2014-01-01

An Optimization Of Neighbor Discovery In Mobile Ad Hoc Networks

Felipe Jovel
University of Texas at El Paso, fjovel@miners.utep.edu

AN OPTIMIZATION OF NEIGHBOR DISCOVERY IN MOBILE AD HOC NETWORKS

Felipe Jovel

Department of Computer Science

APPROVED:

Patricia Teller, Ph.D., Chair

Michael McGarry, Ph.D., Co-Chair

Shirley Moore, Ph.D.

Charles Ambler, Ph.D.
Dean of the Graduate School

to my

MOTHER and WIFE

with love

AN OPTIMIZATION OF NEIGHBOR DISCOVERY IN MOBILE AD HOC NETWORKS

by

Felipe Jovel

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Computer Science

THE UNIVERSITY OF TEXAS AT EL PASO

December 2014

Abstract

This thesis aims to improve the performance of Mobile Ad Hoc Networks (MANETs) that use soft-state signaling for neighbor discovery, specifically OLSR. In particular, it uses simulations and previous work based on experiments conducted on a physical test bed to study the behavior of the neighbor-discovery algorithm used by OLSR to identify and explore ways to optimize the neighbor-discovery process. Candidate optimizations include the actual and relative settings of the refresh and expiry timers used by the algorithm.

Previous studies focused on understanding how the settings of the algorithms Hello_Interval (δ) and TC_ Interval refresh timers affect network performance in terms of protocol message overhead and network throughput. In contrast, this thesis investigates how the setting of the Hello_Validity (τ) timer relative to the setting of the Hello_Interval timer affects network performance in terms of overall network packet loss and how the setting of the Hello_Interval timer affects energy consumption vs. packet loss.

Results obtained from our simulations indicate that the relationship between the settings of these timers impact network performance in terms of the percentage of overall packet loss. In particular, we discovered that: (1) For Hello_Validity values smaller than 2δ, as the Hello_Validity approaches the value of the Hello_Interval, the percentage of packet loss increases due to the proximity of these parameters and collisions; and (2) setting the Hello_Validity timer to twice the value of the Hello_Interval timer results in a configuration with little packet loss. We attribute these results to a phenomenon that we call *"neighbor flapping"*, where neighbor information for a node that is within range is repeatedly placed in and then evicted from the neighbor sets of other nodes. In terms of energy consumption, we discovered that as the Hello_Interval timer increases, energy consumption decreases but packet loss significantly increases.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis represents a continuation of a research effort to understand and optimize the performance of a specific type of wireless network called a Mobile Ad-Hoc Network (MANET). Its main goal is to comprehend the behavior of the neighbor discovery mechanism employed in MANETs, and to identify possible optimizations that will improve its performance in terms of both latency and energy consumption. The work described in this thesis extends, through simulation, the findings of previous work [4], which relied solely on physical experimentation. To facilitate the understanding of the research presented in this thesis, the reader is first introduced to computer networks in Sections 1.1 and 1.2. In Section 1.3 wireless networks and MANETs are described. Finally, Section 1.4 presents the contributions of this thesis and describes its organization.

1.1 Computer Networks

A computer network can be described as a set of computers capable of communicating with each other. Today, the Internet is the largest, well-known computer network. Like many large systems, the Internet is intricate in nature. Abstraction, a powerful concept used to simplify the design of complex systems, was applied in architecting the Internet. The principle of abstraction gave life to abstract layer models used to specify how end-to-end communication between computers is accomplished in the Internet. For example, the Transmission Control Protocol Internet Protocol (TCP/IP) became the most widely used model in computer networking. The Internet Engineering Task Force (IETF) developed TCP/IP. It is a five-layer stack model that consists (top to bottom) of the application,

transport, network, data-link, and physical layers.

1.1.1 TCP/IP stack model overview

Each of the TCP/IP layers provides services to and consumes services from the layer directly below it. To provide these services, protocols for each of the five abstraction layers of TCP/IP were designed. From the perspective of a source node, whenever communication to a remote node is necessary, data has to traverse the TCP/IP stack from top to bottom. Along the way, each layer encapsulates data into its corresponding data unit and then hands it to the layer below. The encapsulation process appends layer-specific information used to guide data to its destination. At the destination node, encapsulated data traverses the TCP/IP stack in a bottom-up fashion. Every layer then performs a decapsulation process, and passes the information to the layer above. This behavior is illustrated in in Figure 1.1.

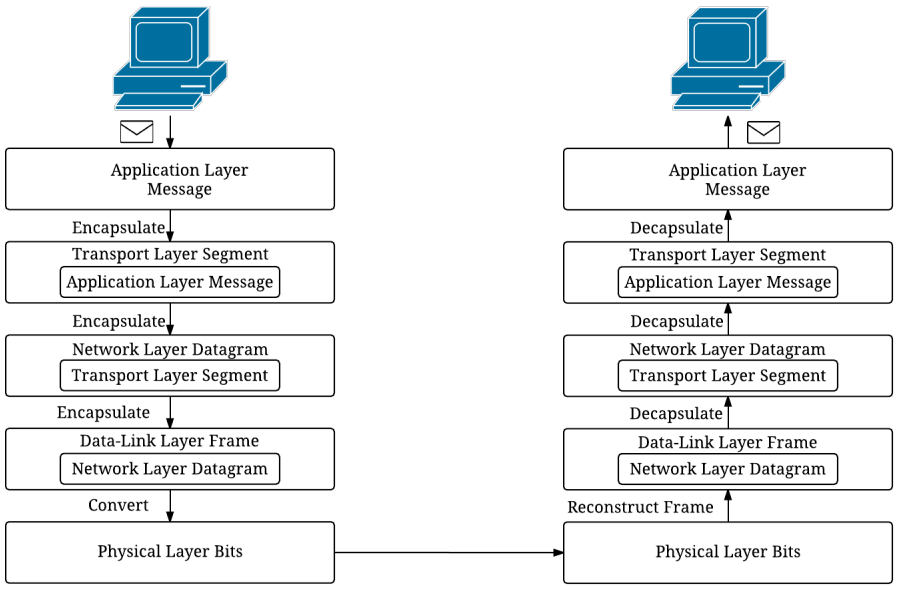


Figure 1.1: An illustration of how data traverses the TCP/IP stack.

At the application layer, protocols specify how applications should exchange messages, which is the data unit at this layer. An example of an application data protocol is the Secure

Shell (SSH) protocol, which enables the establishment of secure connections over insecure networks. In order to transmit messages, the application layer makes use of the process-to-process communication service provided by the transport layer. Depending on the needs of the application employing the transport layer, either TCP or the User Datagram Protocol (UDP) is used to transmit messages encapsulated in segments. In general terms, an application would use TCP when it needs a connection-oriented, reliable, congestion-controlling service. For example, the Post Office Protocol (POP) is an application layer protocol that is used to retrieve email, and because we all want our emails to be displayed exactly as they were sent, it uses TCP's reliable service. On the other hand, an application layer protocol that is intended for video streaming would normally employ UDP since a few frames of the streamed video could be missed without any major repercussions.

