate nodes to deliver packets to the destinations is a primary routing problem in MANET. Each node in the network must have a routing table that maintains routing information to determine the next hop to forward the packet to the intended destination. The routing protocol updates information in the routing table of each node regarding tenable network topology changes. If the routing table becomes inconsistent, a routing loop may occur, and thus, packets may continuously be lost. It becomes harder to maintain the correct routing table information in a large network with hundreds or more nodes.

1.3 Problem Statement

The challenge is to design a routing protocol that performs efficiently in a dynamic environment where nodes may be stationary or mobile. It is sometimes impossible to know what environment the protocol will discover itself in because the environment may change unexpectedly and rapidly. Therefore, the routing protocol must be able to adapt to route changes quickly in order to provide continuous transmission.

Most routing protocols use periodic control messages to detect the changing network topologies, which tends to increase the routing overhead as the network topology changes happen more often, thus, increasing the network load.

In this thesis, the proposed routing protocol inherits the basic route discovery procedure of AODV to overcome the following problems:

- Increased routing overhead and end-to-end delay due to the inability to distinguish between route breaks and collisions
- Degraded throughput due to increased node speed and density for the large-scaled networks
- Invoking the route discovery procedure for every route error

It is insufficient for TCP to use only the end-to-end data reliability checking for fast recovery and delivery of data packets. Therefore, we propose that the reliability of TCP packets is ensured by developing a mechanism that addresses the following problems:

- The inefficiency of end-to-end checking of TCP packets
- Increased processing overhead and delay due to hop-by-hop checking of TCP packets

1.4 Objectives of Study

The objectives of this study are:

- To design and develop a proxy-assisted routing protocol for efficient data transmission where the broadcasting zone is defined with the assistance of a proxy node to reduce the routing overhead, delay and collision at the MAC layer due to the request packet broadcast.

- To monitor the TCP packets at the proxy node and inform the source node of missing packets based on missing sequence numbers to ensure the reliability and fast recovery of TCP.

1.5 Hypotheses

It is expected that the proposed protocol is able to:

- support reliable data transmission with the aid of a proxy node that takes the responsibility for redirecting to a new route as soon as possible, resulting in increased throughput and decreased delay.
- limit the broadcast zone by considering the distance to the proxy node, resulting in reduced routing overhead.
- improve performance compared to other protocols when tested in different scenarios, e.g. varying node topologies, node speed, node density and mobility models.
- increase the reliability and throughput of TCP packets by allowing the proxy node to send a local acknowledgement.

1.6 Contributions of Study

In MANETs, 70% of route errors are due to collisions at the MAC layer. Instead of invoking route discovery procedure for all route errors, our proposed protocol only invokes the route discovery procedure for route breaks that are due to node movements by defining a broadcasting zone with the assistance of a proxy node, resulting in lower routing overhead, especially in the large scaled networks.

In MANETs, it is very important to ensure the reliability of TCP packets. TCP uses the end-to-end checking mechanism at every sender and receiver. As the path length is longer and the number of collisions occurs more often, this basic end-to-end mechanism becomes insufficient.

In this study, a proxy node is responsible for monitoring the TCP sequence number. When the proxy node detects any missing sequence numbers, it sends a local acknowledgement to the source node in advance of any end-to-end acknowledgements so that the source node can retransmit the missing packets. By doing so, the proposed mechanism provides fast retransmission of lost packets, resulting in increased throughput and reliability.

1.7 Overview of Chapters

The rest of the thesis is organized as follows. Chapter 2 is a review of works done by the previous researchers that attempted to solve the routing layer problems for MANETs. Chapter 3 reviews previously proposed transport layer protocols to enhance the performance of protocols for MANETs. The problems and solutions of previous researchers for the transport and routing layers are discussed, followed by a list of the problems they address.

Chapter 4 starts by listing the requirements that must be met by the PART protocol in order to achieve the objectives listed in Section 1.4. It explains the proposed design framework and implementation of the proposed routing layer protocol, followed by an analysis of the experiment results.

Chapter 5 discusses the enhancement between the transport, routing and MAC layers. The proxy local acknowledgement scheme is explained followed by a discussion of the experiments and an analysis of the results. The experiments in mobile environment in Chapters 4 and 5 are carried out using random node movements.

Chapter 6 starts with an overview of mobility models followed by a discussion of the experiments to determine the effectiveness of the proposed protocols when the mobility models are applied.

Lastly, Chapter 7 summarizes the results obtained and discusses to what extent the objectives listed in this chapter have been achieved. This is followed by a discussion on the significance contribution of this research. We conclude with an outline of future work.

Chapter 2

LITERATURE REVIEW FOR ROUTING LAYER PROTOCOLS

2.1 Introduction

This chapter gives an overview of MANET architecture, with detailed descriptions of the problem statement and enhancement of protocols. We propose our concepts to transmit data efficiently in different scenarios of MANETs. The first section provides the key issues of layer and cross layer architectures that are essential to design the protocols in the MANETs. The second section discusses the traditional routing protocols, such as conventional and ad hoc routing protocols. The third section elaborates on the basic AODV (Perkins and Das, 2003) routing protocol and the fourth section presents the layer architecture (i.e. routing layer) enhancement of AODV, such as the reduction of the route discovery frequencies, the addition of multiple paths and multicast techniques, to enhance the performance of the MANETs. The fifth section explains the cross layer enhancements between AODV and other layers to achieve a better performance. Finally, this chapter closes with a conclusion.

2.2 MANET Architecture of 5-layer Reference Model

2.2.1 Layer Architecture

The enhancements on the MANET protocols have been done by modifying a single layer, which is called layer enhancement. For example, the enhancements of the data inefficiency problems are performed at the transport layer, the route instability problems at the network layer, the hidden and exposed terminal problems at the MAC layer, and signal interference problems at the physical layer. Figure 2.1 shows the layer architecture for MANETs and the explanations of each layer are described in the following sub-sections.

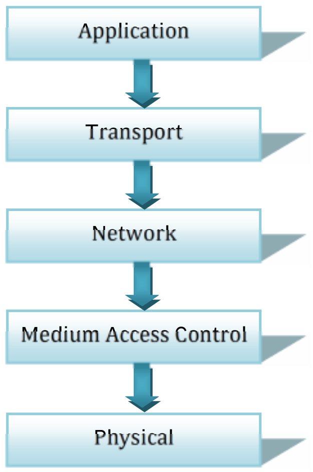


Figure 2.1 : Layer architecture for MANETs

2.2.1.1 Application Layer Issues

The application layer uses the services at the transport layer and supports higher-level protocols. The file transfer and email download applications normally not only require reliable data delivery and high throughput but also tolerate round trip delay and packet jitter. Interactive applications, such as web browsing and remote terminal access, must have a lower delay. TCP is a highly suitable protocol for such applications. For streaming applications, such as video and audio, playback files need to be accommodated without waiting for the entire download to complete. However, such applications only require on-time delivery rather than reliability. A conversational traffic, such as voice over IP (VoIP), requires low latency and delay. UDP is a highly suitable protocol for such applications. Therefore, the application layer protocols need to be designed to handle frequent disconnection and reconnection with peer applications.

2.2.1.2 Transport Layer Issues

The main objective is to transport messages successfully from the source to the destination. TCP (Transport Layer Protocol) (Postel, 1981) and UDP (User Datagram

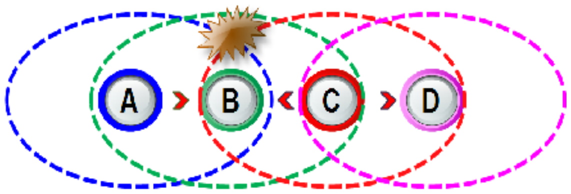
Protocol) (Postel, 1980) are two well-known protocols to meet this goal. TCP ensures end-to-end data delivery and reliability, whereas UDP supports an unreliable connectionless transport. As TCP was initially designed for wired communications, its performance may degrade in wireless networks. In general, control information is embedded in the messages to support flow and error controls. A long message may need to be broken down into shorter messages, called segments, which is a term used to refer to packets at the transport layer. To use TCP efficiently in wireless networks, many enhancements of TCP have been proposed.

2.2.1.3 Network Layer Issues

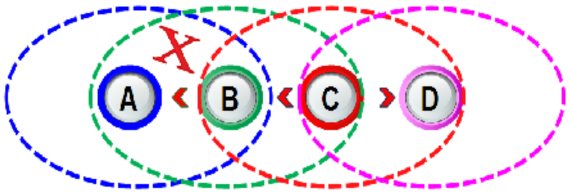
The main objective of the network layer is to deliver the data packets to the transport layer. In wireless networks, due to the mobility of nodes and the lack of infrastructure, a route that is believed to be optimal at a given point in time might not work at all a few moments later. Therefore, it is important that routing protocols adapt the failed routes, detect neighbor nodes and update routing tables efficiently after a route failure. The main goal of a routing strategy is to provide efficient routes quickly as soon as the route failures are detected.

2.2.1.4 MAC Layer Issues

To route the packets through numerous communication channels, a mechanism is required for node-to-node delivery. The responsibility of a MAC or link layer protocol is to detect whether a channel is available to send packets across a communication link. The nature of the wireless channel gives rise to the problems of packet collision or channel long idle due to hidden nodes in the wireless environments. The hidden terminal and exposed terminal problems (Chane et al., 1997) are well-known, especially for the MANETs. Let us consider the hidden terminal problem in Figure 2.2(a). Node B is within the transmission range of nodes A and C, but nodes A and C cannot hear each other. While node A is transmitting to node B, node C also transmits to node B because it cannot detect the transmission between node A and node B, thus causing a packet collision at node B.



(a) Hidden terminal problem



(b) Exposed terminal problem

Figure 2.2 : Illustration of MAC layer problems

Let us consider the exposed terminal problem in Figure 2.2(b). Node B is transmitting to node A. Since node C is within node B's range, it senses the medium using the carrier sense mechanism and decides to defer its own transmission. However, this is an unacceptable situation because there could be no collision at node A. Therefore, the MAC layer protocols must be able to detect the occurrence of collisions, partially support reliable data transport over the shared wireless medium and prevent the hidden or exposed terminal problems. Examples of the most useful link layer protocol are Ethernet, IEEE 802.11, Point-to-Point protocol and Asynchronous Transfer Mode (ATM).

2.2.1.5 Physical Layer Issues

The physical layer is responsible for transmitting packets across a communication link. Its main purpose is to ensure that the transmission parameters, such as a wireless channel, propagation model, antenna and signal power, are set appropriately to achieve low bit error rate. The MANETs inherit the conventional problems of wireless communication and networking. The absence of wires between a source and destination renders the transmitted signal much more susceptible to interferences and background noises. Moreover, the broadcast nature of MANET increases signal flooding. Therefore, the physical layer must resist the interference of outside signal and be designed for robustness.

Finally, the five layers discussed above are common to the Open Systems Interconnection (OSI) layer. There are seven OSI layers and the other two layers, namely session and presentation layers sit on top of the transport layer. In MANETs, each layer protocol is created to fulfill the design goal.

2.2.2 Cross-layer Architecture

In the cross layer architecture, protocols seek to provide the upper layers and lower layers with a reliable communication. The physical layer could adapt transmission power, transmission rate and coding to meet application requirements. The MAC layer must quickly inform the routing layer of link breaks so that the routing layer can discover another route as soon as possible. The transport layer protocols use the assistance of the routing layer or the MAC layer to distinguish between packet losses due to congestion and route breaks. As shown in Figure 2.3, for the enhancement of each layer, the assistance of other layers is taken by operating together with transport and network, transport and MAC, transport and physical, network and MAC, etc.

On the other hand, Conti et al. (2004) pointed out that the cross-layer designs can produce unintended interactions among protocols, such as adaptation loops, that result in performance degradation. Moreover, the cross layer architecture can produce spaghetti-like codes that may not be efficient because every modification must be propagated across all protocols.

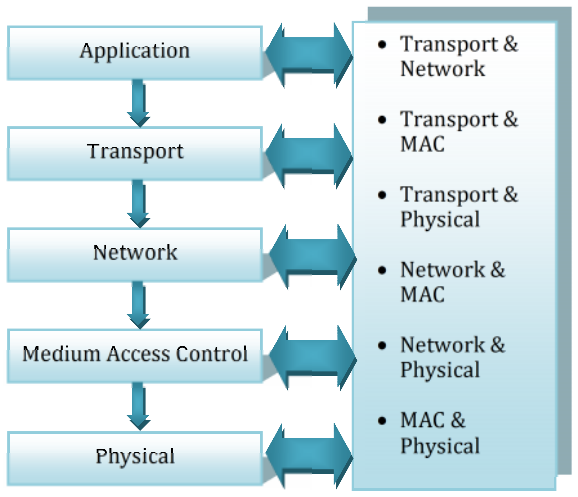


Figure 2.3 : Cross layer architecture for MANETs

The next section gives an overview of ad hoc routing protocols and the basic implementation of AODV. Later, the enhancements of layer architecture of AODV and cross layer enhancements of AODV between the routing and other layers are reviewed.

2.3 Overview of Basic Routing Protocols

In MANETs, providing efficient routes is one of the most critical challenges for the routing operation. Due to the lack of infrastructure support, MANETs require the assistance of intermediate nodes to send data packets to the destinations successfully. The role of intermediate nodes is very essential to route data packets successfully. The movement of nodes also affects the overall network performance. Therefore, routing protocols are needed to adapt to route changes and mobility.

2.3.1 Conventional Routing Protocols

Conventional routing protocols, such as the distance vector and link state, are invented based on the static topology of the wired networks. The protocols work fine in very low mobility scenarios for MANETs. However, the nodes in MANETs may move freely at various speeds and directions, thus they are difficult to control with the conventional routing protocols. Many ad hoc routing protocols are proposed to handle the network topology changes very well under such scenarios. The basic operations of conventional protocols are explained briefly as follows.

a) Distance Vector Routing

Instead of broadcasting route information to all nodes in the network, the distance vector routing (Tanenbaum, 1996) only broadcasts an estimated shortest distance to the neighbors by monitoring the costs of the outgoing links. The nodes that receive this information update their routing tables using the shortest path algorithm.

b) Link State Routing

Each node in the network maintains a complete view of the topology with a cost for each link and broadcasts the information to all other nodes periodically to keep consistent routes (Tanenbaum, 1996). Based on this information, each node in the network chooses the next hop using a shortest path algorithm for the intended destination.

c) Source Routing

Each packet carries the complete path to the destination. A source node determines the route that is used to transmit data packets. Source routing (Perlman, 1992) avoids the formation of routing loops. It may have an additional overhead for each packet while transmitting packets.

2.3.2 MANET Routing Protocols

Now that the basics of conventional routing protocols have been explained, in this section, we move on to explain the basics of MANET routing. There are three categories of routing protocols based on the route discovery and route update mechanism: table driven (proactive), on-demand (reactive) and a hybrid of proactive and reactive protocols.

2.3.2.1 Proactive Routing Protocols

Proactive routing protocols attempt to maintain up-to-date routes by broadcasting control messages throughout the network periodically. These messages incur a higher overhead and may contribute to congestion. Well-known examples of proactive routing protocols are destination-sequence distance vector (DSDV) (Perkins and Watson, 1994) and optimized link state routing (OLSR) (Clausen and Jacquest, 2003).

DSDV is a proactive, hop-by-hop distance vector routing protocol. In DSDV, each node maintains the number of hops to each destination in its routing table. Each node broadcasts its routing table information periodically throughout the network by using monotonically increased sequence numbers. The sequence number not only prevents the occurrence of stale routes, but also avoids the formation of routing loops. If a node does not receive a periodic message from its neighbor for a while, it assumes that the link is broken. The route update algorithm is very simple and guarantees loop free routes by transmitting a smaller update message periodically. Therefore, the entire routing table needs not be transmitted when network topology changes occur.

OLSR is a carefully designed protocol that works in a distributed manner and does not depend on any central entity. Each node chooses its neighbor nodes as multipoint relays (MPR) that are responsible for forwarding control traffic by flooding. MPRs provide the shortest path to a destination by declaring and exchanging the link information periodically for their MPR's selectors. By doing so, the nodes maintain the network topology information. The MPR can reduce the number of nodes that broadcast the routing information throughout the network. To forward data traffic, a node selects its one hop symmetric neighbors, referred to as MPRset that covers two hops away nodes. According to HELLO messages, the information of symmetric neighbors is used to calculate the MPRset. When a node receives a packet, it forwards it if a sender has already selected a MPR node. Otherwise, the node discards it.

For route maintenance, periodic hello messages are used for link sensing, neighbor detection and MPR selection process. The link sensing indicates whether it is symmetric, asymmetric or lost. The neighbor detection indicates either symmetric, MPR or not a neighbor. If the link to the neighbor is symmetric, it is chosen as MPR. After receiving a HELLO message, a node builds its routing table. When a duplicate packet is received with the same sequence number, it is discarded. A routing table is updated either when a neighbor has changed or a route to the destination has expired.

2.3.2.2 Reactive Routing Protocols

Reactive routing protocols establish routes only when they are needed. When a source node requires to transfer packets to a destination, it initiates a route discovery procedure by broadcasting a route request (RREQ) packet. Once a destination node receives an RREQ, it sends a unicast route reply (RREP) packet. To update a routing table, on-demand routing protocols need not use periodic control messages.

Depending on the RREQ and RREP packets, nodes in the network receive up-to-date route information. The drawback is that the reactive routing protocols increase route discovery frequencies whenever a route break occurs. Dynamic source routing (DSR) (Johnson et al., 2007) and Ad-hoc on-demand distance vector (AODV) are well-known examples of reactive routing protocols. Although both protocols share the same on-demand behavior, the natures of the protocols are different from each other.

In DSR, each node is initialized by broadcasting a route request packet when there is no route available in its route cache. Upon receiving this request, each node broadcasts it by attaching its address and forwards the request packet to reach the destination. The destination node replies to the earliest request to the source node. This approach is known as a source routing. Each node not only quickly supports a route when a route break occurs but also tolerates the topological changes due to the monitoring of route discovery. For route maintenance, each node always monitors the links to forward the packets. If a node cannot forward a packet, it sends a route error packet to its upstream nodes towards the source.

AODV is based on DSDV and DSR routing protocols. Each node has a routing table that maintains information about destinations, such as next hop and hop count (i.e. the distance from the current node to the destination node). AODV also uses a sequence number generated by a destination node to indicate fresh-enough routes. Unlike DSR, it only counts the number of hops. It builds the reversed routes to the source node by looking at the information of the route request packets.

The responsibility of intermediate nodes is to check for fresh routes according to the hop count and destination sequence number, and forwards the packets that they receive from their neighbors to the respective destinations. AODV utilizes HELLO packets for route maintenance. If a node does not receive a HELLO packet within a certain time, or it receives a route break signal that is reported by the link layer, it sends a route error packet by either unicast or broadcast, depending on the precursor lists (i.e. active nodes towards the destination) in its routing table. AODV inherits the periodic broadcasting and sequence numbering techniques of DSDV.

Although a route discovery procedure of AODV and DSR is similar, there are a few differences. Each DSR packet carries the complete routing information, whereas AODV packet carries the destination address only. On the other hand, DSR's route reply packet carries all addresses of nodes along a path, whereas AODV's route reply packet carries only the destination address and sequence number. AODV avoids the stale route cache problem of DSR, and it adapts the topology changes quickly by resuming route discovery procedure from the very beginning.

2.3.2.3 Hybrid Routing Protocols

A hybrid routing protocol (Pearlman and Samar, 2002) combines the features of proactive and reactive routing algorithms. Hybrid protocols divide the network into areas called zones. The proactive routing protocols work within the inner zone, and the reactive routing protocols work in the outer zone. These zones may be overlapping or non-overlapping depending on the zone management algorithm.

The responsibility of proactive and reactive protocols is to establish and maintain routes to the destinations located inside or outside zones. The zone-based routing protocol (ZRP) (Pearlman and Samar, 2002) and sharp hybrid routing protocol (SHARP) (Ramasubramanian, 2003) are examples of hybrid routing protocols.

Our research inherits the basic route discovery procedure of AODV because of its quick adaption. Therefore, the brief functions of AODV and its enhancements related to our work are described. Route changes occur frequently due to the movement of nodes, network congestion and contention. Discovering an efficient route between a pair of mobile nodes is an essential operation in order to transfer data packets to a destination successfully. Moreover, establishing a route requires some exchange of control packets that can be quite high in MANETs due to the rapid topology changes.

AODV is selected as a basic protocol because it can quickly adapt to route changes whenever a route break occurs by using its route discovery procedure. Although the network-wide broadcasting of AODV causes large overhead, AODV reduces the delay and routing overhead when node mobility increases.

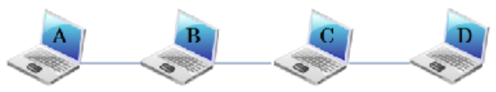
2.4 AODV Routing Protocol

This section discusses the details of AODV, such as message formats, routing table, route discovery procedure and so on.

2.4.1 AODV Basic

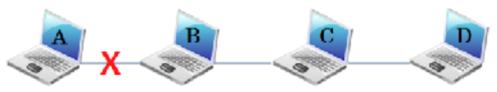
AODV is a distance vector-based protocol. Initial distance vector protocols suffer from a problem called count-to-infinity (Hedrick, 1988) due to incomplete advertised information. This problem is described below.

Initially, each node has routes to the other nodes, and the link cost is "1". In Figure 2.4(a), node A has a route to node B with 1 hop count and node B has routes to nodes C and A with 1 hop count. Node C also has a route to node B and node D with 1 hop count, and to node A with 2 hop counts. In Figure 2.4(b), when a link between node A and B breaks, node B is responsible for re-computing a new route. However, node B does not know that a successor of node C to node A is itself, resulting in a count-to-infinity problem. Sooner or later, in Figure 2.4(c), node B knows that node C has a route to node A with 2 hops and sends a data packet to node C. At that time, node B updates its routing table for node A according to the node C's information (i.e. C+1) and sends packets to node C.



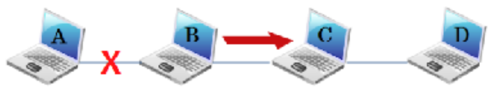
A	B	C	D
1 (B)	1 (A) 1 (C)	2 (A) 1 (B) 1 (D)	3 (A) 2 (B) 1 (C)

(a) Link cost or hop count of initial state



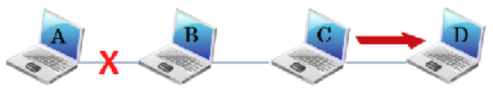
A	B	C	D
1 (B)	1 (A) 1 (C)	2 (A) 1 (B) 1 (D)	3 (A) 2 (B) 1 (C)

(b) Route failure between node A and B



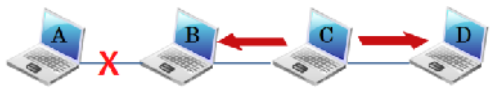
A	B	C	D
1 (B)	3 (A=C+1) 1 (C)	2 (A) 1 (B) 1 (D)	3 (A) 2 (B) 1 (C)

(c) Node B tries to connect to A via C



A	B	C	D
1 (B)	3 (A=C+1) 1 (C)	4 (A=D+1) 1 (B) 1 (D)	3 (A) 2 (B) 1 (C)

(d) Node C tries to connect to A via D



A	B	C	D
1 (B)	5 (A=C+1) 1 (C)	4 (A=D+1) 1 (B) 1 (D)	5 (A=C+1) 2 (B) 1 (C)

(e) Node C tries to send packet to B and D

Figure 2.4 : Count-to-infinity problem of traditional routing protocols

When node C receives the packets, it counts the shortest path to node A and finds that it must go through node D. Thus, in Figure 2.4(d), node C updates its routing table according to the node D's information. In Figure 2.4(e), node C then sends the packet to node B and node D. All nodes suppose that all neighbors have a shortest path to the node A and update their routing tables. The counting situation for node A never stops until there is no allocated memory for the routes.

The simple split horizon scheme is the first attempt to solve this problem. It omits routes learned from one neighbor in updates sent to that neighbor as well as the split horizon with poisoned reverse that includes such routes in updates, but sets their metrics to infinity (Hedrick, 1988). With poison reverse, when a node detects an unreachable route, it removes the route immediately from the routing table. In this way, looping events can be avoided. However, this scheme is not good enough to establish a typical wireless network. Therefore, a distance vector-based protocol called AODV was proposed to build an efficient routing protocol for the MANETs.

AODV was designed to avoid the following issues.

- Huge amount of control overheads
- Huge amount of processing overheads
- Formation of loops

AODV attempts to minimize the overhead by utilizing only on demand messaging instead of sending route updates periodically. Consequently, AODV messages are simple to compute, with low processing overheads. By utilizing a destination sequence number, AODV stringently prevents the formation of routing loops. The features of AODV and its fundamental components are described in the following sections.

2.4.2 Destination Sequence Number

AODV utilizes a technique based on destination sequence numbers to ensure all discovered paths are loop-free. The destination sequence number is updated each time a node receives a new sequence number from the RREQ, RREP, or RRER messages related to a destination. Before a node initiates a new RREQ, it increases its destination sequence number to avoid conflicts with previously established reverse routes. Also, when a node receives a route request, it updates its destination sequence number by taking the maximum value of its current destination sequence number and RREQ's destination sequence number before sending out a reply packet. In this way, a node maintains recent route information of a destination by utilizing destination sequence number technique.

In order to avoid the stale route information, the node always compares its current sequence number with the newly received AODV's sequence number. If the value of the sequence number in AODV message is greater than its current value, the information relating to the destination is considered stale and discarded from the routing table. Utilizing sequence number technique not only prevents routing loops and the initial count-to-infinity problem (Hedrick, 1988) but also ensures the selection of fresh enough route to the destination.

2.4.3 Message Formats

AODV defines three types of messages, namely Route Request (RREQ), Route Reply (RREP), and Route Error (RERR). The message formats of AODV are described in Figure 2.5. The value of the type field determines the packet type (i.e. type 1:RREQ, type 2:RREP, type 3:RERR). A routing agent determines the packet type by looking at this value in the message.

Figure 2.5(a) shows the fields in an RREQ message. *Flags J (join)* and *R (Repair)* are used for multicast transmission. *G (Gratuitous)* is used together with RREP for unicast transmission to the node specified in the destination address field in the message. *D (Destination only)* is set to 1 when only the destination needs to respond to the RREQ message. *U (Unknown)* is set to 1 when the destination sequence number is unknown. *Hop count* is the number of hops from an originator to the node handling the request in the network. *RREQ ID* is a sequence number that is identified as a particular RREQ when taken in conjunction with the originator's address. The Destination IP address is the address of the destination node that is supposed to receive data packets from a source node. The originator IP address is the address of the node that originates the packet transmission.

Bits: 8	5	11	8
Type = 1	J R G D U	Reserved	Hop Count
RREQ ID			
Destination IP Address			
Destination Sequence Number			
Originator IP Address			
Originator Sequence Number			
Expire time			

(a) RREQ message format.

Bits: 8	2	9	4	8
Type = 2	R A	Reserved	Prefix Sz	Hop Count
Destination IP Address				
Destination Sequence Number				
Originator IP Address				
Lifetime				

(b) RREP message format.

Bits: 8	1	15	8
Type = 3	N	Reserved	Dest Count
Unreachable Destination IP Address(1)			
Unreachable Destination Sequence Number(1)			
Additional Unreachable Destination IP Address(if needed)			
Additional Unreachable Destination Sequence Number(if needed)			

(c) RERR message format.

Figure 2.5: AODV message formats

When a node has data packets to transfer to a destination, it adds the destination address in the d*estination IP address field* of RREQ and its sequence number in *a destination sequence number* of RREQ packet. The destination sequence number is the latest sequence number previously received by the originator for any route towards the destination. It also adds its address in the *originator IP address* field and its sequence number in the *originator sequence number* field of the RREQ. The originator sequence number is used in the route entry pointing toward the originator of the RREQ. The e*xpired time* field indicates when the RREQ packets are going to expire due to timeout event.

In an RREP message, the *R* (*Repair*) flag is used for multicast. An *A* (*Acknowledgement*) flag is set when an acknowledgement is required as shown in Figure 2.5(b). If a node forwards an RREP over an unstable or unidirectional link, the node sets the “A” flag to 1 to indicate to the recipient of the RREP that an acknowledgement is required. The *Prefix size* (i.e. nonzero value) means that the indicated next hop can be used for any node with the same routing prefix as the requested information. *Lifetime (milliseconds)* is the time for which the nodes receiving the RREP consider the route to be valid.

The *N (No delete)* flag is reserved for a local repair procedure. The *DestCount* is the number of unreachable destinations. The *Unreachable destination IP address* is the IP address of the unreachable destination address. *Unreachable destination Sequence Number* is the sequence number for the previous unreachable destination address field as shown in Figure 2.5(c).

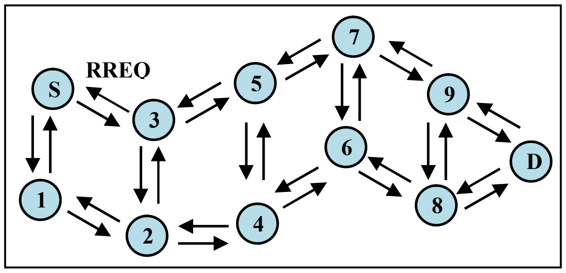
2.4.4 Routing Table

AODV is a hop-by-hop routing protocol, where a routing table keeps all reachable destinations and the address of the next node towards the destination. Each node keeps track of how to deliver data packets based on information in its routing tables that stores the shortest paths or all available paths. An entry in a routing table contains the following information.

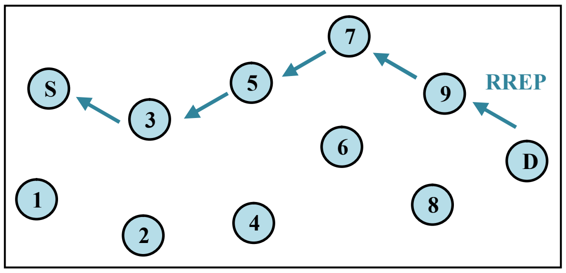
- *Destination IP address*
- *Destination sequence number*: The latest sequence number for the destination
- *Hop Count*: The last known hop count to the destination
- *Next Hop*: The node to forward a data packet in order to reach the destination
- *Lifetime*: How long a route remains valid
- *List of Precursors:* The list of neighboring nodes that are likely to use them as next hops toward a destination
- *Valid Destination sequence number flag*

2.4.5 Route Discovery Procedure

If a node has data packets to send to a destination, it first determines whether there is a route in its routing table for the destination. If it has, it uses that route to send the packets. Otherwise, it requests a route by broadcasting RREQ packets.



(a) RREQ broadcast



(b) RREP unicast

Figure 2.6 : Route discovery procedure of AODV

In Figure 2.6(a), node S has data packets to transmit. It broadcasts an RREQ packet by adding a destination IP address and the last known sequence number, its IP address and current sequence number. A hop count is then initialized to zero. If the source node receives the RREQ packet that it sends, it discards it.

When a neighbor node receives the RREQ packet, it creates a reversed route entry for both the source node and the node from which it receives the request. Then it increases the hop count value and responds to the RREQ packet. The node sends a reply to the request if it is the destination, or if it has a valid route to the destination.

In Figure 2.6(b), when node D receives the RREQ, it sends an RREP to the source node. When other intermediate nodes (i.e. node 1 to node 9) receive the RREQ, they compare their addresses to the destination address of RREQ and forward it to their neighbors if the addresses do not match. Before the destination node, node D, sends the RREP packet, it increases its sequence number by taking MAX ($Dseq_{RTE}$> $Dseq_{RREQ}$) + 1. Then the RREP packet is unicast by adding the IP address of the destination (i.e. node S's address), its record destination sequence number and hop count that is set to zero.

When the next hop (i.e. node 9) receives the RREP, it first increases the hop count value in RREP and compares its address to the IP destination address in RREP packet. If it does not match, it forwards the RREP packet to its next hop that is shown in its routing table entry.

As soon as the source node receives the RREP, it starts transmitting data packets. If the source node receives multiple RREPs, it selects the earliest and shortest route with the greatest sequence number. On the other hand, each node always checks the RREQ messages it receives and discards duplicates received from multiple neighbors.

Referring to Figure 2.6(a), node 3 would discard RREQs it receives from nodes 2 and 5 after receiving the original RREQ from the source node S.

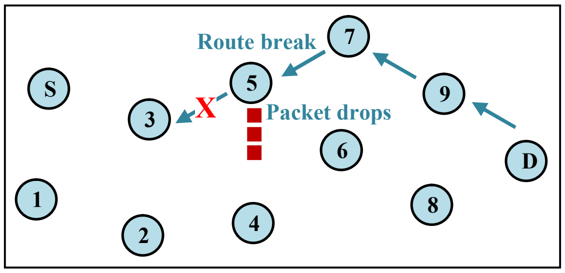
2.4.6 Route Maintenance

Link breaks occur due to node mobility and the temporary nature of wireless channels. Nodes always monitor the condition of links to the next hops in active routes that have recently been utilized for data packet transmission. If a node detects a link failure in an active route, the node upstream of the break invalidates all unreachable destinations in its routing table and sends an RERR packet to notify other nodes about the link failure. The RERR message is sent by either unicast or broadcast, depending on the precursor lists in the routing table of a node that detects the link breaks.

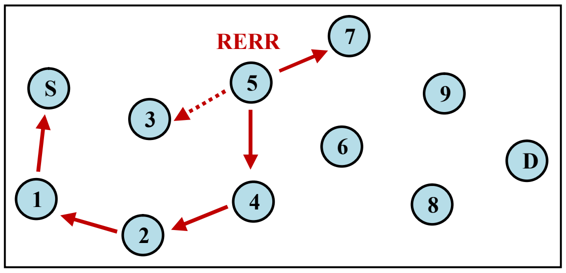
In Figure 2.7(a), when node 5 could not connect to node S due to a link break, or it encounters packet drops due to the CBK problem (DROP_RTR_MAC_CALLBACK)[1], node 5 broadcasts RERR packets to its neighbors (i.e. node 3, 4, 6 and 7). The nodes that receive the RERR forward it to the source node if their routing tables have route information of a source node as shown in Figure 2.7(b).

Once the source node S receives the RERR packet, it can re-invoke the route discovery procedure if it still requires the route. Here, the CBK problem arises due to the contention at the MAC layer, not because of route failure. If the packet drops are due to a link break, node 3 may not forward the RERR packet. In this situation, node 4 forwards the RERR packet to reach the source node.

[1] NS-2 new trace format explanation: http://www.isi.edu/nsnam/ns/doc/node187.html. CBK problem is due to the MAC layer packet collision. When nodes compete to access channel, the hidden terminal problem of MAC layer causes packets to collide with each other.



(a) Route break and packet drops



(b) Sending RERR packets

Figure 2.7 : Route maintenance procedure of AODV

2.5 Layer Approach for the Optimization of AODV

There are many on-demand routing protocols (Jiang et al., 1999; Toh, 1999; Perkins and Das, 2003; Johnson et al., 2007). The most popular, i.e. AODV, has been enhanced for different types of scenarios and networks since the first version of AODV. On demand routing protocols typically use a simplistic form of broadcasting called simple flooding for routing tasks, such as route discovery and topology dissemination. However, this method may lead to unnecessary retransmissions, channel contentions at the MAC layer and packet collisions. Such an occurrence induces the broadcast storm problem (Tseng et al., 2002), which has been proven to degrade network throughput and data delivery latency.

In an effort to reduce this impact, Espes and Mammeri (2007) introduced an expanding search algorithm that reduces the area flooded with RREQ by exploiting spatial locality of nodes. It assumes that if nodes are located physically close to each other, then it is likely to form a route without searching the entire network. However, on-demand routing protocols also have to face a higher routing overhead and delay problems because they invoke a route rediscovery procedure whenever a link becomes unavailable or congested. Therefore, there have been many approaches that enhance the route discovery procedure with the purpose of reducing the route discovery frequencies and delay as shown in Figure 2.8.

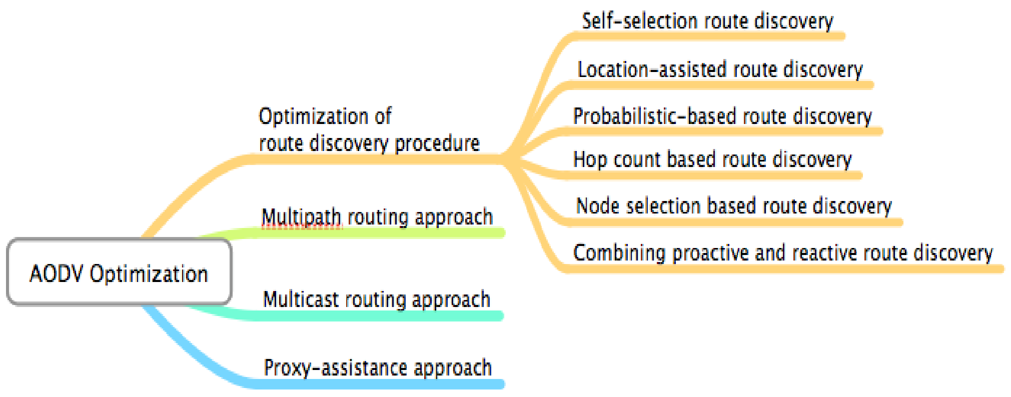

AODV Optimization
Optimization of route discovery procedure
Self-selection route discovery
Location-assisted route discovery
Probabilistic-based route discovery
Hop count based route discovery
Node selection based route discovery
Combining proactive and reactive route discovery
Multipath routing approach
Multicast routing approach
Proxy-assistance approach

Figure 2.8 : Optimization of AODV with the layer approach

AODV has been enhanced with two approaches, such as layer and cross-layer to improve the performance. There are many constraints due to the limitations of the wireless medium. Therefore, sending control messages periodically for routing purposes tends to cause congestion and increase overhead. As a consequence, establishing routes on-demand is becoming mandatory to limit the control messages.

2.5.1 Self-selecting Route Discovery Procedure

Discovering the shortest path by blindly flooding the RREQ packets is not a good solution. To minimize the route discovery overhead due to flooding, Abolhasan and

Lipman (2005) proposed a self-selecting route discovery method, where only nodes that have certain criteria are allowed to forward or rebroadcast RREQ packets.

Two types of self selection strategies are proposed: *Source-Driven* - each intermediate node determines if it should forward an RREQ packet depending on a utility metric specified by the source during a route discovery phase; *Pure Self Selection* - intermediate nodes make a decision independently. Allowing intermediate nodes to actively participate with the route discovery procedure and broadcasting RREQ packets selectively reduce control overhead, channel contention of MAC layer and battery power consumption. Although this approach may reduce the overhead and delay, the impact on throughput was not considered even though throughput is the most important factor to be measured for data transmission.

2.5.2 Location-assisted Route Discovery Procedure

Idrees et al. (2005), Meng et al. (2005), Cha et al. (2007), Giruka and Singhal (2007), Zhao and Zhu (2008) and Asenov and Hnatyshin (2009) enhanced AODV for route discovery latency and throughput improvement with the assistance of Global Positioning System (GPS).

The awareness of the mobility of neighbor nodes and an estimate of the reliable distance that must be smaller than the transmission range can reduce the overall number of control packets traversing the network due to the blind flooding. Looking for the location of nodes using GPS assists to improve the performance of AODV in terms of throughput and overhead. The mobility aware agent proposed by Idrees et al. (2005) improves the throughput of AODV by a factor of up to 1.2. The mobility prediction techniques proposed by Meng et al. (2005) and Cha et al. (2007) reduces the control overhead and delay by a factor of up to 2 compared to AODV. Moreover, OGPR (On-demand Geo-graphic Path-based Routing) and OAODV (Optimized AODV) that was proposed by Giruka and Singhal (2007) and Zhao and Zhu (2008), as well as Geo-AODV (GPS-enhanced AODV) that was proposed by Asenov and Hnatyshin (2009) not only reduces the control overhead and delay, but also increases the throughput and packet delivery ratio by a factor of up to 2 compared to AODV. However, these approaches work only under the GPS coverage area.

2.5.3 Probabilistic-based Route Discovery Procedure

Abdulai et al. (2008) tries to reduce the blind flooding of the RREQ packets using a generic probabilistic method. Taking into account the local density of nodes and setting the probability of forwarding as high in the sparse areas and low in the dense areas, the Fixed Probabilistic Route Discovery with AODV (FPR-AODV) is introduced to reduce routing overhead from 30% to 60% depending on the density of nodes in the MANETs.

Another probabilistic algorithm called Gossip is proposed by Mahesh et al. (2007), where each node forwards a routing packet with some probability; and adds this gossiping to AODV to avoid the flooding of route discovery. Haas et al. (2006) also proposed a similar approach, where packets are forwarded with a defined probability to form a leading set of forwarding nodes to cope with route request flooding problems in the MANETs.

Shi and Shen (2004) proposed the Adaptive Gossip-based Ad Hoc Routing (AGAR) by adding sleep time or a reasonable timeout period. Even though the gossiping algorithm (Haas et al., 2006) reduces the probability of broadcast storms, it may cause only a subset of nodes to forward the packets, resulting in lower throughput.