Next, to provide process-to-process communication the transport layer uses the network layer, which provides host-to-host communication services. The network layer takes segments and encapsulates them into datagrams. Datagrams are routed through the network from source to destination by means of routing protocols. The work described in this thesis is associated with one of the routing protocols in the network layer. Thus, Section 1.2 is devoted to expanding the description of the network layer.

Continuing down the TCP/IP stack model, the data-link layer receives network layer datagrams and encapsulates them into data-link layer frames. The data-link layer is responsible for transmitting frames over individual links. Note that the communication path from source to destination often involves different link types. For example, some may be wireless links, while others might be copper or optical. Thus, different link-layer protocols are needed to handle different link types along a communication path. Another important function of the data-link layer is that of arbitrating what device gets to transmit over a link. This is necessary when multiple computers have access to the same link. If multiple nodes try to transmit over the same link at the same time, collisions occur. To mitigate this, the data-link layer provides media access control mechanisms that coordinate communication. Finally, once the data-link layer hands a frame to the physical layer, the frame is sent over

the link as bits. The physical layer at the destination reconstructs the frame from the received bits, and delivers it to the data-link layer. This decapsulation process continues up the stack until the original message is received at the application layer.

1.2 Network Layer

As previously stated, the network layer is responsible for delivering packets from source to destination. Specialized devices called packet switches are used to guide packets through the network. A packet switch has both input and output ports. The purpose of a packet switch is to receive packets at an input port, and forward them to the correct output port. In order to determine the output port to which a packet should be forwarded, packet switches maintain forwarding tables that contain a mapping of destination addresses to output ports. When a packet arrives at an input port its header is parsed to determine its intended destination address. The destination address is then used to index into the forwarding table and the packet is forwarded accordingly.

It is worth noting that there are data-link and network layer switches. The forwarding table in a data-link layer switch is called an arp table, while a network layer switch has a routing table. To distinguish between the two, data-link switches are usually called link-layer switches and network switches are called routers.

Clearly, the routing of packets from source to destination is enabled by the routing tables in routers. Thus, an important related topic is routing-table configuration. This can be done in a static or dynamic manner. Static routing-table configuration can be done by having network administrators manually set the entries in routing tables. On the other hand, dynamic routing is made possible by designing routing protocols that routers execute. The function of a routing protocol is to obtain global or partial topology information and, based on this, calculate an optimal route to all network destinations. Most routing protocols fall into one of two broad categories: link-state and distance-vector routing protocols.

1.2.1 Link-state routing algorithms

To calculate an optimal path to all nodes in a network, link-state routing algorithms require as input a complete map of the network. This map provides link-state routing algorithms with a description of all the nodes in the network, the links between nodes, and the costs of traversing those links. Since routing algorithms are nothing more than a computer program that is executed by specialized computers called routers, an abstract data type called a graph is often used to logically represent the map of the network. Formally, a graph G = (V, E) consists of a set V of vertices and a set E of edges. Note each edge (x,y) is assigned a cost `c(x,y)`, which is used in the computation of the cost of a path.

To construct the graph of the network, link-state routing algorithms rely on link-state broadcast algorithms. There are three types of transmissions possible in computer networks: unicast, multicast, and broadcast. Unicast, as the name implies, is the transmission of data to a single destination identified by a unique address. Multicast transmissions target a subset of the network's nodes, while broadcast transmissions are intended for all nodes in a network. To provide the link-state routing algorithm with a complete graph of the network, link-state broadcast algorithms broadcast link-state messages. Every node in the network broadcasts link-state messages to advertise its neighbor nodes to message recipients. This information enables each of the nodes running the link-state routing algorithm to construct the graph of the network. Once the graph of the network is available, the link-state routing algorithm identifies an optimal path to every destination in the network and the routing table is configured according to these results. An example of a link-state routing algorithm is Dijkstra's algorithm, which solves the single source, shortest-path problem for a graph.

1.2.2 Distance-vector routing algorithms

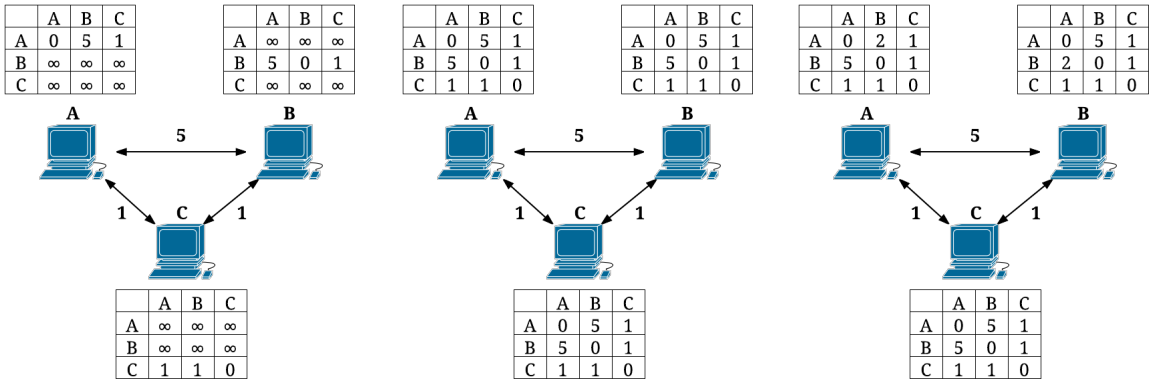
Unlike link-state routing protocols, distance-vector routing protocols do not have access to the global network topology. Instead, optimal paths are iteratively constructed by having each node transmit its distance vector to its directly attached nodes, i.e., its neighbor

nodes. The distance vector of a node consists of the distances (or costs) associated with the current best known paths from this node to all other known destination nodes in the network. For example, assume that a node 'a' has two directly attached neighbors, 'b' and 'c', each with a cost of 2 and 4, respectively. Then, the distance vector of node 'a' is (2,4). A distance vector for a particular node describes the node's current view of the network. In distance-vector routing, every node keeps its own distance vector as well as the distance vectors of its neighbor nodes.

Initially, every node is only aware of its own distance vector and is not aware of the distance vectors of its neighbor nodes. To inform neighbor nodes of its distance vector, every node transmits its distance vector to these nodes. Upon receiving a distance vector from a neighbor node, a recipient node uses it to update its local copy of the distance vector of the corresponding node and its own distance vector. When the receipt of a distance vector from a neighbor node causes a node to update its distance vector, the node transmits its updated distance vector to its neighbors. Eventually when the network becomes stable, nodes stop transmitting distance vectors because their routing tables contain optimal paths to all destinations.

It is easy to see that the distance vectors maintained by the nodes in a network allow them to identify the least-cost (optimal) paths to other nodes in the network. Next, we consider how routing tables are configured. This is accomplished using distance vectors. Whenever a node updates its distance vector, it updates its routing table. For example, consider the network in Figure 1.2a. Initially, node 'a' has a distance vector of (0,5,1), indicating that it can reach itself at a cost of 0, 'b' at a cost of 5, and 'c' at a cost of 1. At this point, the routing table of node 'a' indicates that it can send data to nodes 'b' and 'c' directly, i.e., in one hop. When all nodes send their initial distance vectors to their neighbors, the neighbor nodes store these distance vectors as shown in Figure 1.2b. Next, as shown in Figure 1.2c every node uses the recently acquired information to update its distance vector. For example, using the recently obtained distance vector of node 'c' , node 'a' realizes that 'c' can reach 'b' at a cost of 1. Since node 'a' can reach 'c' at a cost of 1,'a'

updates its distance vector to reflect that it can reach ‘b’ at a cost of 2, i.e., through ‘c’. This information is then used by node ‘a’ to add an entry to its routing table that shows that to send data to ‘b’, it must forward it through node ‘c’.



(a) Initial distance vectors.

(b) Distance vectors after first transmission.

(c) Distance vectors after recalculation.

Figure 1.2: Distance vector configuration.

1.3 Infrastructure vs. Infrastructure-less Wireless Networks

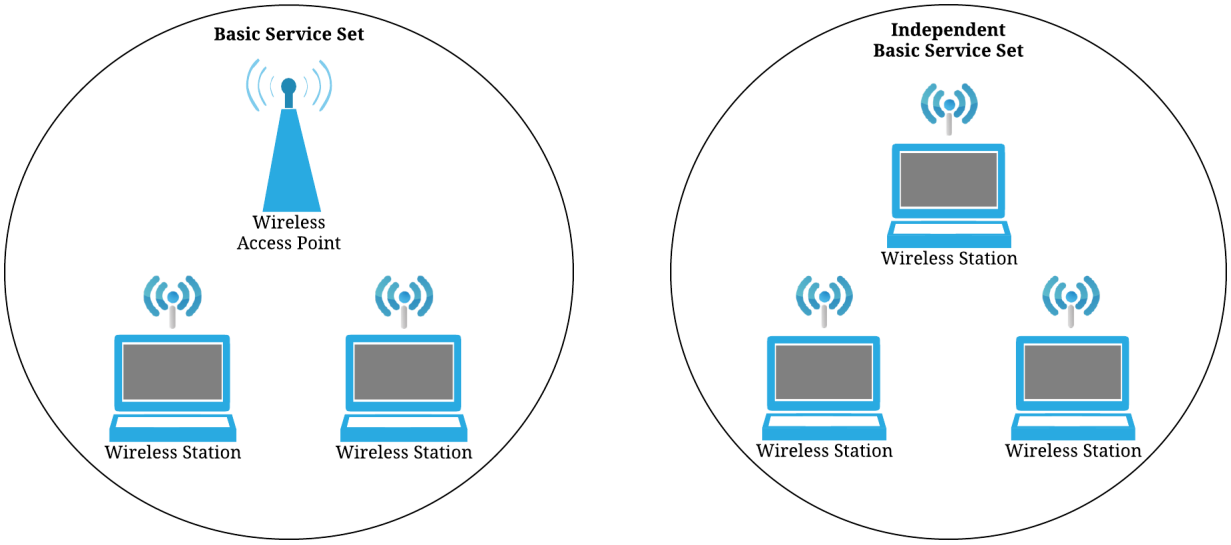
The wireless standard used in this work is the IEEE 802.11 standard, which is commonly referred to as WiFi. The IEEE 802.11 standard defines wireless stations and access points as the main participants in a wireless network. Wireless stations are the actual computers (or hosts) that use the wireless network to communicate. On the other hand, access points are used to establish and manage communication in the wireless network. In order to join a wireless network, a wireless station has to first associate itself with an access point. Once the wireless station does this, it can begin to communicate via the wireless network.

Communication takes place by relaying messages through an access point. Even in the case where two wireless stations, a and b, are within range, messages are first sent to the

access point. If a wants to communicate with b, it sends its message to the access point, and the access point forwards it to b. Since access points forward messages, it is common to have them also function as routers.

The mode of operation just described is referred to as an infrastructure. The fact that an access point is required for the wireless network to exist is what makes this an infrastructure wireless network. The basic building block in this network architecture is a basic service set. A basic service set is composed of an access point and one or more wireless stations.

On the other side of the spectrum there are independent basic service sets. An independent basic service set is composed of only wireless stations and, thus, it is usually referred to as infrastructure-less mode or ad-hoc mode. In an ad-hoc wireless network nodes communicate directly with other nodes within transmission range. A mobile ad-hoc network is defined as an ad-hoc network that is self-configuring. To be self-configuring, a mobile ad-hoc network must have a routing protocol. This work focuses on the Optimized Link State Routing (OLSR) protocol, which is a popular routing protocol for mobile ad-hoc networks (MANETs). Specifically, the neighbor discovery mechanics of the protocol are studied with the purpose of optimizing them.



(a) Infrastructure wireless network.

(b) Infrastructure-less wireless network.

Figure 1.3: Infrastructure vs. infrastructure-less wireless networks

1.4 Thesis Contributions and Organization

The neighbor discovery process is the first step towards establishing a MANET. In the OLSR protocol, neighbor discovery is accomplished through the periodic exchange of Hello_Messages that advertise nodes' links. This periodic transmission of control messages makes OLSR a good candidate for highly mobile networks, since a high frequency of control messages may enable it to quickly discover changes in topology. However, sending control messages too frequently can be wasteful in terms of throughput and energy consumption. Thus, the goals of this research are to:

- Understand how the setting of Hello_Validity timer of OLSR, relative to that of its Hello_Interval timer, affects network performance in terms of overall packet loss, and
- Understand how the setting of OLSR's Hello_Interval timer effects energy consumption versus packet loss.

To this end, a network simulation was developed and validated against a physical platform. With both physical and simulation platforms in place, an extensive parameter sensitivity study was conducted in order to gain insight into the behavior of the neighbor discovery aspects of OLSR. From experimental results, we:

- found that as the Hello_Validity interval decreases, the percentage of packet loss increases;
- setting the Hello_Validity timer to twice that of the Hello_Interval timer results in a configuration with little packet loss; and
- as the Hello_Interval timer increases, energy consumption decreases but packet loss significantly increases.

The organization of the rest of this thesis is as follows. In Chapter 2 we introduce the OLSR protocol, and in Chapter 3 we discuss related work. Chapter 4 describes our experimental methodology; in particular, it describes our simulation platform and our initial step in validation via our experimental physical platform, and provides detailed descriptions of our simulation experiments. In Chapter 5 we present the results of these experiments. Finally, in Chapter 6 we state our conclusions and discuss future work.