2.5.4 Hop Count based Route Discovery Procedure

Taking into account hop count to reduce the routing overhead, Spohn and Garcia-Luna-Aceves (2005) introduced a Three-hop Horizon Pruning (THP) algorithm that broadcasts the RREQs based on the current information about the local neighborhood. Depending on the local topology information makes the performance of THP less reliable when the topology changes frequently. Although the simulation results of AODV-THP show significant performance in terms of a PDF, delay and overhead, they do not show the throughput improvement, which is a very important factor.

The Direct Forwarding Routing Protocol (DFRP) proposed by Mohamed and Hassan (2008) forwards RREQ through a double function agent node that has at least two-hop connectivity. Instead of forwarding route request packets throughout the entire network, the source node broadcasts them to only half of the network in order to avoid network congestion and deliver the packet to the destination routes with minimum overhead. Using his approach, performance improvement was achieved only in a very small-scaled network. The consideration of large-scaled networks is an important factor.

An adaptive timeout mechanism based on hop count was proposed by Tamilarasi et al. (2007), namely AAODV (Adaptive AODV), to enhance AODV by assigning the lifetime for the removal of stale routes from the routing tables. When a source node receives a route error packet, the current entries using the failed link are marked as inactive in its routing table. Then bad link time is set by dividing the default bad link time by an average hop count that is calculated according to the route error message. Then the lifetime of the routes is updated using the failed link's bad link life time. The simulation results show that AAODV reduces an average control packet load by 10% and delay by 5% and increases the packet delivery ratio by 2%.

2.5.5 Node Selection based Route Discovery Procedure

By categorizing mobile hosts as normal nodes and selfish nodes, Zhang and Agrawal (2004) proposed a routing protocol with the purpose of reducing the route discovery frequency by allowing only normal nodes to forward packets for other nodes. By using a probabilistic approach like gossiping, a proper number of selfish nodes are set up. Although this approach can reduce the route discovery cost by minimizing the number of broadcast packets, the network may not operate if all nodes are selfish. Therefore, Wang et al. (2005) proposed a method to distinguish selfish peers based on local observations of the AODV routing protocol. Building a statistical description of each neighbor assists to separate the set of neighbor nodes as cooperative and selfish nodes. However, this approach does not consider the mobility of nodes and might increase RREQ drop rates when node movement is introduced.

On the other hand, Zhou et al. (2004) also proposed the Priority Route Discovery Strategy (PRDS) that constructs quality routes to forward RREQ messages. By exploiting a competitive technique, only good quality nodes take part in the construction of routes to prevent unnecessary route discovery and reduce route discovery overhead. Lee and Kim (2006) tried to recover the path by selecting designated candidate nodes from the pre-connected routing nodes. Ramakrishnan and Shanmugavel (2006) tried to repair broken routes with the aid of a virtual node instead of sending the RERR packets to the source node. Enhancements of AODV are able to reduce the routing overhead, but throughput improvement is rarely considered.

2.5.6 Combining Proactive and Reactive Route Discovery Procedure

Proactive routing protocols that keep routes in hand do not need to initiate route discovery once a route break occurs, whereas reactive routing protocols discover routes on demand. Even though the proactive routing protocols support routes immediately, there may have *stale routes*[2] in their route caches, and there is no proper mechanism to effectively detect and remove stale routes. On the other hand, although reactive routing protocols are able to adapt to route changes as soon as possible, they are not as effective as proactive routing protocols in small-scaled networks. Therefore, combining the proactive and reactive approaches is an effective way to enhance the performance of routing layer protocols.

Liu and Lin (2005) proposed a refinement-based routing protocol that takes advantage of both proactive and reactive protocols by selecting the routes proactively and maintaining the routes reactively. Bai and Singhal (2006) introduced DOA (DSR over AODV) routing protocol by dividing the routes as segments through the waypoints and selecting intermediate nodes as waypoints. Waypoint nodes, called a start node and an end node, start communication in a segment with the assistance of intermediate nodes. Intra-segment route repair is invoked when a primary route fails. If it fails, inter-segment or primary route repair is invoked. Only when both of them fail are route error messages sent to a source node.

By applying proactive and reactive approaches separately and concurrently, Mase and Kameyama (2005a) enhanced the Multihop Hello Guided Routing with proactive (MHGR-P) that was proposed by Mase and Kameyama (2004) with Reactive approach (MHGR-R). Each node in MHGR-R diffuses hello message to the entire network when it has data packets for a source and destination with the purpose of reducing the routing overhead of MHGR-P. By combining the concept of proactive and reactive MHGR, Mase and Kameyama (2005b) also proposed a unified MHGR (MHGR-U), where each node proactively originates multihop hello (P-mode) or

[2] A stale route is an expired route in the route cache that is used for routing purposes. It could result in inconsistencies during the route reconstruction phase.

reactively does so (R-mode). Consequently, the P mode is able to create and maintain routes to nodes and the R mode does so reactively. In this way, all these proposals try to reduce the routing overhead of the original AODV by combining the two approaches: proactive and reactive. The mentioned approaches reduce the average delay and overhead by almost 25% in the best case, but the consideration for throughput is lacking in the enhanced protocols. Therefore, in the next section, we describe a different approach that considers multiple paths between the source and destination to improve throughput.

2.5.7 Multipath Routing Approach

Keeping only a single path between a source and destination pair is a big challenge in MANETs because mobile nodes are prone to disconnections. Therefore, more aggressive and adaptable routing strategies are required to be tolerant of dynamic route changes. To be robust to link breaks, it is possible to keep one or more alternative next hops to the destination in routing tables.

Lee and Gerla (2000) introduced a backup routing, called AODV-BR, that only modifies RREP to establish a mesh structure and alternative paths. To obtain an alternative path, a node listens to the neighbors' RREP packets promiscuously. If a node that is not part of the primary route overhears an RREP packet, it records that neighbor as its next hop to the destination in its alternative routing table. If a node that is part of the primary route overhears multiple RREPs, it selects the best route based on the shortest path and inserts it to its alternative routing table. When a primary route is not available, a node performs a one hop data broadcast to its immediate neighbors with backup paths from the alternative routing table. Upon receiving this packet, the neighbors unicast it to their next nodes if their alternative routing tables have an entry for the destination. In this way, AODV-BR prevents the possibility of dropped packets by delivering through one or more alternative routes. Although it reduces packet drop rates, it cannot direct the packets to the destination with less delay. Buffering packets in nodes that are parts of alternative routes increases packet latency when TCP traffic is used. Moreover, it has a limitation that the alternative routes can only be selected within one hop distance.

Chen and Lee (2005) extended the idea of AODV-BR by counting the nodes that are two-hop distance from the primary route. During the route discovery procedure, backup paths with two-hop distance are built while replying to an RREP packet. Lai et al. (2007) also extended AODV-BR to adapt the topology changes by overhearing data packets besides the RREP packets. The alternative paths that only depend on the RREP packets may fail as the node speed increases. AODV-adaptive backup routing (ABR) and AODV adaptive backup with local repair route (ABL) proposed by Lai et al. (2007) can overcome the weakness of AODV-BR.

When a link break is detected, AODV-ABR tries to repair the broken route by handshaking with its immediate neighbors instead of sending one-hop data broadcast that may cause duplicate data packets. For handshake purposes, Backup Route Request (BRRQ) and Backup Route Reply (BRRP) are introduced. When the distance between the broken link and the destination is not farther than the specific hops, AODV-ABL tries to repair the link by sending RREQ packets. Otherwise, it repairs the link using a handshake process. Even though these approaches perform better than AODV-BR, floating extra packets may increase control overhead.

Another way to provide backup paths was introduced by Jiang et al. (2002) with the short-term name Multiple Next Hop (MNH) routing protocol. Unlike the AODV-BR, MNH modifies AODV's RREQ to set up forward paths to the source node. MNH also modifies RREP to add multiple next hops in the routing table for the destination. However, the weakness of MNH is the lack of route update because the multiple routing paths are only constructed at the beginning. As the network topology changes happen more often due to node mobility, the number of used backup paths becomes invalid. The problems with this method are the occurrence of routing loops and the generation of many routing packets.

In order to address the weakness of the above mentioned AODV's variants, Marina and Das (2006) considered the Ad-hoc On-demand Multipath Distance Vector (AOMDV) routing protocol that provides multiple alternative paths at every node. AOMDV computes multiple paths and observes each route advertisement to define an extra path to the intended destination during a route discovery procedure. RREQ packets arriving at the nodes are copied and sent back to the source nodes. This approach may cause the formation of loops due to accepting all copied routes. To eliminate any possibility of routing loops, it uses an advertised hop count field in the routing tables of nodes.

The advertised hop count of a node S for a destination D is set to the maximum hop count of the multiple paths for D at S. The advertised hop count is initialized whenever the sequence number is updated. By doing so, AOMDV only accepts alternative routes with lower hop counts. Each RREQ conveys an additional first hop field to indicate the first neighbor of the source node. The intermediate nodes do not discard duplicate copies of RREQ immediately as long as each RREQ provides a new node-disjoint path to the source. If an intermediate node offers a new path, a reverse path is established by sending an RREP back to the source.

By computing multiple paths in a single route discovery attempt, a new route discovery is needed only when all paths fail. Even though the multiple path technique is good enough to adapt to the broken route quickly with lower overheads and higher throughput, it must use the scarce resources of MANETs, such as bandwidth, memory and battery power for calculating and maintaining multiple paths at every node.

2.5.8 Multicast Routing Approach

In addition to unicast and broadcast transmission, AODV could be extended as multicast through the multicast trees (Royer and Perkins, 1999). In multicast AODV (MAODV), when a node wants to be a part of the multicast group, it broadcasts an RREQ by attaching the multicast group address. Nodes that receive the RREQ set up the reverse routes to the source node and rebroadcast it. As soon as the RREP packets are received, intermediate nodes create the forward routing table entries to the multicast group. Members of the multicast tree can only reply to the source node.

The source node may receive multiple RREPs, thus it explicitly needs to send an activation message along the path. There is a short period to wait for the RREPs. As soon as that period expires, the source node unicasts a Multicast Activation (MACT) message to update the routing tables of nodes to become members of multicast trees.

Royer and Perkins (1999) has described the detail mechanism about the responsibility of the group leader, group merge and multicast tree maintenance. Nodes in MAODV use a Hello message to detect a link break among their neighbors. However, flooding Hello messages continuously is not a good approach in order to reduce routing.

2.5.9 Proxy-assistance Approach

Choi and Das (2002) proposed a proxy-based indirect routing scheme, namely an applicative indirect routing (AIR) that is proactive in nature for link failure and congestion problems. AIR uses proxies to manage unreliable links in the original path.

The rerouting mechanism of AIR directs packets that encounter congestion or broken links to an alternative path. Simulation results show that although AIR achieves performance improvement over one of the proactive routing protocols DSDV, it cannot mimic the reactive routing protocols AODV and DSR, and thus it is just a competitive proactive protocol.

Tiwari (2006) proposed a proxy-AODV routing protocol that appoints the proxy nodes for data transmission over a partially connected ad hoc network. In proxy-AODV, the source nodes select the proxy nodes to hold data on behalf of the destination and then the proxy nodes try to transfer data packets to the destination. In the situation that the source and destination are in different partitions, the farthermost nodes are assumed to be the nearest nodes towards the destination and need to store the same data packets for a destination node. Storing data packets at the proxy nodes may assist data transmission to the desired destination, but may incur not only the unnecessary data packet forwarding, but also network overloading due to the extra proxy requests and replies when the movement of a proxy node is not as expected.

In proxy-AODV, the source node sends an RREQ packet if it wants to establish a route to a destination. In the meantime, the node that is farther enough hop away replies to the source node with a proxy RREP (P-RREP). The source node keeps all P-RREP, and if it does not receive any reply packets from the destination after the timeout period, the source node sends a data packet to nodes that are assumed to reach the destination quickly. If those nodes cannot reach the destination after the timeout period, they discard the data packets. This approach sometimes causes out-of-order packets at a TCP destination. If there is no packet reordering technique at the destination, this may degrade performance due to the acknowledgement technique of TCP.

Boice et al. (2009) also considered the possibility of temporary or long-lived network partitions and proposed the space-content-adaptive-time routing (SCaTR) framework that is able to deliver data even in the situation of connectivity disruptions. In the SCaTR framework, proxies are selected based on past connectivity information. The proxy nodes need to maintain content-adaptive contact tables and update its tables whenever it receives a timely hello packet. After a proxy node has buffered the data packets, it must take the responsibility of sending the packets to the destination during the simulation. If the proxy node no longer wants to act as a proxy, it initiates a route discovery process for buffered packets to find another proxy or destination. As soon as the proxy node knows that its buffer space is going to be low, it selects the first node that has contact with the destination as a proxy node even if this node has a lower contact value. By doing so, data packets can be sent to the destination even if the disruption time is very long in the SCaTR framework.

To reduce message replication and routing overhead, the SCaTR protocol is constrained to selecting at most two proxies for each message. However, the SCaTR protocol does not take part in data transmission as long as the connectivity is still available in the network. SCaTR acts exactly like on-demand routing. Only when source and destination do not have a direct connection does SCaTR choose a node which is closer to the destination as a proxy for that destination and takes part in data transmission.

2.6 Cross-layer Approach for the Optimization of AODV

There are many proposals that attempt to enhance the performance of the MANETs. Most of them modify only one layer, such as the routing layer, transport layer, link layer or physical layer, to reduce the routing overhead, average end-to-end delay, channel contention and collision for overall network performance. The following subsections and Figure 2.9 describe the cross layer concepts between AODV and the other layers.

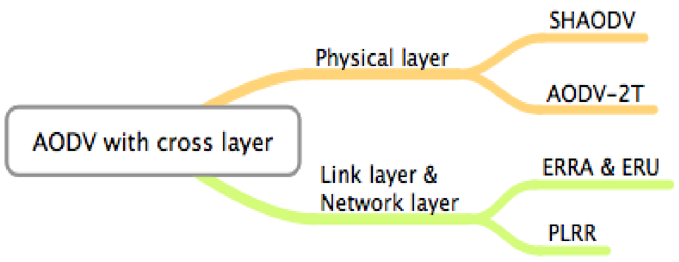


Figure 2.9 : Optimization of AODV with the cross layer approach

2.6.1 SHAODV

Routes in the MANETs can be easily broken by a node along the route that moves out of the transmission range or runs out of battery power. Therefore, taking advantage of the physical layer parameters, such as a signal to noise ratio (SNR) and signal strengths, can assist to improve the performance of the MANETs. By collecting the signal strength data, Feng et al. (2004) proposed a self-healing routing scheme based on AODV (SHAODV) that triggers the edge effect and route break problems occurring at the same time. SHAODV can establish a new route with a long lifetime according to the collected signal strength. However, the weak point of SHAODV is that the MAC device has to be remodeled to collect the SNR for the received data frame when using SHAODV.

2.6.2 AODV-2T

Otakahn and Lertwatechakul (2008) proposed an efficient routing protocol called AODV-2T that uses two level thresholds of battery power and receivable signal power to repair and redirect routes before an actual route break happens. Alternative routes are prepared at the first level threshold check; and connections are directed to the backup path immediately at the critical state of the second level threshold. Simulation results show that AODV-2T reduces routing overhead by 15 to 18 % lower than AODV.

2.6.3 AODV-ERRA & ERU

A new method that exchanges link conditions between link and network layers was proposed by Alsharabi et al. (2005). According to the link condition of IEEE 802.11a, the proposed methods, namely ERRA (Early Route ReArrangement) and ERU (Early Route Update), establish alternative routes by predicting the link stability and link lifetime, resulting in not only prevention of link breaks but also keeping optimized routes. Alsharabi et al. (2005) also consider limiting the necessary signaling overhead to maintain an optimum route. Even though this paper mentions the theory, no comparison was done to other protocols.

2.6.4 AODV-PLRR

AODV-PLRR (Preemptive Local Route Repair) proposed by Crisostomo et al. (2005) avoids route failures by repairing routes preemptively when a link break is about to occur. This approach utilizes HELLO messages to probe the link stability of neighbors and triggers the local repair procedure to find a new route, or a node that has a fresh and stable route to a destination before an actual route break occurs. As Crisostomo et al. (2005) do not consider the locations of the source and destination, the non-optimal route establishments can happen more, especially for large-scaled ad hoc networks.

2.7 Chapter Summary

Previous researchers have focused on solving the following problems.

- Control broadcast storm problems, such as blind flooding
- Route discovery overhead due to broadcast technique
- Route errors due to link breaks, battery power, transmission rate

The approaches taken to solve the above problems are:

- Use the information of other layers to detect route errors
- Learn topology changes using the information from other layers
- Enhancement of route discovery procedure
- Adding multiple paths
- Proposing multicast techniques
- Proxy-based approaches for intermittent connections

While the proposed solutions are able to address the problems of routing layer, there are few outstanding problems as listed below:

- To the best of our knowledge, there are few researches for routing that use one or more proxy nodes in the network, such as SCaTR routing protocol proposed by Boice (2009). Nevertheless, this technique considers the participation of proxies for intermittent and disruptive networks. On the other hand, proxy-AODV proposed by Tiwari (2006) considered a risky condition by assuming that the farthermost nodes as the nearest nodes towards the destination and this assumption may not be right at all time. Therefore, in this thesis, we consider the involvement of a proxy node for not only the intermittent connections, but also the longer distance or hop count. If the destination node knows that the distance to the source node is more than the predefined value, it considers using a proxy node. The responsibility of a proxy node is to reduce the unnecessary broadcasting route discovery latency by repairing routes between a source and a proxy; or a proxy and a destination locally, resulting in not only a lower overhead and delay but also a better throughput.

- In MANETs, route errors happen more often due to route failures or collisions at the MAC layer. On-demand routing protocols invoke the route discovery procedure for all route errors, resulting in the worst effect on routing overhead, delay and throughput. Although there are many solutions to solve the route discovery latency, so far no proper mechanism is proposed yet to react to the route errors that happen due to collisions at the MAC layer or route errors with mobility. In this thesis, when a proxy node receives a route error packet, it tries to repair the route locally if this error is due to the collision at the MAC layer instead of sending the route error packets to the source node and broadcasting the request packets throughout the whole network. If this error is due to the route failure, in other words, ‘no route available’, either the source node or proxy node broadcasts the request packets within a limited zone to reduce the routing overhead.

As a summary, in this thesis, we propose a Proxy-Assisted Routing for efficient data Transmission (PART) protocol that considers the participation of a proxy node for a longer distance that is higher than the pre-defined value, according to the hop count information. The justification for our proposed solution is discussed in Chapter 4.

Chapter 3

LITERATURE REVIEW FOR TRANSPORT LAYER PROTOCOLS

3.1 Introduction

The Transmission Control Protocol (TCP) (Postel, 1981) is the *de facto* standard for the reliability of end-to-end data delivery in the wired networks. Normally, TCP is an independent protocol that is not related to the underlying network technology. TCP's reliability depends on the retransmission of lost packets. Packet retransmission technique is used if the sender receives duplicate acknowledgments or no acknowledgment before a timeout period.

Packet losses occur frequently as the amount of traffic to access Internet applications, such as World Wide Web (WWW), e-mail, multimedia and file transfer applications increases. As there are no resource reservation and admission control to monitor the imposed network load, the total number of packets in the network is more than they should be, leading to a congestion.

The TCP congestion control mechanism controls the sending rate by keeping track of the congestion window (CWND). When a new TCP connection is established, the CWND is set to one maximum segment size (MMS) (Allman and Floyd, 2002). When an acknowledgement packet is received, the CWND is increased exponentially. To limit the CWND size, there is a slow-start threshold that is set to 65Kbytes (Stevens, 1997). After the CWND size has reached the slow start threshold, it is increased linearly. This region is called congestion avoidance as shown in Figure 3.1. In the congestion avoidance region, the initial window is increased linearly. Whenever the timeout event occurs, the slow start procedure is initialized by setting the congestion window size to one, whereas the CWND is halved (i.e. *CWND = CWND/2*) when three duplicate ACKs are received. Initiating the slow start procedure whenever a timeout event occurs reduces throughput.

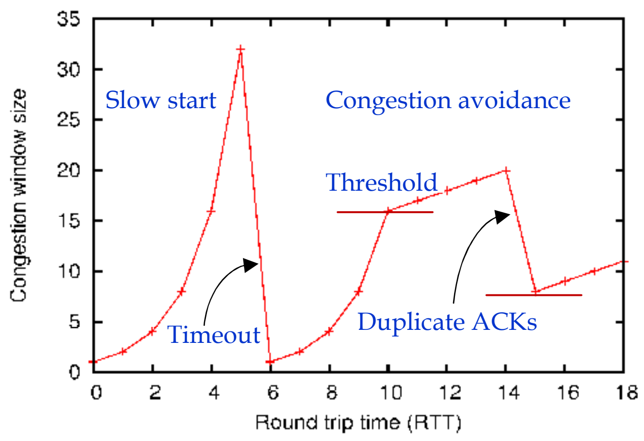


Figure 3.1 : Congestion control algorithm of TCP

As a transport layer protocol, TCP controls network congestion and bridges a file transfer application to a lower layer. TCP limits the transmission rate depending on the congestion window and provides reliability using the packet retransmission technique. It sends ack packets for each received TCP segment. Sending ack packets for all received TCP segments reduce the performance throughput, especially in longer path length. Therefore, Chen et al, (2008) not only points out that the path length is an important factor to consider when choosing appropriate delay window sizes, but also proposes a delay acknowledgement scheme that delays and balances ack flow and burst loss. This approach improves throughput up to 30% in static networks. In addition, a number of TCP variants have been introduced for the traditional TCP with layer approach (i.e. enhance within the single layer) and the mobile wireless environments with cross layer approach (i.e. use the aid of other layers). The next sections describe how to enhance the transport layer protocol (i.e. TCP) to achieve the better performance in mobile environments.

3.2 Enhancements of Traditional TCP

Reno (Allman, 1999), New Reno (Floyd & Henderson, 1999), Vegas (Brakno et al., 1994) and Westwood (Caasetti et al., 2002) proposed TCP enhancements with a single layer approach.

3.2.1 TCP-Reno

Instead of starting transmission from a slow start after a relatively long idle period, Allman (1999) introduced TCP-Reno that adds fast retransmit and fast recovery algorithms. With fast retransmit, Reno attempts to retransmit packets before a timeout, but the sender initiates the slow-start procedure as if a timeout causes the retransmission. With fast recovery, Reno uses additive increase/multiplicative decrease (AIMD) at all time, and only initiates the slow start when either a connection is established or a timeout occurs. In other words, Reno with fast recovery omits the slow start if no timeout occurs as shown in Figure 3.2.

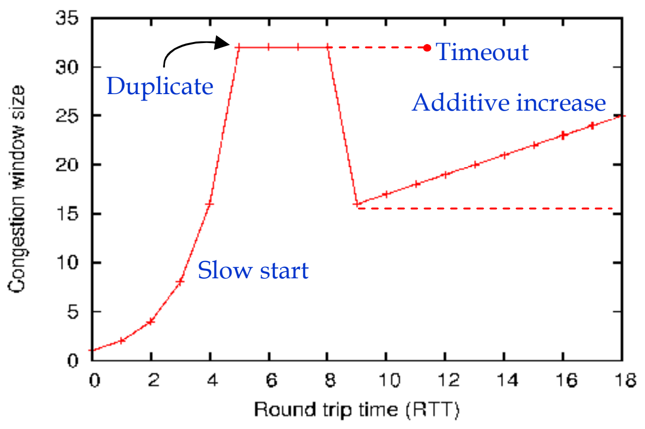


Figure 3.2 : Congestion control algorithm of TCP-Reno

3.2.2 TCP-New Reno

TCP-New Reno (Floyd & Henderson, 1999) is an improvement of Reno, and it is advanced fast transmit, where three duplicate acknowledgments signal a retransmission without a timeout with fast recovery. The fast recovery means that once a certain ACK threshold is received, the window size is decreased by half rather than starting over with a slow start. Only during a timeout does it go back into slow start. New Reno increases the adoption of the TCP selective acknowledgements (SACK) (Mathis & Mahdavi, 1996) modification.

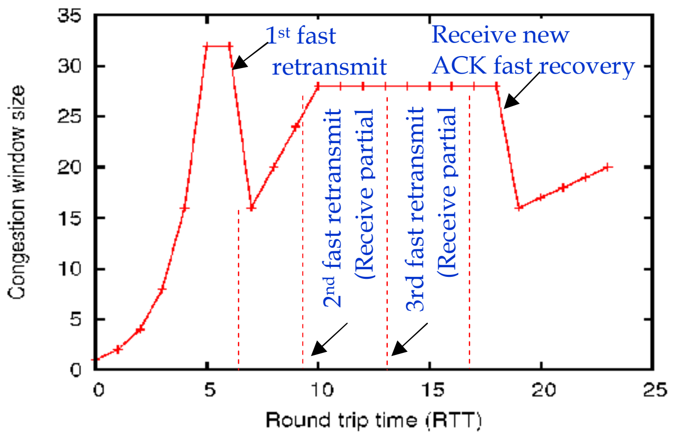


Figure 3.3 : Congestion control algorithm of TCP-New Reno

TCP-New Reno involves two kinds of ACKs: partial ACK and full ACK. The partial ACK acknowledges some segments at the fast recovery stage while the full ACK acknowledges all outstanding data. Upon receiving the full ACK, the sender sets the congestion window to slow start threshold and terminates the fast recovery as shown in Figure 3.3. Then the congestion avoidance mechanism is resumed. In this way, the New Reno maintains a high throughput.

3.2.3 TCP-Vegas

Tahoe, Reno and New Reno variants are window-based transport protocols that adjust the congestion window upon packet losses. On the other hand, Brakno et al., (1994) introduce a delay-based TCP, called TCP-Vegas, which does not violate the congestion avoidance paradigm of TCP. Vegas prevents packet losses by reducing the sending rate once it senses initial congestion. Vegas uses packet delay as an indication of congestion.

In a situation where a duplicate ACK is received, the timestamp for the ACK is compared to a timeout value. If the timestamp is greater than the timeout value, then Vegas will retransmit rather than waiting for three duplicate ACKs. Vegas estimates the available bandwidth using the difference between expected and actual flow rates.

When the network is congested, the actual flow rate will be smaller than the expected flow rate. Otherwise, the actual flow rate and the expected flow rate are close to each other. TCP-Vegas estimates the congestion level and updates the window size accordingly. The difference between the flow rates can be easily calculated during the round trip time using the equation

$$Diff = (Expected - Actual)BaseRTT$$

where $BaseRTT$ is the minimum round trip time. Based on Diff, the source updates its window size as follows.

$$CWND = \begin{cases} CWND + 1 & if\, Diff < \alpha \\ CWND - 1 & if\, Diff > \beta \\ CWND & otherwise \end{cases}$$

3.2.4 TCP-Westwood

TCP-Westwood (Caasetti et al., 2002) modifies the congestion window algorithm of TCP at a sender-side. The idea behind is to estimate the available bandwidth to control the congestion window size by monitoring the ACK packets. A sender measures the rate of ACKs that it receives and estimates the current bandwidth according to that connection. Once packet losses occur (i.e. timeout or duplicate ACKs), the sender set appropriate congestion window according to the estimated bandwidth. Instead of halving the congestion window like Reno and New Reno, TCP Westwood backs off some value of cwnd and threshold based on the estimated value to ensure faster recovery. The improvement of Westwood is more significant in lossy-link wireless networks due to its bandwidth estimation.

3.3 Layer Approach for Enhancements of TCP

The transport layer protocol (i.e. TCP) has been enhanced with layer approach and cross layer approach for the mobile environments. Due to the inherent reliability of wired networks, there is an implicit assumption made by TCP that any packet loss is due to congestion (Caceres and Lftode, 1995). TCP simply invokes the congestion control mechanism as soon as packet losses are detected.

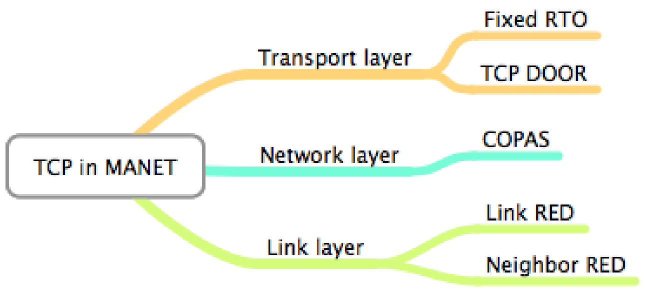


Figure 3.4 : Layer TCP enhancements for MANETs

To adjust a congestion window due to packet losses and to be robust to sporadic losses due to channel errors, Gao, et al (2008) proposed TCP SPC (Statistic Process Control) by using RTT statistics that judge the network status. Although the wider use of TCP provides reliable data transmission for the wired Internet, there has been significant performance degradation when it is implemented in the wireless world. Node mobility, misinterpretation of packet losses, frequent route breaks, congestion and the effects of path length are common problems for the TCP. Figure 3.4 shows the layer approaches to apply TCP in the mobile environment.

3.3.1 Fixed RTO

To distinguish between route failures and congestion, Dyer and Boppana (2001) proposed the fixed retransmission timeout (fixed RTO) technique by enhancing the transport layer. There are two RTOs to determine whether the packet loss is due to route break or congestion. The two timeout events occur in sequence, and then if the missing acknowledgement is not received after the expiry of the second timeout, the source node assumes that this packet loss is due to route failure. Instead of using an exponential backoff algorithm of the traditional TCP, the fixed RTO is maintained without doubling the timeout value until the route is re-established. This technique is evaluated for three ad hoc routing protocols, i.e. AODV, DSR and ADV (Adaptive proactive) (Boppana and Konduru, 2001). ADV with fixed RTO technique achieves a higher throughput up to 12%, with lower overhead compared to AODV and DSR.

3.3.2 TCP-DOOR

Due to the out-of-order delivery events occurring at the TCP destination, an end-to-end protocol called TCP Detection of Out-of-Order and Response (TCP-DOOR) was proposed by Wang and Zhang (2002) by enhancing the transport layer. The out-of-order events are detected by using the non-decreasing property of the ACK sequence numbers. Out-of-order packets are detected by looking at either the sequence number of packets at the receiver side or the sequence number of ACKs at the sender side. These out-of-order events are interpreted as route failures and the congestion control algorithm is disabled temporarily. However, the assumption of out-of-order packet as due to route failure deserves further analysis because the sender or the receiver may receive the out-of-order packets due to collision or congestion. Simulation results of TCP-DOOR show that it improves throughput up to 50% on average due to its out-of-order detection.

3.3.3 COPAS

Cordeiro et al. (2003) proposed the Contention-based Path Selection (COPAS) to address the TCP performance by considering the TCP packet drop due to the wireless channel. COPAS not only uses the routing layer information and selects disjoint routes but also redirects packets whenever a route failure occurs. It monitors the backup threshold to measure the contention levels in the network. Once the contention of a route exceeds the backup threshold, an uncongested route or a new route is selected to redirect traffic from a highly congested route. Simulation results show that COPAS achieves 90% better throughput than DSR in a static network. Although this technique is the best for static networks, it may not perform well for mobile networks.

3.3.4 Link RED

The link Random Early Detection (RED) algorithm proposed by Fu et al. (2003) monitors the average number of retransmissions at the link layer to reduce the wireless channel contention. When the average number of retransmissions exceeds a given threshold, the backoff timer at the MAC layer is increased without notifying the sender. In this way, the link RED technique is able to improve TCP throughput between 5% to 30% in various simulated topologies.

3.3.5 Neighborhood RED

Xu et al. (2003) proposed the neighborhood Random Early Detection (RED) to enhance TCP fairness. This algorithm uses a new distributed queue called neighborhood. It also measures RED dropping and neighbor contention level based on ideal and busy time slots. The neighborhood RED aggregates queues of its one-hop neighbors to compute the average queue size. Simulation results verify its effectiveness in improving TCP fairness.

3.4 Cross-layer Approach for Enhancements of TCP

There are cross layer enhancements that address the problem of TCP's inability. Figure 3.5 shows the modified versions of the mentioned problems to improve the performance with the cross layer approaches.

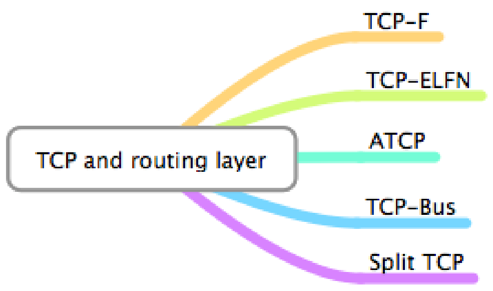


Figure 3.5 : Cross layer TCP enhancements for MANETs

3.4.1 TCP-F

TCP-Feedback (TCP-F) was proposed by Chandran et al. (2001) to overcome the problem of invoking the congestion control algorithm by mistake. It depends on an intermediate node at the routing layer to monitor the mobility of a downstream neighbor along the route. To signal TCP about the route failure, a route failure notification (RFN) is set whenever a route break is detected at the routing layer. Upon receiving RFN, the TCP sender goes into the snooze state, stops sending packets and freezes timers. The sender can resume transmission after receiving a route reestablishment notification (RRN) from the network layer. Although entering the snooze state avoids unnecessary retransmission and invoking congestion control, if a route to the sender is not established at the routing layer due to mobility, additional control packets, such as RFN and RRN, tends to increase congestion and collision in the network. This could lead to the performance degradation, such as increased end-to-end delay and decreased throughput.

3.4.2 TCP-ELFN

The explicit link failure notification (ELFN) technique is similar to TCP-F and was proposed by Holland and Vaidya (2002) to inform a TCP sender about a route failure. Upon receiving the ELFN message, the TCP sender disables its retransmission timers and enters a standby mode, but it periodically sends a small packet to probe the network. Unlike the TCP-F, the TCP-ELFN uses an explicit notification to monitor a new route. As soon as an acknowledgement for the probe packet is received, the TCP sender defers the standby mode, resumes its retransmission timer, and continues the packet transmission. The TCP-ELFN, which uses the DSR routing protocol, requires only a link failure notification from the lower layer. However, both mechanisms, TCP-F and TCP-ELFN, do not consider bit error that is very common in the MANETs.

3.4.3 Ad Hoc TCP (ATCP)

ATCP (Liu and Singh, 2001) inserts a thin layer between IP and TCP to handle packet losses or high bit error rate. Contrary to previous TCP F/ELFN, ATCP considers the packet losses due to channel errors. Explicit Congestion Notification (ECN) and destination unreachable message are used to learn the network state information so that the sender can pick a proper state correctly. If the sender receives a destination unreachable message, it goes into persist state and packets are not transmitted until a new route is found instead of invoking a congestion control. As soon as the ECN is received, congestion control is immediately started instead of waiting for a time out period.

The high bit error is detected when packet losses occur, but if the ECN bit is not set, ATCP assumes such losses as bit error and simply retransmits the lost packets. ATCP is only active at the TCP layer, and has four possible states: Normal, Congested, Loss and Disconnect. In the normal state, ATCP counts the number of duplicate ACKs and put TCP in the persist mode if either three duplicate ACKs are received or the retransmission timer expires. Then it goes to the loss state and transmits the unacknowledged segments of TCP buffer. If the ECN flag that indicates congestion is active, it goes to the congested state, does nothing, and returns to the normal state on receiving a packet from the TCP layer. If the TCP sender receives an unreachable message, it enters a Disconnected state as shown in Figure 3.6.

Although the ATCP is transparent, meaning that nodes with and without ATCP can set up TCP connections normally, the ATCP layer may require to change the layer interfaces currently in use.

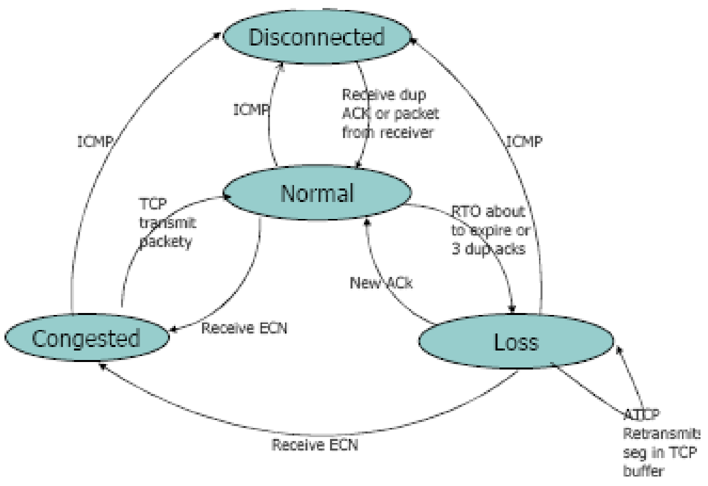

Disconnected
ICMP
ICMP
Receive dup ACK or packet from receiver
ICMP
Normal
TCP transmit packety
RTO about to expire or 3 dup acks
New ACk
Congested
Receive ECN
Loss
ATCP Retransmit seg in TCP buffer
Receive ECN

Figure 3.6 : State transition diagram for ATCP at the sender (Liu and Singh, 2001)

3.4.4 TCP-Bus

Buffering capacity and sequence information (TCP-Bus) was proposed by Kim et al. (2000). TCP-Bus uses feedback information of a node that detects a path break. The source initiated on-demand Associatively Based Routing (ABR) (Toh, 1999) is selected to get the feedback information from the routing layer. TCP-Bus uses two control messages, i.e. explicit route disconnection notification (ERDN) and explicit route successful notification (ERSN) for route maintenance. On detecting a route break, the ERDN is generated at the intermediate node or pivoting node (PN). After discovering a new partial route from the source to destination, the PN sends an ERSN message to the source node. During the route reconstruction phase, packets are buffered until a new partial route is established. When a route failure occurs, timeout values for the buffered packets are doubled. To retransmit the lost packets, the receiver node requests a selective retransmission by sending an indication to the source. Although the TCP-Bus outperforms TCP and TCP-F, it does not consider the failure of the PN to establish a new route to the destination.

3.4.5 Split TCP

Split TCP (Kopparty et al., 2002) mainly solved the unfairness problems between TCP sessions because sessions with short paths achieve much higher throughput than sessions with long paths. The split-TCP also splits a TCP connection into a set of shorter TCP connections, and proxy nodes are used as end points of these short connections as shown in Figure 3.7. The number of proxies depends on the path length – longer paths require more proxy nodes. The routing agent determines if a node has a proxy duty according to the inter-proxy distance. The responsibility of a proxy is to capture TCP packets, store them, and send local acknowledgement (LACK) packets to the source (or the previous proxy). When the proxy receives

LACK, the stored packets are cleared. In the end-to-end delivery, the destination node also sends end-to-end ACKs to the source upon receiving packets.

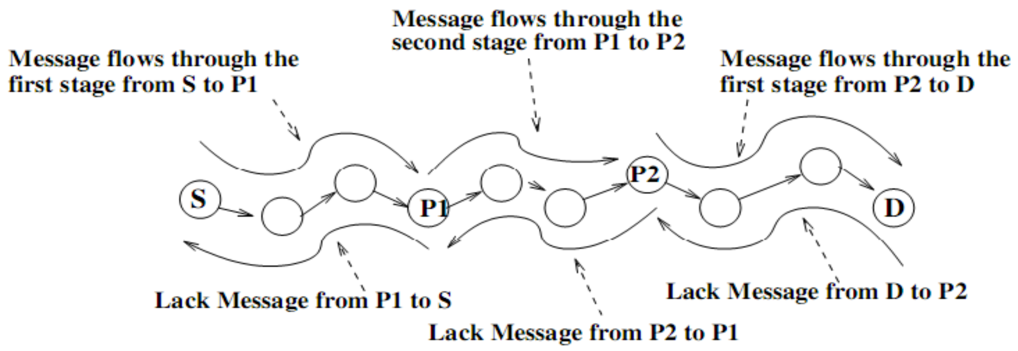

Message flows through the first stage from S to P1
Message flows through the second stage from P1 to P2
Message flows through the first stage from P2 to D
S
P1
P2
D
Lack Message from P1 to S
Lack Message from D to P2
Lack Message from P2 to P1

Figure 3.7 : TCP-Split (Kopparty et al., 2002)

The intermediate nodes decide whether they should act as a proxy or a normal node depending on the number of hops. The split-TCP improves throughput and fairness. As the split TCP still requires the end-to-end ACKs, when a proxy fails or becomes disconnected from the network, the overall throughput is higher than that of regular TCP. Large buffers and network overload are the weaknesses of split TCP. On the other hand, as the number of TCP connections increases, the number of proxies also increases. A node may act as a proxy for two or three TCP connections, giving rise to the possibility of buffer overflow at the proxy. This approach works best with fewer TCP connections. As the number of TCP connections increases, not only the number of proxies, but also the overhead used for LACK, increases. A similar split TCP technique was proposed by Luglio et al, (2004) for satellite networks.

3.4.6 Hop-by-hop Transport Protocol

As the end-to-end data reliability checking tends to decrease the TCP performance in the MANETs, Scofield (2007) and Heimlicher et al. (2007) designed a hop-by-hop framework to overcome the end-to-end inefficiency of TCP. According to the simulation results, the hop-by-hop transport protocol delivers data packets three times faster than the end-to-end TCP in the MANETs. In the hop-by-hop framework, data packets are forwarded from node to node in a store-and-forward manner, and the participation of intermediate nodes is very important. The hop-by-hop protocol is run on every node, and provides per-link flow control and congestion control. Ensuring end-to-end data efficiency still requires the end-to-end flow and congestion control. However, checking data packets at every node are very expensive for scarce resources, such as battery power and bandwidth. As we mentioned above, the transport protocols are enhanced to improve throughput and fairness of TCP with the aid of routing layer protocols. In this thesis, to provide a reliable data transmission, we use a proxy node for each TCP connection, instead of using many proxies for one TCP connection, with the assistance of the routing layer.