Chapter 2

Overview of Optimized Link-State Routing (OLSR)

As previously mentioned in Chapter 1, this work focuses on neighbor discovery in MANETs. To facilitate this research we studied the neighbor discovery process associated with OLSR, a popular MANET routing protocol. OLSR is well suited for large and dense mobile networks because of the performance optimization achieved through the use of multipoint relays (MPRs), which works well in this context [1]. Recall from Section 1.2.1 that link-state routing algorithms use broadcast algorithms to disseminate the network topology to every node in a network. Such algorithms usually have every node broadcast messages that flood the entire network with link-state information. However, in OLSR only a subset of nodes, called MPRs, flood the network with link-state information, effectively reducing link-state message overhead and increasing the scalability of the algorithm. This MPR scheme is precisely why the protocol name includes the word optimized. In this chapter, the reader is mainly introduced to the neighbor discovery process used by OLSR. However, for completeness, we also describe topology discovery and routing table calculation.

2.1 Link Sensing and Neighbor Discovery

To explain link sensing and neighbor discovery, we first define links and neighbors. Nodes in an OLSR MANET have one or more wireless network interfaces, of which one or all participate in OLSR. A link is defined as a pair of wireless interfaces corresponding to two nodes. Due to uncertainties associated with radio propagation, some links are asymmetric

(unidirectional), while others are symmetric (bidirectional). Every node maintains a record of known links and their types (asymmetric or symmetric) in what OLSR calls the node's link set. Likewise, there can exist asymmetric and symmetric neighbors. An asymmetric neighbor of node A is a node to which node A has at least one asymmetric link and no symmetric links. In contrast, a symmetric neighbor of node A is a node to which node A has at least one symmetric link. In addition to maintaining a link set, a node also maintains a neighbor set in order to keep track of its neighbors. Due to the relationship between links and neighbors, the neighbor set is constructed from the link set. Observe that the primary function of link sensing and neighbor discovery is to populate both the link set and the neighbor set.

2.1.1 Populating the link set

Link sensing and neighbor discovery are accomplished through periodic broadcasts of Hello_Messages. This means that Hello_Messages are used to populate the link sets and the neighbor sets of the nodes in a network. However, before discussing Hello_Messages and how they are used to populate link sets and neighbor sets, we introduce OLSRs basic packet layout, which is shown in Figure 2.1. The definition of each of its fields follows:

Packet Length:

Length (in bytes) of the packet.

Packet Sequence Number:

A separate packet sequence number is maintained for each interface that participates in OLSR to cause all packets transmitted over an interface to be enumerated sequentially.

Message Type:

Type of OLSR message encapsulated within the OLSR packet (e.g., Hello, TC, and MID).

Vtime:

Amount of time for which the information contained within the packet are considered valid.

Message Size:

Size of the message in bytes.

Originator Address:

Recall that a node may have one or more interfaces that participate in the OLSR network. The address assigned to one of those interfaces is selected as the originator address, which is used as an identifier for the node.

Time-To-Live:

Number of hops that a packet is allowed to make. Before retransmitting a packet, its Time-To-Live is decremented by one.

Hop Count:

Number of hops that the packet has made so far. Before retransmitting the packet the hop count is incremented by one.

Message Sequence Number:

Packet sequence number associated with each OLSR packet/message. Since an OLSR packet may be received more than once by a node, the Message Sequence Number enables a node to determine if it had previously received the packet and, thus, to avoid reprocessing it.

Message:

OLSR message contained within the OLSR packet. For example, this could be a Hello_Message, a TC_Message, or any other message type supported by OLSR.

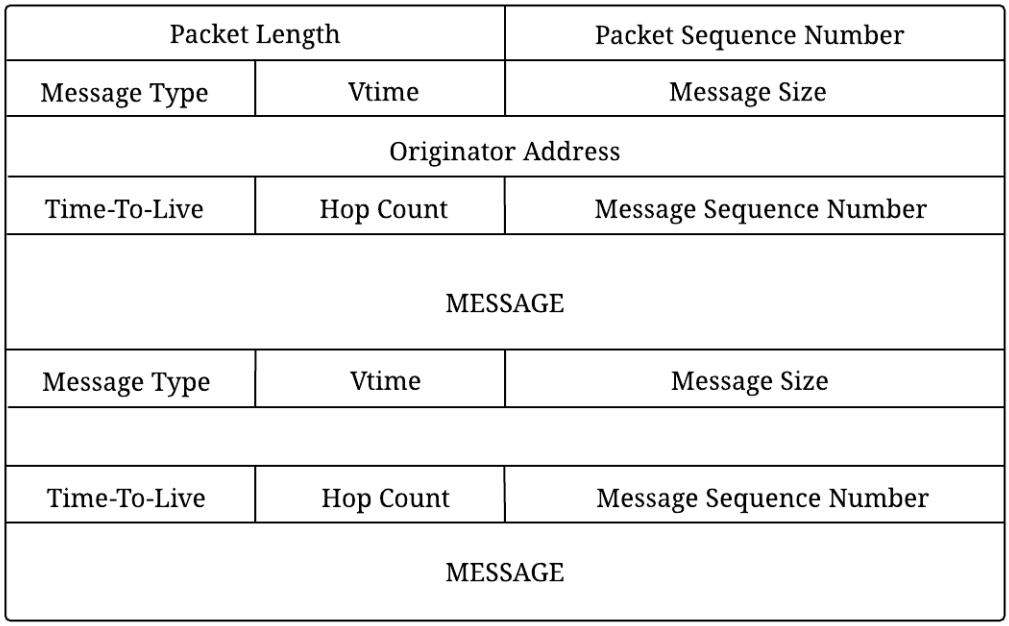


Figure 2.1: The structure of an OLSR packet.

To transmit an OLSR message (e.g., Hello, TC, and MID) a node creates an OLSR packet and initializes each of these fields. Recall that link sensing is accomplished through the periodic exchange of Hello_Messages. By definition, this involves nodes that are within transmission range of each other. Thus, Hello_Messages are intended for nodes that] are within transmission range of the emitters, i.e., that are one hop away. To enforce this, the Time-To-Live field of a packet with a Message Type of Hello_Message is set to one. When a Hello_Message reaches a node, the node decrements the Time-To-Live to zero so that it will not be retransmitted. The Vtime field specifies how long, from the time that the packet is received, the data within it will be considered valid. For example, if a Hello_Message causes an asymmetric link to be established and the time defined by Vtime expires before the link is refreshed by another Hello_Message, the link is removed. Observe that the Message Type field identifies the purpose of the message contained within an OLSR packet. If the type is set to Hello_Message, the node uses it to perform link sensing, neighbor discovery, and

MPR signaling. On the other hand, if the type is set to TC_Message, the node uses it to perform topology discovery. Finally, the Message field contains the message (e.g., Hello, TC, and MID) itself. We now proceed to describe the structure of a Hello_Message, which is shown in Figure 2.2.