3.5 Chapter Summary

Previous researchers have focused on solving the following problems.

- The difficulty to distinguish route failures and congestion
- Out-of-order delivery packets at the destination node
- Packet drop problems of TCP due to the wireless channel
- Contention and collision problems
- TCP unfairness problems
- TCP's end-to-end inefficiency problems

The approaches taken to solve the above problems are:

1. Enhancing protocol within a single layer
    - Congestion control algorithm

    - Retransmission and timeout considerations of TCP

2. Sharing information between layers
    - Use the information of other layers to detect bit error and contentions
    - Learn topology changes with the information from other layers

While the proposed solutions above are able to address the problems at the transport layers, there is an outstanding problem as listed below:

- For the reliability of TCP data packets at the transport layer, the sender and receiver check whether the packets are received correctly by using acknowledgment packets. However, not only are the routes very unstable, but the end-to-end data reliability checking of TCP is insufficient in the MANETs. The hop-by-hop transport protocols proposed by Scofield (2007) and Heimlicher et al. (2007) checks the reliability of data packet at every node to improve the correctness of data and throughput. However, this approach is very expensive on scarce resources, such as bandwidth, power and memory usages. Therefore, in this thesis, the technique that we proposed checks the reliability of data packets at a proxy node for a TCP connection while maintaining the end-to-end reliability of TCP.

As a summary, we propose a proxy acknowledgement (PACK) technique for the TCP senders. For the efficiency of TCP data packets, instead of checking TCP packets either at every hop or at many proxies like split TCP, the proposed protocol monitors data packets at one proxy node between a source and destination. The main objective is to address the problems of end-to-end inefficiency and hop-by-hop inefficiency at the transport layer by using a proxy-assisted approach. The justification for our proposed solution is discussed in Chapter 5.

Chapter 4

RESEARCH METHODOLOGY: THE CROSS LAYER ENHANCEMENT BETWEEN THE ROUTING AND MAC LAYERS

4.1 Introduction

This chapter discusses the methodology of PART protocol, followed by implementation and experimental results. The proposed protocol, Proxy-assisted routing protocol for data transmission (PART) must satisfy the following requirements to transmit data packets efficiently.

Resilience to route failures: The protocol is designed to be resilient to route failure with the aid of a proxy node.

Broadcast zone limitation: The protocol limits the broadcast zone to reduce unnecessary forwarding of control packets.

Cross-layer information: It needs the address of downstream and upstream nodes from the MAC layer for unicast transmission.

4.2 Components of PART Protocol

PART consists of two components, which are explained below.

4.2.1 Packet Types

The packets that are used in PART protocol are shown in Figure 4.1. A broadcast route request (RREQ) packet and a unicast route reply (RREP) packet are to discover a route. For reporting a route error or collision, a broadcast or unicast route error (RERR) packet is sent depending on the precursor lists[1] in the routing table. For

[1] The precursor lists of a routing table entry contain the neighbor nodes to which a route reply is generated or forwarded.

repairing a route locally after receiving a route failure notification, RREQ and RREP with flags are used. For notifying a proxy address, a P-INFORM packet is used by using cross-layer information for the unicast transmission.

RREQ/ RREQ-with-flag

Flag (P,D)	Bcast-id	HC	PHC	dest	dest-seq-num	src	src-seq-num	padd

RREP/RREP-with-flag

Flag (P,D)	HC	PHC	dest	dest-seq-num	src	padd

RERR

dest-count	proxy-count	unreach-dest	unreach-dest-seq-num	unreach-proxy

P-INFORM

dest	dest-seq-num	Padd	HC	PHC

Figure 4.1 : Formats of control packets for PART

RREQ/RREQ-with-flag

Pflag	Proxy only flag; indicates only the proxy may respond to RREQ.
Dflag	Destination only flag; indicates only the destination may respond to RREQ.
Bcast-id	An ID number that indentifies the particular RREQ with the IP address of a source node.
Hop Count (HC)	The number of hops from the originator IP address to the node handling the request.
Destination IP address (Dest)	The IP address of the destination that is intended for data transmission.

Destination sequence number (dest-seq-num)

The latest SN previously received by the originator for any route toward the destination.

Proxy hop count (PHC)

The number of hops to the proxy from the current node.

Source IP address (src)

The address of the source node that sends data packets.

Source sequence number (src-seq-num)

The current sequence number to be used in the route entry pointing toward the originator of the route request.

Proxy Address (Padd)

The address of the proxy node.

RREP/RREP-with-flag

Most fields are the same as RREQ packet. The only difference is the transmission method, i.e. unicast or broadcast transmission.

RERR

Unreachable destination IP address

The IP address of the destination that has become unreachable due to a link break.

Unreachable destination sequence number

The sequence number in the routing table for the destination listed in the previous unreachable destination IP address field.

Unreachable proxy address

The IP address of the proxy that has become unreachable due to a link failure or proxy failure.

DestCount	The number of unreachable destinations in the message.
Proxy-count	The number of hops to the proxy from the current node.

P-INFORM

P-INFORM is used to notify the destination node the proxy's address after assigning a proxy node.

4.2.2 Routing Table

Each node needs to keep track of how to deliver data packets, and for this purpose, a routing table is used. The routing table is like a database to store the shortest paths or all available paths to provide nodes with routing information as shown in Figure 4.2.

ROUTING TABLE

dest	seq-num	hops	p-hops	next hop	proxy	rep_downstream	flag

Figure 4.2 : Routing table of PART

PART is a hop-by-hop routing protocol, where each routing table keeps all reachable destinations and the address of the next node towards the destination. An entry in a routing table contains the following information.

- *Destination IP address*
- *Hop Count*: The last known hop count to the destination
- *Destination sequence number*: The latest sequence number for the destination
- *Proxy Hop Count*: The number of hops away from source/destination to a proxy.
- *Next Hop*: The node to forward a data packet in order to reach the destination

- *Proxy Hop*: The node that needs to send local acknowledgement or repair the broken routes
- *Lifetime*: How long a route remains valid
- *List of Precursors:* The list of neighboring nodes that are likely to use them as next hops toward a destination
- *Valid Destination sequence number flag*

When a node receives a control message from a neighbor, it checks its routing table for an entry of the destination. If an entry does not exist, one is created. If an entry exists, it compares the destination sequence number in the entry and in the message. The route is updated if either

- the sequence number (SN) in route table entry (RTE) is lower than the one in the control message, i.e., ($SN_{RTE} < SN_{CONTROLmessage}$) or
- the sequence numbers are equal ($SN_{RTE} == SN_{CONTROLmessage}$), but the hop count of control packets is smaller than the existing hop count in the table, i.e., ($HC_{CONTROLmessage} < HC_{RTE}$) or
- the sequence number is unknown.

Depending on the control message, the lifetime of an active route is updated each time the route is used, and is initialized to *ACTIVE_ROUTE_TIMEOUT*. Each time a route is used to forward a data packet, each node on the route updates this field to (*CURRENT_TIME + ACTIVE_ROUTE_TIMEOUT*).

4.3 Overview of PART Protocol

The characteristics of PART include broadcast and unicast techniques and proxy repair function to repair the routes locally.

- PART builds routes using a Route Request (RREQ) broadcast with or without a flag, and a Route Reply (RREP) unicast with or without a flag. P-INFORM is used to let the other nodes know the proxy's address.
- The proxy calculation function is the responsibility of the destination. A node receiving the RREQ does the following.
    - Check if it is with or without a flag.
    - If a destination node receives RREQ-NOflag, it considers using a proxy node according to the hop count information.
    - Reply with RREP-NOflag packet to the sender when an intended destination receives RREQ-NOflag packet.
    - Reply with RREP-Pflag to the sender when a proxy node receives RREQ-Pflag packet.
    - Reply with RREP-Dflag to the proxy when a destination node receives RREQ-Dflag packet.
- To reduce the number of RREQ broadcast packets, a limited broadcasting zone is created to decrease routing overhead.
- The proxy node is responsible for detecting a route break and repairing it locally by broadcasting the RREQ-Dflag packet (i.e. Proxy repair function).

4.4 Route Discovery Procedure and Functionalities of Nodes

In the following sections, the fundamental operations of the nodes (i.e. source node, destination node, intermediate node and proxy node) and how to assign a proxy node during a route discovery procedure are outlined.

The source node decides whether the RREQ is broadcast with or without flag, depending on its routing table information. Firstly, the source node checks its routing table (RT) to find information related to a proxy node. If the source node finds the proxy address (Padd) in its routing table, it sends RREQ-Pflag to the proxy node instead of broadcasting the RREQ packet to reach the destination. Otherwise, it broadcasts RREQ-NOflag packet throughout the network (cf. Algorithm 4.2).

Receive Data from Application layer (Source)

```
1:  upon  receiving a data packet from the application layer  do
2:      put  data packets  into  send buffer
3:      call  Algorithm 4.2
4:  end upon
```

Algorithm 4.1 : Data from the application layer

Send control packet (RREQ-NOflag) at the source node

```
1:  upon  receiving a  data packet  intended for a node  do
2:      if  proxy address  is not already known in the  routing table  then
3:          send  RREQ-NOflag
4:      else
5:          send  RREQ-Pflag
6:      end if
7:  end upon
```

Algorithm 4.2 : Functions of a source node

The basic route discovery procedure of PART is shown in Figure 4.3. When a destination node receives a RREQ-NOflag, it decides whether to use the assistance of a proxy node by checking the hop count of RREQ. In this case, the minimum allowable hop count value, which is an adjustable value, is defined as three. When the distance between the source and destination node is less than the defined value, a proxy node is not used.

To repair a route and start data transmission quickly, a middle node in the path is assigned as a proxy node. If the hop count of RREQ-Noflag is greater than the predefined value, the proxy calculation function is invoked (cf. Algorithm 4.3). If the hop count is even, proxy hop count (PHC) is taken by dividing the hop count by two.

Otherwise, the hop count is added by one and divided by two to assign a proxy node along the path (cf. Algorithm 4.4).

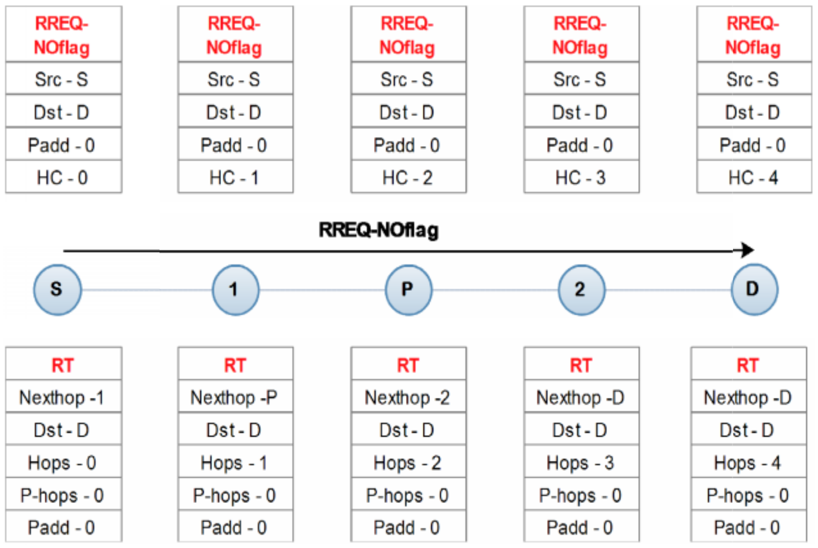


Figure 4.3 : RREQ-NOflag packet broadcasting and routing table updating

Receive control packet (RREQ-NOflag) at the destination node

```
1:  upon  receiving  RREQ  at destination  do
2:      check  HC of RREQ
3:      if  HC is greater than a predefined value  then
4:      call  Algorithm 4.4
5:      end if
6:  end upon
```

Algorithm 4.3 : Consideration of a proxy involvement

Proxy calculation function at destination node

```
1:  if  HC  is even  then
2:      HC is divided by two to obtain  PHC  value
3:  else
4:      HC is added by one and divided by two to obtain  PHC  value
5:      Add PHC value to the  RREP-NOflag  packet
6:  end if
```

Algorithm 4.4 : Proxy calculation function

Proxy node assignation at intermediate nodes

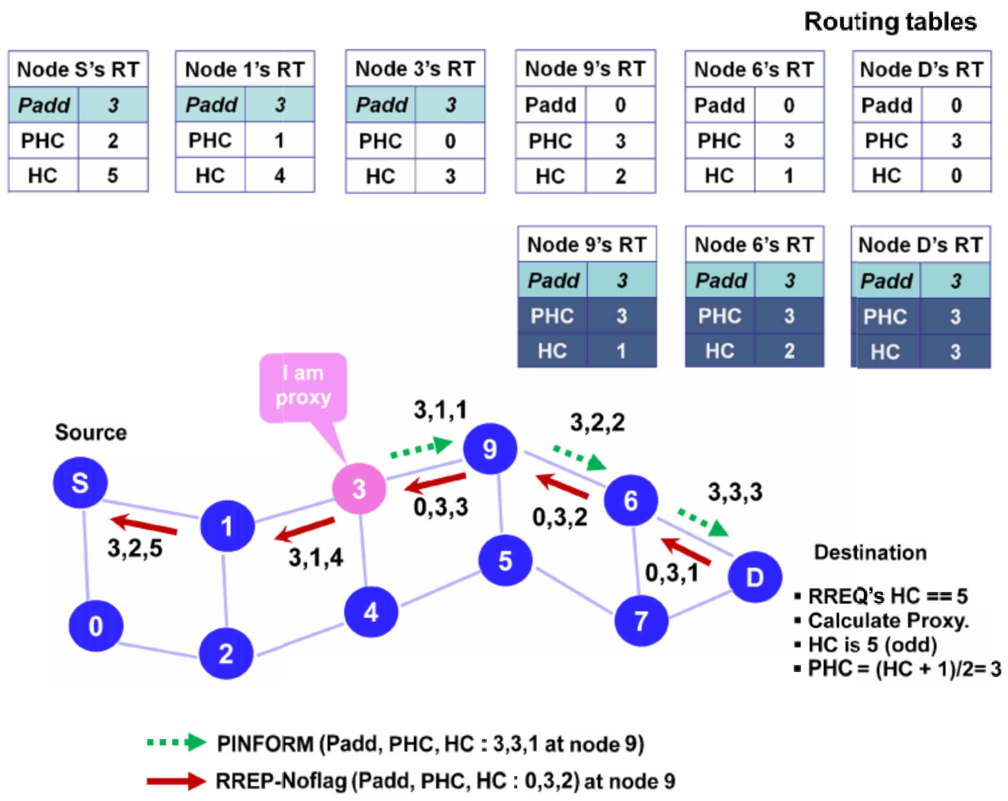
```
1: upon receiving RREP-NOflag at the intermediate nodes do
2:     check PHC value and HC value
3:     if PHC and HC are equal then
4:         I am proxy node
5:         call Algorithm 4.6
6:     else
7:         forward RREP-NOflag
8:     end if
9: end upon
```

Algorithm 4.5 : Proxy node assignation process

After calculating a PHC value, the destination node attaches this value in the RREP packet to assign a proxy node. At this point, the destination node only knows how many hops away the proxy node is, but does not know the proxy address yet. The intermediate nodes that receive the RREP-NOflag packet compare the calculated PHC value and hop count value of RREP-NOflag packet, which increases monotonically. If these values are equal, this node is assigned to perform as a proxy node. Otherwise, RREP-NOflag packet is forwarded to the next hop according to the routing table information (cf. Algorithm 4.5).

The PART protocol comes to life after the destination node has decided the PHC value when it receives the RREQ-NOflag packet. The destination node updates its routing table first after calculating the PHC value. In other words, the PHC value is 3, which is added to the routing table of the destination node as shown in Figure 4.4. Then, this PHC value is attached in RREP-NOflag packet (Padd, PHC, HC, for example, 0, 3, 1). The destination node does not know the proxy address yet, thus Padd is 0; PHC, which is a calculated value is 3; and HC, which always starts at 1.

Referring to Figure 4.4, when the intermediate nodes (i.e. nodes 6 and 9) receive RREP-NOflag packet, they compare whether the PHC and HC values are equal, update their routing tables and forward RREQ-NOflag packet because the values are not equal.



Node S's RT	
Padd	*3*
PHC	2
HC	5

Node 1's RT	
Padd	*3*
PHC	1
HC	4

Node 3's RT	
Padd	*3*
PHC	0
HC	3

Node 9's RT	
Padd	0
PHC	3
HC	2

Node 6's RT	
Padd	0
PHC	3
HC	1

Node D's RT	
Padd	0
PHC	3
HC	0

Node 9's RT	
Padd	*3*
PHC	3
HC	1

Node 6's RT	
Padd	*3*
PHC	3
HC	2

Node D's RT	
Padd	*3*
PHC	3
HC	3

Figure 4.4 : Proxy assignation with reply packet (RREP-NOflag)

When the packet arrives at node 3, and it detects that the values are equal, it takes the responsibility of a proxy node. Node 3 then sends a unicast P-INFORM packet to the destination node to inform its proxy address. All nodes (i.e. nodes 9, 6 and D) update their routing tables by adding the Padd value when they receive P-INFORM. At the same time, node 3 attaches its address in the RREP-NOflag packet; resets the proxy hop count value and counts it from the default value to determine the distance between the source and proxy node. In this way, all nodes along the path to the source can update their routing tables. Before the source node starts data transmission, all nodes along the path know the proxy address without broadcasting.

4.4.1 Cross-layer Information for Unicast P-INFORM

The unicast P-INFORM packet is sent by taking the address information at the MAC layer, in other words, taking the cross layer information from the MAC layer. To use the address information of the MAC layer, *rep_downstream* field is added to the routing table. When RREP-NOflag packet is received from the destination node, intermediate nodes update the *rep_downstream* in the routing table. The nodes that receive the RREP-Noflag must keep the source's Ethernet address at the MAC layer so that the proxy node can send a P-INFORM to the downstream node to reach the destination while sending RREP-NOflag packet to the upstream node to reach the source as shown in Figure 4.4 and Algorithm 4.6.

Sending unicast P-INFORM packet at the proxy node

```
1:  upon receiving RREP-NOflag at the proxy node do
2:      add rep_downstream field in the routing table
3:      take source's Ethernet address at the MAC layer
4:      update rep_downstream address of routing table
5:      send P-INFORM to this downstream address
6:      while receiving P-INFORM at the nodes then
7:          if I am a destination node then
8:              add proxy address in the Padd of routing table
9:          else
10:             forward P-INFORM to downstream address
11:         end if
12:     end while
13: end upon
```

Algorithm 4.6 : Sending unicast P-INFORM

4.5 Limiting of the Broadcast Zone

In order to reduce the routing overhead, the source node defines a broadcast zone after it receives a route failure signal. A source node decides on a broadcasting zone depending on the nature of the route error. When a source node detects that a route error is due to the packet collision at the MAC layer "COL" (DROP_MAC_COLLISION) *(new-trace explanation)*, it sends a broadcast RREQ-

Pflag packet to the proxy node. To limit the broadcast zone, the source node attaches the PHC value in the RREQ-Pflag packet while repairing the route as shown in Figure 4.5 and Algorithm 4.7.

Source node limits a broadcast zone depending on the ERROR

```
1:  upon  receiving  ERROR  at the source node  do
2:      check  ERROR type
3:      if  collision at MAC layer  then
4:          send  RREQ-Pflag to the  proxy node
5:      else
6:          send  RREP-NOflag  to the source node
7:      end if
8:  end upon
```

Algorithm 4.7 : Limiting a broadcast zone depending on the error type

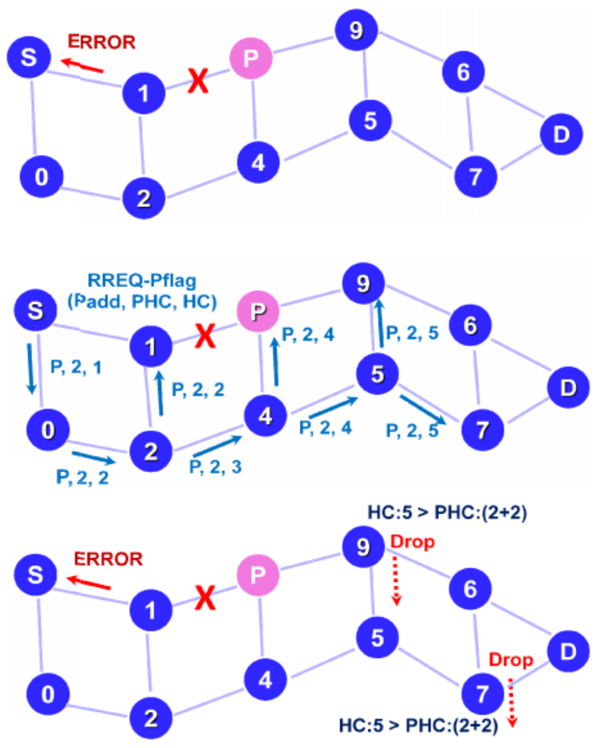


Figure 4.5 : Limitation of broadcasting zone with hop count consideration

When the intermediate nodes receiving the RREQ-Pflag packet find that the hop count is greater than PHC plus two, they discard the RREQ-Pflag packet without forwarding it. Referring to Figure 4.5, the PHC value of the source node is 2. Before the source node sends RREQ-Pflag packet, it attaches the PHC value 2. The RREQ-Pflag packet is forwarded as long as the hop count is less than (PHC+2) or the node is not the proxy.

If an intermediate node discovers the hop distance is higher than (PHC+2), it discards the RREQ-Pflag packet. For example, in Figure 4.5, when node 9 receives RREQ-Pflag packet, it detects that the hop count is 5, which is higher than (PHC + 2 = 4). A similar situation is seen at node 7. Node 9 and node 7 will not forward the RREQ-Pflag packet and discard it. When the proxy node receives the RREQ-Pflag packet, it sends a RREP-Pflag packet to the source.

4.6 Repairing Routes at the Proxy Node

When a proxy node detects that there is a packet drop or receives an ERROR packet, it tries to repair the route by sending a RREQ-Dflag packet intending that the destination should reply to it. The direction of packet flow determines which node acts as a source and destination, i.e. the sender of the packet is the source and the receiver is the destination. Even though the data source and destination are exactly the same for the RREQ and TCP packets, they may not be the same for the RREP and ACK packets. The proxy node tries to repair a route depending on the destinations of the packets. Referring to Figure 4.6, if the dropped packet is a request or data packet, such as TCP, the proxy node sends RREQ-Dflag to the data destination. If it is a reply or acknowledgement packet, the proxy node sends RREQ-D to the data source. Before sending the RREQ-Dflag packet, a proxy node keeps track of not only how many times the route errors have occurred, but also the CURRENT_TIME that the first error occurs. Then the proxy node tries to repair a route by calling a proxy repair function. However, when the proxy node finds that there have been three continuous failures within a predefined time (i.e. 0.5 seconds), the proxy node sends ERROR with RESET flag to inform the source node to discover a new route. (cf. Algorithm 4.8).

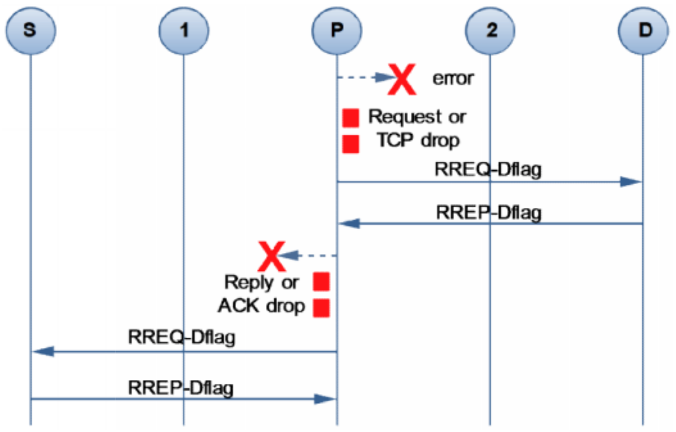


Figure 4.6 : Error handling at proxy node

The proxy node decides whether to repair a route or to give up the proxy duty

```
1:  upon receiving ERROR at the proxy node do
2:      increase proxy-fail-to-forward count by one
3:      if proxy-fail-to-forward is equal to 1 then
4:          add CURRENT_TIME to proxy_firsttime_failure
5:          call Algorithm 4.9
6:      else if proxy-fail-to-forward is equal to 2 then
7:          call Algorithm 4.9
8:      else if proxy-fail-to-forward is equal to 3 then
9:          add CURRENT_TIME to proxy_lasttime_failure
10:         if the time difference between proxy_lasttime_failure and
              proxy_firsttime_failure is less than or equal to predefined reset time then
11:             send ERROR with RESET flag to the source node
12:         else
13:             call Algorithm 4.9
14:         end if
15:     else if proxy_fail_to_forward is greater than 3 then
16:         reset proxy_firsttime_failure
17:         reset proxy_lasttime_failure
18:         reset proxy_fail_to_forward
19:     end if
20: end upon
```

Algorithm 4.8 : Detection a route error at a proxy node

In Algorithm 4.8, the *proxy_fail_to_forward* (int), *proxy_firsttime_failure* (double), and *proxy_lasttime_failure* (double) fields are defined in the routing tables to handle a route error at a proxy node. Once a route break is detected, a proxy node increases *proxy_fail_to_forward* by 1 and records the *CURRENT_TIME* that a failure occurs, and repairs a broken route by calling a *proxy_repair* function that sends RREQ-Dflag (cf. Algorithm 4.9).

Proxy node tries to repair a route when it detects an ERROR

```
1: upon notification to perform proxy_repair do
2:     buffer data packet
3:     mark RTF_IN_REPAIR in the flag of routing table
4:     while proxy_timer is not expire do
5:         send RREQ_Dflag
6:     end while
7: end upon
```

Algorithm 4.9 : Proxy-repair function

4.7 Detection of Proxy Failure Conditions

There are two ways to detect proxy failure conditions.

a) Detection by Source Node

After the source node has sent the RREQ-Pflag packet to the proxy node, it sets the RREP_P_WAIT timer. If the source node does not receive the RREP-Pflag packet after the timeout period, it retries. If it does not receive a reply after the second attempt, it assumes that the proxy node is dead, and discovers a new path and another proxy if the path length is still longer than the predefined value.

Another technique to check the availability of a proxy node is the consideration of hop count. When the source node receives RREP-Pflag packet from the proxy node, it checks the hop count. If the hop count is getting higher, the source node considers finding a new proxy soon after receiving an ERROR signal.

b) Detection by Proxy Node

The proxy node has to be intelligent to detect whether it should give up the proxy duty. Whenever an ERROR message is received, the proxy node tries to repair the route first. However, when the proxy encounters more than three ERRORs continuously within a very short time, it assumes that it is out of the transmission range or has moved far away from the source and destination, and sends ERROR with RESET flag.

As the proxy node moves out of the transmission range, the source node uses the *timer* and *retry techniques* to discover a new route if it does not receive the ERROR signal from the proxy node. The best thing the proxy node can do is to repair a route with less overhead as soon as possible and supports data transmission with less delay.

4.8 The Implementation of PART Protocol

This section gives an overview of the PART protocol and the core considerations of its design. Further, we present how the protocol interfaces with the simulator and how cross-layer information is managed.

The building of real testbeds for wireless connection does come at a price, and it is an impossible task when network factors like mobility, simulating area and movement pattern, are taken into account. Moreover, it is impossible to test hundreds or thousands of nodes using different topologies and movement patterns. Therefore, simulators have become compulsory for wired and wireless network environments to overcome the above mentioned problems.

There are many simulators, for example, GloMoSim (Bagrodia et al., 1998), NS-2 (*VINT Group, Network Simulator*), NS-3 (Henderson et al., 2006), OMNeT++ (Varga & Hornig, 2008), OPNET (*OPNET Technologies Inc. OPNET modeler*), and QualNet (*QualNet Network Simulator*). A comparison of the strengths and weaknesses of simulators has been discussed by Schilling (2005) and Weingartner and Wehrle (2009).

> *NS-2 has many and expanding uses, including: To evaluate the performance of existing network protocols; To evaluate new network protocols before use; To run large scale experiments impossible in real experiments; To simulate a variety of IP networks* (Meeneghan, 2004).

In this thesis, we use the Network Simulator (NS), mainly due to the provision of TCP variants, a variety of ad hoc routing protocols and wireless network interfaces. The NS-2 is an event-driven packet level network simulator developed as a part of the VINT project (Virtual Internet Testbed). The CMU Monarch group extended NS-2 to provide new elements at each layer and to construct detailed and accurate wireless simulations.

4.8.1 Introduction to Network Simulator (NS)

NS is objective-oriented and a discrete event simulator for the networking research. Objects can be defined in two key languages: C++ and Object-oriented Tool Command Language (OTcl).

The C++ and the OTcl are linked together using TclCL. These objects are combined using Tcl scripts. The simulation structure of NS is shown in Figure 4.7. During simulation, the NS creates an output trace file to analyze simulation results and output network animation file (NAM) to visualize the network simulation scenarios.

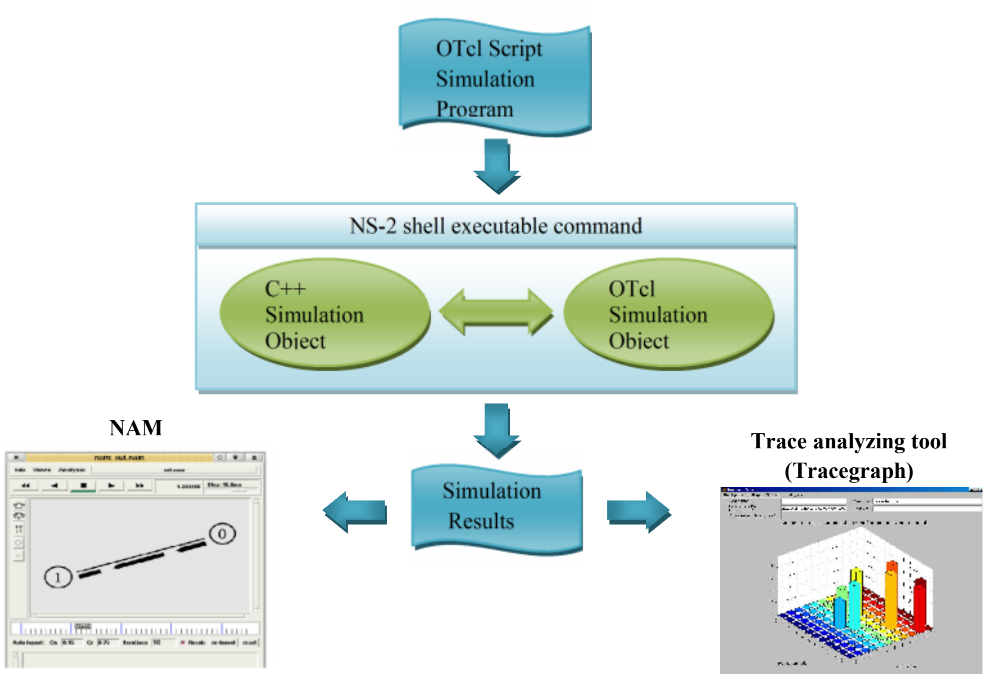

OTcl Script Simulation Program
NS-2 shell executable command
C++ Simulation Object
OTcl Simulation Object
NAM
Simulation Results
Trace analyzing tool (Tracegraph)

Figure 4.7 : User's view and basic architecture of network simulator

4.8.2 Components of NS

Figure 4.8 shows that all objects derived from class *NsObject* consists of two classes: *connectors and classifiers*.

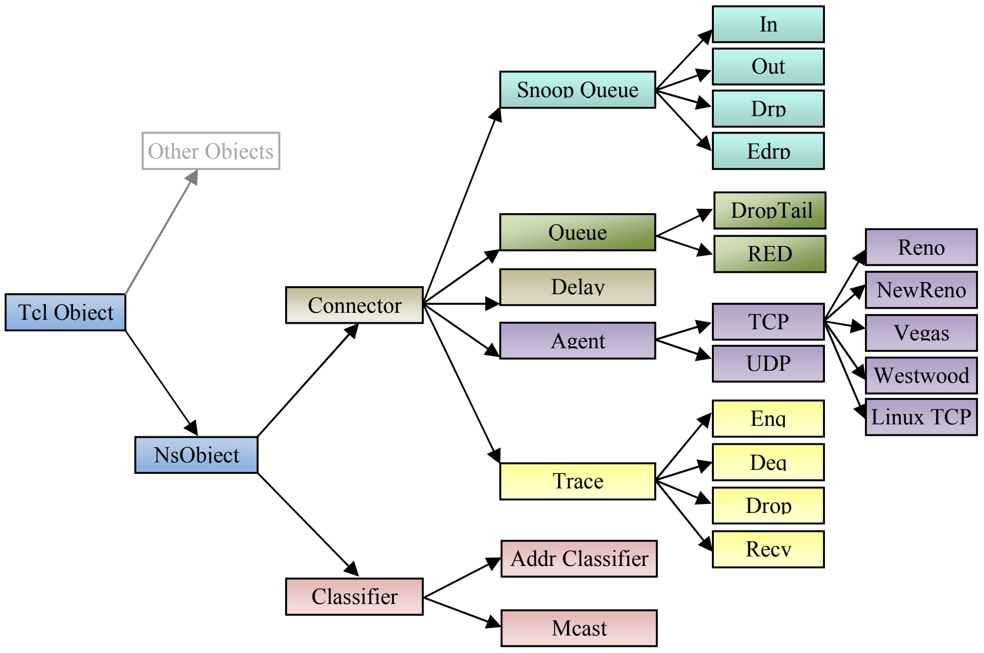


Figure 4.8 : Network components of NS

While connectors link queue, delay and agents, classifiers examine packets (i.e. multicast or unicast) and direct them to the respective destinations.

4.8.3 Basic Protocol Implementation in NS

Network components include queue management techniques (i.e. Drop Tail queue, RED queue, etc.), agents (i.e. TCP and UDP), routing, broadcasting and multicasting techniques, and traced output files. The detailed structure of installation, compilation, running and analyzing could be found in the NS manual (*The Network Simulator NS-2: Documentation*).

In NS-2, a new protocol is implemented as a C++ class that provides the basic functionality of a protocol, referred to as "routing agent", "transport agent", etc. Figure 4.9 shows the basic layering concepts of protocols. The TCP agents run the varieties of transport protocols, such as UDP and TCP (Tahoe, Reno, New Reno, Vegas, Westwood, and Sack).

The PART agents run the PART protocols. The LL agent supports data link protocols, such as link layer retransmissions, queuing, packet fragmentation and reassembly.

The IFq runs as an interface queue between link and MAC layers and determines the contention level. NetIF is the interface between the node and the channel that determines the propagation model, such as transmission power and frequency.

The RPM is a radio propagation model to implement an antenna and two-way ground propagation. Multiplexers are used as a bridge among layers and nodes. The address and port multiplexers (i.e. *addr demux* and *port demux*) are used as demultiplexers that bridge a node to a receiving transport or routing agent as shown in Figure 4.9.

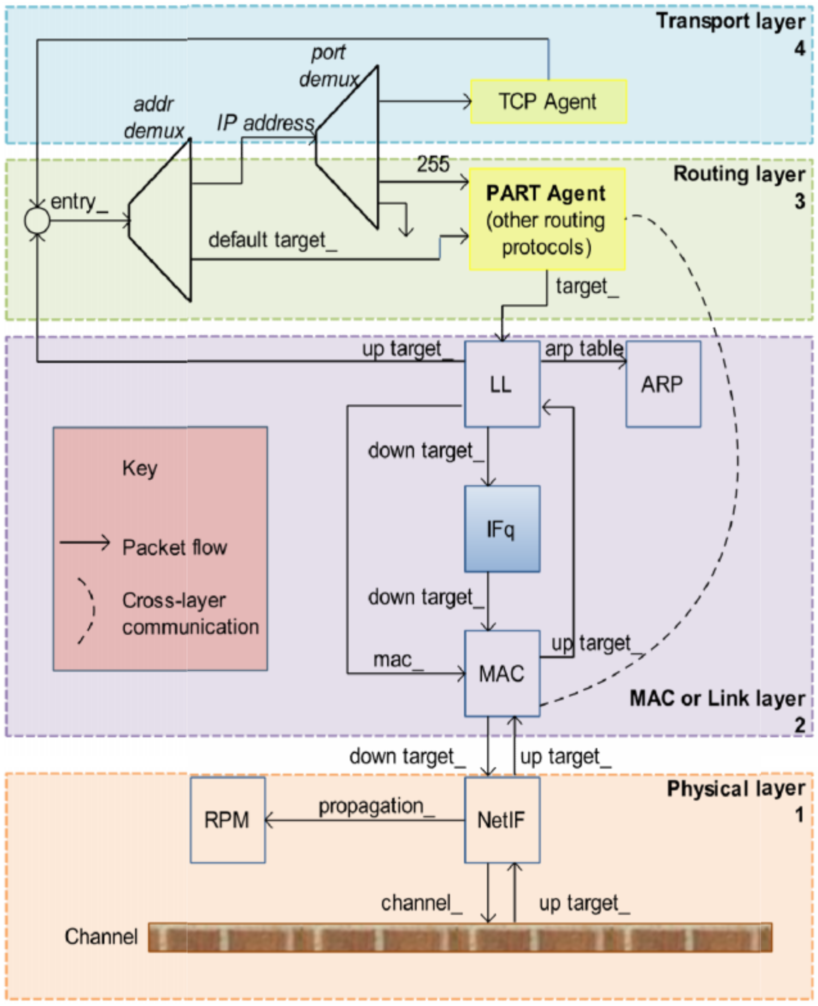


Figure 4.9 : A basic mobile node structure of NS

A routing agent is implemented, whose main responsibility is to compute the routing table. The PART routing protocol is implemented in the network simulator version 2.34. We firstly create a new directory called *part* inside the base directory of ns2.34, including:

part.h	This is the header file where all necessary predefined values, timers, and routing agent are defined for the functionalities of protocols.

part.cc	The routing agent, all timers and Tcl hooks are implemented in this file.
part_packet.h	All PART packet types are declared to enable the exchange of packets among nodes.
part_rtable.h	Header files where the routing table information of PART are maintained.
part_rtable.cc	Implement the routing table
part_rqueue.h	A header file where the data packets are queued.
part_rqueue.cc	Implement the queue size, queue function.

The files above are compulsory to build a new routing protocol. To implement a routing protocol in NS, an agent must be created by inheriting from *Agent* class that offers a link with a Tcl interface so that the PART protocol can be controlled through the simulation scripts written in Tcl.

4.8.4 Necessary Changes

Before coding the PART protocol inside NS, there are some changes to integrate our codes inside the simulator.

4.8.4.1 Packet Type Declaration

The new packet type, *part*, is indicated as *PT_PART* inside *common/packet.h* as shown in Program 4.1. In order to allow a dynamic definition of part within dynamic libraries, the predefined packet type is implemented as static constant.

Program : 4.1 //**~ns/common/packet.h**

```
static const packet_t PT_PART = 45;
  class PacketClassifier {
…
name_[PT_PART]= "PART";
}
```

4.8.4.2 Tcl Library

The new packet type, *PART*, is added in the following Tcl libraries to provide the necessary infrastructure and create wireless nodes running the PART routing protocol. (Program 4.2 to 4.5)

Program : 4.2 //**~ns/tcl/lib/ns-packet.tcl**

```
foreach prot {
PART
DSR
AODV
ARP
…
#
}
```

Program : 4.3 //**~ns/tcl/lib/ns-agent.tcl**

```
Agent/PART set sport_   0
Agent/PART set dport_   0
```

Program : 4.4 //**~ns/tcl/lib/ns-lib.tcl**

```
Simulator instproc create-wireless-node args {
# …
   switch –exact $routingAgent_{
      PART {
        set ragent [$self create-part-agent $node]
      }
      # …
}
```

Program : 4.5 //**~ns/tcl/lib/ns-mobilenode.tcl**

```
# Special processing for PART
set partonly[string first "PART" [$agent info class]]
if {$partonly != -1 } {
      $agent if-queue [$self set ifq_(0)];
# ifq between LL and MAC   }
```

4.8.4.3 Tracing Support

Tracing support is the most important part of the simulator in order to keep track of what happens during the simulation. A trace object describes the detail information of packets during the simulation, such as which node receives what types of control packet (i.e. request, reply, error or data) at how many seconds later after running the simulation.

The CMU trace objects provide the trace format for wireless simulations, and the new trace information of PART is needed to add in *cmu-trace.h* (Program 4.6) and *cmu-trace.cc* (Program 4.7). There are two trace formats: old traces and new traces. Table 4.1 gives that the detail explanation for new-trace format. Here is the example of output new trace lines.

```
s -t 1.000000000 -Hs 0 -Hd -2 -Ni 0 -Nx 200.00 -Ny 297.00 -Nz 0.00 -Ne -1.000000 -Nl AGT -Nw ---
-Ma 0 -Md 0 -Ms 0 -Mt 0 -Is 0.0 -Id 1.0 -It tcp -Il 40 -If 0 -Ii 0 -Iv 32 -Pn tcp -Ps 0 -Pa 0 -Pf 0 -Po 0
```

Program : 4.6 //~**ns/trace/cmu-trace.h**

```
class CMUTrace: public Trace {
   Private:
void format_part(Packet *p, int offset);
```

Program : 4.7 //~**ns/trace/cmu-trace.cc**

```
void
CMUTrace::format(Packet* p, const char *why)
{
    case PT_AODV:
       break;
    case PT_PART:
       break;
default:
}
```

In this thesis, the AWK scripts (Robbins, 2001) are utilized to analyze output trace files. A graph generation tool (i.e. gnuplot) is used to generate output graphs automatically (Janert, 2009).