Reserved:

Set to all zeros to be in compliance with RFC3626.

Htime:

Frequency at which Hello_Messages are transmitted.

Willingness:

A value that specifies the willingness of a node to become an MPR.

Link Code:

Link type between the interface of the sender and the following list of neighbor interfaces. It also specifies the neighbor type.

Link Message Size:

Size (in bytes) of the message.

Neighbor Interface Address:

Address of an interface of a neighbor node.

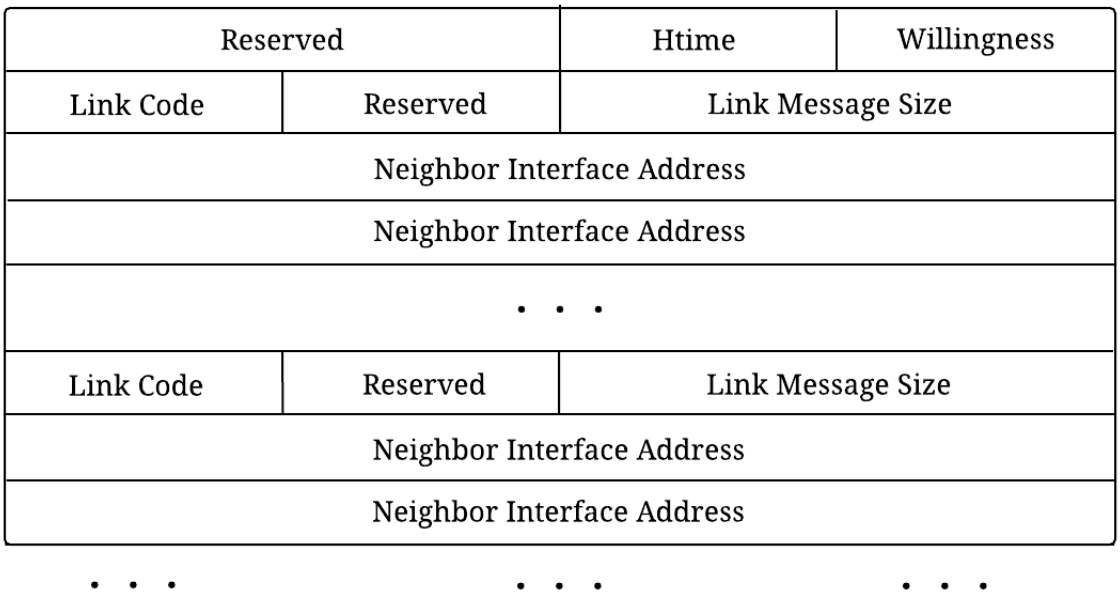


Figure 2.2: The structure of a Hello_Message.

Recall that nodes running link-state routing algorithms broadcast link-state messages to advertise known links to nodes in the network. A recipient of such a message uses it to construct a map of the network. Since Hello_Messages are intended for one-hop neighbors, we can think of Hello_Messages as link-state messages that are used for local topology discovery. OLSR nodes advertise their links through Hello_Messages by including a list of addresses in each message. As shown in Figure 2.2, a Hello_Message may have multiple Neighbor Interface Address fields, which correspond to the interface addresses of one-hop neighbors of the node that is sending the Hello_Message. In addition, a Hello_Message may include multiple such lists, each of which is differentiated by the "Link Code" field, which specifies the link type and neighbor type associated with each of the addresses in a list. For example, links can be of type "asymmetric" or "symmetric". Neighbors can be of type "symmetric", "MPR", or "not-a-neighbor (asymmetric)". The "Link Code" field allows a Hello_Message to contain multiple lists that are differentiated by different "Link Code"

fields. A list can contain MPR neighbors, symmetric neighbors, or asymmetric neighbors.

Having discussed the structure of OLSR packets and Hello_Messages, we now proceed to describe how Hello_Messages are used to populate the link and neighbor sets. Figure 2.3 illustrates the link sensing process, which starts when any two nodes, e.g., nodes A and B, come into proximity of one another. Every node transmits Hello_Messages at a specified frequency (t). The first Hello_Message sent by a node at time t contains an empty list of advertised nodes, meaning that the node has not established any links yet. Upon receiving this first message, each of the nodes adds an asymmetric link entry to its link set, which indicates it can receive messages from the other node. When node A (B) sends its next Hello_Message at time $2t$, it includes a list of all the links in its link set, using the "Link Code" field to identify the type of each link. In this case, the Hello_Message includes the address of node B (A) and a "Link Type" of "asymmetric". Upon receipt of this message, node B (A) observes that the Hello_Message lists its interface addresses as an asymmetric link, which indicates that node A (B) received Hello_Messages that it sent. Accordingly, node B (A) updates the entry for node A (B) in its linkfrom an asymmetric link to a symmetric link, which means that these two nodes can communicate with each other. When node A (B) sends its next Hello_Message at time $3t$, it advertises the newly established symmetric link with node B (A). As long as the nodes keep within transmission range of each other and they continue to deliver Hello_Messages in a timely manner, each node will continue to have a symmetric link to the other.

Next, we describe how a link is lost. Recall that the OLSR packet that encapsulates a Hello_Message has a field "Vtime", which specifies the time during which the information in the Hello_Message remains valid. Also recall that the Hello_Message itself contains the "Htime" field, which specifies the frequency at which Hello_Messages are emitted. From this point on, we refer to the Vtime and Htime fields as Hello_Validity (τ) and Hello_Interval (δ), respectively. Referring to Figure 2.3, which provides an example of the link sensing process, δ is two seconds, while τ is four seconds. If after time $3t$, node B goes out of range of node A, the links previously established between these two nodes will eventually

be removed from their link sets. Since τ is four seconds, each of these nodes expect to receive a Hello_Message from the other node that will refresh the link information within a four-second interval. If they do not receive such a message, the links are lost and the corresponding entries in the link sets are removed.

2.1.2 Populating the neighbor set

At this point, we have seen how Hello_Messages are used to perform link sensing, which, in turn, is used to populate the link sets of the nodes in a network. Next, the neighbor sets of the nodes are derived from their link sets. Note that although a node can have multiple link set entries for a neighbor node, it can have only one neighbor set entry for a local (one-hop) neighbor. When a link entry to a neighbor node is added to a nodes link set, an associated neighbor entry is also inserted into its neighbor set. As with entries in the link set, each entry in a neighbor set is associated with a field that specifies if the corresponding neighbor is symmetric or asymmetric. If a link entry is changed, the neighbor set is updated using the following rule: If the neighbor has a symmetric link entry, the status of the neighbor entry is set to symmetric; otherwise, it is set to asymmetric.

When a Hello_Message is created, the status of the neighbor entry is used to set the neighbor type of the Neighbor Interface Address list to which the neighbor belongs. The neighbor type field is specified within the "Link Code" field and can be set to symmetric, asymmetric , or MPR. The latter type, MPR, is used to inform other nodes that they have been selected as MRPs by the sending node. When a node receives a Hello_Message that includes its address with a neighbor type of MPR, it knows it has been selected as MPR by the node sending the message.

A neighbor entry is removed from a neighbor set when all associated link entries are removed from the link set. The neighbor set is constantly updated. For example, if the type of a neighbor node changes from asymmetric to symmetric or vice versa, the corresponding neighbor entry should be updated to reflect this.

2.1.3 Populating the two-hop neighbor set

If node Ay is a symmetric neighbor of Bx, and Cz is any symmetric node of Ay, Cz is a two-hop neighbor of Bx. A strict two-hop neighbor of B is any two-hop neighbor of B except B and any symmetric neighbor of B. The addition of a two-hop neighbor into a nodes two-hop neighbor set also is triggered by Hello_Messages. When a node receives a Hello_Message, if the sending node is a symmetric or MPR neighbor, for every node listed in the Hello_Message as symmetric, a two-hop entry is added to the two-hop neighbor set if it complies with the definition of a strict two-hop neighbor. If the Hello_Message lists a neighbor of the sending node as asymmetric and there is a corresponding two-hop neighbor entry for that node in the receiving nodes two-hop neighbor set, the entry is removed. Also, if the neighbor through which a node has a two-hop neighbor is lost, the two-hop neighbor entry must be removed. This concludes our description of link sensing and neighbor discovery.

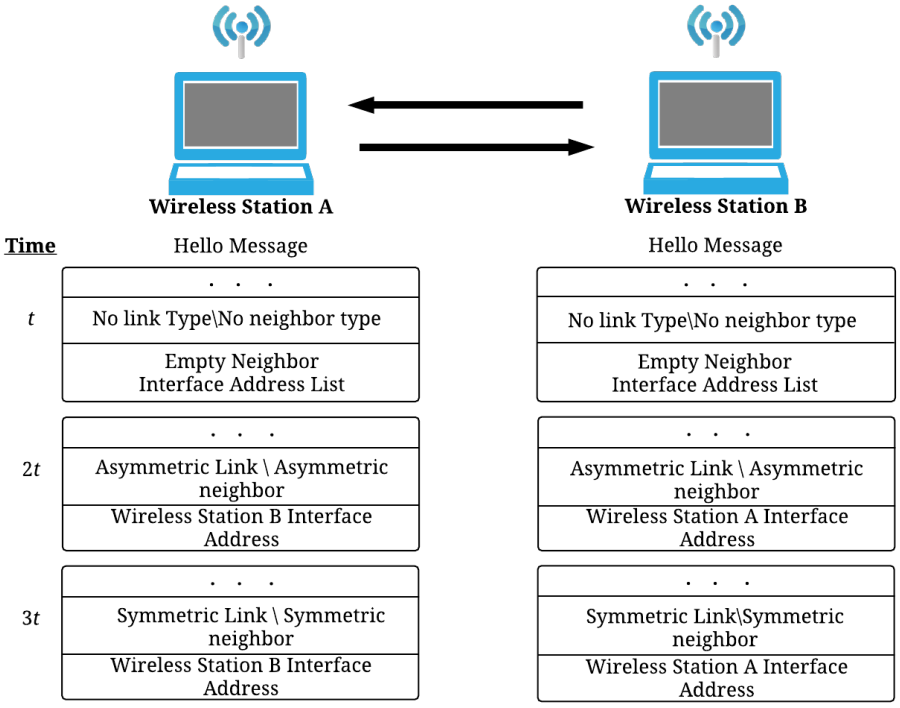


Figure 2.3: Example of the link sensing process.

2.2 Topology Discovery

In Section 2.1 we described how link sensing and neighbor discovery provide nodes with a set of neighbor nodes with which direct communication is possible, i.e., we described the process of local topology discovery. In this section we describe how TC_Messages, broadcasted by MPR nodes, are used for global topology discovery. We have seen that link sensing and neighbor discovery are accomplished through the periodic exchange of Hello_Messages. Also, we briefly mentioned that Hello_Messages provide an MPR signaling mechanism. In this section we also elaborate on MPR signaling and selection.

2.2.1 Populating the MPR set and MPR selector set

Every OLSR node independently selects MPR nodes from its set of symmetric neighbors that are willing to be MPR nodes. The willingness of a node to carry traffic on behalf of other nodes is provided in the willingness field of Hello_messages. The nodes that receive these Hello_Messages keep this information in the sending nodes neighbor entry in their neighbor sets. A node chooses its MPR set so that, through it, every one of its strict two-hop neighbors can be reached. Note that MPRs are chosen by each OLSR interface of a node. Thus, a nodes MPR set is the union of all MPRs chosen by all of its OLSR interfaces.

Once a nodes MPR set is chosen, each node in the MPR set is informed that it has been selected as an MPR. As mentioned in Section 2.1.2, a "Link Code" describes each list of addresses advertised in a Hello_Message. When constructing a Hello_Message, a node uses these lists. In particular, it adds the addresses associated with each entry in its neighbor set that has a neighbor type of MPR and sets the associated "Link Code" to "MPR neighbor". The inclusion of a recipient nodes address in this list informs the recipient that it was selected as an MPR node by the sending node. Accordingly, the recipient node adds the sending node to its MPR selector set. An entry is removed from the MPR selector set of a node if it does not receive another Hello_Message that refreshes the entry within the Hello_Validity time specified in the OLSR packet.

2.2.2 Populating the topology set

As described in Section 1.2.1, link-state routing algorithms require global topology information. Classical link-state routing algorithms obtain this information by flooding the network with link-state messages. In OLSR only a subset of the network nodes broadcast these messages, thus, reducing link-state message overhead. MPR nodes have the responsibility of emitting topology control (TC) messages, which are OLSR link-state messages. The structure of a TC_Message is shown in Figure 2.4. The fields of a TC_Message are

defined below:

ANSN:

A sequence number that identifies an advertised neighbor set. Every time a change occurs in the advertised neighbor set, the ANSN is incremented by one. This helps a receiving node determine if the information within a TC_Message is recent.

Reserved:

Set to all zeroes to comply with RFC3626

Advertised Neighbor Main Address:

Although an OLSR node may have multiple OLSR interfaces, only the address of one is assigned as the main address of a node. This address uniquely identifies the node. Via its Advertised Neighbor Main Address fields, a TC_Message provides a list of advertised neighbor main addresses, which correspond to every node in the advertised neighbor set.

Accordingly, a TC_Message contains a list called the advertised neighbor set. This advertised neighbor set consists of the addresses of the neighbors found in the MPR selector set of the MPR node that sent the TC_Message. TC_Messages are distributed to every node in an OLSR network. To this end, every MPR node broadcasts TC messages, and only MPR nodes retransmit TC_Messages. Nonetheless, non-MPR nodes receive TC_Messages and process them. This means that a TC_Message travels through MPR nodes, from local neighborhood to local neighborhood until every node in the network receives a copy of the message. Recall from Section 2.1.1 that an OLSR packet uses the Time-To-Live field to limit the number of hops a packet travels. Thus, OLSR packets that encapsulate TC_Messages set the Time-To-Live field to the maximum value (255) so that the message can flood the entire network.