Table 4.1 : New trace format explanation

1	2	3	4	5	6	7	8	9	10	11
eve nt	**Time**	**Node info:**	**Node ID**	**Node coordinates**	**Node energy**	**Node trace level**	**Reasons for event**	**Packets info: at MAC**	**Packets info: at IP**	**Protocol s Types**
s r d f	t	Hs Hd	Ni	Nx Ny Nz	Ne	Nl	Nw	Ma Md Ms Mt	Is Id It Il If Ii Iv	Vary on Protocol s

1	s, r, d, f	send, receive, drop, forward	
2	t	time	
3	Hs, Hd	id for this node, id for next hop towards the destination	
4	Ni	Node id	
5	Nx, Ny, Nz	X, Y, Z coordinates of node	
6	Ne	Node energy level	
7	Nl	AGT – Agent trace (i.e. TCP or CBR)	
		RTR – Router trace (i.e. AODV, DSR or DSDV)	
		MAC – MAC trace (i.e. RTS, CTS, ARP)	
8	Nw	"END" →	DROP_END_OF_SIMULATION
		"COL" →	DROP_MAC_COLLISION
		"DUP" →	DROP_MAC_DUPLICATE
		"ERR" →	DROP_MAC_PACKET_ERROR
		"RET" →	DROP_MAC_RETRY_COUNT_EXCEEDED
		"STA" →	DROP_MAC_INVALID_STATE
		"BSY" →	DROP_MAC_BUSY
		"NRTE" →	DROP_RTR_NO_ROUTE (i.e no route is available)
		"LOOP" →	DROP_RTR_ROUTE_LOOP (i.e there is a routing loop)
		"TTL" →	DROP_RTR_TTL (i.e TTL has reached zero)
		"TOUT" →	DROP_RTR_QTIMEOUT (i.e packet has expired)
		"CBK" →	DROP_RTR_MAC_CALLBACK
		"IFQ" →	DROP_IFQ_QFULL (i.e no buffer space in IFQ)
		"ARP" →	DROP_IFQ_ARP_FULL (i.e dropped by ARP)
		"OUT" →	DROP_OUTSIDE_SUBNET (i.e dropped by base stations on receiving routing updates from nodes outside its domain)
9	Ma	Duration	
	Md	Destination's Ethernet address	
	Ms	Source's Ethernet address	
	Mt	Ethernet type	
10	Is	Source address. Source port number	
	Id	Destination address. Destination port number	
	It	Packet type	
	Il	Packet size	
	If	Flow id	
	Ii	Unique id	
	Iv	TTL value	
11	Parp	Po	ARP Request/Reply
		Pm	Source MAC Address
		Ps	Source address
		Pa	Destination MAC address
	Paodv	Pd	Destination address
		Pt	packet type
		Ph	hop count
		Pb	Broadcase ID
		Pd	Destination
		Pds	Destination Sequence Number
		Ps	Source
	Pcbr	Pss	Source Sequence Number
		Pl	Lifetime
		Pc	Operation (REQUEST, REPLY, ERROR, HELLO)
	Ptcp		
		Pi	Sequence number
		Pf	how many times this packet was forwarded
		Po	optimal number of forwards
		Ps	sequence number
		Pa	ack number
		Pf	how many times this packet was forwarded
		Po	optimal number of forwards

4.8.4.4 Priority Queue

The priority queue defines routing packets as high priority packets by inserting them at the beginning of the queue. Therefore, the new packet type, *part*, is added in the queue/priqueue.cc to be treated as high priority as coded in line 11 of Program 4.8.

Program : 4.8 //**~ns/queue/pri-queue.cc**

```
1:   void
2:   PriQueue::recv(Packet* p, Handler *h) {
3:     struct hdr_cmn *ch = HDR_CMN(p);
4:      If(Prefer_Routing_Protocols) {
5:         switch(ch->ptype()) {
6:             case PT_DSR:
7:             case PT_MESSAGE:  #DSDV
8:             case PT_AODV:
9:             case PT_AOMDV:
10:            case PT_PART:
11:        recvHighPriority(p, h);
12:    break;
13:    default:
14:       Queue::recv(p,h)
15:          }
16:  }
```

4.8.4.5 Makefile

Before compiling a new protocol into the compiler, the *Makefile* is edited by adding the object files of *part* (lines 10 and 11) of Program 4.9 inside OBJ_CC variable of NS. These are the basic procedures to implement a new protocol in the simulator.

Program : 4.9 //**~ns/Makefile**

```
1:   OBJ_CC = \
2:       tools/random.o tools/rng.o tools/ranvar.o
3:       common/misc.o common/timer-handler.o \
4:        …
5:   aodv/aodv_logs.o aodv/aodv.o \
6:   aodv/aodv_rtable.o aodv/aodv_rqueue.o \
7:   aomdv/aomdv_logs.o aomdv/aomdv.o \
8:   aomdv/aomdv_rtable.o aomdv/aomdv_rqueue.o \
9:   …
10:  part/part_logs.o part/part.o \
11:  part/part_rtable.o part/part_rqueue.o \
12:  …
13:  $(OBJ_STL)
```

We are going to explain how to implement the PART protocol in the simulator in the following section. The PART protocol has to maintain an internal state, such as a new class inside the routing agent, a routing table and control packet types.

4.8.5 Packet Header Declaration

The PART packet types are coded in lines 4-6 of Program 4.10. All data structures, constants and macro lines 7-11 of Program 4.10 are put to a new packet type. Lines 12-20 of Program 4.10 declare the PART header format that can be shared among the packet types of PART.

Lines 22-35 of Program 4.10 are an example of request packet format (RREQ).

nsaddr_t	Every time the network address needs to be declared in ns2.
u_int32_t	32 bits unsigned integer.
u_int8_t	8 bits unsigned integer.
Double	Floating arithmetic for timer

Program : 4.10 //~**ns/part/part_packet.h**

```
1:      #ifndef __part_packet.h__
2:      #define __part_packet_h__
3:      /* Packet formats */
4:      #define PARTTYPE_RREQ          0x02
5:      …
6:      #define PARTTYPE_PINFORM   0x12
7:      /* Header Macros */
8:      #define HDR_PART(p)((struct hdr_part*)hdr_part::access(p))
9:      …
10:     #define HDR_PART_REQUEST(p)((struct
11:                   hdr_part_request*)hdr_part::access(p))
12:     /* General PART Header which is shared by all formats */
13:       struct hdr_part {
14:          u_int8_t       ah_type;
15:          static int offset_;
16:          inline static int& offset() { return offset_; }
17:          inline static hdr_part* access(const Packet* p) {
18:          return (hdr_part*) p->access(offset_);
19:           }
20:         };
21:       …
22:       /* example of packet type */
23:       struct hdr_part_request {
24:          nsaddr_t   rq_src;  //Node which originated the packets
```

```
    u_int32_t  rq_seqno; //Packet sequence number
    u_int8_t   rq_hop_count; //The number of hops
    Double     rq_timestamp;
    …
inline int size() {
int sz = 0;
sz = size*sizeof(u_int32_t);
assert(sz>=0);
return sz;
};
};
…
union hdr_all_part { // for size calculation for header
hdr_part ah;
};

#endif
```

4.8.6 Routing Table Implementation

The main responsibility of a routing table is to maintain up-to-date routing information. When a node receives a PART control packet from a neighbor, it checks its routing table for an entry for the destination. If there is no entry, a new entry must be created.

Program : 4.11 **//~ns/part/part_rtable.h**

```
#ifndef __part_rtable_h__
#define __part_rtable_h__
class part_rt_entry {

public:
    part_rt_entry();
    void nb_insert(nsaddr_t id);
    void pc_insert(nsaddr_t id);
      PART_Precursor* pc_lookup(nsaddr_t id);
    void        pc_delete(nsaddr_t id);
    void        pc_delete(void);
      double    rt_req_timeout;
      u_int8_t  rt_flags;
      nsaddr_t rt_rep_downstream; //Unicast P-INFORM
…
protected:
      LIST_ENTRY(part_rt_entry) rt_link;
      nsaddr_t   rt_dst;
      u_int32_t  rt_seqno;
      u_int16_t  rt_hops;         // hop count
      Nsaddr_t   rt_padd;  // proxy address
}
```

According to the information contained in the control packet, the sequence number is determined to ensure up-to-date routes. It is also necessary to maintain a list of precursors (i.e. the neighborhood information of nodes) in the routing table entry.

Program : 4.12 //**~ns/part/part_rtable.cc**

```
part_rt_entry::part_rt_entry()
{
 int i;
   rt_req_timeout = 0.0;
   rt_rep_downstream = 0;
   rt_dst = 0;
   rt_seqno = 0;
   rt_hops = rt_last_hop_count = INFINITY2;
   rt_nexthop = 0;
   rt_padd = 0;
   …
  LIST_INIT(&rt_pclist);
   rt_expire = 0.0;
   rt_flags = RTF_DOWN;
}
```

In part_rtable.h, lines 7-11 of Program 4.11 are for inserting and deleting the neighbors. Lines 16-22 of Program 4.11 are examples of routing table entries, such as the timeouts, flags, destination (dst), sequence number, hops, proxy address (padd) etc. In part_rtable.cc, the initial values are assigned for each routing entry shown in Program 4.12. After that, a classifier-port that classifies the packets is declared. In Program 4.13, function classify (p) returns the destination port number of the IP header of the incoming packet p.

Program : 4.13 //**~ns/classifier/classifier_port.cc**

```
int PortClassifier::classify(Packet *p)
{
hdr_ip* iph = hdr_ip::access(p);
return iph->dport();
}
```

4.8.7 PART Agent

The PART agent performs functions for the proxy local repair, local acknowledgement for TCP packets, adaption of route between source and proxy; or proxy and destination vice versa. The necessary functions and headers are declared in the *part.h* of the *part* directory of ns in Program 4.14.

Program : 4.14 **//~ns/part/part.h**

```
#include <cmu-trace.h>
#include <priqueue.h>
#include <part/part_rtable.h>
#include <part/part_rqueue.h>
#include <classifier/classifier-port.h>
…
…
#define PART_PROXY_REPAIR
#define RREP_WAIT_TIME        1.0  // sec
#define RREP_P_WAIT_TIME      0.5        // sec
#define PRO_RESET_TIME         0.5   // sec
…
class  PARTProxyRepairTimer: public Handler {
public:
    PARTProxyRepairTimer(PART *a): agent(a) {}
        void handle(Event*);
private:
    PART *agent;
    Event intr;
};
…
void  proxy_calculation(Packet *p);
void  forward(part_rt_entry *rt, Packet *p,
            double delay);
void  sendHello(void);
void  sendRequest(nsaddr_t dst, nsaddr_t padd,
                u_int8_t phc, u_int8_t flags);
void  sendReply(nsaddr_t ipdst, u_int32_t
              hop_count, u_int32_t phop_count,
              nsaddr_t rpdst, u_int32_t rpseq,
              nsaddr_t padd, u_int32_t lifetime,
              double timestamp, u_int8_t flags);
void  sendError(Packet *p, u_int8_t flags,
              bool jitter = true);
void        sendProxyInform(nsaddr_t dst);
…
void  recvPART(Packet *p);
void  recvHello(Packet *p);
void  recvRequest(Packet *p);
void  recvReply(Packet *p);
void  recvError(Packet *p);
void  recvProxyInform_D(Packet *p);
…
PortClassifier *dmux_;
};
```

The proxy_calculation function (line 23 of Program 4.14) is called to determine whether a proxy node should be used. On the other hand, the *Timer* is required for the controlled packets that are sent periodically or after some events have occurred. Lines 14-21 from Program 4.14 are the codes for the PARTProxyRepairTimer, which is used while a proxy node is trying to repair a broken route.

4.8.7.1 Tcl Hooks

The packet header of *part* in Program 4.10 (Section 4.8.5) has to be found to a Tcl interface. Tcl (Tool control language) is used for writing the protocol testing codes, such as random topology, grid topology, chain topology, etc. To access Tcl scripts for testing scenarios, the offset of the packet header and a portion of *part.cc*, Program 4.15, performs such functions.

Program : 4.15 //**~ns/part/part.cc**

```
int hdr_part::offset_;
static class PARTHeaderClass : public PacketHeaderClass {
public:
   PARTHeaderClass():PacketHeadeClass("PacketHeader/PART",
                              sizeof(hdr_all_edtp)) {
   bind_offset(&hdr_edtp::offset_)
     }
}
Class_rtProtoPART_hdr;
```

4.8.7.2 Cross-layer Communication

The unicast transmission for P-INFORM packet is done by taking the address of the upstream node from MAC layer while the unicast RREP packet is sent to the downstream node. Therefore, the mac-802.11 header from the MAC layer needs to be declared in *part.cc*. When a node receives a unicast RREP-NoFlag packet, it looks at the MAC layer destination address in the packet so that the routing table (*rt_rep_downstream*) can be updated. Lines 10-15 in Program 4.16 are the updating procedure of the routing table.

Program : 4.16 **//~ns/part/part.cc**

```
1:        #include <mac-802_11.h>
2:        …
3:        void
4:        PART::recvReply(Packet *p) {
5:        struct hdr_ip *ih = HDR_IP(p);
6:        struct hdr_part_reply *rp = HDR_PART_REPLY(p);
7:        struct hdr_mac802_11 *mh = HDR_MAC802_11(p);
8:        part_rt_entry *rt;
9:        If(rp->rp_flags == 0x00) { // RREP-NOFlag
10:       // to update routing table with MAC layer address
11:       rt_update(rt, rp->rp_dst_seqno, rp->rp_hop_count,
12:               rp->rp_phop_count, rp->rp_src, index,
13:               (ETHER_ADDR(mh->dh_ta)), rt->rt_datadst,
14:                rt->rt_datasrc, CURRENT_TIME+rp->rp_lifetime);
15:       …
16:       }
```

4.9 Statistical Analysis

Statistical analysis is used to compare the sample and population means to ascertain whether there is a significant difference. The most useful statistical tests are z-test and t-test. The z-test is applied to analyze in large sample sizes ($n > 30$), whereas the t-test is the best to apply in a small sample sizes ($n < 30$). Every hypothesis is stated as a null hypothesis and an alternative hypothesis. If the sample mean has many standard deviations from the mean stated in the null hypothesis, the null hypothesis is rejected. If the results observed in the sample are not under the assumption of the null hypothesis, the result is statistically significant.

Suppose that we want to conduct a hypothesis test to determine whether the average output results for PART is significantly different from the AODV routing protocol.

1. Null hypothesis

   H_0: $\mu_1 = \mu_2$ (H_0: PART = AODV)

2. Alternative hypothesis

   H_a: $\mu_1 < \mu_2$ (Ha: PART < AODV)

   For throughput, PART does not perform as well as AODV.

   For average end-to-end delay, PART performs better than AODV.

3. Test statistics

$$z = \frac{\bar{X} - \mu_0}{\frac{s}{\sqrt{n}}} \quad or \quad t = \frac{\bar{X} - \mu_0}{\frac{\sigma}{\sqrt{n}}}$$

4. Critical value

   $\alpha = 0.01$ (1%)

   $\alpha = 0.05$ (5%)

   $z = -z_\alpha$ or $t = -t_\alpha$

5. Decision Rule

   If $z < -z_\alpha$ or $t < -t_\alpha$; Reject H_0.

   Otherwise Accept H_0.

4.10 Experiments on the Effects of Node Movements

Objective of experiment: To test the efficiency of a proxy node in the situation where a source node moves towards a proxy node; a proxy node moving towards a source node or a destination node; and a destination node moving towards a proxy node during the simulation.

Expected outcome: The PART protocol tolerates the effects of node movements in MANETs, in terms of throughput, delay and overhead. Although the occurrences of proxy changes could be more often, the PART could adapt to find a new proxy or give up a proxy duty depending on the movement of nodes.

For this purpose, we simulate 11 nodes in a simulation area of 1200 × 600 for 120 seconds. A TCP connection starts transmitting from a source to a destination one second after the simulation starts. During the simulation, each node (i.e. a source or a destination) starts moving towards the proxy node after 50 seconds at high speed 50 m/s and moderate speeds between 1 to 15 m/s. Later, the proxy node moves towards a source or a destination node. Figure 4.10 shows the possibility of proxy nodes between a source and destination pair while the source node (i.e. node 10) is communicating to the destination node (i.e. node 8).

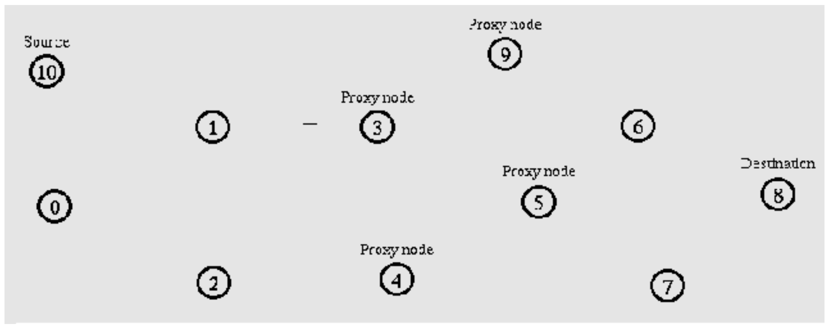


Figure 4.10 : Possibilities of proxy nodes for a given topology

We measure the following parameters as performance metrics and evaluate the performance of protocols.

1. Packet Loss Rate (%) — It is the number of packet loss at the application layer.

$$PLR = \frac{Dropped\ Packets}{Highest\ Packet\ ID + 1} \times 100$$

2. Average end-to-end delay (Avg. EtE delay) is the sum of delay experienced by each packet making up the flow per number of packets.
3. Packet delivery fraction (PDF) is the ratio of the packets delivered to the destinations to those generated from the sources.

$$PDF = \frac{Total\ number\ of\ received\ packets}{Total\ number\ of\ send\ packets} \times 100$$

4. Routing Overhead is calculated as the total number of control packets sent divided by the number of data packets delivered successfully.
5. Normalized Routing Load (NRL) is the number of routing packets transmitted per data packet delivered at the destination.

$$NRL = \frac{Total\ number\ of\ routing\ packets}{Total\ number\ of\ received\ packets}$$

6. Throughput is the rate of successfully delivered data per second to individual destinations during the network simulation.

4.10.1 Source Node Movement

Figure 4.11 demonstrates the movement of a source node. Even though the source node moves towards the destination node through the proxy nodes, it can communicate with the destination without using any proxy node for a path length that is shorter than 3 hops.

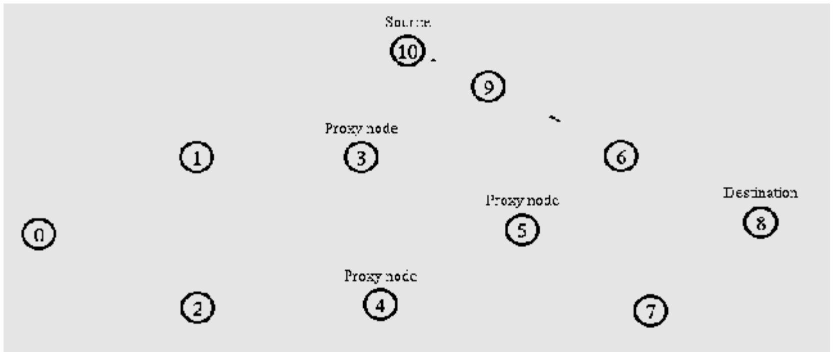


Figure 4.11 : Source node movement

Figure 4.12 shows the throughput measurement of PART and AODV. PART achieves almost 29% higher throughput than AODV. If we look at the Table 4.2, PART reduces the average end-to-end delay almost 17% lower than AODV and 72% lower than DSR. The packet loss rate is 16% lower than AODV, and the routing overhead is almost 22% significantly lower than AODV. If we look at the PDF, the performance differences between AODV and PART are not significant, whereas PART suffers slightly lower PDF if compared to DSR. Tables 4.3 to 4.8 show the z-test and t-test results, where the improvements are statistically significant.

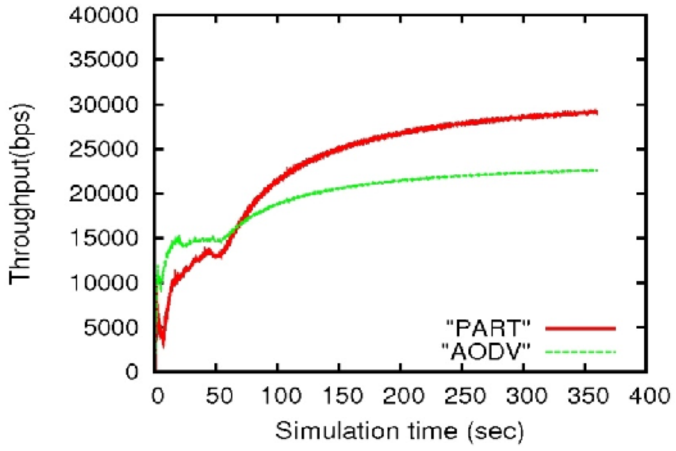


Figure 4.12 : Throughput measurement for source node movement

Table 4.2 : Performance measurement for source node movement

Node Speed / Metrics	PART			AODV			DSR		
	5m/s	**10m/s**	**15m/s**	**5m/s**	**10m/s**	**15m/s**	**5m/s**	**10m/s**	**15m/s**
Average delay	429.32	442.71	465.63	521.89	457.23	581.97	758.78	761.49	779.19
Dropped pkts	160	130	99	179	170	117	16	17	17
Pkt loss (%)	1.29	1.26	0.74	1.73	0.96	1.11	0.08	0.09	0.1
PDF	97.4	97.46	98.48	96.56	98.05	97.77	99.64	99.65	99.64
NRL	0.49	0.39	0.26	0.65	0.33	0.4	0.26	0.2	0.17

Table 4.3 : Throughput measurement across source node movement

z-Test: Two Sample for Means		
	AODV	***PART***
Mean	2.0E+04	2.5E+04
Known Variance	7.8E+06	2.7E+07
Observations	5.3E+03	6.8E+03
Hypothesized Mean Difference	0.0E+00	
Z	-6.0E+01	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

Table 4.4 : Average end-to-end delay measurement across source node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	4.0E+02	4.4E+02
Variance	1.4E+04	2.4E+04
Observations	5.0E+00	5.0E+00
Pooled Variance	1.9E+04	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-4.9E-01	
P(T<=t) one-tail	3.2E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	6.4E-01	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	4.0E+02	7.4E+02
Variance	1.4E+04	2.6E+03
Observations	5.0E+00	5.0E+00
Pooled Variance	8.1E+03	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-6.1E+00	
P(T<=t) one-tail	1.5E-04	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	3.0E-04	
T Critical Two-tail	2.3E+00	

Table 4.5 : Dropped packets measurement across source node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	1.9E+02	2.2E+02
Variance	1.9E+04	2.7E+04
Observations	5.0E+00	4.0E+00
Pooled Variance	2.2E+04	
Hypothesized Mean Difference	0.0E+00	
df	7.0E+00	
t Stat	-3.1E-01	
P(T<=t) one-tail	3.8E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	7.7E-01	
T Critical Two-tail	2.4E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	1.9E+02	1.6E+01
Variance	1.9E+04	1.3E+01
Observations	5.0E+00	5.0E+00
Pooled Variance	9.6E+03	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	2.7E+00	
P(T<=t) one-tail	1.3E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	2.5E-02	
T Critical Two-tail	2.3E+00	

Table 4.6 : Packet losses measurement across source node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	1.9E+00	2.0E+00
Variance	3.8E+00	3.8E+00
Observations	5.0E+00	5.0E+00
Pooled Variance	3.8E+00	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-1.0E-01	
P(T<=t) one-tail	4.6E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	9.2E-01	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	1.9E+00	8.4E-02
Variance	3.8E+00	1.1E-03
Observations	5.0E+00	5.0E+00
Pooled Variance	1.9E+00	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	2.1E+00	
P(T<=t) one-tail	3.6E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	7.2E-02	
T Critical Two-tail	2.3E+00	

Table 4.7 : PDF measurement across source node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	9.6E+01	9.6E+01
Variance	1.4E+01	1.3E+01
Observations	5.0E+00	5.0E+00
Pooled Variance	1.3E+01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	9.8E-02	
P(T<=t) one-tail	4.6E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	9.2E-01	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	1.9E+00	8.4E-02
Variance	3.8E+00	1.1E-03
Observations	5.0E+00	5.0E+00
Pooled Variance	1.9E+00	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	2.1E+00	
P(T<=t) one-tail	3.6E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	7.2E-02	
T Critical Two-tail	2.3E+00	

Table 4.8 : NRL measurement across source node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	7.3E-01	7.5E-01
Variance	7.3E-01	5.9E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	6.6E-01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-5.5E-02	
P(T<=t) one-tail	4.8E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	9.6E-01	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	7.3E-01	3.9E-01
Variance	7.3E-01	1.9E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	4.6E-01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	7.8E-01	
P(T<=t) one-tail	2.3E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	4.6E-01	
T Critical Two-tail	2.3E+00	

4.10.2 Proxy Node Movement

Figure 4.13 illustrates the movement conditions of the proxy node. As the proxy node moves rapidly and randomly for a short period, PART suffers a slight throughput degradation – almost 6% lower than AODV (Figure 4.14). In Table 4.9, PART reduces the average end-to-end delay by almost 8% lower than AODV and 222% lower than DSR. However, PART suffers a slightly lower degradation compared to AODV and DSR in terms of dropped packets and PDF. Tables 4.10 to 4.15 show the z-test and t-test results, where the improvements are statistically significant in all situations, except for the throughput.

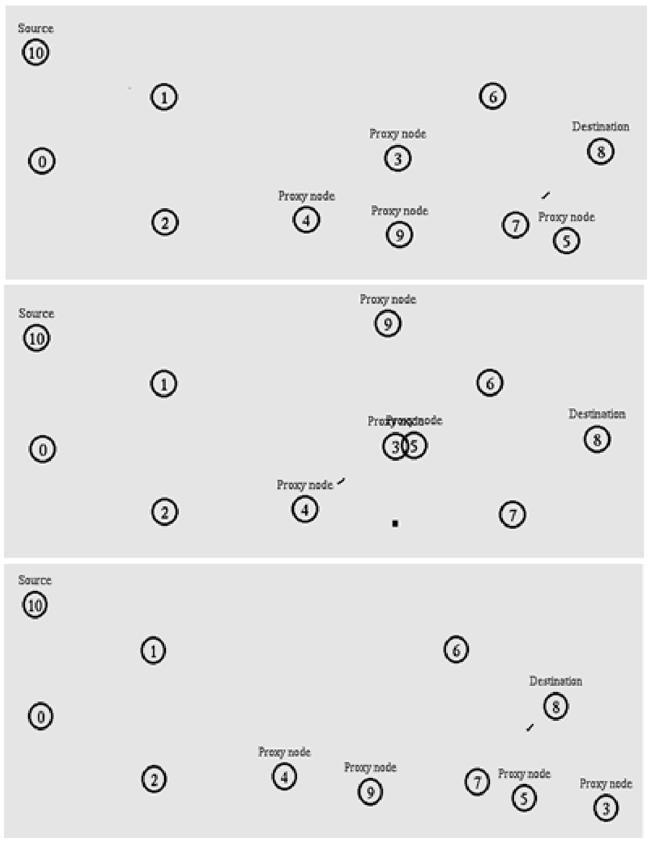


Figure 4.13 : Movements of proxy nodes

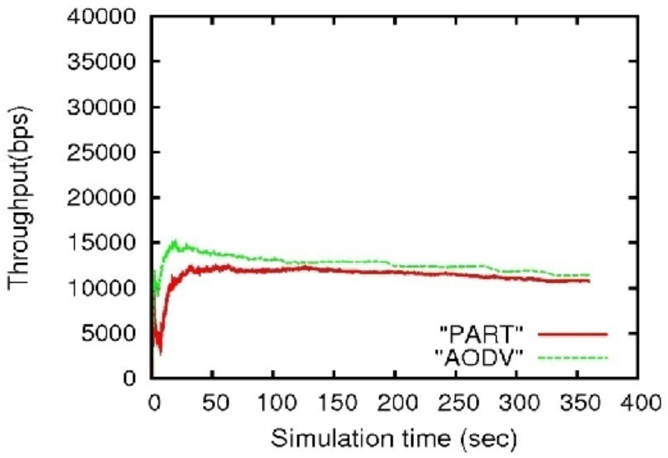


Figure 4.14 : Throughput measurement for movements of proxy nodes

Table 4.9 : Performance measurement for proxy node movement

Node Speed / Metrics	PART			AODV			DSR		
	5m/s	**10m/s**	**15m/s**	**5m/s**	**10m/s**	**15m/s**	**5m/s**	**10m/s**	**15m/s**
Average delay	188.93	199.32	196.09	193.25	235.1	201.51	634.22	619.72	624.94
Dropped pkts	300	291	297	326	260	281	30	31	29
Pkt loss (%)	5.55	5.4	6.02	5.56	5.05	4.81	0.1	0.1	0.09
PDF	89.43	89.76	88.61	89.47	90.35	90.76	98.52	98.25	98.42
NRL	3.92	4.31	4.13	3.43	3.79	3.73	2.37	2.52	2.51

Table 4.10 : Throughput measurement across proxy node movement

z-Test: Two Sample for Means		
	AODV	***PART***
Mean	1.1E+04	1.3E+04
Known Variance	5.8E+07	1.0E+06
Observations	2.5E+03	2.7E+03
Hypothesized Mean Difference	0.0E+00	
z	-8.5E+00	
P(Z<=z) one-tail	0.0E+00	
z Critical one-tail	1.6E+00	
P(Z<=z) two-tail	0.0E+00	
z Critical two-tail	2.0E+00	

Table 4.11 : Average end-to-end delay measurement across proxy node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	1.9E+02	2.4E+02
Variance	1.9E+01	4.1E+03
Observations	5.0E+00	5.0E+00
Pooled Variance	2.1E+03	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-1.6E+00	
P(T<=t) one-tail	7.5E-02	
T Critical one-tail	1.4E+00	
P(T<=t) two-tail	1.5E-01	
T Critical Two-tail	1.9E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	1.9E+02	6.4E+02
Variance	1.9E+01	8.2E+02
Observations	5.0E+00	5.0E+00
Pooled Variance	4.2E+02	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-3.5E+01	
P(T<=t) one-tail	2.6E-10	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	5.2E-10	
T Critical Two-tail	2.3E+00	

Table 4.12 : Dropped packets measurement across proxy node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	3.1E+02	2.7E+02
Variance	4.8E+02	3.0E+03
Observations	5.0E+00	5.0E+00
Pooled Variance	1.7E+03	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	1.6E+00	
P(T<=t) one-tail	7.2E-02	
T Critical one-tail	1.4E+00	
P(T<=t) two-tail	1.4E-01	
T Critical Two-tail	1.9E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	3.1E+02	3.0E+01
Variance	4.8E+02	8.0E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	2.4E+02	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	2.9E+01	
P(T<=t) one-tail	1.2E-09	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	2.3E-09	
T Critical Two-tail	2.3E+00	

Table 4.13 : Packet losses measurement across proxy node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	5.7E+00	5.2E+00
Variance	5.3E-02	1.3E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	9.3E-02	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	2.3E+00	
P(T<=t) one-tail	2.4E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	4.9E-02	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	5.7E+00	9.4E-02
Variance	5.3E-02	3.0E-05
Observations	5.0E+00	5.0E+00
Pooled Variance	2.6E-02	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	5.4E+01	
P(T<=t) one-tail	7.3E-12	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	1.5E-11	
T Critical Two-tail	2.3E+00	

Table 4.14 : PDF measurement across proxy node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	8.9E+01	9.0E+01
Variance	1.8E-01	4.1E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	3.0E-01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	2.4E+00	
P(T<=t) one-tail	2.2E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	4.4E-02	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	8.9E+01	9.8E+01
Variance	1.8E-01	1.0E-02
Observations	5.0E+00	5.0E+00
Pooled Variance	9.4E-02	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	4.7E+01	
P(T<=t) one-tail	2.2E-11	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	4.4E-11	
T Critical Two-tail	2.3E+00	

Table 4.15 : NRL measurement across proxy node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	4.0E+00	3.6E+00
Variance	5.2E-02	6.2E-02
Observations	5.0E+00	5.0E+00
Pooled Variance	5.7E-02	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	2.8E+00	
P(T<=t) one-tail	1.2E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	2.3E-02	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	4.0E+00	2.4E+00
Variance	5.2E-02	4.7E-03
Observations	5.0E+00	5.0E+00
Pooled Variance	2.9E-02	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	1.5E+01	
P(T<=t) one-tail	2.5E-07	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	5.0E-07	
T Critical Two-tail	2.3E+00	

4.10.3 Destination Node Movement

We consider the situation of destination node's movement as shown in Figure 4.15. If we look at the throughput, PART does not perform very well at the beginning of the simulation time, but outperforms the others significantly starting from around 220 seconds as shown in Figure 4.16.

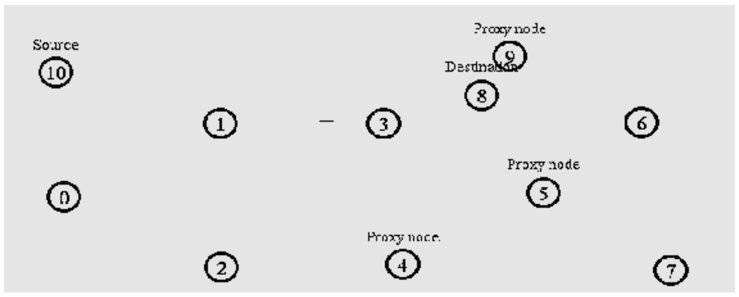


Figure 4.15 : Destination node movement

When the destination node moves towards the source node, the distance between the source and destination is shorter. Therefore, PART routing protocol operates exactly like AODV at the beginning. However, as the simulation time progresses, the possibility of route failures becomes higher. In this situation, the proxy node is able to redirect the broken routes quickly, resulting in higher throughput at time 220 s. If we compare the average throughput throughout the simulation time, PART incurs 18% higher throughput on average compared to AODV.

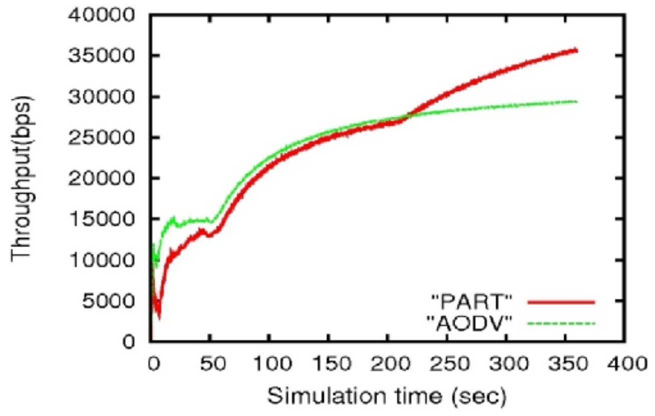


Figure 4.16 : Throughput measurement for destination node movement

Table 4.16 shows that although PART reduces the average end-to-end delay up to 4% lower than AODV and 25% lower than DSR, there is a little difference for other parameters such as PDF and packet losses. PART allows the nodes to broadcast RREQ-Pflag packets until the proxy node. The proxy replies to the source after receiving the packet. In this way, the proxy node reduces overhead lower than AODV. The z-test and t-test results in Table 4.17 and Table 4.11 show that the performance improvement is statistically significant.

Table 4.16 : Performance measurement for destination node movement

Node Speed / Metrics	PART			AODV			DSR		
	5m/s	**10m/s**	**15m/s**	**5m/s**	**10m/s**	**15m/s**	**5m/s**	**10m/s**	**15m/s**
Average delay	433.15	464.9	466.29	435.87	477.29	496.33	560.31	576.99	562.89
Dropped pkts	184	137	123	201	136	137	16	16	20
Pkt loss (%)	1.45	1.31	0.91	1.57	1.02	1.01	0.06	0.07	0.08
PDF	97.14	97.4	98.16	96.88	97.97	97.97	99.74	99.75	99.74
NRL	0.48	0.28	0.25	0.53	0.38	0.3	0.21	0.15	0.12

Table 4.17 : Throughput measurement across destination node movement

z-Test: Two Sample for Means		
	AODV	*PART*
Mean	2.7E+04	8.8E+04
Known Variance	6.4E+08	4.3E+08
Observations	8.4E+03	6.9E+03
Hypothesized Mean Difference	0.0E+00	
z	-1.6E+02	
P(Z<=z) one-tail	0.0E+00	
z Critical one-tail	1.6E+00	
P(Z<=z) two-tail	0.0E+00	
z Critical two-tail	2.0E+00	

Table 4.18 : Average end-to-end delay measurement across destination node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	*AODV*
Mean	3.8E+02	3.9E+02
Variance	1.5E+04	1.9E+04
Observations	5.0E+00	5.0E+00
Pooled Variance	1.7E+04	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-6.9E-02	
P(T<=t) one-tail	4.7E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	9.5E-01	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	*DSR*
Mean	3.8E+02	5.7E+02
Variance	1.5E+04	4.1E+01
Observations	5.0E+00	5.0E+00
Pooled Variance	7.7E+03	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-3.4E+00	
P(T<=t) one-tail	4.9E-03	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	9.8E-03	
T Critical Two-tail	2.3E+00	

Table 4.19 : Dropped packets measurement across destination node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	*AODV*
Mean	2.2E+02	2.4E+02
Variance	3.4E+04	3.9E+04
Observations	5.0E+00	5.0E+00
Pooled Variance	3.7E+04	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-1.1E-01	
P(T<=t) one-tail	4.6E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	9.1E-01	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	*DSR*
Mean	2.2E+02	2.4E+02
Variance	3.4E+04	3.9E+04
Observations	5.0E+00	5.0E+00
Pooled Variance	3.7E+04	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-1.1E-01	
P(T<=t) one-tail	4.6E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	9.1E-01	
T Critical Two-tail	2.3E+00	

Table 4.20 : Packet losses measurement across destination node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	2.2E+02	1.5E+01
Variance	3.4E+04	1.3E+01
Observations	5.0E+00	5.0E+00
Pooled Variance	1.7E+04	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	2.5E+00	
P(T<=t) one-tail	1.8E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	3.6E-02	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	2.1E+00	2.1E+00
Variance	4.8E+00	5.6E+00
Observations	5.0E+00	5.0E+00
Pooled Variance	5.2E+00	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-8.3E-03	
P(T<=t) one-tail	5.0E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	9.9E-01	
T Critical Two-tail	2.3E+00	

Table 4.21 : PDF measurement across destination node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	9.6E+01	9.6E+01
Variance	1.7E+01	1.9E+01
Observations	5.0E+00	5.0E+00
Pooled Variance	1.8E+01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	7.5E-04	
P(T<=t) one-tail	5.0E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	1.0E+00	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	9.6E+01	1.0E+02
Variance	1.7E+01	3.8E-02
Observations	5.0E+00	5.0E+00
Pooled Variance	8.4E+00	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-2.1E+00	
P(T<=t) one-tail	3.6E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	7.3E-02	
T Critical Two-tail	2.3E+00	

Table 4.22 : NRL measurement across destination node movement

t-Test: Two-Sample Assuming Equal Variances		
	PART	***AODV***
Mean	6.4E-01	7.1E-01
Variance	5.8E-01	6.3E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	6.0E-01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-1.3E-01	
P(T<=t) one-tail	4.5E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	9.0E-01	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances		
	PART	***DSR***
Mean	6.4E-01	4.0E-01
Variance	5.8E-01	3.0E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	4.4E-01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	5.9E-01	
P(T<=t) one-tail	2.9E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	5.7E-01	
T Critical Two-tail	2.3E+00	

4.10.4 Random Movement of Nodes

Figure 4.17 illustrates the example of random movements for the performance testing. The proxy node, destination node and other nodes move randomly during the simulation. If we measure the performance differences, PART achieves a higher throughput from the beginning till the end of the simulation time as shown in Figure 4.18. Table 4.23 shows that the throughput improvement is statistically significant.