Every node, MPR or not, maintains a topology set. An entry in a topology set includes the address of a neighbor advertised by an MPR node, the address of the MPR node that advertised it (not always the MPR that delivered the TC_Message), and the ANSN of the

neighbor set to which the advertised node belongs. When a node receives a TC_Message, it first checks its topology set to determine if it previously received a TC_Message that was broadcast by the advertising MPR. If it did, it compares the ANSN stored in its topology set entry to the one in the TC_Message. If the received ANSN is less than the one stored in the topology set entry, it knows the information in the TC_Message is outdated and discards the TC_Message. Otherwise, it either updates the existing entry or creates a new entry if there is no preexisting entry. Similar to the process associated with the information provided by Hello_Messages, if a TC_Message does not refresh an entry in the topology set before it expires, the entry is removed. In this way, Hello_Messages enable nodes to discover local nodes with which they can communicate directly, while TC_Messages enable them to discover remote nodes with which communication is possible by messages relayed through MPR nodes. Thus, the use of Hello_Messages and TC_Messages provides a global topology configuration to every node.

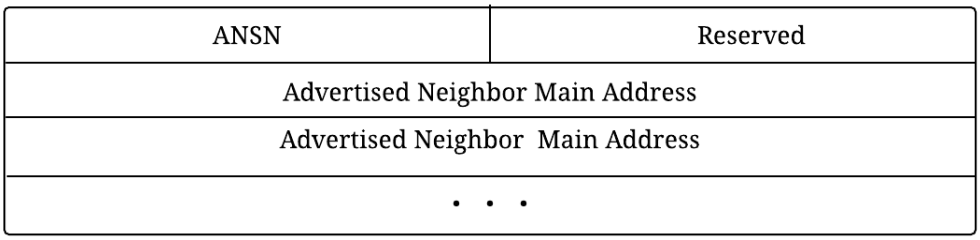


Figure 2.4: Structure of a TC_Message.

2.3 Routing Table Calculation

In OLSR every node maintains a routing table computed from the information obtained through link sensing, neighbor discovery, and topology discovery. Thus, when any of the sets populated through these mechanisms changes, the associated routing tables are recomputed. In other words, the routing tables need to be recomputed any time a change occurs

in any of the following sets: link set, neighbor set, two-hop neighbor set, and topology set. Figure 2.5 displays the structure of an OLSR routing table. The fields in the routing table follow:

dest_addr:

Address of the destination node.

next_addr:

Address of the next node in the route to the destination node.

distance:

Distance (number of hops) to the destination node.

iface_addr:

Local interface through which the node with **next_addr** is reachable.

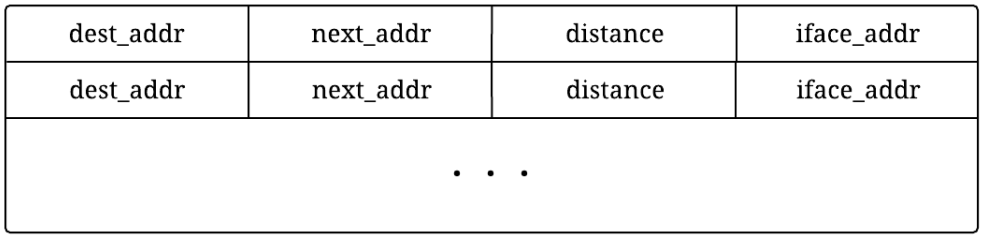


Figure 2.5: Structure of an OLSR routing table.

Next, we give a high level overview of how the routing table is computed. The first step is to delete previous entries if any exist. Then, routes to symmetric one-hop neighbors are added. For every one-hop neighbor, an entry is added with **dest_addr**: interface address of the one-hop neighbor, **next_addr**: **dest_addr**, **distance**: 1, and **iface_addr**: address of the local interface.

Routing entries to two-hop neighbors are added next. For every two-hop neighbor, a routing entry is added such that **dest_addr**: address of the two-hop neighbor, **next_addr**: address of the one-hop neighbor through which the two-hop neighbor is identified, **distance**:

2, and **iface_addr**: address of the local interface.

Finally, we explain how routes to nodes h+1 hops away, where h initially has a value of two, are added to the routing table. For every topology entry in the topology set, if the routing table does not include an entry with dest addr equal to the address of an advertised node stored in a topology set entry and the address of the MPR that advertised the node corresponds to the dest addr of a routing entry with a distance value of h, a new routing entry is added. The fields of the new routing entry are set as follows: **dest_addr**: address ofthe advertised node, **next_addr**: next address of the routing entry for which **dest_addr** is equal to the address of the MPR node that advertised the node, distance: h+1, and iface_addr: iface address of the routing entry for which **dest_addr** is equal to the address of the MPR that advertised the node. This is an iterative process where h is incremented by one at each iteration. If during an iteration of this process no new routing entry is added, the process stops.

Chapter 3

Related Work

An important component of many network protocols is discovery and maintenance of network state. This is accomplished by the exchange of messages between nodes. Network protocols that maintain network state are classified as either hard-state or soft-state protocols. Hard-state protocols create or install a network state that is permanent, i.e., does not change or remains hard, for an unbounded period of time [12]. Once a state is installed in the routing tables of the nodes in the network, no further messages have to be exchanged to maintain that state. However, the occurrence of certain events trigger protocol responses that create and delete the network state in a fully deterministic manner through cooperation among the routers. When state deletion is required, the state installer is required to generate a state removal message. Since state installation and removal is accomplished through individual messages, these messages must be strictly acknowledged. Thus, protocols that take a hard-state approach to state signaling are considered reliable state protocols.

As may be expected, the counterparts to hard-state protocols are soft-state protocols. Unlike hard-state protocols, soft-state protocols rely on the periodic exchange of state messages [12]. Two timers directly control the behavior of soft-state protocols: refresh timers and expiry timers. The refresh timers specify the frequency at which state messages are transmitted. Since state messages are transmitted frequently, there is no delivery guarantee (reliability) requirement. If a state message fails to arrive due to collisions or other environment-related reasons, another state message will follow after a period equal to the refresh timer. For this reason, soft-state protocols are considered best-effort state-signaling protocols. Soft-state protocols have no explicit removal of state. Instead, the expiry timers

are used to determine when to invalidate state information. If state information is not refreshed by a state message before the expiry timer interval expires, the corresponding state information is invalidated. As one might expect based on the discussion in Chapter 2, OLSR is a soft-state protocol. The main refresh timers used in OLSR are the Hello_Interval and TC_Interval timers. Recall that these timers control the frequency at which Hello_Messages and topology control messages are transmitted. Obviously, the main expiry timers are the Hello_Validity and Topology_Validity timers. Other soft-state protocols include RSVP, PIM, and IGMP [12].