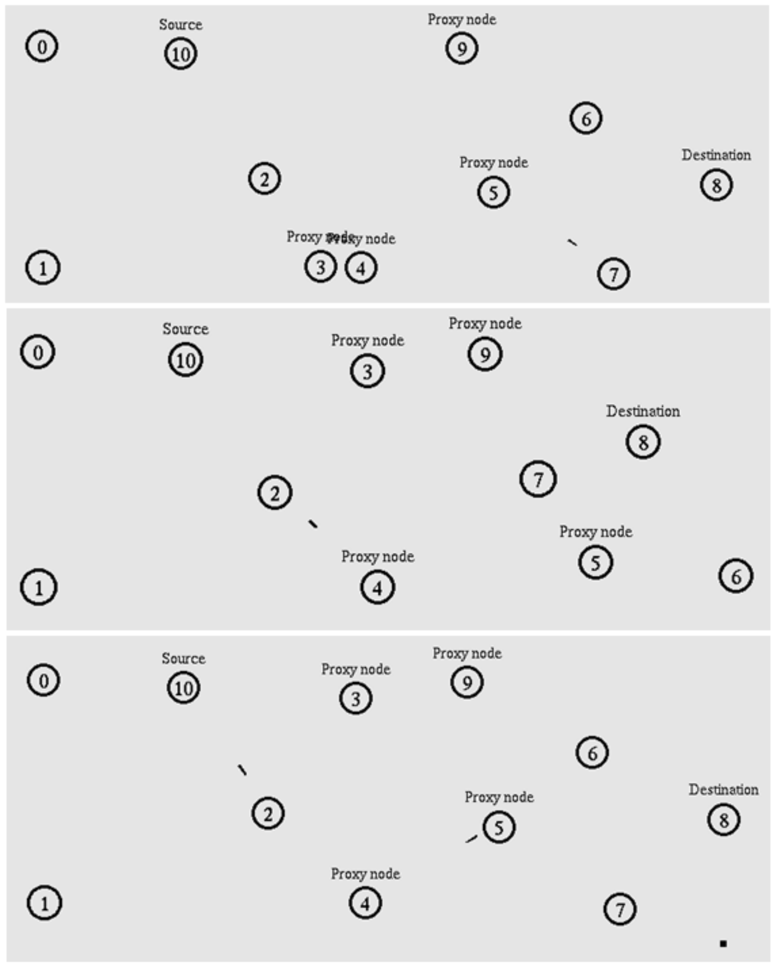


Figure 4.17 : Random movements of nodes

The simulation results are as expected, i.e. the PART protocol tolerates the effects of node (source and destination) movements. However, when the proxy node movement occurs, slight throughput degradation occurs.

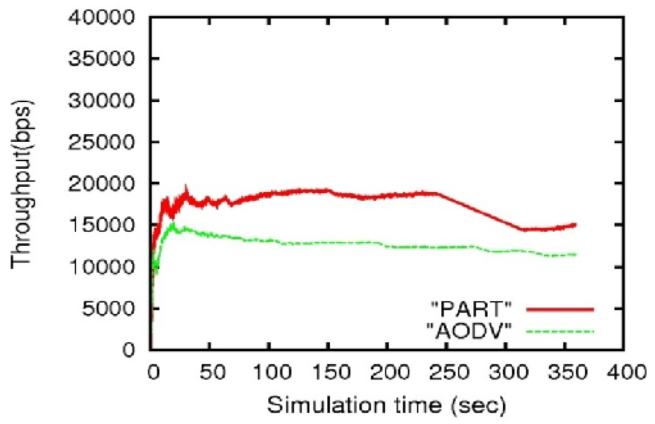


Figure 4.18 : Throughput measurement for random movements of nodes

Table 4.23 : Throughput measurement across random node movement

z-Test: Two Sample for Means (random node movement)		
	AODV	***PART***
Mean	1.3E+04	1.8E+04
Known Variance	6.5E+07	4.3E+08
Observations	2.7E+03	3.5E+03
Hypothesized Mean Difference	0.0E+00	
z	-1.3E+01	
P(Z<=z) one-tail	0.0E+00	
z Critical one-tail	1.6E+00	
P(Z<=z) two-tail	0.0E+00	
z Critical two-tail	2.0E+00	

4.11 Performance Evaluations in the Large-scaled Networks

Objective of experiment: To test the scalability and robustness of PART protocol in large-scaled networks with random topologies.

Expected outcome: Although the node density is varied between 100 nodes to 500 nodes, the PART protocol performs very well in terms of packet losses, average delay, routing overhead and throughput under the large-scaled networks.

The parameters for analysis are listed in Table 4.24. 100 to 500 nodes move randomly within the 2500 × 1500 area at 10m/s speed for about 600 seconds. 70 TCP applications are generated with a default window size of 32 and the packet size is 512 bytes. To generate the random movement, NS-2 contains the mobility generator, called "*setdest*", which is generated using the command *setdest -n <num_of_nodes> -p pause_time –s <max_speed> -t <sim_time> -× <max_×> -y <max_y>* under the NS-2 directory *ns-2/indep-utils/cmu-scen_gen*. To generate the application traffic, "*cbrgen.tcl"* under the directory *ns-2/indep-utils/cmu-scen_gen* is used.

Generation examples are:

For mobility generation → *setdest –n 50 –p 0 –s 10 –t 360 –× 1200 –y 600*
(*50 nodes are moving in the area of 1200 and 600 at 10 m/s speed and no pause time.*)

For traffic generation → *ns cbrgen.tcl –type cbr –nn 50 –seed 1 –mc 20 –rate 4*
(*50 pair of nodes generate the 20 connections at 4 packets per second*)

Table 4.24 : Parameters for large-scaled networks

Parameters	
Topography areas size	2500 × 1500
Simulation time	600 second
Number of nodes	100 nodes to 500 nodes
Routing protocols	PART, AODV, DSR, DSDV, AOMDV, OLSR
Transport protocols	TCP
Number of TCP connections	70 FTPs
Window size of TCP	32
Packet size	512 bytes
Average speed	10 m/s with 0 pauses time
Mac protocol	Mac/802.11
Antenna	OmiAntenna

4.11.1 Packet Loss Rate Measurement

First, we measure the percentage of packet loss rate. Proactive routing protocols achieve the lowest losses compared to reactive protocols. Proactive routing protocols (i.e. DSDV and OLSR) always keep the routing tables and link state information in hand and the source route caching technique of DSR achieves significantly lower losses, even though it is a reactive routing. Among the on-demand routing protocols, PART reduces packet losses 28.70% lower than AODV and 6.90% lower than AOMDV as shown in Figure 4.19. As the number of nodes increases, the packet loss rates of proactive routing protocols decrease gradually, whereas the packet loss rates of reactive routing protocols increase gradually. Table 4.25 shows the t-test results, where packet loss rate improvement is statistically significant over DSDV, DSR and OLSR.

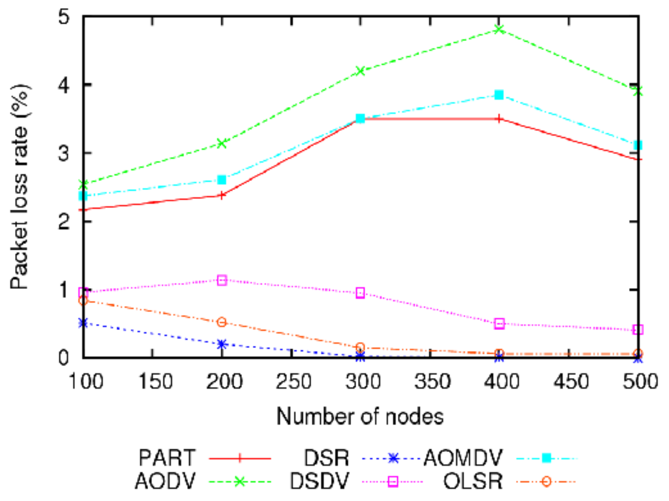


Figure 4.19 : Packet loss rate measurement in random topology

Table 4.25 : Packet losses measurement in large-scaled networks

t-Test: Two-Sample Assuming Equal Variances

	PART	*AODV*
Mean	2.9E+00	3.7E+00
Variance	3.8E-01	7.7E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	5.7E-01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-1.8E+00	
P(T<=t) one-tail	5.7E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	1.1E-01	
T Critical Two-tail	2.3E+00	

	PART	*DSDV*
Mean	2.9E+00	7.9E-01
Variance	3.8E-01	1.0E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	2.4E-01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	6.8E+00	
P(T<=t) one-tail	7.2E-05	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	1.4E-04	
T Critical Two-tail	2.3E+00	

	PART	*DSR*
Mean	2.9E+00	1.5E-01
Variance	3.8E-01	4.8E-02
Observations	5.0E+00	5.0E+00
Pooled Variance	2.1E-01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	9.4E+00	
P(T<=t) one-tail	6.9E-06	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	1.4E-05	
T Critical Two-tail	2.3E+00	

	PART	*AOMDV*
Mean	2.9E+00	3.1E+00
Variance	3.8E-01	3.7E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	3.8E-01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-5.1E-01	
P(T<=t) one-tail	3.1E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	6.2E-01	
T Critical Two-tail	2.3E+00	

	PART	*OLSR*
Mean	2.9E+00	3.3E-01
Variance	3.8E-01	1.2E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	2.5E-01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	8.1E+00	
P(T<=t) one-tail	2.0E-05	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	3.9E-05	
T Critical Two-tail	2.3E+00	

4.11.2 Average Delay Measurement

When we measure the average end-to-end delay, the performance of PART and AOMDV protocols is not so different. AOMDV has a large processing overhead and memory usage for calculating multiple paths at every node, especially in the large-scale networks. On the other hand, the performance of PART is almost similar to AOMDV without having multiple paths resulting in the lower memory usage and energy consumption. Moreover, PART reduces the average delay 23% lower than DSDV and 22% lower than AODV as shown in Figure 4.20. We encounter that DSR performs the worst if compared to the others. In the 300-node scenarios, the average end-to-end delay of DSR is the highest.

A possible explanation for this situation could be the instant use of route caching technique. In the large-scaled networks, the cache size increases significantly due to the large amount of nodes, and also the possibilities of the stale routes increase due to the lack of the detection technique for those stale routes in the route cache of DSR. Table 4.26 shows the t-test results, where delay improvement of PART is statistically significant over AODV and DSR.

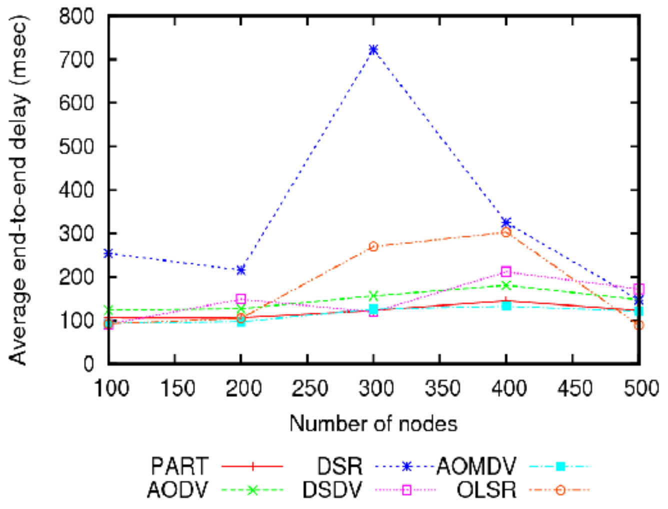


Figure 4.20 : Average delay measurement in random topology

Table 4.26 : Average end-to-end delay measurement in large-scaled networks

t-Test: Two-Sample Assuming Equal Variances
(large-scale networks)

	PART	*AODV*
Mean	1.2E+02	1.5E+02
Variance	2.6E+02	5.4E+02
Observations	5.0E+00	5.0E+00
Pooled Variance	4.0E+02	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-2.1E+00	
P(T<=t) one-tail	3.5E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	7.0E-02	
T Critical Two-tail	2.3E+00	

	PART	*DSDV*
Mean	1.2E+02	1.1E+02
Variance	2.6E+02	3.4E+03
Observations	5.0E+00	5.0E+00
Pooled Variance	1.8E+03	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	3.7E-01	
P(T<=t) one-tail	3.6E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	7.2E-01	
T Critical Two-tail	2.3E+00	

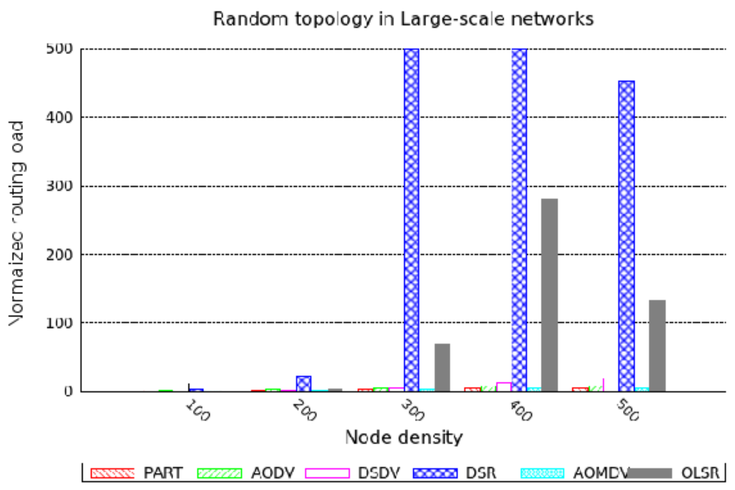
	PART	*DSR*
Mean	1.2E+02	3.3E+02
Variance	2.6E+02	5.2E+04
Observations	5.0E+00	5.0E+00
Pooled Variance	2.6E+04	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-2.1E+00	
P(T<=t) one-tail	3.5E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	7.1E-02	
T Critical Two-tail	2.3E+00	

	PART	*AOMDV*
Mean	1.2E+02	1.1E+02
Variance	2.6E+02	3.1E+02
Observations	5.0E+00	5.0E+00
Pooled Variance	2.9E+02	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	6.4E-01	
P(T<=t) one-tail	2.7E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	5.4E-01	
T Critical Two-tail	2.3E+00	

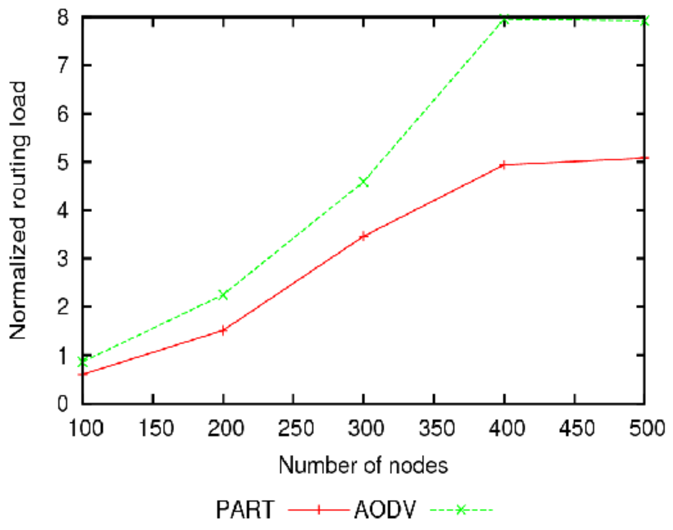
	PART	*OLSR*
Mean	1.2E+02	1.7E+02
Variance	2.6E+02	1.1E+04
Observations	5.0E+00	5.0E+00
Pooled Variance	5.7E+03	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-1.1E+00	
P(T<=t) one-tail	1.6E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	3.2E-01	
T Critical Two-tail	2.3E+00	

4.11.3 Normalized Routing Load Measurement

If we look at the comparison of NRL in Figure 4.21(a), proactive routing protocols (i.e. DSR and OLSR) perform the worst when the number of nodes increases. To ascertain the significance of performance improvement in PART, Figure 4.21(b) compares only PART and AODV. As the number of nodes increases, the NRL of both protocols increases but PART significantly reduces the NRL 51.3% lower than AODV.



(a)



(b)

Figure 4.21 : NRL measurement in random topology

In MANETs, route errors happen all the time. 70% of errors are due to collisions at the MAC layer. Whenever a route error occurs, AODV invokes the route discovery procedure, resulting in increased controlled packets in the network, which in turn causes more congestion and collisions. In PART, if the route error is due to collisions, the proxy node repairs the broken route by broadcasting RREQ-D flag packets to the destination. If the route error is due to a broken route, the source node re-invokes the route discovery procedure by broadcasting RREQ-Pflag packets within the limited zone. Therefore, as the node density increases, PART reduces routing overhead significantly lower than AODV. Table 4.27 shows the t-test results, where NRL improvement is statistically significant over AODV and DSR.

Table 4.27 : NRL measurement in large-scaled networks

t-Test: Two-Sample Assuming Equal Variances (large-scale networks)

	PART	*AODV*
Mean	2.4E+00	4.7E+00
Variance	4.1E+00	1.0E+01
Observations	5.0E+00	5.0E+00
Pooled Variance	7.2E+00	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-9.4E-01	
P(T<=t) one-tail	1.9E-01	
T Critical one-tail	3.2E-00	
P(T<=t) two-tail	3.8E-01	
T Critical Two-tail	2.3E+00	

	PART	*DSDV*
Mean	3.1E+00	7.7E+00
Variance	4.1E+00	7.0E+01
Observations	5.0E+00	5.0E+00
Pooled Variance	3.7E+01	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-1.2E+00	
P(T<=t) one-tail	1.3E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	2.7E-01	
T Critical Two-tail	2.3E+00	

	PART	*DSR*
Mean	3.1E+00	4.2E+02
Variance	4.1E+00	1.6E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	7.8E+04	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-2.3E+00	
P(T<=t) one-tail	2.4E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	4.8E-02	
T Critical Two-tail	2.3E+00	

	PART	*AOMDV*
Mean	3.1E+00	2.7E+00
Variance	4.1E+00	2.8E+00
Observations	5.0E+00	5.0E+00
Pooled Variance	3.4E+00	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	3.3E-01	
P(T<=t) one-tail	3.8E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	7.5E-01	
T Critical Two-tail	2.3E+00	

	PART	*OLSR*
Mean	3.1E+00	9.7E+01
Variance	4.1E+00	1.3E+04
Observations	5.0E+00	5.0E+00
Pooled Variance	6.7E+03	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	-1.8E+00	
P(T<=t) one-tail	5.3E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	1.1E-01	
T Critical Two-tail	2.3E+00	

4.11.4 Throughput Measurement

If we look at the measurement of throughput, PART achieves a significantly better performance than the others. PART performs 4.6 % better than AODV, 4.2 % better than AOMDV, 56.8% better than DSDV, 70.7% better than DSR, and 51.5% better than OLSR. In Figure 4.22, PART can mimic AOMDV in terms of not only average delay and NRL but also throughput without requiring multiple paths at every node. Table 4.28 shows the t-test results, where throughput improvement is more statistically significant over AODV, DSDV, DSR and OLSR.

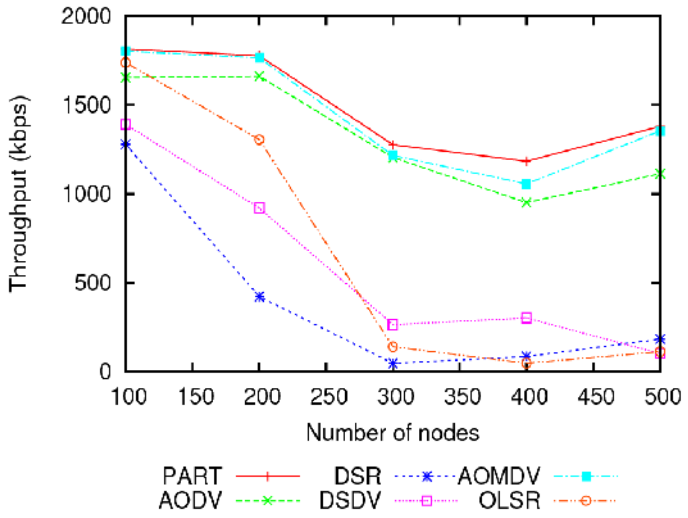


Figure 4.22 : Throughput measurement in random topology

The simulation results are significant as expected — the PART protocol performs very well in terms of packet losses, average delay, routing overhead and throughput under the large-scaled networks.

Table 4.28 : Throughput measurement in large-scaled networks

t-Test: Two-Sample Assuming Equal Variance (large-scale networks)

	PART	*AODV*
Mean	6.3E+02	2.4E+02
Variance	5.3E+05	8.2E+05
Observations	1.7E+01	1.7E+01
Pooled Variance	6.8E+05	
Hypothesized Mean Difference	0.0E+00	
df	3.2E+01	
t Stat	1.4E+00	
P(T<=t) one-tail	8.7E-02	
T Critical one-tail	1.3E+00	
P(T<=t) two-tail	1.7E-01	
T Critical Two-tail	1.7E+00	

	PART	*DSDV*
Mean	1.5E+03	6.0E+02
Variance	8.4E+04	2.9E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	1.9E+05	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	3.3E+00	
P(T<=t) one-tail	5.8E-03	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	1.2E-02	
T Critical Two-tail	2.3E+00	

	PART	*DSR*
Mean	1.5E+03	4.0E+02
Variance	8.4E+04	2.6E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	1.7E+05	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	4.1E+00	
P(T<=t) one-tail	1.6E-03	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	3.3E-03	
T Critical Two-tail	2.3E+00	

	PART	*AOMDV*
Mean	1.5E+03	1.4E+03
Variance	8.4E+04	1.1E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	9.7E+04	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	2.6E-01	
P(T<=t) one-tail	4.0E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	8.0E-01	
T Critical Two-tail	2.3E+00	

	PART	*OLSR*
Mean	1.5E+03	6.7E+02
Variance	8.4E+04	6.3E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	3.6E+05	
Hypothesized Mean Difference	0.0E+00	
df	8.0E+00	
t Stat	2.2E+00	
P(T<=t) one-tail	3.1E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	6.2E-02	
T Critical Two-tail	2.3E+00	

4.12 Chapter Summary

In MANETs, the route errors happen more often due to collisions and route breaks, resulting in increased routing overhead and decreased throughput. The nature of on-demand routing is to re-invoke a route discovery procedure from the beginning. When we analyze the detail simulation, the route errors due to collisions at the MAC layer are almost 70%. PART distinguishes the route errors due to collision or route breaks. If it is a collision, PART simply invokes a proxy repair function without sending error packets. Otherwise, it sends a route error message to the source node to rediscover a route. However, as long as a proxy node is still available on the path, it broadcasts the request packets within the limit of the broadcast zone. Therefore, the participation of a proxy node is important to adapt to route failures with less overhead and delay not only in the intermittent connection, but also in every connection in MANETs. The simulation results show that PART is able to detect the proxy node with the effects of node movement and outperforms other protocols in most situations. In the large-scale networks, PART reduces packet losses by almost 30% and NRL by almost 55% if compared to the others. It has a significantly higher throughput, i.e. almost 70% compared to others.

Chapter 5

RESEARCH METHODOLOGY: THE CROSS LAYER ENHANCEMENT BETWEEN TRANSPORT, ROUTING AND MAC LAYERS

5.1 Introduction

In this chapter, we propose a local acknowledgement mechanism for a proxy node (PACK) to improve the reliability and throughput of the transport layer protocols with the aid of lower layer protocols. The proposed mechanism must satisfy the following requirements to transmit data efficiently.

Reliable delivery: This mechanism ensures the delivery of all application-layer data packets to the destination node with the aid of a middle proxy node. Hence, the protocol guarantees the reliability of data packets and throughput improvement by detecting the missing TCP sequence numbers and acknowledging them to the source node in advance during the transmission.

Interoperability with the application layer: The mechanism does not affect the end-to-end interface for reliable transmission of TCP packets.

Cross-layer information: The protocol requires more than a minimum of cross-layer information. In particular, it needs the following information:

- The MAC address of upstream nodes for the unicast transmission.
- Proxy selection from the routing layer to detect the missing TCP sequence numbers.
- Monitoring ACK packets from the TCP destination at the routing layer to inform the missing sequence number to the source node.

5.2 Local Acknowledgement Scheme of a Proxy Node

For the reliability of TCP, previous researchers proposed techniques such as hop-by-hop checking of packets (Heimlicher, 2007) and split-TCP (i.e. using multiple proxies) (Kopparty et al, 2002). Using multiple proxies for each connection and hop-by-hop techniques are not cost-effective in MANETs. Instead of doing that, a proxy node is selected depending on the path length according to the PART routing protocol, which is proposed in Chapter 4. In the proxy node acknowledgement scheme (PACK), the proxy node always monitors the TCP sequence number, records missing sequence numbers, and informs these to the source node. In this way, the source node can retransmit the missing sequence numbers in advance of end-to-end acknowledgement that the packets are not received. The following are the research questions that must be addressed to do so.

1. How does a proxy node know which TCP sequence numbers are missing?
2. How does a proxy node inform the missing sequence numbers to the source?
3. What is the function of the intermediate nodes upon detecting of the missing sequence numbers?
4. How does the source node retransmit the missing sequence number before the end-to-end ACK is received?

5.2.1 TCP Packet Header

Figure 5.1 shows the two extra fields added in the TCP's packet header (Postel, 1981) to detect missing sequence numbers and how many sequence numbers are missing while TCP packets are transmitting.

- Missing Sequence Number: 32 bits
- Number of Missing Sequence Number: 32 bits

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Missing Sequence Number                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Number of  Missing Sequence Number               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5.1 : TCP header format

5.2.2 Proxy Selection

The proxy selection process is the function of routing protocol (PART). As described in Chapter 4, a middle node is selected as a proxy according to the path length.

5.2.3 Sequence Number Checking at the Proxy Node

The responsibility of a proxy node for a TCP connection is to

- detect the missing TCP sequence number (*miss_seqno*) and the number of missing sequence numbers (*num_miss_seqno*) by monitoring the data packets going through the routing layer of a proxy node,
- send a PACK to the source node to inform it of *miss_seqno*.

To detect the *miss_seqno*, a proxy node uses a simple algorithm to check the current sequence number (*cur_seqno*) which is in the TCP packet header, and the expected sequence number (*exp_seqno*) which is increased linearly. In Figure 5.2, while a source node is sending TCP data packets to the destination, a proxy node monitors the sequence number and checks for missing sequence numbers. Then, the proxy node puts the *seqno* of the TCP header in the *cur_seqno*.

Initially, the *seqno* starts at one. In order to increase *exp_seqno* linearly, the *cur_seqno* and *exp_seqno* must be the same. However, if the proxy node change occurs, the *seqno* may not start from one. Therefore, the value of *cur_seqno* must be updated according to the *exp_seqno*. To detect the proxy change, the current proxy node ID is recorded in the routing table of each node. In other words, when the proxy node detects that *cur_seqno* is equal to one (i.e. the foremost seqno), this proxy node's id is added in the *now_proxy* field in the routing table (line 6 in Algorithm 5.1). Later, if the current proxy is not equal to the *now_proxy*, this means that a proxy change event has occurred. The *now_proxy* field of the routing table is then updated with a new proxy (line 10 in Algorithm 5.1), and *exp_seqno* is updated with *cur_seqno*. While the TCP *seqno* is monitored, *exp_seqno* is increased by one for every received TCP sequence number, and then the *cur_seqno* and *exp_seqno* are compared.

In Figure 5.2, as long as the *cur_seqno* and *exp_seqno* are the same, the proxy node assumes that there is no missing sequence number. After receiving *seqno* 4, it is supposed to receive *seqno* 5 in *cur_seqno* according to the *exp_seqno*, however, it receives *cur_seqno* 7. In other words, *cur_seqno* is greater than *exp_seqno*, which indicates a missing sequence number. Therefore, the proxy node adds *seqno* 5 in the missing sequence number and calculates the number of missing sequence number by subtracting *cur_seqno* and *exp_seqno*. Then, the *exp_seqno* is updated by adding *cur_seqno* value, for example, in Figure 5.2, *exp_seqno* is 7 to continue checking the sequence numbers.

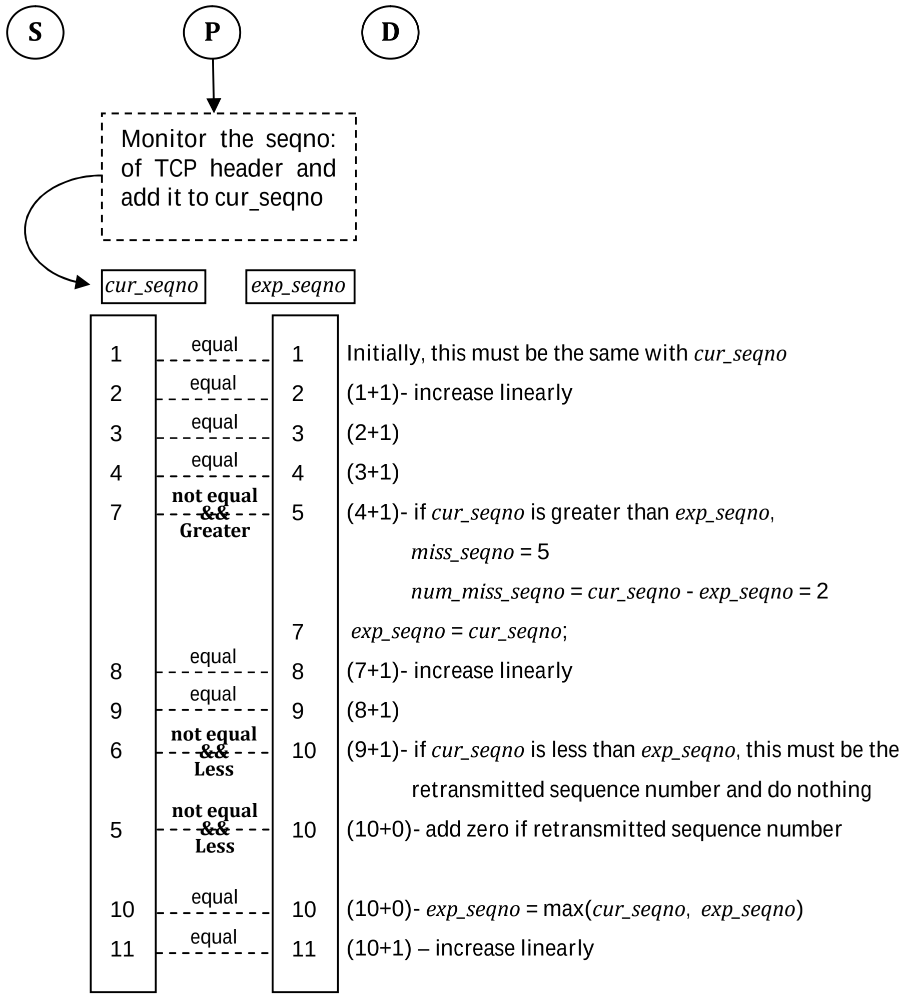

S
P
D
Monitor the seqno: of TCP header and add it to cur_seqno
cur_seqno
exp_seqno
1 equal 1 Initially, this must be the same with cur_seqno
2 equal 2 (1+1)- increase linearly
3 equal 3 (2+1)
4 equal 4 (3+1)
7 not equal && Greater 5 (4+1)- if cur_seqno is greater than exp_seqno,
miss_seqno = 5
num_miss_seqno = cur_seqno - exp_seqno = 2
7 exp_seqno = cur_seqno;
8 equal 8 (7+1)- increase linearly
9 equal 9 (8+1)
6 not equal && Less 10 (9+1)- if cur_seqno is less than exp_seqno, this must be the retransmitted sequence number and do nothing
5 not equal && Less 10 (10+0)- add zero if retransmitted sequence number
10 equal 10 (10+0)- exp_seqno = max(cur_seqno, exp_seqno)
11 equal 11 (10+1) – increase linearly

Figure 5.2 : Sequence number checking algorithm

After receiving *seqno* 9, the proxy node detects that the *cur_seqno* and *exp_seqno* are not equal, and the *cur_seqno* is less than *exp_seqno*, meaning that the retransmitted missing sequence number is received. Instead of increasing *exp_seqno* linearly, *exp_seqno* is added to zero as shown in Figure 5.2 and line 15 of Algorithm 5.1.

To update *exp_seqno* for checking the rest of the sequence number, the proxy node takes the maximum between *cur_seqno* and *exp_seqno* as shown in line 25 of Algorithm 5.1. As soon as the proxy node detects that the sequence numbers are missing, it sends a PACK packet by adding *miss_seqno* and *num_miss_seqno*, which is a subtracted value from *cur_seqno* and *exp_seqno*, to inform the source node. Line 21 of Algorithm 5.1 calls Algorithm 5.2, which is unicast PACK sending algorithm.

Checking the missing sequence number at the proxy node

```
1:  upon monitoring TCP packet at the proxy node in the routing layer do
2:      access tcp_ seqno at the routing layer
3:      add tcp_seqno to the cur_seqno field in the routing table
4:      if tcp_seqno is the equal to one then
5:        add cur_seqno to exp_seqno variable
6:        record the current proxy node in now_proxy field of the routing table
7:      end if
8:      if now_proxy is not equal to current proxy then
9:        add cur_seqno to the exp_seqno
10:       record a new proxy in now_proxy in the routing table
11:     else if tcp_seqno is greater than one then
12:         if cur_seqno is greater than or equal to exp_seqno then
13:           add exp_seqno plus one
14:         else
15:           add exp_seqno plus zero
16:         end if
17:         if exp_seqno is not equal to cur_seqno &&
18:             cur_seqno is greater than exp_seqno then
19:               add exp_seqno to first_miss_seqno in the routing table
20:               calculate num_miss_seqno = cur_seqno – exp_seqno
21:               call Algorithm 5.2
22:               add cur_seqno to the exp_seqno to reset value
23:         else if exp_seqno is not equal to cur_seqno &&
24:                 cur_seqno is less than exp_seqno then
25:               update exp_seqno = max(exp_seqno, cur_seqno)
26:         end if
27:     end if
28: end upon
```

Algorithm 5.1 : Sequence number checking at a proxy node

5.2.4 Cross-layer Information for Unicast PACK

The PACK packet is sent by unicast towards the source node by using the MAC layer information as shown in Figure 5.3. When a PACK control packet is received, the intermediate nodes are responsible for checking whether it is the source node. If it is, it updates its routing table with the *miss_seqno*. Otherwise, it updates the routing table and forwards the packet to the next hop towards the source node. Algorithm 5.2 shows the procedure for sending PACK. Before a proxy node sends PACK, it looks at the MAC layer address of the upstream nodes, adds it in the *next_hop* field of the routing table, and sends PACK with the *miss-seqno* information to the source node.

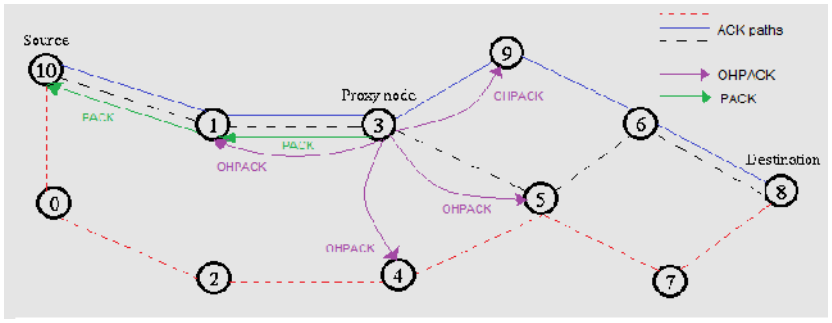


Figure 5.3 : Control packets for updating missing sequence numbers

Sending unicast PACK packet from a proxy node to the source node

```
1:  upon forwarding TCP at the proxy node do
2:      if packet type is TCP &&
3:          packet size is greater than or equal to 512 then
4:          access source's Ethernet address
5:          add this address to upstream field of the routing table
6:          add upstream to the next_hop of the routing table for unicast
            transmission
7:          send PACK with the miss_seqno and num_miss_seqno
            information
8:      end if
9:  end upon
```

Algorithm 5.2 : Unicast PACK packet transmission

5.2.5 One-hop Broadcast

One hop broadcast (OHPACK) is used so that the neighboring nodes of a proxy can update the *miss_seqno* information correctly. After the proxy node has known the *miss_seqno*, it broadcasts OHPACK packet to its one-hop neighbors as shown in Figure 5.3. When the neighbors receive the OHPACK, they update their routing tables by adding *miss_seqno* and *num_miss_seqno*. The possibilities of ACK paths throughout the network are shown in Figure 5.3. Two control packets, i.e. PACK and

OHPACK, assist to attach *miss_seqno* and *num_miss_seqno* values in the ACK packets from the destination to inform to the source node.

5.2.6 Monitoring ACK Packet from the Destination

An important problem to address is how to inform the source node's transport layer of the *miss_seqno*. Because PACK and OHPACK are lower routing layer packets, it is impossible to send *miss_seqno* to the upper transport layer. Moreover, all the functions mentioned above work at the routing layer locally. Therefore, there is only one way to modify the end-to-end ACK packet, that is by adding new fields in the TCP packet header. The new fields in the TCP header are shown in Figure 5.1.

When the end-to-end ACK packets are received, the responsibilities of an intermediate node are to check whether it is the next hop towards the source node. If it is, the node adds the *miss_seqno* and *num_miss_seqno* values to the end-to-end ACK packets to inform the source node which TCP sequence numbers are missing as shown in Algorithm 5.3. Otherwise, the node simply forwards the ACK packets.

Receiving ACK packet from the TCP destination

```
1:  if a node is the next hop to the source node then
2:      add first_miss_seqno in the routing table to miss_seqno field of ACK
        packet
3:      add num_miss_seqno in the routing table to num_miss_seqno field of
        ACK
4:      reset the miss_seqno fields in the routing table
5:  end if
```

Algorithm 5.3 : Monitoring ACK from data destination

5.2.7 Functions of TCP Data Source

The TCP data source has to check every received ACK packet. When the *miss_seqno* value is not equal to zero, the source node retransmits the *miss_seqno*, and decides the number of missing sequences according to the *num_miss_seqno* as shown in Algorithm 5.4. If the *num_miss_seqno* is equal to one, TCP source adds the *miss_seqno* in the TCP's header, sets the *retransmit_timer* and sends the TCP packet in advance before receiving the end-to-end ACKs. Otherwise, the TCP source makes sure to send all missing sequences by counting the *num_miss_seqno*. (cf. Algorithm 5.5 and 5.6)

Receiving ACK packet at a source node

```
1:  upon receiving ACK at the source node do
2:      if miss_seqno of tcp header is not equal to zero then
3:          add miss_seqno to retransmit_miss_seqno variable
4:          add num_miss_seqno to num_retransmit_miss_seqno variable
5:
6:            if num_retransmit_miss_seqno is equal to one then
7:                call Algorithm 5.5
8:            else if num_retransmit_miss_seqno is greater than one
9:
10:                for the num_retransmit_miss_seqno value is reduced till one do
11:                    call Algorithm 5.5
12:                end for
13:           end if
14:     end if
15: end upon
```

Algorithm 5.4 : ACK processing at TCP source

```
1: if retransmit_miss_seqno is satisfied the seqno checking of TCP's sender then
2:     call Algorithm 5.6
3: else
4:     miss_seqno plus one
5: end if
```

Algorithm 5.5 : Decision upon the retransmit missing sequence number

```
1: upon receiving miss_seqno do
2:     add miss_seqno to seqno of TCP's header
3:     set the retransmit-timer
4:     send TCP packet
5: end upon
```

Algorithm 5.6 : Retransmission of missing sequence number

We apply the PACK mechanism to the TCP's variants, such as TCP-Tahoe, TCP-New Reno, TCP-Vegas and TCP-Westwood and implement them in NS. Then, we analyze the performance of each transport protocol with PART and AODV routing protocols and different topologies in static environments. Later on, node speeds are varied to test the efficiency of PACK mechanism in mobile environments. As performance metrics, throughput, average end-to-end delay, packet loss rates, packet delivery fraction and routing overhead are measured.

5.3 Implementation of PACK over TCP

A TCP source sends and forwards packets to a lower level network depending on the user demand from an application. It limits the sending rate (i.e. congestion window) to control the network congestion. It utilizes an acknowledgement mechanism to provide the reliability of TCP connections. A TCP destination is responsible for acknowledging every received TCP packet. A TCP source decides whether the transmitted packet is lost based on the acknowledgement scheme and retransmits it.

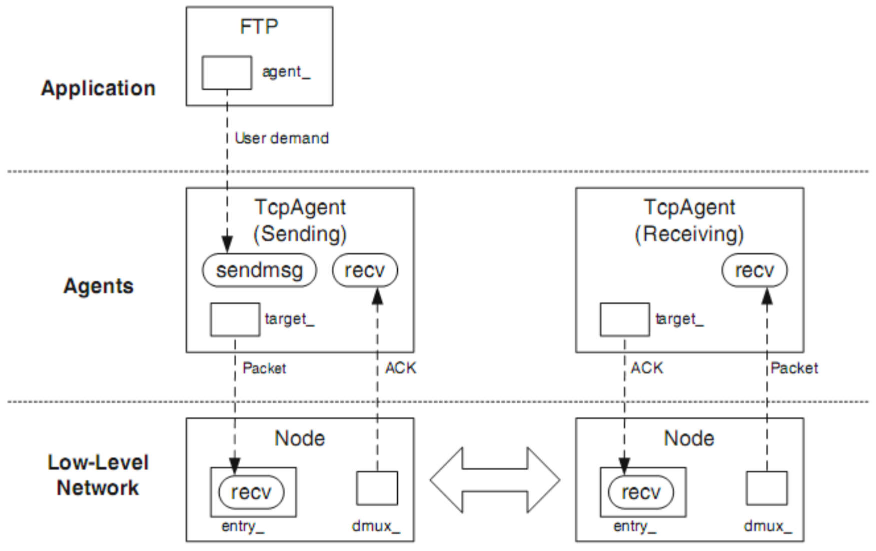


Figure 5.4 : TCP packet transmission and acknowledgement mechanisms

(Issariyakul and Hossain, 2008)

Figure 5.4 shows the basic procedures of TCP. The packet transmission procedure begins when a user executes the file transfer application (FTP) by invoking the *sendmsg* function of the TCP agent. The TCP agent from the sender side sends packets to its downstream nodes by executing *target_*. The low-level network layer delivers them to reach the destination node, which in turn forwards the packets to the TCP agent from the receiver side installed in the demultiplexer (i.e. *dmux_*). Once the TCP packet is received, the TCP receiver sends an ACK packet across the lower level network and forwards it via its demultiplexer.

5.3.1 Local Acknowledgement Mechanism

As we perform the sequence number checking at the routing layer, the nodes in the routing layer monitor TCP packets before forwarding them. Therefore, we add the packet type checking process in *forward* function of PART as shown in Program 5.1. The missing sequence checking techniques are done in *forward* function of PART. In addition, *sendPACK* and *sendOHPACK* voids assist to inform the missing seqno to the source node. The intermediate nodes that receive those packets update their routing tables. However, as we created these packets at the routing layer, it is impossible to send missing seqno information to the transport layer of the source node. Therefore, all intermediate nodes that know the missing seqno monitor the ACK packet from the destination and attach the missing value in the packet header as shown in Program 5.2 and 5.3.

Program : 5.1 `//~ns/part/part.cc`

```
1: void
2: PART::forward(part_rt_entry *rt, Packet *p, double delay){
3: struct hdr_cmn *ch = HDR_CMN(p);
4: struct hdr_ip *ih = HDR_IP(p);
5: struct hdr_mac802_11 *mh = HDR_MAC802_11(p);
6: //check whether the node is proxy or not.
7:   if (ch->ptype()== PT_TCP && ch->direction()==hdr_cmn::UP
     &&(ch->proxy_hop_ == here_.addr_)&&(ch->size() >= 512)){
     rt->rt_upstream = ETHER_ADDR(mh->dh_ta);
     hdr_tcp *tcpr = hdr_tcp::access(p);
     rt->rt_tcp_cur_seqno = tcpr->seqno();
     // seqno is the first for the first proxy node, current
     //and expected must the the same.
     if(rt->rt_tcp_cur_seqno == 1) {
         rt->rt_tcp_exp_seqno = rt->rt_tcp_cur_seqno;
     // record the first proxy node’s id
         rt->rt_now_proxy = ch->proxy_hop_;
     // when the proxy node changes occur, exp_seqno is reset
         if(rt->rt_now_proxy != ch->proxy_hop_) {
            rt->rt_tcp_exp_seqno = rt->rt_tcp_cur_seqno;
            rt->rt_now_proxy = ch->proxy_hop_;
         }
// Later, current and expected seqno are greater than 1.
else if(rt->rt_tcp_cur_seqno > 1)   {
// exp_seqno is increased linearly as long as the seqno is
// received in order.
      if (rt->rt_tcp_cur_seqno >= rt->rt_tcp_exp_seqno)
          rt->rt_tcp_exp_seqno = rt->rt_tcp_exp_seqno + 1;

// exp_seqno is not increased if miss_seqno is received.
   else if (rt->rt_tcp_cur_seqno < rt->rt_tcp_exp_seqno)
          rt->rt_tcp_exp_seqno = rt >rt_tcp_exp_seqno + 0;
/* If current and expected values are not equal && current
 * is greater than expected, Add missing value (expected
 * value) in first_miss_seqno. To know how many seqnos are
 * missing, substract (current-expected. Inform the source
 * node by adding first_miss and no: of miss in PACK.Reset
 * the expected value by adding current seqno: to continue
 * checking.
 */
if( (rt->rt_tcp_exp_seqno != rt->rt_tcp_cur_seqno)  &&
    (rt->rt_tcp_cur_seqno > rt->rt_tcp_exp_seqno) )  {
    rt->rt_first_miss_seqno = rt->rt_tcp_exp_seqno;
    rt->rt_num_miss_seqno = rt->rt_tcp_cur_seqno –
                            rt->rt_tcp_exp_seqno;
      send_PACK(ih->saddr(), rt->rt_upstream,
          rt->rt_first_miss_seqno, rt->rt_num_miss_seqno);

//To make sure that miss_seqno reaches to the source node
// send another one hop broadcast.
      sendOHPACK(rt->rt_first_miss_seqno,
                         rt->rt_num_miss_seqno);
      rt->rt_tcp_exp_seqno = rt->rt_tcp_cur_seqno;
}
/* If current and expected values are not equal && current
 * is less than expected, that means, retransmit
 * miss_seqno are received. As mentioned above, no incre
 * ment can be done by adding zero. To successfully check
 * the rest seqno:, we need to reset the exp_seqno by
 * taking maximum value of (exp_seq and cur_seq)
 */
else if((rt->rt_tcp_exp_seqno != rt->rt_tcp_cur_seqno)
    &&(rt->rt_tcp_cur_seqno < rt->rt_tcp_exp_seqno) )  {
    rt->rt_tcp_exp_seqno = max(rt->rt_tcp_exp_seqno,
                               rt->rt_tcp_cur_seqno);
}
}
```

Program : 5.2 **//~ns/part/part.cc**

```
void
PART::recv(Packet *p, Handler*)
{
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);

/*************************************************
* If packet type is ACK
*************************************************/
if(ch->ptype() == PT_ACK && ch->size_ == 40 )
{
  recvACK(p);
}
```

When a TCP source node receives the ACK packets, it checks whether any missing sequence number is included in it. If it is, the source node retransmits the missing sequence number instead of waiting for duplicate ACK and timeout period. Note that, we do not violate the original end-to-end checking mechanism at the TCP receiver and sender.

Program : 5.3 **//~ns/part/part.cc**

```
void
PART::recvACK(Packet *p)
{
struct hdr_cmn *ch = HDR_CMN(p);
struct hdr_ip *ih = HDR_IP(p);
part_rt_entry *rt;
if(rt->rt_dst == ih->daddr()) {
   hdr_tcp *tcpr = hdr_tcp::access(p);
if(rt->rt_first_miss_seqno != 0)
{
  tcpr->miss_seqno_ = rt->rt_first_miss_seqno;
  tcpr->no_miss_seqno_ = rt->rt_num_miss_seqno;
  rt->rt_first_miss_seqno = 0;
  rt->rt_num_miss_seqno = 0;
  }
 }
}
```

This mechanism is applied to the variants of TCP, such as TCP-Tahoe, New Reno, Vegas and Westwood. When an ACK packet is received with non-zero *miss_seqno*, Program 5.4 calls the `retransmit_miss_seqno()` function (Program 5.5) that checks whether the sequence number is fresh enough to retransmit. If it is, program 5.6 retransmits the *miss_seqno*.

Program : 5.4 **//~ns/tcp/*.cc (tcp, newreno, vegas, westwood)**

```
void
TcpAgent::recv(Packet *pkt, Handler*)
{
hdr_tcp *tcph = hdr_tcp::access(pkt);
int valid_ack = 0;
 if (tcph->miss_seqno_ != 0) {
    retransmit_miss_seqno_ = tcph->miss_seqno_;
    no_retransmit_miss_seqno_ = tcph->no_miss_seqno_;
if (no_retransmit_miss_seqno_ == 1) {
  retransmit_miss_seqno_ = tcph->miss_seqno_;
  retransmit_miss_seqno();
 }
else if (no_retransmit_miss_seqno_ > 1) {
   for (int i = -1 ; i < no_retransmit_miss_seqno_; i++)
    retransmit_miss_seqno_ = tcph->miss_seqno_ + i;

  }
}
```

Program : 5.5 **//~ns/tcp/tcp.cc**

```
void
TcpAgent:: retransmit_miss_seqno()
{
 If (retransmit_miss_seqno_ <= highest_ack_+ wnd_ &&
     retransmit_miss_seqno_ < curseq_ &&
     retransmit_miss_seqno_ <= highest_ack_+ cwnd_+dupacks_
     {
     handle_seqno(retransmit_miss_seqno_);
  }
return;
}
```

Program : 5.6 **//~ns/part/part.cc**

```
Void
TcpAgent::handle_seqno(int miss_seqno) {
Packet* p = allocpkt();
hdr_tcp *tcph = hdr_tcp::access(p);
tcph->seqno() = miss_seqno;
tcph->ts() = Scheduler::instance().clock();
send(p,0);

}
```

5.4 Experimental Analysis

Objective of experiment: We analyze the performance of the PACK scheme with the TCP variants by changing the hop distances in the chain topologies (Figure 5.5), the grid sizes in the grid topologies and random topologies with node movements.

Expected outcome: The sequence number checking of the PACK mechanism positively influences the best performance for TCP variants in terms of the throughput, delay and packet delivery fraction and number of collisions at the MAC layer.

5.4.1 Chain Topology in Static Network

We use NS-2 to carry out the simulations. All nodes are allocated in the respective simulation area, and the simulation time is set to 360 seconds. Each node has a transmission range of 200 m, and the default bandwidth is set to 11Mbps. The FTP application is generated. The TCP packet size is 512 bytes. Firstly, we measure the performance throughput for all hop distances across the TCP-Tahoe, New Reno, Vegas and Westwood by using AODV and PART as based routing protocols. We apply the PACK mechanism to all TCP variants by using PART as a base protocol.

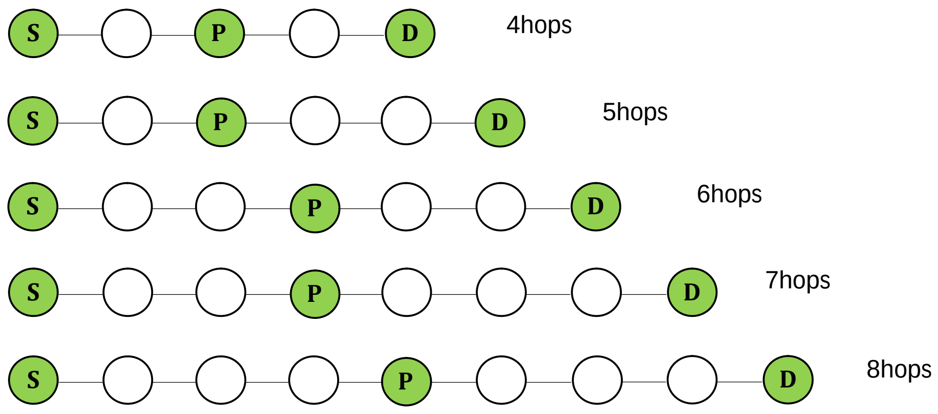
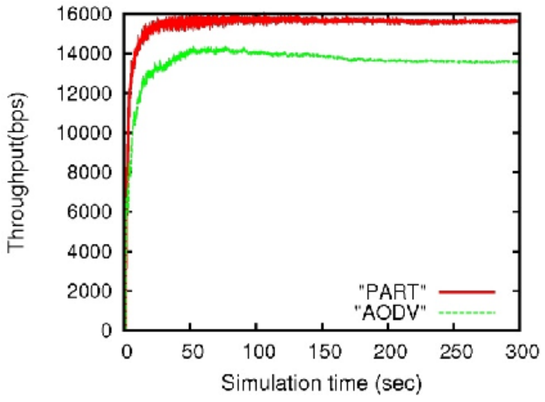


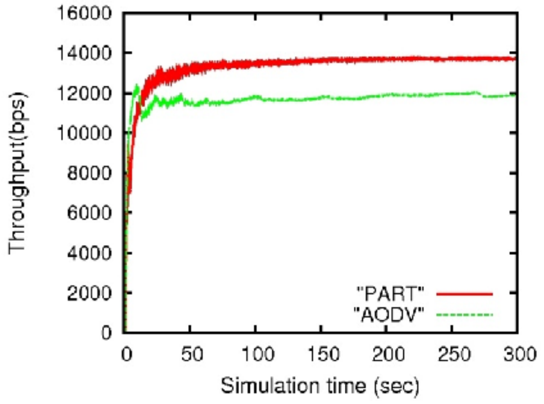
Figure 5.5 : Analysis of hop distance changes

5.4.1.1 Throughput Measurements across Variants of TCP

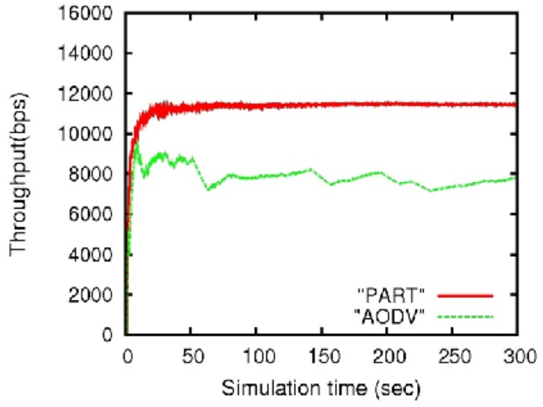
We measure the throughput of each TCP variant with different routing protocols, and Figure 5.6 shows the throughput comparisons for Tahoe. As the path length is longer, the throughput of Tahoe gradually decreases. Tahoe-PACK over PART outperforms Tahoe almost by 15% in 4 hops, 16% in 5 hops, 51% in 7 hops and 50% in 8 hops over AODV. Moreover, Tahoe over AODV incurs a higher fluctuation, especially when the hop count is higher.



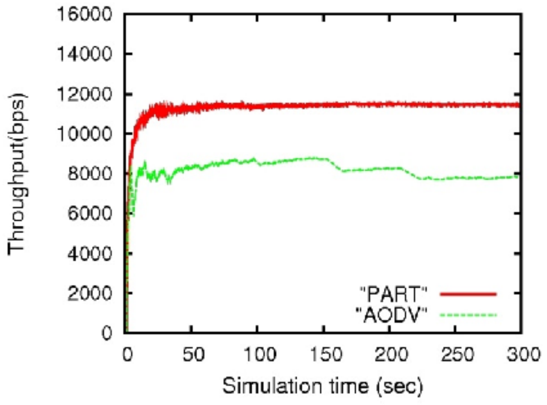
(a) 4 hops



(b) 5 hops



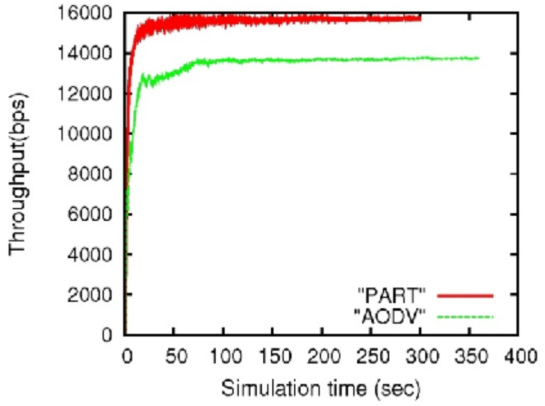
(c) 7 hops



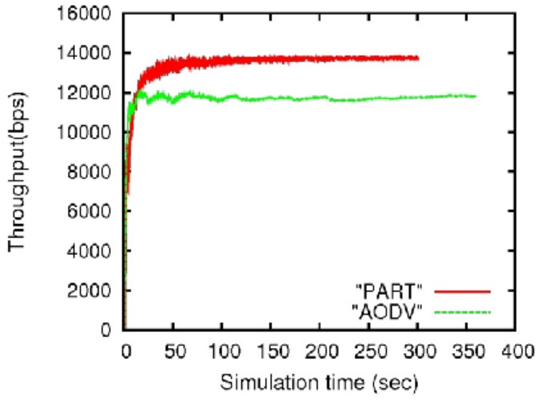
(d) 8 hops

Figure 5.6 : Throughput measurement across TCP-Tahoe

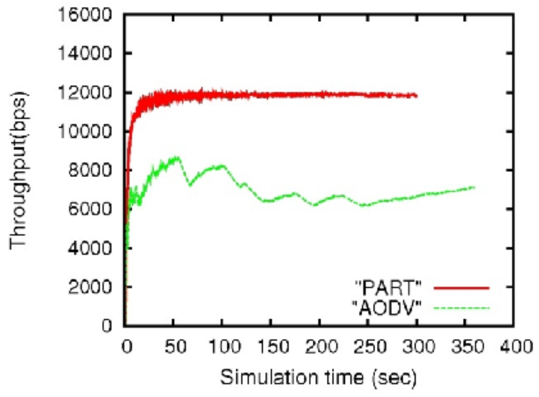
Even though we analyze the performance in a static network, the route errors happen more often due to the collision at the MAC layer. As we use the fast retransmission of PACK mechanism with the PART protocol, the significant performance improvement can be seen in Figure 5.6. Again, as we examine the PACK mechanism over New Reno, similar improvements are encountered. While New Reno-PACK maintains a stable throughput, the oscillation of New Reno is higher with AODV. The route errors happen at all time due to the collision even in the static networks.



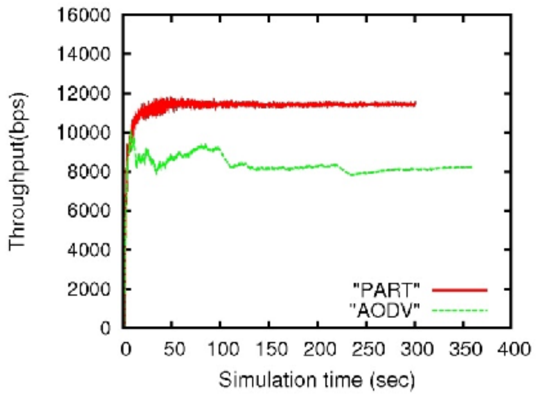
(a) 4 hops



(b) 5 hops



(c) 7 hops



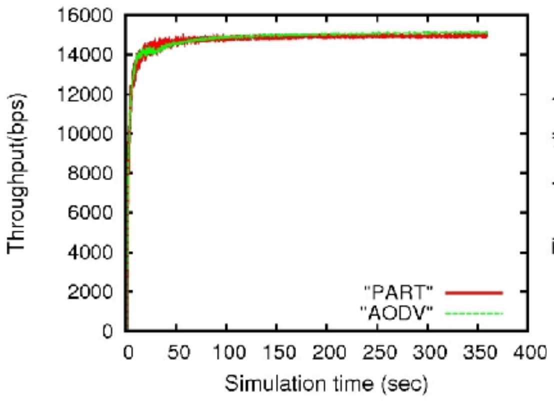
(d) 8 hops

Figure 5.7 : Throughput measurement across TCP-New Reno

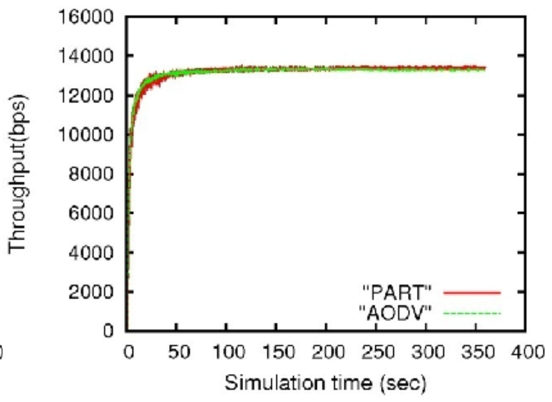
The PACK mechanism has the ability to retransmit the missing sequence number in advance after checking them at a proxy node. Therefore, New Reno-PACK over

PART improves performance almost by 15% in 4 hops, 16% in 5 hops, 67% in 7 hops and 39% in 8 hops better than New Reno over AODV as shown in Figure 5.7.

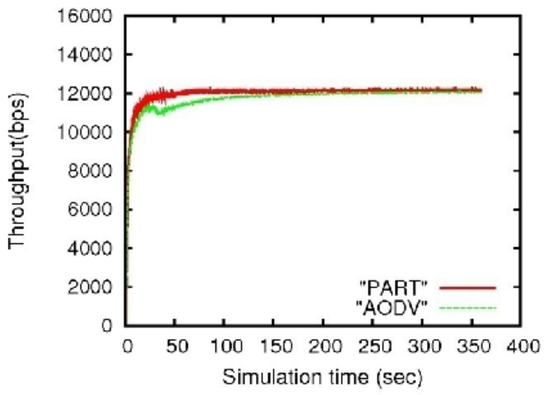
We perform similar measurements across Vegas and Westwood. In Figure 5.8, the Vegas-PACK is only slightly better than AODV in terms of throughput: around 0.7% in 5 hops, 0.9% in 7 hops and 0.2% in 8 hops.



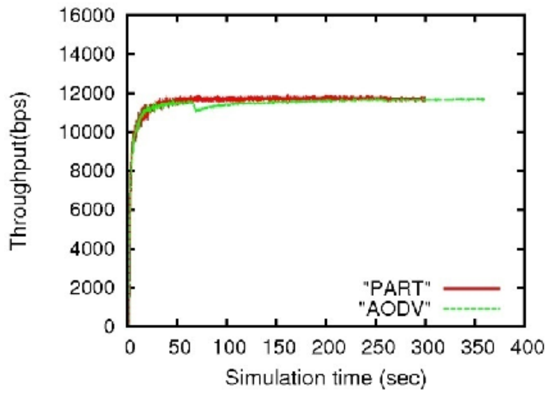
(a) 4 hops



(b) 5 hops



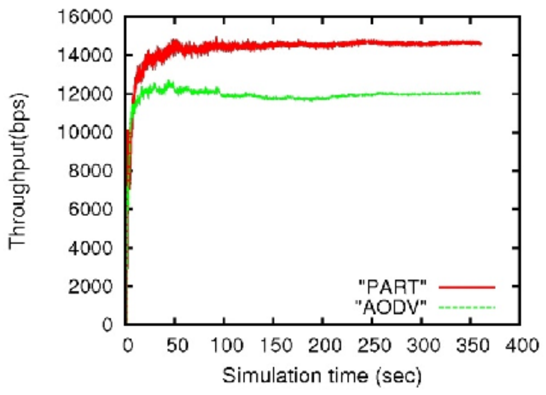
(c) 7 hops



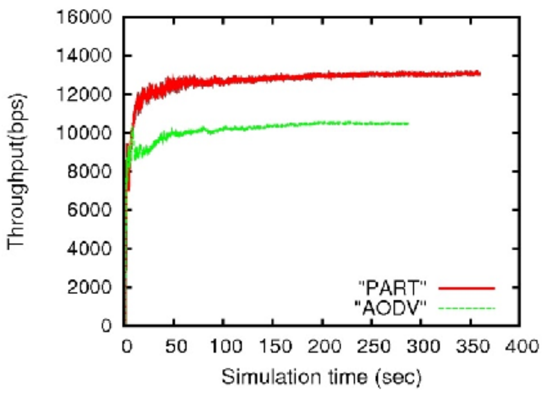
(d) 8 hops

Figure 5.8 : Throughput measurement across TCP-Vegas

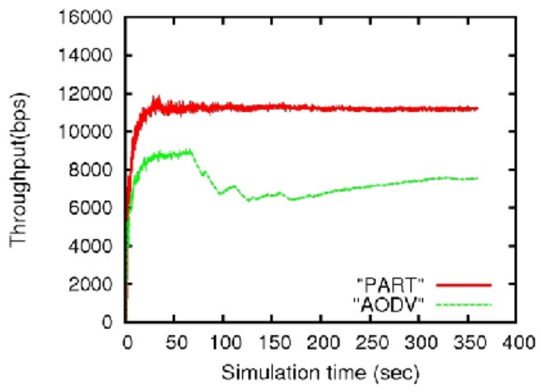
The PACK mechanism for the window-based TCP protocols is not as effective as Vegas in terms of throughput. Because Vegas has problem re-routing packets, the sequence number checking algorithm at the proxy node may not always work when a connection changes. Therefore, the performance of PACK mechanism is almost the same over AODV and PART routing.



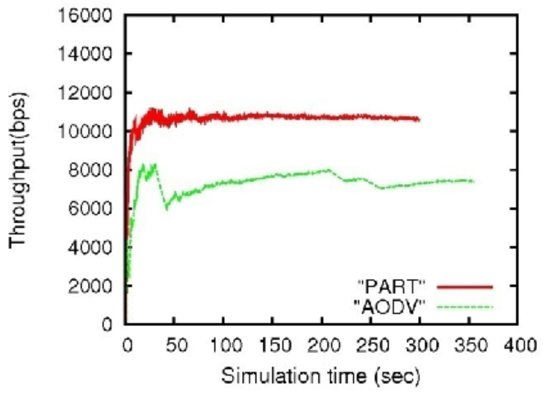
(a) 4hops



(b) 5hops



(c) 7hops



(d) 8hops

Figure 5.9 : Throughput measurement across TCP-Westwood

Figure 5.9 shows the comparison for Westwood. Westwood-PACK over PART shows a significantly higher throughput: almost 22% in 4 hops, 35% in 5 hops, 27% in 7 hops and 44% in 8 hops compared to Westwood over AODV. Westwood monitors the ACK packets to control the slow start threshold and estimate bandwidth for the congestion window adjustment. Tables 5.1 to 5.4 show the z-test results, where the improvements are statistically significant across TCP variants in chain topologies.

Table 5.1 : Throughput measurement of TCP-Tahoe across a chain topology in a static network

z-Test: Two Sample for Means (4 hops)		
	PACK with PART	***AODV***
Mean	1.5E+04	1.4E+04
Known Variance	8.0E+05	9.5E+05
Observations	8.5E+03	7.4E+03
Hypothesized Mean Difference	0.0E+00	
Z	1.3E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means (5 hops)		
	PACK with PART	***AODV***
Mean	1.3E+04	1.2E+04
Known Variance	9.4E+05	3.9E+05
Observations	7.5E+03	6.5E+03
Hypothesized Mean Difference	0.0E+00	
Z	1.2E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means (7 hops)		
	PACK with PART	***AODV***
Mean	1.1E+04	7.8E+03
Known Variance	4.6E+05	3.8E+05
Observations	6.2E+03	4.3E+03
Hypothesized Mean Difference	0.0E+00	
Z	2.7E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means (8 hops)		
	PACK with PART	***AODV***
Mean	1.1E+04	8.2E+03
Known Variance	4.6E+05	3.3E+05
Observations	6.2E+03	4.3E+03
Hypothesized Mean Difference	0.0E+00	
Z	2.6E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

Table 5.2 : Throughput measurement of TCP-New Reno across a chain topology in a static network

z-Test: Two Sample for Means (4 hops)		
	PACK with PART	***AODV***
Mean	1.5E+04	1.3E+04
Known Variance	7.7E+05	8.4E+05
Observations	8.6E+03	9.0E+03
Hypothesized Mean Difference	0.0E+00	
Z	1.5E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means (5 hops)		
	PACK with PART	***AODV***
Mean	1.3E+04	1.2E+04
Known Variance	9.3E+05	3.1E+05
Observations	7.5E+03	7.7E+03
Hypothesized Mean Difference	0.0E+00	
Z	1.4E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means (7 hops)		
	PACK with PART	***AODV***
Mean	1.2E+04	7.0E+03
Known Variance	5.0E+05	5.4E+05
Observations	6.4E+03	4.6E+03
Hypothesized Mean Difference	0.0E+00	
Z	3.4E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means (8 hops)		
	PACK with PART	***AODV***
Mean	1.1E+04	8.3E+03
Known Variance	4.6E+05	3.0E+05
Observations	6.2E+03	5.4E+03
Hypothesized Mean Difference	0.0E+00	
Z	2.6E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

Table 5.3 : Throughput measurement of TCP-Vegas across a chain topology in a static network

z-Test: Two Sample for Means (4 hops)		
	PACK with PART	***AODV***
Mean	1.5E+04	1.4E+04
Known Variance	8.0E+05	9.5E+05
Observations	8.5E+03	7.4E+03
Hypothesized		
Mean Difference	0.0E+00	
Z	1.3E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means (5 hops)		
	PACK with PART	***AODV***
Mean	1.3E+04	1.3E+04
Known Variance	5.6E+05	5.1E+05
Observations	9.4E+03	9.4E+03
Hypothesized		
Mean Difference	0.0E+00	
Z	2.9E+00	
P(T<=t) one-tail	1.8E-03	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	3.7E-03	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means (7 hops)		
	PACK with PART	***AODV***
Mean	1.2E+04	1.2E+04
Known Variance	4.2E+05	5.0E+05
Observations	8.6E+03	8.5E+03
Hypothesized Mean		
Difference	0.0E+00	
Z	2.6E+01	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means (8 hops)		
	PACK with PART	***AODV***
Mean	1.2E+04	1.1E+04
Known Variance	5.0E+05	4.1E+05
Observations	6.9E+03	8.2E+03
Hypothesized		
Mean Difference	0.0E+00	
Z	9.8E+00	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

Table 5.4 : Throughput measurement of TCP-Westwood across a chain topology in a static network

z-Test: Two Sample for Means (4 hops)		
	PACK with PART	***AODV***
Mean	1.4E+04	1.2E+04
Known Variance	8.2E+05	3.9E+05
Observations	9.6E+03	7.8E+03
Hypothesized		
Mean Difference	0.0E+00	
Z	2.1E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means (5 hops)		
	PACK with PART	***AODV***
Mean	1.3E+04	1.0E+04
Known Variance	6.6E+05	4.2E+05
Observations	8.6E+03	5.5E+03
Hypothesized		
Mean Difference	0.0E+00	
Z	2.1E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means (7 hops)		
	PACK with PART	***AODV***
Mean	1.1E+04	7.0E+03
Known Variance	4.5E+05	5.4E+05
Observations	7.3E+03	4.6E+03
Hypothesized Mean		
Difference	0.0E+00	
Z	2.7E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means (8 hops)		
	PACK with PART	***AODV***
Mean	1.1E+04	8.3E+03
Known Variance	3.7E+05	3.0E+05
Observations	5.8E+03	5.4E+03
Hypothesized		
Mean Difference	0.0E+00	
Z	2.8E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

5.4.1.2 Average Delay Measurements across Variants of TCP

We measure the average delay for all TCP variants as shown in Figure 5.10. Tahoe-PACK over PART offers a lower delay almost 10% if compared to Tahoe over AODV. Although New Reno-PACK performs almost 4% in 5 hops, 2% in 6 hops and 7 hops, 9% in 8 hops better over PART than AODV, it suffers almost 3% higher delay in the 4-hop chain topology. The worst situations are encountered in Vegas-PACK that suffers 41% higher delay in 4 hops and 33% in 6 hops because of the window-based PACK mechanism. The mechanism performs worse in terms of average end-to-end delay in chain topology, whereas Westwood-PACK achieves almost 11% lower delay over PART protocol.

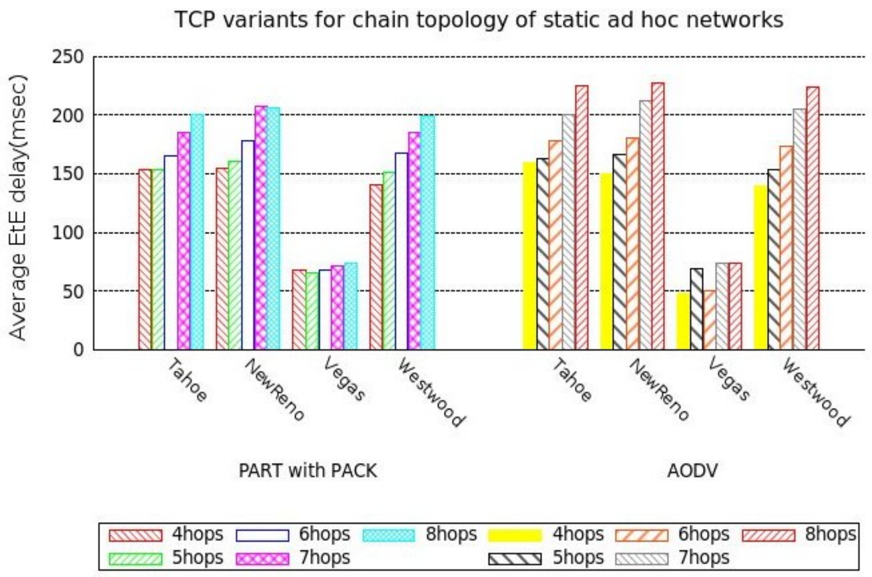


Figure 5.10 : Average delay measurement across TCP variants

5.4.1.3 Packet Delivery Fraction Measurements across Variants of TCP

Finally, we analyze the packet delivery fraction (PDF) for all TCP variants using PART and AODV. If we look at Figure 5.11, PACK with PART mechanism shows significant performance improvement for all TCP variants, especially in Tahoe, New Reno and Westwood if compared to AODV. Tahoe-PACK over PART achieves almost 5% higher PDF than AODV. New Reno-PACK improves PDF around 3% compared to AODV and Westwood-PACK also obtains almost 4% higher PDF if compared to Westwood over AODV. However, among the TCP variants, the delivery rate of Vegas is the highest because the rate-based algorithm of Vegas benefits on the delivery rate well. Vegas attempts to reduce the sending rate before the actual congestion occurs and uses packet delay as a signal of congestion. Table 5.5 shows the t-test results, where the improvements are statistically significant across TCP variants in chain topologies.

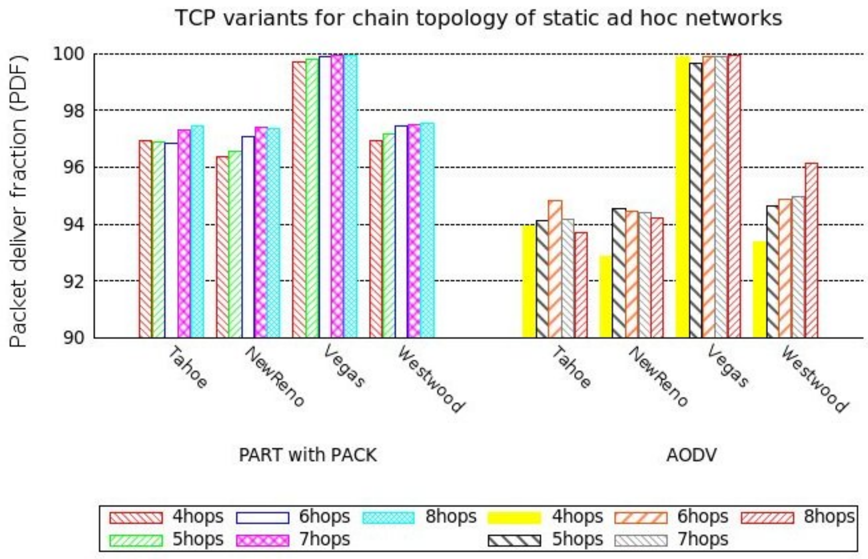


Figure 5.11 : PDF measurement across TCP variants

Table 5.5 : PDF measurement of TCP Variants across a chain topology

in a static network

t-Test: Two-Sample Assuming Equal Variances (Tahoe)		
	PACK with PART	***AODV***
Mean	9.7E+01	9.4E+01
Variance	7.1E-02	1.8E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	1.2E-01	
df	8.0E+00	
t Stat	1.3E+01	
P(T<=t) one-tail	5.2E-07	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	1.0E-06	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances (New Reno)		
	PACK with PART	***AODV***
Mean	9.7E+01	1.9E+02
Variance	2.2E-01	1.0E+03
Observations	5.0E+00	5.0E+00
Pooled Variance	3.5E-01	
df	8.0E+00	
t Stat	7.6E+00	
P(T<=t) one-tail	3.1E-05	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	6.1E-05	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances (Vegas)		
	PACK with PART	***AODV***
Mean	1.0E+02	1.0E+02
Variance	1.1E-02	1.5E-02
Observations	5.0E+00	5.0E+00
Pooled Variance	1.3E-02	
df	8.0E+00	
t Stat	-2.7E-02	
P(T<=t) one-tail	4.9E-01	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	9.8E-01	
T Critical Two-tail	2.3E+00	

t-Test: Two-Sample Assuming Equal Variances (Westwood)		
	PACK with PART	***AODV***
Mean	9.7E+01	9.5E+01
Variance	6.4E-02	9.8E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	5.2E-01	
df	8.0E+00	
t Stat	5.5E+00	
P(T<=t) one-tail	2.7E-04	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	5.5E-04	
T Critical Two-tail	2.3E+00	

5.4.2 Grid Topology in Static Network

To ascertain the efficiency of the PACK mechanism, we analyze its performance in a grid topology with different grid sizes, for example 5 × 5 grid (25 nodes) and 7 × 7 grid (49 nodes), as shown in Figures 5.12 and 5.13. We set the simulation area to 3500 m × 500 m field for 5 × 5 grid topology and 6000 m × 2000 m for 7 × 7 grid topology. The simulations are run for 360 seconds. The FTP applications are executed between each pair of source and destination. We examine the PACK mechanism with TCP variants across PART and AODV by measuring throughput, delay and packet loss rates as performance metrics.

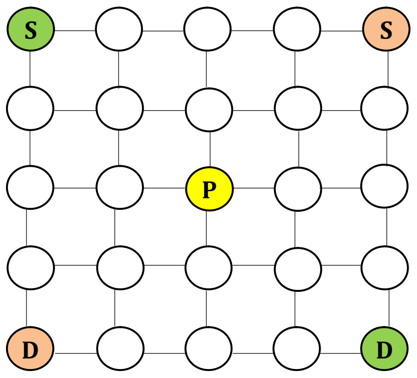


Figure 5.12 : 5 × 5 grid topology

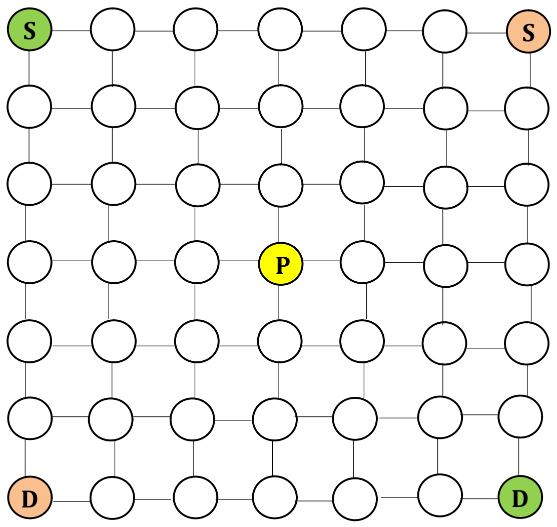
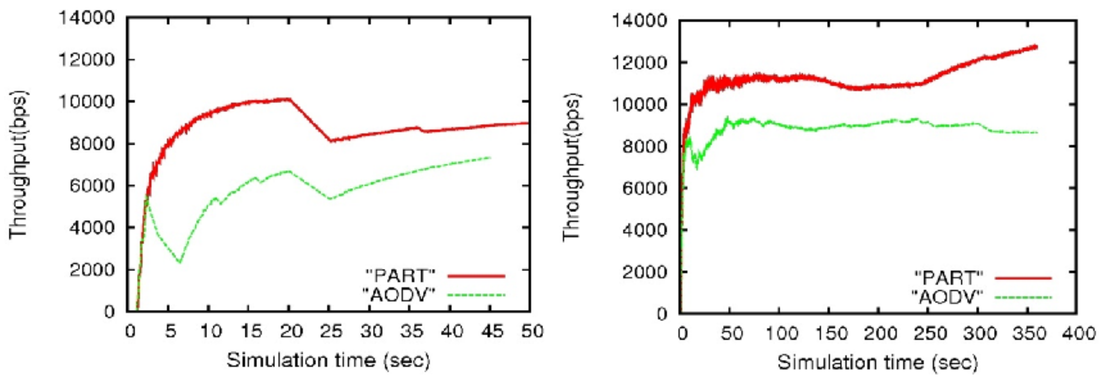


Figure 5.13 : 7 × 7 grid topology

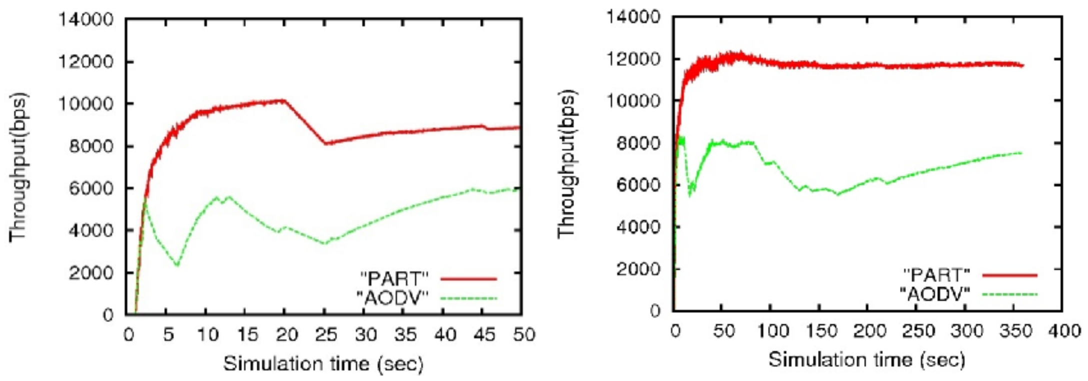
5.4.2.1 Throughput Measurements across Variants of TCP

We investigate the throughput of TCP variants with PACK mechanism by varying grid sizes. Tahoe-PACK over PART still performs 22% better in 5 × 5 and 47% better in 7 × 7 if compared to Tahoe over AODV in Figure 5.14. Also in Figure 5.15, New Reno-PACK offers a higher throughput — almost 49% in 5 × 5 grid and 55% in 7 × 7 if compared to New Reno over AODV.



(a) 5 × 5 grid topology (b) 7 × 7 grid topology

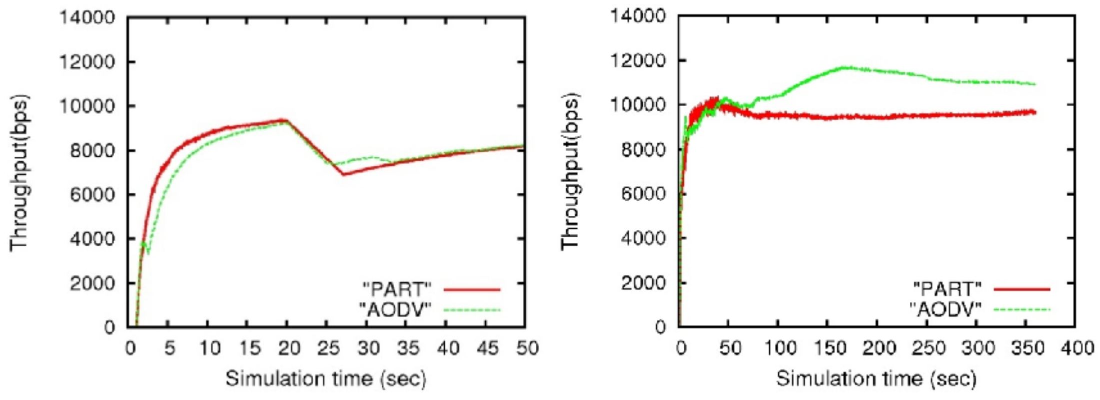
Figure 5.14 : Throughput measurement across TCP-Tahoe



(a) 5 × 5 grid topology (b) 7 × 7 grid topology

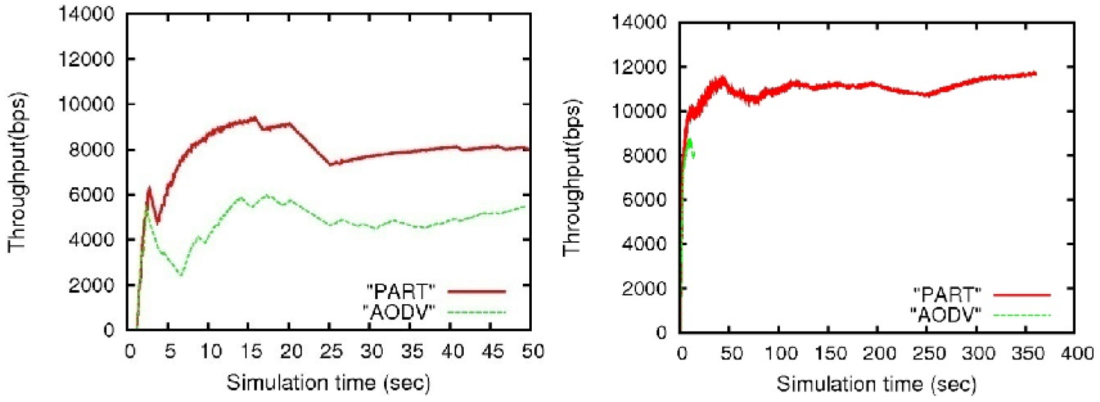
Figure 5.15 : Throughput measurement across TCP-New Reno

However, like in the chain topology, the performance of PACK mechanism is worse than Vegas with PART as shown in Figure 5.16. As it suffers a slightly lower throughput 0.6% in 5 × 5 grid, it also suffers a higher throughput degradation about 12% lower in 7 × 7 grid in Figure 5.16. Figure 5.17 shows better performance of PACK mechanism over Westwood, in terms of throughput. Table 5.6 shows the z-test results, where the improvements are statistically significant across TCP variants in grid topologies.



(a) 5 × 5 grid topology (b) 7 × 7 grid topology

Figure 5.16 : Throughput measurement across TCP-Vegas



(a) 5 × 5 grid topology (b)7 × 7 grid topology

Figure 5.17 : Throughput measurement across TCP-Westwood

Table 5.6 : Throughput measurement of TCP Variants across a grid topology in a static network

z-Test: Two Sample for Means Tahoe (5×5 grid)

	PACK with PART	*AODV*
Mean	8.7E+03	6.0E+03
Known Variance	1.7E+06	1.5E+06
Observations	8.2E+02	6.0E+02
Hypothesized Mean Difference	0.0E+00	
Z	4.1E+01	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means Tahoe (7×7 grid)

	PACK with PART	*AODV*
Mean	1.1E+04	8.8E+03
Known Variance	7.6E+05	3.4E+05
Observations	8.3E+03	5.7E+03
Hypothesized Mean Difference	0.0E+00	
Z	2.0E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means New Reno (5×5 grid)

	PACK with PART	*AODV*
Mean	8.7E+03	4.8E+03
Known Variance	1.7E+06	9.7E+05
Observations	8.1E+02	5.4E+02
Hypothesized Mean Difference	0.0E+00	
Z	6.4E+01	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means New Reno (7×7 grid)

	PACK with PART	*AODV*
Mean	1.2E+04	6.8E+03
Known Variance	4.7E+05	6.1E+05
Observations	7.6E+03	4.9E+03
Hypothesized Mean Difference	0.0E+00	
Z	3.5E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means Vegas (5×5 grid)

	PACK with PART	*AODV*
Mean	7.9E+03	7.8E+03
Known Variance	1.4E+06	1.5E+06
Observations	8.0E+02	8.1E+02
Hypothesized Mean Difference	0.0E+00	
Z	1.8E+00	
P(T<=t) one-tail	3.8E-02	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	7.7E-02	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means Vegas (7×7 grid)

	PACK with PART	*AODV*
Mean	9.5E+03	1.1E+04
Known Variance	2.8E+05	7.6E+05
Observations	6.8E+03	7.7E+03
Hypothesized Mean Difference	0.0E+00	
Z	-1.1E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means Westwood (5×5 grid)

	PACK with PART	*AODV*
Mean	7.9E+03	4.8E+03
Known Variance	1.5E+06	7.0E+05
Observations	7.3E+02	4.9E+02
Hypothesized Mean Difference	0.0E+00	
Z	5.2E+01	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means Westwood (7×7 grid)

	PACK with PART	*AODV*
Mean	11002.36	7347.275
Known Variance	457606.4	3380350
Observations	7645	215
Hypothesized Mean Difference	0	
Z	29.09448	
P(T<=t) one-tail	0	
Z Critical one-tail	1.644854	
P(T<=t) two-tail	0	
Z Critical two-tail	1.959964	

5.4.2.2 Average Delay Measurements across Variants of TCP

When we measure the average delay across grid topologies, PACK mechanism maintains optimum delay almost similar to AODV in 5 × 5 grid, whereas it incurs a slightly higher delay if compared to AODV for Westwood. In the 7 × 7 grid, all TCP variants with PACK perform worse than AODV as shown in Figure 5.18. In the grid topologies, nodes are close to one another, making it easy for them to exchange information. A proxy node assists to increase throughput by checking for missing sequence numbers and retransmitting the missing packets in advance. However, this checking may cause longer delay in the grid topologies.

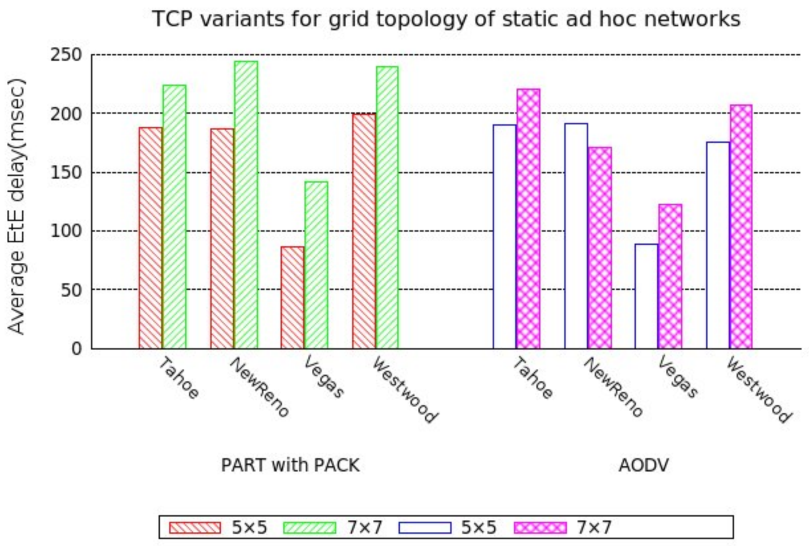


Figure 5.18 : Average delay measurement across TCP variants

5.4.2.3 Packet Loss Rate Measurements across Variants of TCP

Even though PACK does not significantly reduce delay, PACK reduces the percentage of packet loss rates for the grid topologies as shown in Figure 5.19. Tahoe-PACK over PART reduces packet losses by almost 58% in 5 × 5 and by 13% in 7 × 7 topologies. Also a significant achievement of New Reno-PACK is observed where it reduces packet losses by almost 34% in 5 × 5 and 46% in 7 × 7 grid topologies. The percentages of packet loss rate for Westwood-PACK over PART are 36% lower in 5 × 5 grid and 54% lower in 7 × 7 grid topologies if compared to Westwood over AODV. Although the PACK mechanism does not provide a significant performance improvement in Vegas in terms of throughput and delay, it reduces the packet losses significantly by almost 16% in 5 × 5 grid and 28% in 7 × 7 grid topologies when the performances are compared to the AODV.

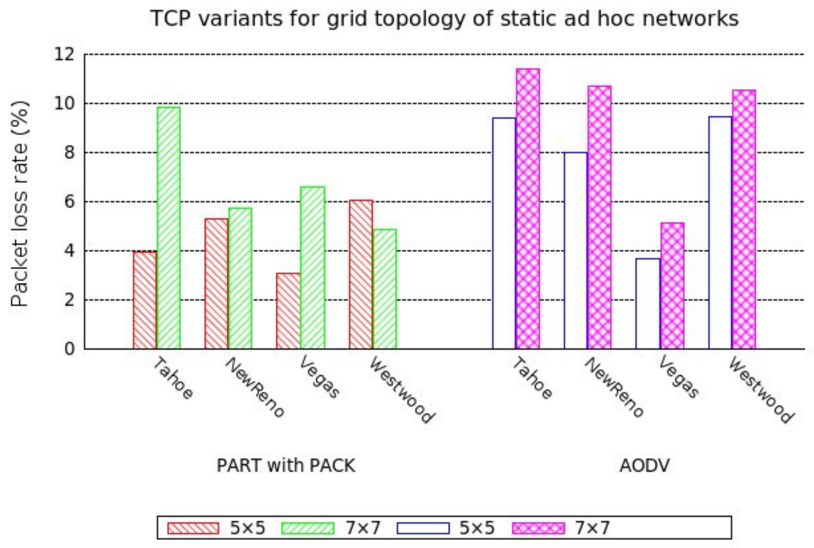


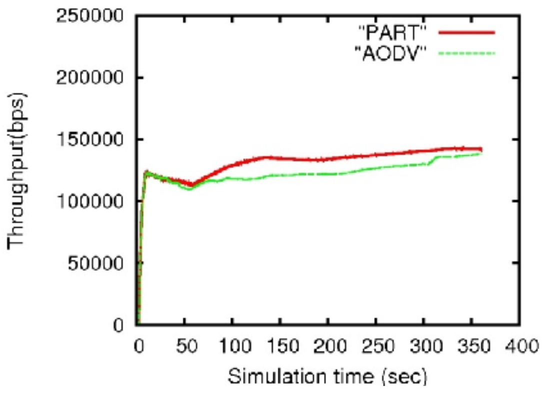
Figure 5.19 : Packet loss rate measurement across TCP variants

5.4.3 Random Topology in Mobile Network

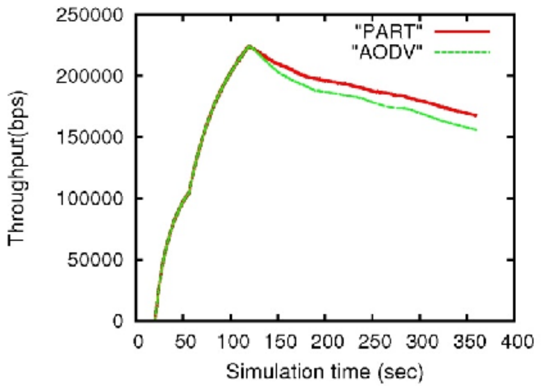
To assure the effectiveness of the PACK mechanism, the random topologies are simulated across the mobile environment, and the performance metrics measured. For this purpose, we run 30 nodes in 1500m x 300m simulation area for 360 seconds. The 10 FTP applications are exchanged between each pair of source and destination. We vary node speed from 1m/s to 20m/s with pause time 100 seconds. The random movements are generated using "*setdest*" command under NS-2 directory, which is described in Chapter 4.

5.4.3.1 Throughput Measurements across Variants of TCP

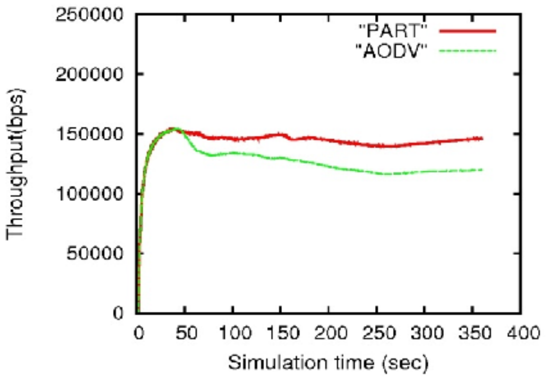
When we measure the performance of throughput by varying node speeds, Tahoe-PACK over PART outperforms Tahoe over AODV by 3% higher at 1m/s, 7% higher at 10m/s, 22% higher at 15m/s and 11 % higher at 20m/s as shown in Figure 5.20.



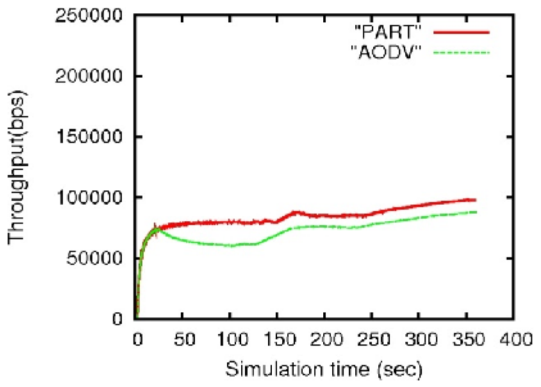
(a) Node speed 1m/s



(b) Node speed 10m/s



(c) Node speed 15m/s



(d) Node speed 20m/s

Figure 5.20 : Throughput measurement across TCP-Tahoe

As the node speed increases, the possibility of route errors increases due to the collisions and route breaks. However, the base protocol, PART, reduces the possibility of route breaks with the assistance of proxy nodes.

Tables 5.7 and 5.8 show the number of collisions and route breaks. Tahoe-PACK over PART reduces the amount of collisions and route breaks, resulting in increased throughput.

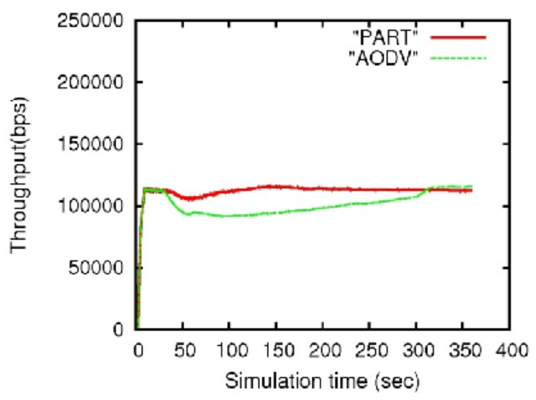
Table 5.7 : Number of collisions at the MAC layer

Speed / Protocols	1m/s	5m/s	10m/s	15m/s	20m/s
Tahoe-PACK with PART	22,685	31,447	15,856	19,934	25,012
Tahoe with AODV	25,342	40,136	23,565	31,321	31,516

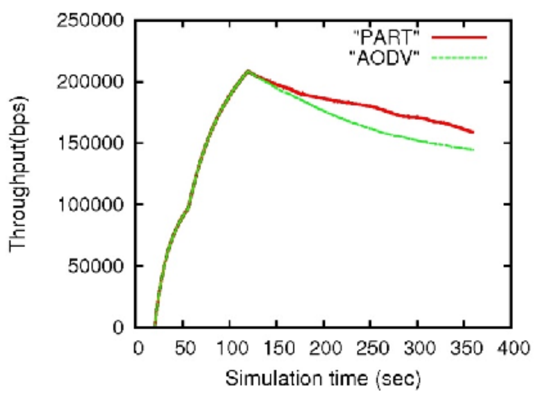
Table 5.8 : Number of route breaks at the MAC layer

Speed / Protocols	1m/s	5m/s	10m/s	15m/s	20m/s
Tahoe-PACK with PART	21	27	41	15	34
Tahoe with AODV	45	144	68	69	98

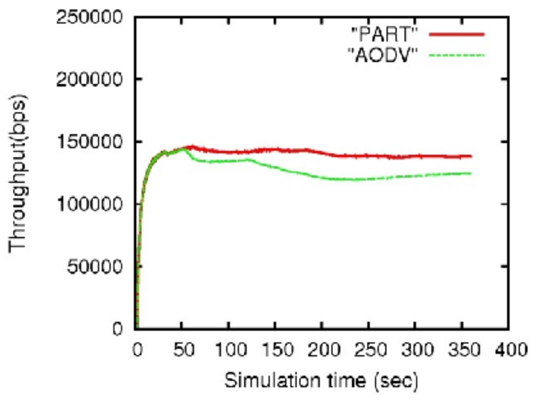
In Figure 5.21, New Reno-PACK over PART outperforms AODV by almost 1% higher at 1m/s, 14% higher at 10m/s, 8% higher at 15m/s and 4% higher at 20m/s. Also in Figure 5.22, Vegas-PACK over PART performs 3% higher at 1m/s, 7% higher at 10m/s, 22% higher at 15m/s and 11 % higher at 20m/s when compared to Vegas over AODV. Although we have encountered the weakness of PACK mechanism over Vegas in the static chain and grid topologies, the significant throughput improvements are observed as the node movements and TCP traffic are randomized. Caasetti et al., 2002 pointed out that Westwood performs well over lossy links. Our simulation results also show that Westwood's performance is more significant when the node speed increases. In Figure 5.23, Westwood-PACK performs 10% higher at 10m/s, 11% higher at 15m/s and 11 % higher at 20m/s if compared to AODV. Tables 5.9 to 5.11 show the t-test results, where the improvements are significant across TCP variants in random topologies.



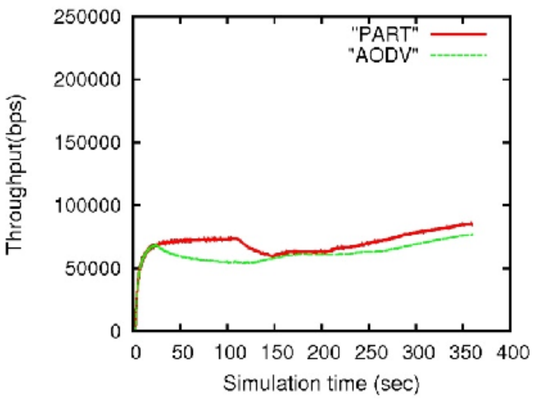
(a) Node speed 1m/s



(b) Node speed 10m/s



(c) Node speed 15m/s



(d) Node speed 20m/s

Figure 5.21 : Throughput measurement across TCP-New Reno

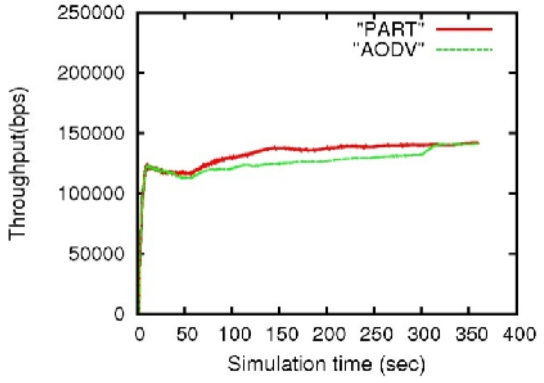
Table 5.9 : Number of collisions of TCP Variants in a mobile network

t-Test: Two-Sample Assuming Equal Variances (Tahoe)

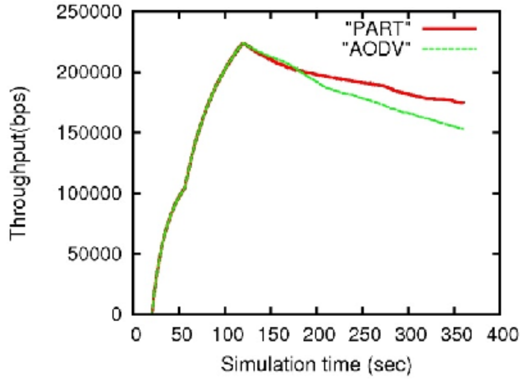
	PACK with PART	***AODV***
Mean	2.3E+04	3.0E+04
Variance	3.4E+07	4.2E+07
Observations	5.0E+00	5.0E+00
Pooled Variance	3.8E+07	
df	8.0E+00	
t Stat	-1.9E+00	
P(T<=t) one-tail	4.8E-02	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	9.5E-02	
T Critical Two-tail	2.3E+00	

Table 5.10 : Number of route breaks of TCP Variants in a mobile network

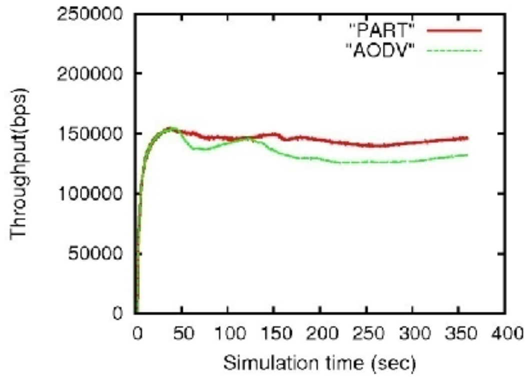
t-Test: Two-Sample Assuming Equal Variances (Vegas)		
	PACK with PART	***AODV***
Mean	2.8E+01	8.5E+01
Variance	1.1E+02	1.4E+03
Observations	5.0E+00	5.0E+00
Pooled Variance	7.8E+02	
df	8.0E+00	
t Stat	-3.2E+00	
P(T<=t) one-tail	5.9E-03	
T Critical one-tail	1.9E+00	
P(T<=t) two-tail	1.2E-02	
T Critical Two-tail	2.3E+00	



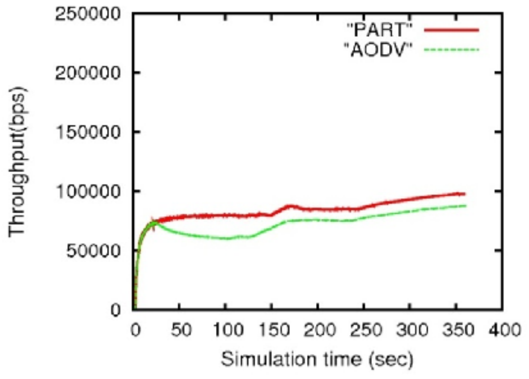
(a) Node speed 1m/s



(b) Node speed 10m/s

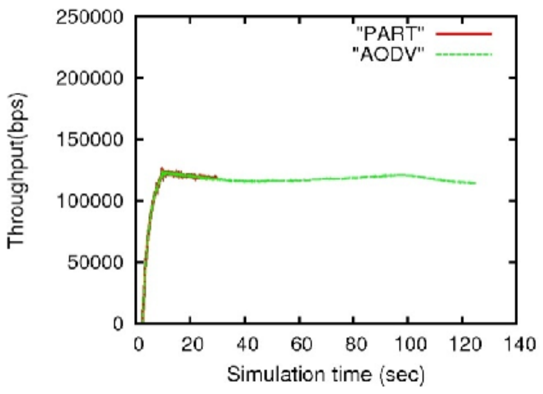


(c) Node speed 15m/s

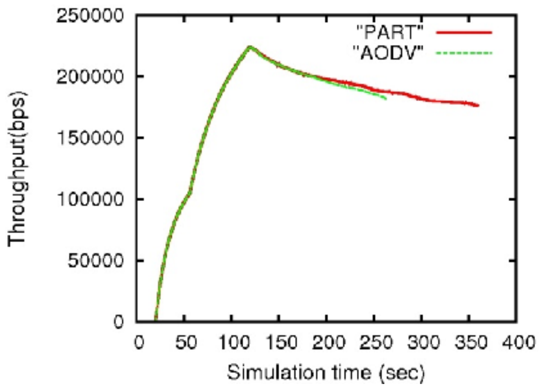


(d) Node speed 20m/s

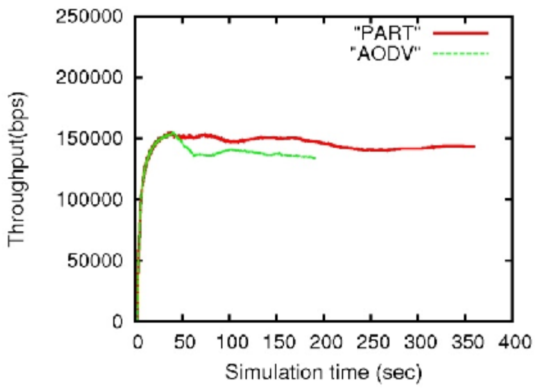
Figure 5.22 : Throughput measurement across TCP-Vegas



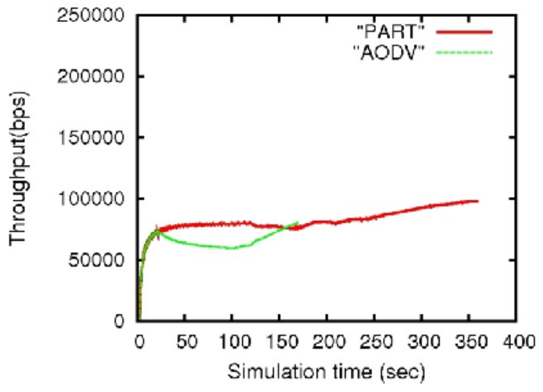
(a) Node speed 1m/s



(b) Node speed 10m/s



(c) Node speed 15m/s



(d) Node speed 20m/s

Figure 5.23 : Throughput measurement across TCP-Westwood

Table 5.11 : Throughput measurement of TCP Variants across a random topology in a mobile network

z-Test: Two Sample for Means Tahoe		
	PACK with PART	***AODV***
Mean	1.6E+04	1.5E+04
Known Variance	5.7E+05	8.1E+05
Observations	3.8E+03	3.5E+03
Hypothesized Mean Difference	0.0E+00	
Z	6.6E+01	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means New Reno		
	PACK with PART	***AODV***
Mean	1.6E+04	1.5E+04
Known Variance	6.6E+05	9.0E+05
Observations	3.8E+03	3.6E+03
Hypothesized Mean Difference	0.0E+00	
Z	5.8E+01	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means Vegas		
	PACK with PART	***AODV***
Mean	1.7E+04	1.5E+04
Known Variance	5.3E+05	1.1E+06
Observations	4.1E+03	3.7E+03
Hypothesized Mean Difference	0.0E+00	
Z	7.5E+01	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

z-Test: Two Sample for Means Westwood		
	PACK with PART	***AODV***
Mean	1.4E+04	1.2E+04
Known Variance	5.2E+05	3.3E+05
Observations	3.3E+03	2.8E+03
Hypothesized Mean Difference	0.0E+00	
Z	1.2E+02	
P(T<=t) one-tail	0.0E+00	
Z Critical one-tail	1.6E+00	
P(T<=t) two-tail	0.0E+00	
Z Critical two-tail	2.0E+00	

5.4.3.2 Average Delay Measurements across Variants of TCP

The PACK mechanism over TCP variants suffers a higher delay at moderate speeds (i.e. 1 and 5m/s) and starts achieving a lower delay starting from 10m/s speed. Figure 5.24 shows the average delay measurement with TCP variants. On average, the PACK mechanism reduces the average delay about 10% lower in Tahoe, 5% lower in New Reno, 8% lower in Vegas and 9% lower in Westwood over PART protocol compared to AODV.

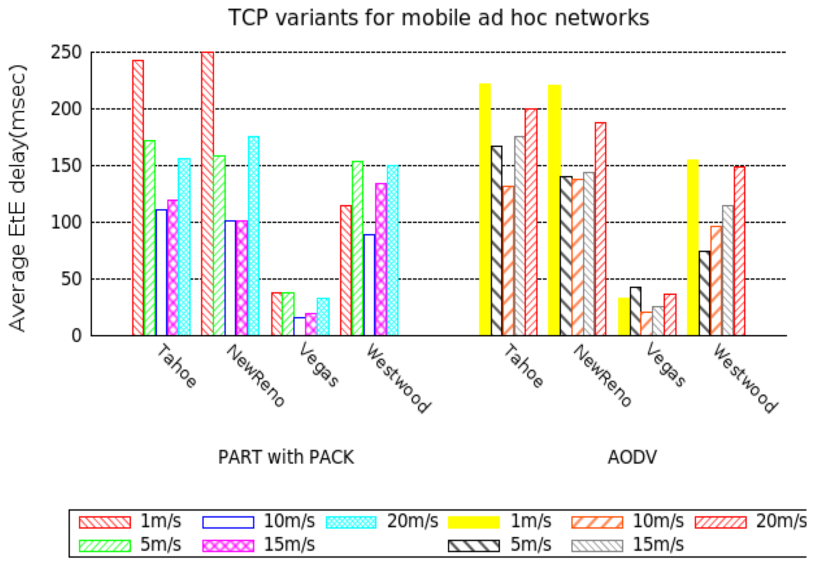


Figure 5.24 : Average delay measurement across TCP variants

5.4.3.3 Routing Overhead Measurements across Variants of TCP

Finally, we investigate the effects of node speed on the PACK mechanism with TCP variants while the nodes move randomly. As shown in Figure 5.25, TCP variants with PACK over PART achieve a significantly lower overhead than AODV. Whenever route errors occur, AODV invokes the route discovery procedure and discovers a new route across the whole network. On the contrary, the PART protocol has a special mechanism depending on the types of route error.

If the route errors occur due to the collision, PART repairs the route locally instead of sending route error messages to the source node. These upshots affect PART protocol over variants of TCP with PACK. Moreover, PACK is able to retransmit the missing sequence number as soon as possible. Figure 5.25 shows that the PACK mechanism with TCP variants reduces routing overhead almost 66% lower in Tahoe, 68% lower in New Reno, 83% lower in Vegas and 95% lower in Westwood over PART when compared to AODV. The simulation results are significant as expected — the sequence number checking of PACK positively influences the best performance for TCP variants.

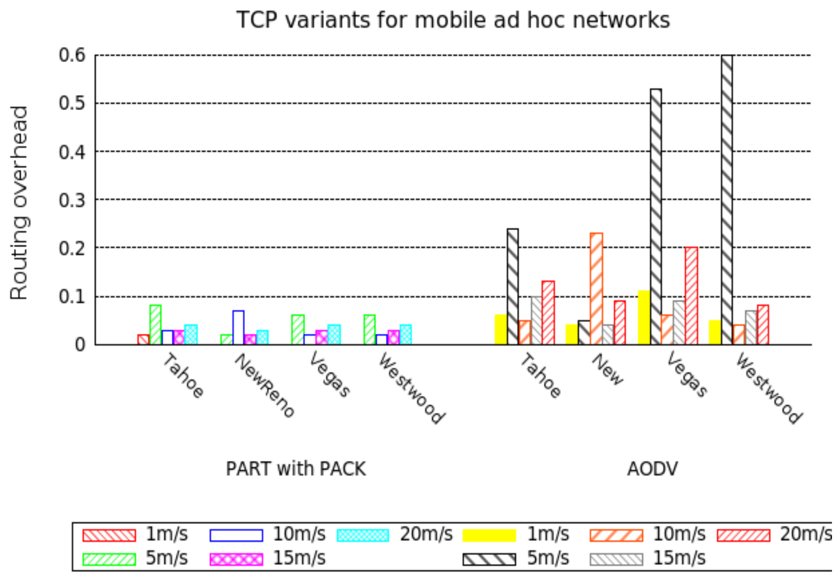


Figure 5.25 : Routing overhead measurement across TCP variants

5.5 Chapter Summary

We present a new mechanism called local acknowledgement from proxy (PACK) that detects the missing sequence number between each pair of source and destination. The PACK mechanism is applied to the variants of TCP, such as Tahoe, New Reno, Vegas and Westwood. We measured the performance differences by varying network topologies, such as the chain, grid and random in static and mobile ad hoc environments. Simulation results show that PACK provides a better performance with TCP variants over PART protocol. In the chain topology, as the path length increases, PACK achieves a significantly greater throughput up to 50% over Tahoe, 70% over New Reno and 45% over Westwood. Moreover, PACK with TCP variants reduces the average delay up to 10% and increases the PDF almost 5% over PART. In the grid topology, PACK over PART with TCP variants provides a better throughput up to 55% and a lower packet loss up to 60%. In random topology, the node speeds are varied from 1m/s to 20m/s. Simulation results show that PACK with TCP variants has a higher throughput up to 22%, a lower delay up to 10% and a lower routing overhead up to 95% over PART as the nodes move randomly in the network.

Chapter 6

ANALYTICAL STUDIES OF THE INTERACTION BETWEEN MOBILITY MODELS AND ROUTING PROTOCOLS

6.1 Introduction

As mentioned in Chapters 2 and 3, routing and transport protocols have been a subject of research in the networking community for many years. The responsibilities of a routing protocol are to detect route failure, maintain an optimal route, and support data transmission efficiently. So far, many ad hoc routing protocols have been proposed and each protocol is developed with some advantage over the others. However, as we mentioned in our paper (Oo and Othman, 2011), protocols are not equally good across all metrics, such as average delay, throughput, routing overhead and so on. For example, Oo and Othman (2011) clarified that on-demand routing protocols do not always perform better than table-driven ones. Likewise, the multipath routing protocols are not always able to reduce routing overhead compared to the single path routing protocols when the node speed increases.

Routing protocols for MANETs have been tested and evaluated in so many literatures. Broch et al. (1998) compared four routing protocols: DSR, AODV, DSDV and TORA, and performed experiments using a realistic physical layer measurement and the IEEE 802.11 protocol with Distributed Coordination Function (DCF). Divecha et al. (2007) also studied the effects of various mobility models with two routing protocols.

For realistic measurement, Johansson et al. (1999) compared DSDV, AODV and DSR by introducing three realistic scenarios to test the protocols in more specialized contexts. Prabhakaran et al. (2006) incorporated more realistic mobility models, including entity mobility models and group mobility models in the multipath fading environments and studied the performance metrics such as the energy goodput, packet delivery ratio, and control overhead packets generated by using AODV.

Most performance measurements are based on CBR traffic, whereas Oo and Othman (2011) examined the performance comparisons based on the application traffic (i.e. TCP and CBR traffic) with various mobility models, single path and multipath (Proactive and Reactive) routing protocols. In general, DSR is better than AODV due to the source routing technique. However, as the network size increases, AODV performs better due to its quick adaptation.

The performance measurements among routing protocols discussed in Chapter 4 were done without applying any mobility models. The objective of this chapter is to incorporate mobility models and build an analytical framework that evaluates the performance of PART against other protocols in terms of topology changes, various TCP traffic types, and node movement patterns (i.e. using different mobility models).

6.2 Overview of Mobility Models

In this section, we give a brief overview of mobility models. It is very important to measure the mobility models together with the routing protocols for the realistic movements of mobile users.

6.2.1 Random Waypoint Mobility Model (RWP)

The Random waypoint mobility model (RWP) (Camp et al, 2002) has been used to evaluate the ad hoc routing protocols because of its simplicity and wide availability. The RWP Model includes pause times, minimum speed and maximum speed. A pause time is chosen between changes in direction and/or speed. A node starts moving from a randomly chosen position and stay in one location for a certain period of time (i.e. a pause time). Once this time expires, the node chooses a destination and moving speed randomly. This speed is uniformly distributed between minimum speed and maximum speed [*minspeed*, *maxspeed*]. The node then moves toward the newly chosen destination at the selected speed. Upon arrival at the destination, the above process is started over again. The movement pattern of this model is illustrated in Figure 6.1. The RWP model is used in many prominent simulation studies of ad hoc network protocols. Due to its flexibility, it appears to create realistic mobility patterns for the way people might move in, for example, a conference setting or museum.

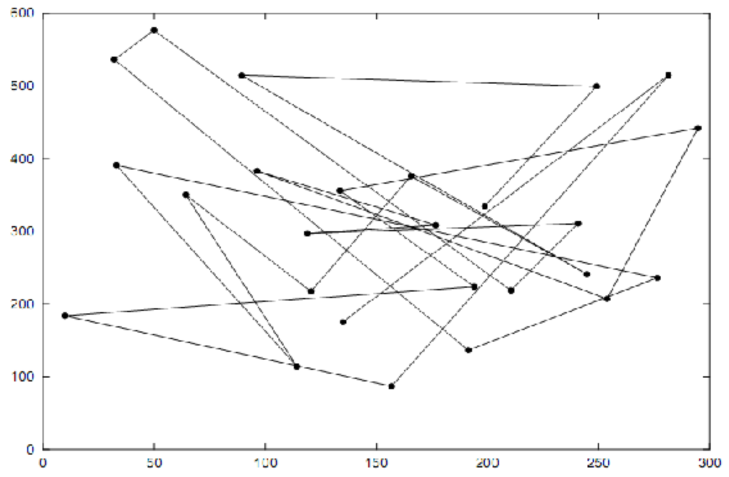


Figure 6.1 : Movement patterns of RWP model (Camp et al, 2002)

6.2.2 Manhattan Grid Mobility Model (MG)

MG represents a street network to model a city section and uses a grid road topology as shown in Figure 6.2 (Jayakumar and Gopinath, 2008). This model is considered to apply in realistic movements representing node movements in an urban area, where the streets are organized along the grid of horizontal and vertical directions on the urban map. At the intersection, the mobile nodes choose between moving on the same street, or turning left or right based on a probability.

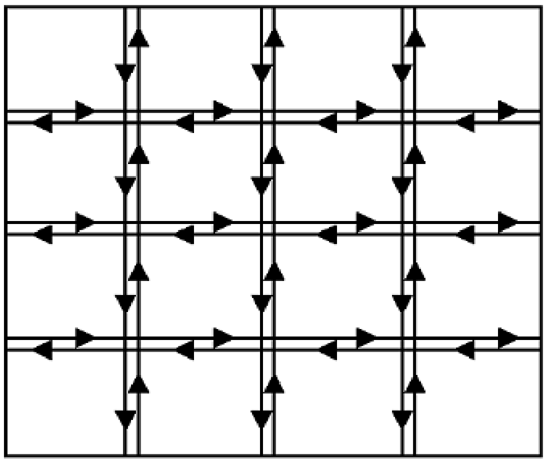


Figure 6.2 : Movement patterns of MG model (Jayakumar and Gopinath, 2008)

6.2.3 Gauss-Markov Mobility Model

Liang and Hass (2003) proposed this model. It adapts to different levels of randomness via one tuning parameter. Initially, each node is assigned a current speed and direction. Later, the movement of a node is updated based on its past speed and direction. By allowing past directions to influence future directions, Gauss-Markov eliminates sudden stops and sharp turns.

6.2.4 Reference Point Group Mobility Model (RPGM)

RPGM (Hong et al., 1999) is a group model which represents the random movement of a group and each node within the group. Group movements of nodes are based on a logical center, which defines the group motion behavior including location, speed, acceleration, etc. There are two vectors: group motion vector and individual motion vector to define the movement of each node in the network. Each individual node within the group has a reference point that follows the group movement. The group motion is specified with check points. By changing the check points, the various moving scenarios are created. Usually, nodes are uniformly distributed within the geographic scope of a group. A node is randomly placed in the neighborhood of its reference point at each step. Whenever the group reaches its destination, all nodes inside the group pause for a specific time and then start moving again. The sample movement of RPGM model is shown in Figure 6.3.

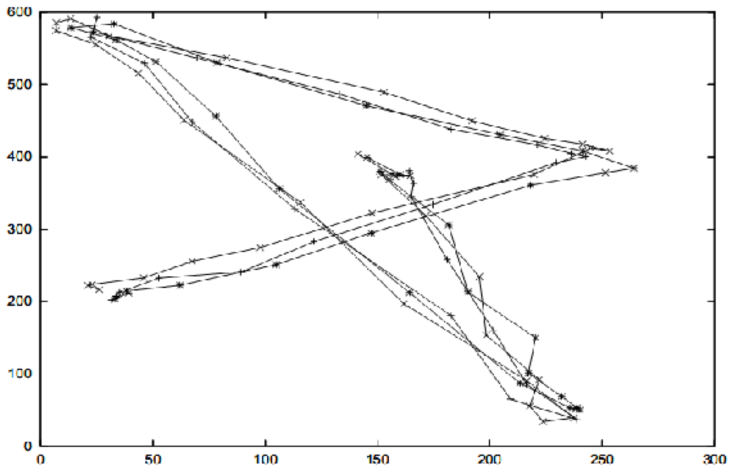


Figure 6.3 : Movement patterns of RPGM model (Camp et al, 2002)

6.3 Generation of Mobility Models with NS-2

There is a separate mobility generator, called BonnMotion, which was developed by the Communication Systems group at the Institute of Computer Science IV of the University of Bonn, Germany (BonnMotion: a Mobility Scenario Generation and Analysis Tool 2009). It is a very useful tool for GloMoSim/QualNet and NS-2.

The BonnMotion supports many mobility models: Random Waypoint (Camp et al., 2002), Gauss-Markov (Liang and Haas, 2003), Manhattan Grid (Buruhanudeen et al., 2007), Reference Point Group Mobility Model (Hong et al., 1999), Disaster Area Model (Aschenbruck et al, 2009), Random Street (Aschenbruck and Schwamborn, 2010), Random Direction (Camp et al, 2002), Random Walk, Probabilistic Random Walk Model (Camp et al. 2002), Column Mobility Model (Camp et al, 2002), Nomadic

Community (Camp et al, 2002) and Purse Mobility Model (Camp et al, 2002) are generated using the following command.

"*bm –f <output_name> <model_type> -n <numbers_of_node> -d <sim_time> -i<default_value> -x <simulation area (x-axis)> -y <simulation area (y-axis)>*"

For Random Waypoint → *bm –f scenario1 RandomWaypoint –n 100 –d 900 –i 3600 –x 1200 –y 600*

The "*i*" is the cutting value, and must be a high default value because nodes have a higher probability of being near the center of the simulation area, while they are initially distributed over the simulation area. The "*n*" is the number of nodes and the "*d*" is the simulation seconds. The "*x* and *y*" are the simulation areas.

6.4 Analytical Framework for Network Performance Tests

Figure 6.4 shows an analytical framework that illustrates an analysis structure to test the overall network performance. The FTP applications are generated by the application layer protocol. Different transport layer protocols, such as Tahoe, New Reno and Vegas, are used by the applications. Then, the movement patterns of nodes are generated according to the mobility models used. As the performance metrics, the properties of mobility models, such as average node degree, average number of partitions, link duration, etc. are measured first. Finally, the performance of the routing protocols is examined by utilizing the mobility models and TCP variants together.

The analytical framework contains mobility models (RWP, Manhattan, Gauss-Markov and RPGM), routing protocols (PART, AODV, DSR, OLSR, and AOMDV), and traffic types (Tahoe, New Reno and Vegas) for FTP applications that are combined as an integrated framework to compare the overall performance of the network.

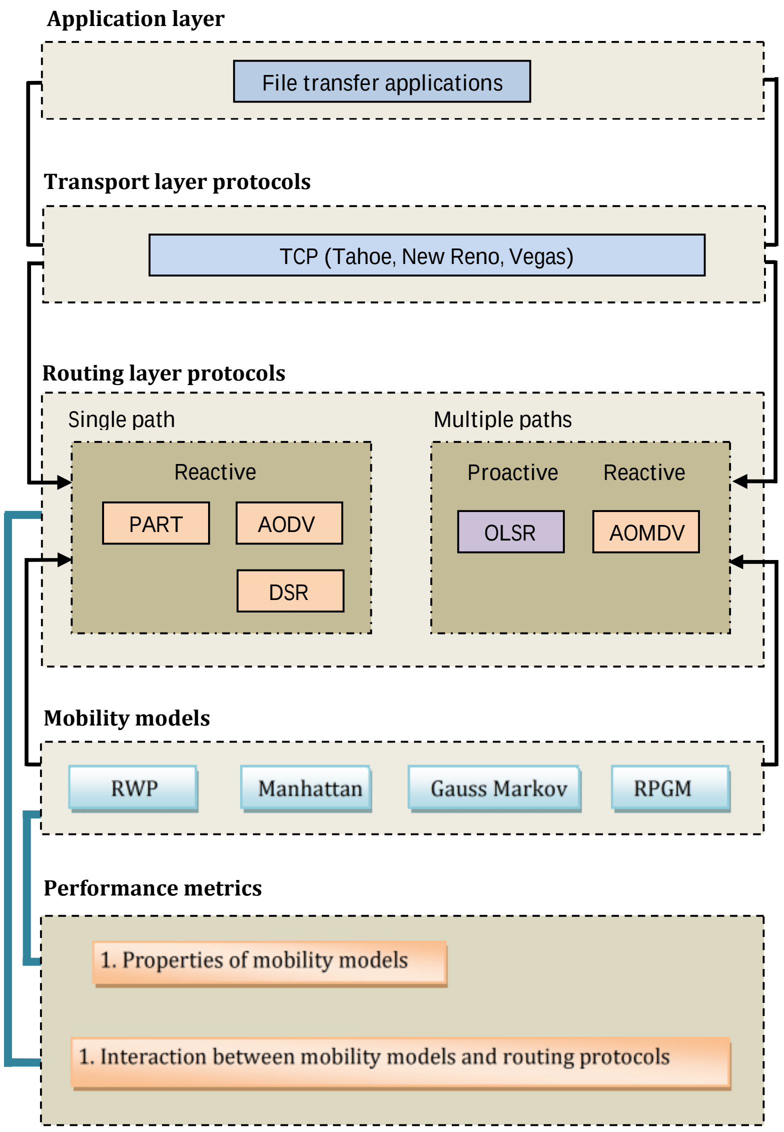


Figure 6.4 : Analytical framework for overall network performance

6.5 Comparison of Mobility Models' Properties

Objective of experiment: To ascertain how the properties of mobility models vary with different node velocities.

Expected outcome: The properties of mobility models depend on the variations of node speed, which in turn, affect the link conditions of the mobility models.

We simulated the mobility models using Bonnmotion version 1.3a. We run each simulation for 600 s with the following parameters to measure the properties of mobility models with speed changes.

- In RWP, the pause time is set to zero for the continuous movement of nodes and node speed is set to either one of the following: 1m/s (low speed), 10 m/s (medium speed) and, 20 m/s, 30 m/s and 40 m/s (high speed) with the simulation area (1200 x 600).
- In Gauss-Markov, initially each mobile node is assigned a speed and direction. At fixed time intervals, movement occurs by updating the speed and direction of mobile nodes. Specifically, the value of speed and direction is calculated based on the value of current speed and direction. Therefore, we set a standard deviation of velocity changes to 0.5 and the update frequency of speed and direction is 2.5.
- In Manhattan, we define a grid of 3 blocks at the horizontal axis and 2 blocks at the vertical axis, an update distance equal to 5 m, a turn probability equal to 0.5, a speed change probability equal to 0.2, pause probability equal to 0.
- In RPGM, when a node comes into the area of another group, it switches to this new group with a probability of 0.01. The maximum distance is equal to 2.5 m.

To measure the properties of mobility models, the following parameters are analyzed.

- An *average node degree* (*ND*) is defined as the number of neighbor nodes averaged over the number of nodes at every instant time. Two nodes are neighbors if they are within transmission range of each other.

$$ND = \frac{\sum_{t=1}^{T}\sum_{a=1}^{N} N(a,t)}{T.N} \qquad (1)$$

where, N is the number of nodes, T is the simulation time, and $N(a, t)$ is the number of neighbor nodes for node a at time t (Lu et al, 2004).

- An *average number of partitions* define that if the number of partitions that exists in the network after the occurrence of event e is denoted as $\left|\prod(e)\right|$, then the average number of partitions over the time interval T is defined as follows:

$$\prod_{avg}(T) = \frac{1}{t_{\max} t_{\min}} = \sum_{a=1}^{\left|E_{part}(T)\right|-1} (\in(a+1).t - \in(a).t \left|\prod(\in(a))\right| \qquad (2)$$

where $E_{part}(T) = \{e \mid e.t \in T\}$, the set of partition events in T. (Hahner et al, 2007)

- The *Link Duration (LD)* is an average duration that a link exists between two nodes a and b. It is a measurement of link stability. Formally,

$$LD(a,b) = \begin{cases} \dfrac{\sum_{t=1}^{T} X(a,b,t)}{LC(a,b)} & \text{if LC(a,b)} \neq 0 \\ \sum_{t=1}^{T} X(a,b,t) & \text{Otherwise} \end{cases} \qquad (3)$$

- The *average link duration* is the value of $LD(a,b)$ averaged over node pairs. (Bai et al, 2003). Formally,

$$\overline{LD} = \frac{\sum_{a=1}^{N} \sum_{b-a+1}^{N} LD(a,b)}{P} \qquad (4)$$

where P is the number of pairs (a,b) and $LC(a,b)$ is number of link changes for a pair of nodes.

$$\overline{LC} = \frac{\sum_{a=1}^{N} \sum_{b-a+1}^{N} LC(a,b)}{P} \qquad (5)$$

- *Link breaks* are how many links break down during the simulation time.
- *Total links* are the total number of links during the simulation time.

We compare the average node degree, average number of partitions, link breaks, average link duration and total links of four mobility models within a 100-m transmission range by varying the node speed. In all mobility models, while the average node degree and the average number of partitions are approximately constant for all speed rates, the link breaks increase immediately. The average link duration decreases at once when the node speed increases as shown in Table 6.1.

Not surprisingly, the average node degree and the average link duration of the RPGM are the highest and the average number of partitions is the lowest due to its group movement pattern. The nodes in the RWP merge together to the center quickly due to the zero pause time and having more neighbors result in a higher average node degree compared to the other two models. Similar to (Liang and Haas, 2003), the average node degree of Manhattan model is almost the same at different speeds due to the restrictions imposed by the map (e.g. streets) to limit node deployment.

The RWP model has lower average numbers of partitions, although its movement pattern is random when compared to the Gauss-Markov and Manhattan models. Sometimes, the random movement patterns may reduce the number of network partitions because nodes randomly move close to the center within the predefined area in an organized manner. The closer the node moves, the lesser the number of partitions. The average node degree and the number of partitions are inversely proportional. In the Gauss Markov, the average node degree is the least and the number of partitions is the highest.

Although Gauss-Markov incurs the largest number of partitions, its average link duration is higher than the RWP model because it takes more time to calculate the new speed and direction in comparison to RWP. The average link duration increases in the Manhattan model because the duration of a link depends on the directions that the nodes choose at crossings. Larger blocks result in a long connection period for nodes that take the same turn at a crossing. Apart from our results, Hahner et al, (2007) also discovered that the size of blocks in the Manhattan model has a major impact on link stability. The RWP model incurs less average link duration because of its randomness.

Table 6.1 : Performance comparison of the mobility models

Metric	Node Speed (m/s)	RWP	Gauss Markov	Manhattan	RPGM
Average node degree	1	1.5	1.0	1.5	4.7
	10	1.7	1.2	1.5	5.2
	20	1.8	1.2	1.5	4.8
	30	1.8	1.1	1.5	4.5
	40	1.7	1.1	1.5	5.0
	Mean	1.7	1.1	1.5	4.8
	SD	0.1	0.1	0.0	0.3
Average number of partitions	1	19.6	26.3	20.4	10.2
	10	20.1	25.4	22.8	12.2
	20	19.9	26.1	24.0	11.4
	30	19.6	28.1	23.6	11.4
	40	20.4	26.2	23.0	10.6
	Mean	19.9	26.6	22.8	11.2
	SD	0.3	1.0	1.4	0.8
Average link duration	1	128.9	179.2	255.6	296.5
	10	31.1	21.7	33.7	86.4
	20	18.7	10.7	16.2	42.1
	30	15.3	6.0	11.5	40.4
	40	10.6	5.7	9.0	36.2
	Mean	40.9	44.9	65.4	100.5
	SD	49.8	75.4	106.8	111.4
Link beaks	1	83	45	41	73
	10	724	783	591	780
	20	1329	1659	1212	1489
	30	1718	2267	1889	1543
	40	2254	2964	2460	1824
	Mean	1222	1544	1239	1141
	SD	847	1161	971	710

The link break conditions of the RWP, Manhattan and RPGM models are not much different from each other. In Gauss-Markov, the growing network partition results in an increased number of average link breaks. Setting the pause time to zero makes the frequency of updating previous speed and direction higher. Similar to our analysis, Hahner et al, (2007) also outlines that in the Gauss-Markov mobility model, if a link between two nodes moving in opposite direction breaks, the number of links goes down because a node takes more time to traverse another node's coverage area with the typical relative speed.

Two factors affect the dynamics of links between nodes: the distance between the nodes and the relative speed between them. If two nodes are connected and are moving away then the links between them will break. As nodes move closer to each other, a link is created. Our simulation results show that Gauss-Markov is the worst, especially for the average number of partitions and link breaks when the node velocity increases in the network.

6.6 Interaction between Routing Protocols and Mobility Models

Objective of experiment: To examine the interaction between routing protocols and mobility models.

Expected outcome: The interaction between mobility patterns and routing protocols contributes significantly to the overall network performance.

In MANETs, civilian and military applications play a vital role for users who move round and share information with each other. The movement of users varies depending on the environment, e.g. people may move randomly in different directions (Random waypoint model); or walk, run and drive in two directions in the street (Manhattan Mobility Model); or move as a group (Reference Point Group Mobility model).

Table 6.2 lists the simulation parameters that were examined. The 100 nodes move within the 2500 x 1500 simulation area at various speeds for about 600 seconds. 70 FTP (File Transfer Protocol) applications of TCP are generated with default TCP window size 32 and the packet size 512 bytes. We test five speeds: 1m/s for walking, 5m/s for driving a motorcycle, 10m/s, 15m/s and 20m/s for driving cars. The mobility and traffic generations are done by using a Bonnmotion and network simulator. We evaluate the performance of routing protocols with each mobility model.

Table 6.2 : Simulation parameters for the performance tests

Parameters across RWP, Manhattan, RPGM mobility models	
Topography areas size	2500 x 1500
Simulation time	600 second
Number of nodes	100
Routing protocols	Single path - PART, AODV, DSR, DSDV, Multipath - AOMDV, OLSR
Transport protocols	TCP
Number of TCP connections	70 FTPs
Window size of TCP	32
Packet size	512 bytes
Maximum speed	1, 5, 10, 15, 20 m/s
Pause time	0 (continuous move)
Wireless channel	Two Ray Ground
Mac protocol	Mac/802.11
Antenna	OmiAntenna

6.6.1 Performance Evaluations of Routing Protocols in RWP Model

Firstly, we measure the average end-to-end delay to investigate the effect of node velocity on the relative rankings of routing protocols with TCP traffic. Figure 6.5 shows that PART incurs the lowest average delay compared to AODV and DSR as the node speed increases. DSR always suffers the highest delay due to its source routing.

On the other hand, PART could not mimic the multipath routing protocols, such as AOMDV and OLSR, at high speed. Although PART performs 16.2% better than AODV and 174.3% better than DSR, it encounters 5.8% higher delay than AOMDV and 14.1 % higher delay than OLSR.

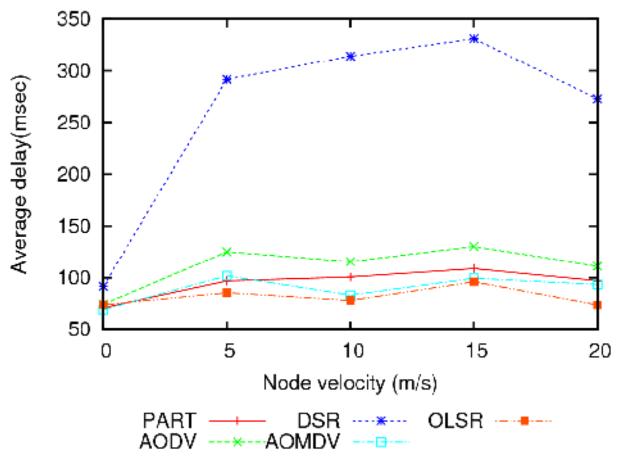
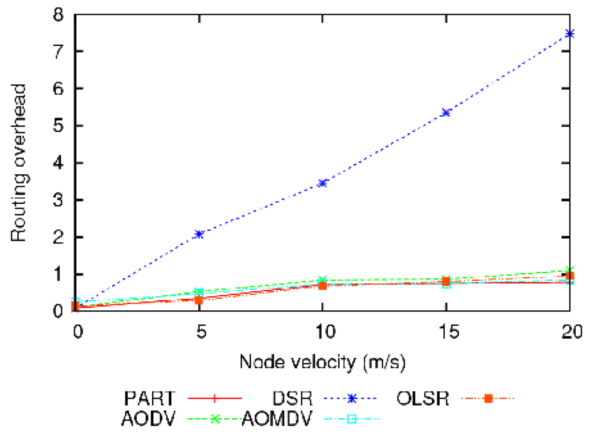
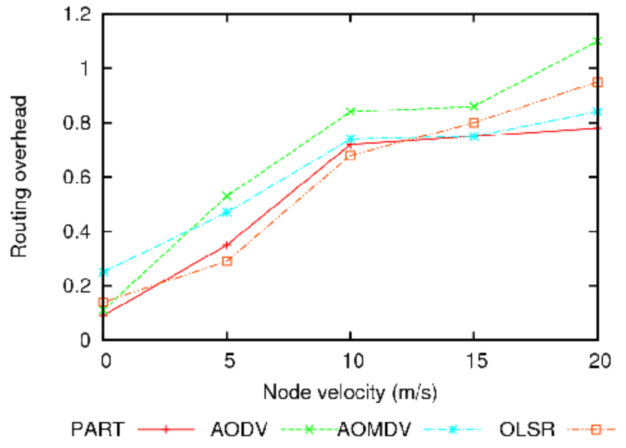


Figure 6.5 : Average delay measurement in RWP model



(a) DSR included



(b) without DSR

Figure 6.6 : NRL measurement in RWP model

Secondly, the performance comparisons of NRL are discussed for all routing protocols. As shown in Figure 6.6 (a), DSR incurs the worst routing overhead and suffers almost 600% higher overhead compared to the others as the node speed increases. PART reduces the routing overhead 27.9% lower than AODV, 13.4% lower than AOMDV and 6.3% lower than OLSR. Figure 6.6(b) compares the overhead without including DSR so that the performance comparisons of other routing protocols can be seen in the graph.

If a primary route fails in AOMDV and the backup routes cannot be provided, invoking a route discovery from the very beginning and building multiple paths tend to incur a greater routing overhead (Oo and Othman, 2011). While nodes move at a moderate speed, the chances of route failures are less and proactive routing protocols, like OLSR, can provide a ready route due to its topology-based routing information whenever route error messages are received. Even if route breaks occur under a moderate speed, its proactive routing nature allows it to provide routes at once. However, as node speed increases, the topology changes occur quickly, and thus the proactive protocols have fewer chances to provide routes at once.

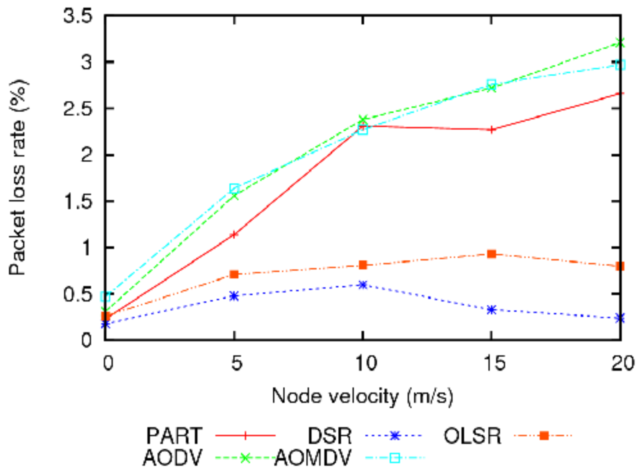


Figure 6.7 : Packet loss rate measurement in RWP model

Thirdly, we measure packet loss rates with velocity changes. As the node speed increases, the packet loss rates also increase, especially for reactive routing protocols. Even though DSR is also a reactive protocol, it reduces packet loss rates compared to the other reactive protocols. The routing caching technique of DSR suffers the highest delay as the node speed and node density increases, but it reduces packet losses. In Figure 6.7, although PART reduces the packet loss almost 19% lower than AODV and AOMDV, it suffers a higher loss rate of almost 80% higher than DSR and OLSR.

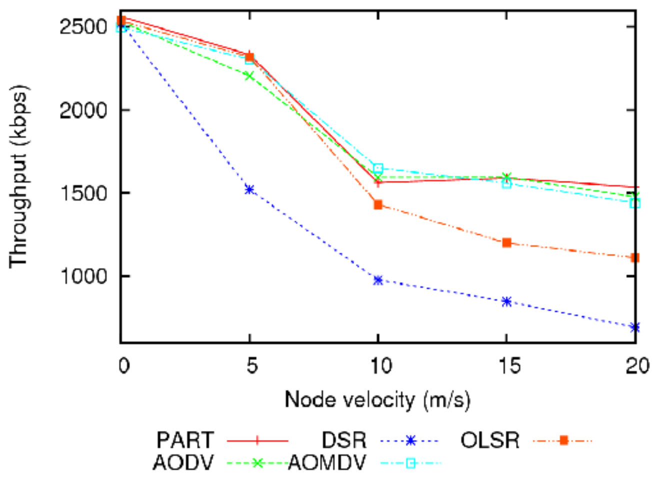


Figure 6.8 : Throughput measurement in RWP model

Finally, we measure the throughput of the routing protocols. PART performs 1.9% better than AODV, 31.5% better than DSR and 10.3% better than OLSR as shown in Figure 6.8. As the node speed increases, the performance of DSR and OLSR suffers due to the stale route problem and rapid topology changes.

The important participation of a proxy node can be seen in most situations. The assistance of proxy node succeeds not only in lowering the routing overhead, but also in reducing collisions and congestion problems due to its limited broadcast zone while maintaining optimum throughput. Table 6.3 shows the t-test results, where the improvement is statistically significant in RWP model.

Table 6.3 Performance comparisons of routing protocols in RWP model

t-Test: Two-Sample Assuming Equal Variances (delay)	***PART***	***DSR***
Mean	9.5E+01	2.6E+02
Variance	2.0E+02	9.4E+03
Observations	5.0E+00	5.0E+00
Pooled Variance	4.8E+03	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	-3.8E+00	
T Critical one-tail	2.7E-03	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	5.4E-03	

t-Test: Two-Sample Assuming Equal Variances (overhead)	***PART***	***DSR***
Mean	5.2E-01	3.6E+00
Variance	8.6E-02	7.9E+00
Observations	5.0E+00	5.0E+00
Pooled Variance	4.0E+00	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	-2.5E+00	
T Critical one-tail	2.0E-02	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	3.9E-02	

t-Test: Two-Sample Assuming Equal Variances (packet losses)	***PART***	***DSR***
Mean	1.8E+00	3.7E-01
Variance	1.2E+00	3.0E-02
Observations	5.0E+00	5.0E+00
Pooled Variance	6.2E-01	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	2.9E+00	
T Critical one-tail	9.6E-03	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	1.9E-02	

t-Test: Two-Sample Assuming Equal Variances (packet losses)	***PART***	***OLSR***
Mean	1.8E+00	7.0E-01
Variance	1.2E+00	6.7E-02
Observations	5.0E+00	5.0E+00
Pooled Variance	6.4E-01	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	2.2E+00	
T Critical one-tail	2.9E-02	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	5.8E-02	

6.6.2 Performance Evaluations of Routing Protocols in MG Model

This mobility model is more realistic and is employed for node movements within a city. When we analyze the average delay, PART incurs a lower delay if compared to DSR and AODV. The average delay of PART and AOMDV is almost similar. However, the topology-based OLSR incurs 8.7% lower delay than PART as shown in Figure 6.9. Needless to say, the topology information does not change very much in such a mobility model. As usual, DSR suffers the highest delay.

If we examine the routing overhead, PART reduces the overhead far better when compared to others (i.e. 13.8% lower than AODV, 92.2% lower than DSR, 106.9% lower than AOMDV, 64.7% lower than OLSR) as shown in Figure 6.10.

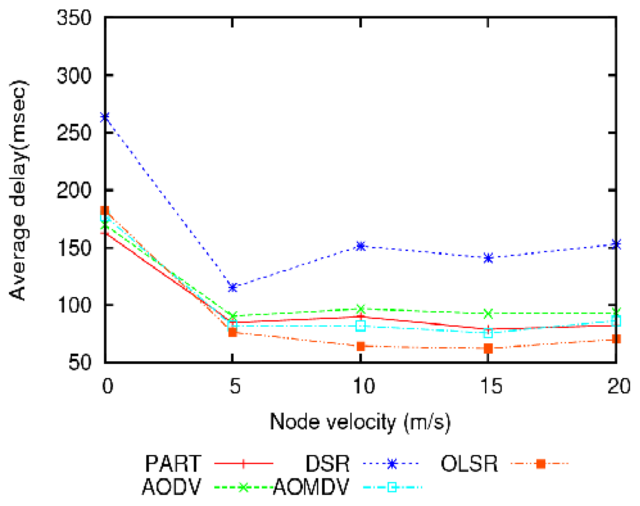


Figure 6.9 : Average delay measurement in MG model

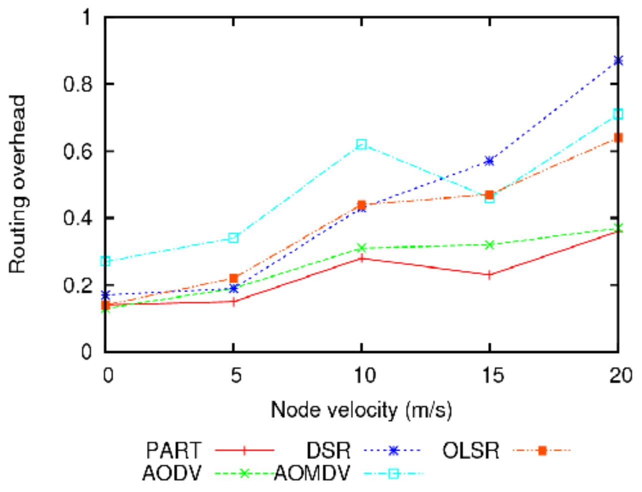


Figure 6.10 : NRL measurement in MG model

In the situation of realistic movements, the huge routing overhead of AOMDV is encountered as shown in Figure 6.10. By virtue of focusing on nodes moving along horizontal or vertical streets, the source routes of DSR and multiple routes of AOMDV cause the network to be congested as node speed and the number of TCP connections increase. Then these effects drive the stale and unusable route problems leading to the increased routing overhead.

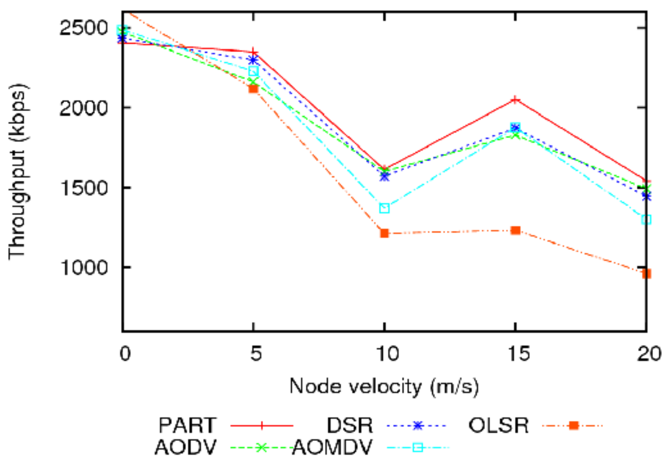


Figure 6.11 : Throughput measurement in MG model

If we look at the throughput measurement in Figure 6.11, the best performance of PART can be seen. It achieves a significantly higher throughput: about 4.1% better than AODV, 3.4% better than DSR, 6.0% better than AOMDV and 18.2% better than OLSR. PART performs very well under the realistic mobility model in terms of throughput. Table 6.4 shows the t-test results, where the improvement is statistically significant in the MG model.

Table 6.4 : Performance comparisons of routing protocols in MG model

t-Test: Two-Sample Assuming Equal Variances (throughput)

	PART	*AODV*
Mean	2.4E+03	1.9E+03
Variance	1.5E+05	1.6E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	1.5E+05	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	1.8E+00	
T Critical one-tail	5.5E-02	
P(T<=t) two-tail	1.4E+00	
T Critical Two-tail	1.1E-01	

t-Test: Two-Sample Assuming Equal Variances (throughput)

	PART	*DSR*
Mean	2.4E+03	1.9E+03
Variance	1.5E+05	2.1E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	1.8E+05	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	1.6E+00	
T Critical one-tail	7.7E-02	
P(T<=t) two-tail	1.4E+00	
T Critical Two-tail	1.5E-01	

t-Test: Two-Sample Assuming Equal Variances (throughput)

	PART	*AOMDV*
Mean	2.4E+03	1.9E+03
Variance	1.5E+05	2.7E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	2.1E+05	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	1.8E+00	
T Critical one-tail	5.7E-02	
P(T<=t) two-tail	1.4E+00	
T Critical Two-tail	1.1E-01	

t-Test: Two-Sample Assuming Equal Variances (throughput)

	PART	*OLSR*
Mean	2.4E+03	1.6E+03
Variance	1.5E+05	5.0E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	3.2E+05	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	2.1E+00	
T Critical one-tail	3.7E-02	
P(T<=t) two-tail	1.4E+00	
T Critical Two-tail	7.4E-02	

t-Test: Two-Sample Assuming Equal Variances (overhead)

	PART	*DSR*
Mean	2.3E-01	4.4E-01
Variance	7.9E-03	8.3E-02
Observations	5.0E+00	5.0E+00
Pooled Variance	4.5E-02	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	-1.6E+00	
T Critical one-tail	7.9E-02	
P(T<=t) two-tail	1.4E+00	
T Critical Two-tail	1.6E-01	

t-Test: Two-Sample Assuming Equal Variances (overhead)

	PART	*DSR*
Mean	2.3E-01	3.8E-01
Variance	7.9E-03	3.9E-02
Observations	5.0E+00	5.0E+00
Pooled Variance	2.3E-02	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	-1.5E+00	
T Critical one-tail	8.2E-02	
P(T<=t) two-tail	1.4E+00	
T Critical Two-tail	1.6E-01	

t-Test: Two-Sample Assuming Equal Variances (packet losses)

	PART	*AOMDV*
Mean	1.3E+00	1.8E+00
Variance	1.5E-01	2.8E-01
Observations	5.0E+00	5.0E+00
Pooled Variance	2.2E-01	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	-1.7E+00	
T Critical one-tail	6.8E-02	
P(T<=t) two-tail	1.4E+00	
T Critical Two-tail	1.4E-01	

t-Test: Two-Sample Assuming Equal Variances (packet losses)

	PART	*OLSR*
Mean	1.3E+00	5.7E-01
Variance	1.5E-01	1.6E-02
Observations	5.0E+00	5.0E+00
Pooled Variance	8.5E-02	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	3.9E+00	
T Critical one-tail	2.4E-03	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	4.9E-03	

6.6.3 Performance Evaluations of Routing Protocols in RPGM Model

We analyze the performance of routing protocols with the group mobility model and measure the throughput and delay. As shown in Figure 6.12, there is not much performance difference between PART, AODV and DSR in terms of throughput. However, PART performs slightly better than the multipath AOMDV and OLSR routing protocols. If we look at the performance of average delay, the similar performances of PART and AODV can be seen in Figure 6.13, where PART reduces the average delay almost 38% lower than DSR and 7% lower than AOMDV and OLSR. The simulation results are significant as expected. Table 6.5 shows the t-test results, where the improvements are statistically significant in the RPGM model.

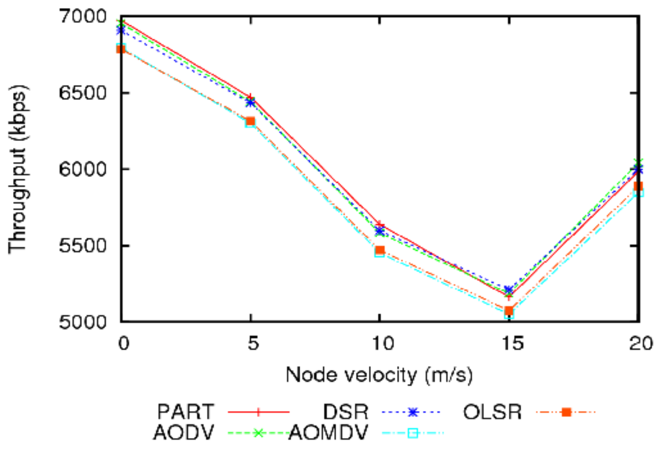


Figure 6.12 : Throughput measurement in RPGM model

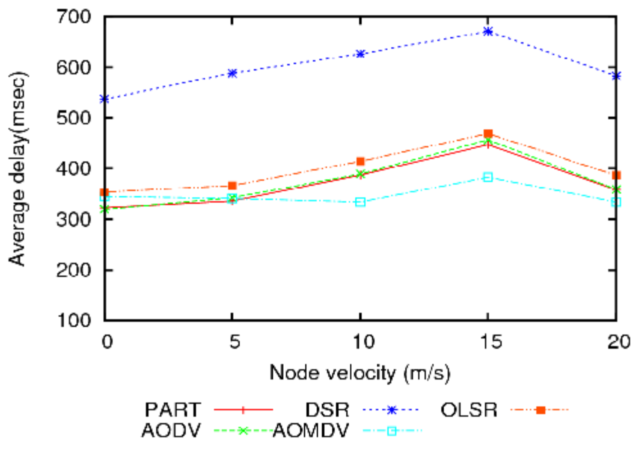


Figure 6.13 : Average delay measurement in RPGM model

Table 6.5 : Performance comparisons of routing protocols in RPGM model

t-Test: Two-Sample Assuming Equal Variances (throughput)

	PART	*AODV*
Mean	6.2E+03	6.0E+03
Variance	5.0E+05	4.8E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	4.9E+05	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	3.3E-01	
T Critical one-tail	3.8E-01	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	7.5E-01	

t-Test: Two-Sample Assuming Equal Variances (throughput)

	PART	*DSR*
Mean	6.2E+03	6.0E+03
Variance	5.0E+05	4.5E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	4.7E+05	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	3.6E-01	
T Critical one-tail	3.6E-01	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	7.3E-01	

t-Test: Two-Sample Assuming Equal Variances (throughput)

	PART	*AOMDV*
Mean	6.2E+03	5.9E+03
Variance	5.0E+05	4.7E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	4.8E+05	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	6.8E-01	
T Critical one-tail	2.6E-01	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	5.2E-01	

t-Test: Two-Sample Assuming Equal Variances (throughput)

	PART	*OLSR*
Mean	6.2E+03	5.9E+03
Variance	5.0E+05	4.5E+05
Observations	5.0E+00	5.0E+00
Pooled Variance	4.8E+05	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	6.4E-01	
T Critical one-tail	2.7E-01	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	5.4E-01	

t-Test: Two-Sample Assuming Equal Variances (delay)

	PART	*AODV*
Mean	3.7E+02	3.7E+02
Variance	2.7E+03	2.8E+03
Observations	5.0E+00	5.0E+00
Pooled Variance	2.8E+03	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	-2.0E-01	
T Critical one-tail	4.2E-01	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	8.4E-01	

t-Test: Two-Sample Assuming Equal Variances (delay)

	PART	*DSR*
Mean	3.7E+02	6.0E+02
Variance	2.7E+03	2.5E+03
Observations	5.0E+00	5.0E+00
Pooled Variance	2.6E+03	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	-7.2E+00	
T Critical one-tail	4.5E-05	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	9.0E-05	

t-Test: Two-Sample Assuming Equal Variances (delay)

	PART	*AOMDV*
Mean	3.7E+02	3.5E+02
Variance	2.7E+03	4.1E+02
Observations	5.0E+00	5.0E+00
Pooled Variance	1.6E+03	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	7.7E-01	
T Critical one-tail	2.3E-01	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	4.7E-01	

t-Test: Two-Sample Assuming Equal Variances (delay)

	PART	*OLSR*
Mean	3.7E+02	4.0E+02
Variance	2.7E+03	2.1E+03
Observations	5.0E+00	5.0E+00
Pooled Variance	2.4E+03	
df	0.0E+00	
t Stat	8.0E+00	
P(T<=t) one-tail	-1.0E+00	
T Critical one-tail	1.7E-01	
P(T<=t) two-tail	1.9E+00	
T Critical Two-tail	3.4E-01	

6.7 Chapter Summary

Our first analysis for the properties of mobility models clearly shows that the node speed variations do affect the link conditions of the mobility models, which in turn, impacts the performance of the routing protocols in different ways. As node speed increases, the link stability of the RPGM is the best due to its group movement pattern. Apart from this model, Manhattan Grid is the best and a more realistic model because it is based on grid road topology. Even though Gauss-Markov eliminates the sudden stop and sharp turn behaviors, it suffers a greater number of partitions and link breaks when the node speed increases. RWP stands as a moderate model among them.

The second analysis shows the performance analysis of routing protocols for each mobility model. In RWP, although PART incurs the lowest delay as the node speed increases, it could not mimic the multipath routing protocols. Its delay is almost 180% lower compared to DSR. PART reduces the routing overhead by almost 30% lower than other protocols. When comparing the packet loss rate, PART incurs almost 20% lower losses than on-demand routing protocols, whereas it suffers higher losses compared to proactive routing protocols. In addition, PART's throughput is almost 35% higher than the others for the RWP model. For the Manhattan model, PART incurs almost 110% lower delay and routing overhead compared to the others. The performance of PART is the best under this realistic model. In the RPGM model, although the throughput comparison of PART is not highly significant compared to the others, PART reduces average delay almost 40% lower than the others.

Chapter 7

CONCLUSION AND FUTURE DIRECTION

7.1 Conclusion

We propose two mechanisms for the routing layer and transport layer enhancements. The first mechanism involves designing and developing a new routing layer protocol, named proxy-assisted routing (PART), for efficient data transmission. The assistance of a proxy node is used, depending on the distance between a source and destination.

The responsibilities of a proxy node are to avoid invoking a new route discovery procedure for route failures that occur due to collisions at the MAC layer and repair the route errors locally. Furthermore, the proxy node defines a broadcasting zone and only nodes within the zone are allowed to forward the control packets to reduce the routing overhead. Without using additional control packets, a proxy node is defined depending on the hop count information by adding new fields in the reply packet. After defining a proxy node, the intermediate nodes between a proxy and source node update their routing tables by adding the proxy node's id. Intermediate nodes are informed about the proxy using a unicast transmission that includes the MAC layer information.

In MANETs, nodes may move in different directions. For example, a node that is assigned as a proxy node at 0.5 seconds may not be able to act as a proxy a few moments later. We use a node as a proxy node during one TCP connection. As the number of TCP connections increases, the number of proxy nodes may increase if the path lengths are longer. We inherit the basic route discovery of AODV because it can adapt to route failures quickly and it works well under high mobility.

To ensure that the objectives listed in Section 1.4 are achieved, we carried out experiments to test the effects of node movements. In Chapter 4, we compare the performance of the proposed proxy-based routing protocols in large-scaled networks for robustness and scalability. The simulation results prove that PART protocol not only reduces average delay and routing overhead, but also increases throughput.

The second mechanism, proposed in Chapter 5, involves a proxy local acknowledgement (PACK) mechanism. The proxy node monitors the sequence numbers of the packets it receives. As soon as the proxy detects a missing sequence number, it attaches the information about the missing packets in the ACK packets it receives from the destination to inform the source node of the missing packets. This mechanism requires the cooperation between the routing layer protocol, PART and MAC layer, and is applied to the transport layer protocols, i.e. Tahoe, New Reno, Vegas and Westwood.

The experiment results in Section 5.4 prove our hypothesis that PACK is able to increase the throughput of TCP packets in both static and mobile networks with different topologies.

Finally, an analytical framework is proposed to compare the performance of routing protocols by incorporating mobility models. The properties of mobility models and their influence on routing protocols are analyzed in Chapter 6. The experiment results in Section 6.6 prove our hypothesis that PART is able to improve the performance with different mobility models, in terms of throughput, delay, packet loss rate and routing overhead.

7.2 Significance of Contribution

The proxy node plays an important role in adapting to route failures with less overhead and delay. It also supports fast route adaptation in a MANET and successfully limits the broadcast zone. Even when the proxy has moved out of range or is unavailable due to power failures, the source node is able to discover a new proxy immediately without extra overhead. The PART protocol is capable of improving the throughput by up to 75% in a large-scaled network and even mimics the multipath routing protocol, AOMDV, with much lower overhead.

The proxy node combined with the PACK mechanism improves performance significantly over various TCP variants. PACK is able to detect missing TCP sequence numbers and inform the source node of missing packets in advance of the end-to-end acknowledgement. As the path length increases, the performance improvement of PACK becomes more significant due to its ability to initiate fast retransmission of lost packets. Because the source node knows the missing TCP sequence number in advance, it results in a significant performance in static and mobile environments. PACK increases throughput up to 70% in chain topology, 60% in grid topology of static network, and 25% in random topology of mobile network. It also performs well when combined with the various TCP variants, showing higher throughput and lower delay in a static ad hoc network.

An analytical framework that incorporates mobility models with single path and multiple paths routing protocols shows that PART performs well, resulting in higher throughput and lower delay in mobile environments.

In a nutshell, all of the stated objectives have been successfully achieved.

7.3 Future Direction

This work focuses on single-path protocols only. An interesting future work to explore is the effectiveness of PART and PACK when applied to multipath protocols. In the proposed PACK mechanism, the proxy node monitors TCP packets to detect missing sequence numbers. The source node is responsible for retransmitting missing packets after it is notified by the proxy. A useful extension to the PACK mechanism would be to allow the proxy to buffer packets and retransmit missing packets.