Since soft-state protocols such as OLSR are directly affected by the behaviors of their associated timers, studies similar to ours have analyzed the impact of their behaviors on network performance. In this chapter we present some of these studies and indicate how our work differs from theirs.

3.1 Message Overhead and Throughput Study via OLSR/OPNET

In [8] the authors construct an OLSR simulation in OPNET, a discrete event simulator. They experiment with four different topology configurations. Each of the four is composed of two separate OLSR networks that are connected by two routers. Each router has two links, one that participates in its corresponding OLSR network and one that runs the Routing Internet Protocol (RIP) to communicate with the other router. In each experiment a client node at the far left of the first OLSR network sends continuous voice traffic to a server node at the far right of the second OLSR network. The first three topologies are linear topologies that differ in terms of the number of hops (and, thus, the number of MPR nodes) between the client and server nodes and their corresponding routers. The fourth topology changes both OLSR networks so that there is only one MPR node surrounded by other non-MPR nodes, including the client and server nodes; this is called the clutter scenario.

The goal of this work is to understand the impact that the Hello_Interval and TC_Interval parameters have on network performance with respect to protocol message overhead and route convergence time. First, experiments are conducted with a static TC_Interval of 5 seconds and a Hello_Interval that varies, using the values from the set {.5, 1, 2, 4, 6, 8, 10}. Then, a second set of experiments are conducted with a static Hello_Interval of 2 seconds and a TC_Interval that varies, using the values from the set {.5,1,2,4,6,8,10}. The results of the experiments in which the Hello_Interval is varied indicate that, except for the cluster scenario, this parameter has little effect on protocol message overhead. This is most likely due to the fact that the neighbor table entries that must be broadcast in the Hello_Messages by nodes in the clutter scenario are larger as a consequence of the nodes having more neighbors. In terms of route convergence times, the experimental results indicate that as the Hello_Interval increases, so do the route convergence times. This is due to the fact that the first step for topology discovery is neighbor discovery, and this is accomplished through Hello_Messages. The results of the experiments that vary the TC_Interval show that as the TC_Interval increases, the protocol overhead decreases significantly. This is due to the fact that in all cases MPR nodes have more neighbors than non-MPR nodes. In terms of route convergence, it was observed that for linear topologies the TC_Interval does not impact this metric as much as the Hello_Interval.

3.2 Message Overhead and Throughput Study via OLSR/NS-2

A similar simulation study is performed in [9]. In this research, the authors use NS-2 to simulate an OLSR network with various densities and mobility. Networks with 20 nodes are considered low density, while networks with 50 nodes are considered high density. Similar to [8], throughput and routing protocol overhead are the metrics observed. To observe the impact of the Hello_Interval on network performance, experiments are conducted with the TC_Interval fixed at 5 seconds and the Hello_Interval varied using values from the set

{1,2,3}. The results of these experiments indicate that smaller Hello_Intervals, as compared to larger ones, result in higher throughput.It was also shown that as the speed of nodes increases from 0-30 m/s in steps of 5 m/s, the performance improvement (in terms of throughput) of setting smaller Hello_Intervals increases. In terms of message overhead, contrary to what was found in [8], this work shows that: (1) smaller Hello_Intervals increase the protocol overhead and (2) the larger the network density, the larger the protocol overhead. The larger overhead observed in this study is due to the higher node density, which means that nodes have more neighbors. And, as the number of neighbors increase, so does the size of the Hello_Messages. Another set of experiments varies the Hello_Interval from 1 to 10 seconds in increments of .2 seconds. From the results of these experiments, it was observed that as the Hello_Interval increases, throughput decreases in a linear fashion. Compared to the decrease in throughput, the overhead drops faster as the Hello_Interval increases. To analyze the effect of the TC_Interval on performance, experiments were conducted with the Hello_Interval varied using the values in the set {1, 2, 5} and with the TC_Interval increased from 1 to 10 seconds in increments of .2 seconds. The results of these experiments show that throughput remains almost constant as the TC_Interval increases. When the TC_Interval is between one and three seconds the protocol overhead is the largest, significantly decreasing between four and five seconds.

3.3 Message Overhead and Throughput Study via M-OLSR/NS-2

The authors of [11] adopt a similar approach to that taken in [8],[9] to study the impact of the Hello_Interval and TC_Interval on network performance. However, instead of studying OLSR, they work with Modified-OLSR (M-OLSR), an adaptation of OLSR for Wireless Mesh Networks (WMN). A WMN shares most of the properties of a MANET except for the fact that it provides infrastructure-based services. In a WMN there usually is a base station that is connected to a larger network, and nodes have to relay their information

through other wireless nodes in order to reach this larger network [12]. Clients in a WMN usually view the network as an infrastructure-based network. Although M-OLSR added functionality to enable WMN, the soft-state component of the protocol is still present. Thus, it is of interest for the authors to understand how the Hello_Interval and TC_Interval affect network performance. To do this, an NS-2 simulation was developed in which 20 nodes act as router running M-OLSR; these routers form the backbone (or mesh) of the network. Four of these nodes are also connected to a wired infrastructure network and act as gateways. In addition, 14 client nodes view the WMN as an infrastructure-based network that provides them with access to the wired network. The simulations vary traffic loads from 25-60 packets/second, with each packet being 512 bytes. Client nodes move through the mesh at speeds of 1-5 meters/second. In the first set of experiments the TC_Interval is fixed at 5 seconds, while the Hello_Interval is varied using the values in the set {1, 2, 3}. The metrics observed were throughput and protocol overhead. The simulation results show that: (1) as the traffic load (packets/second) increases, a smaller (1 second) Hello_Interval provides higher throughput; and (2) with respect to protocol overhead, as the traffic load increases, there is no noticeable effect on protocol overhead. These are straightforward results because: (1) increasing data traffic does not increase protocol control traffic; and (2) smaller Hello_Intervals cause more Hello_Messages to be sent and, thus, increase protocol overhead. The second set of experiments varies the TC_Interval using values in the set {1, 3, 5, 7, 9}. The results of these experiments show that smaller (1- or 3-seconds) TC intervals have no significant effect on throughput. However, smaller TC intervals have a large (3-4 times larger than the default value) impact on protocol overhead.

3.4 Summary and Comparison

As the related work presented in this chapter indicates, several studies have analyzed the effects that refresh timers have on the performance of soft-state protocols. However, there is no study that we know of that has analyzed the effect that expiry timers have on the performance of such protocols. Thus, part of our work focuses on understanding the rela-

tionship between the refresh timers and expiry timers. Specifically, we investigate how the Hello_Validity timer should be set with respect to the Hello_Interval timer. We also analyze the effect that the Hello_Interval parameter has on network energy consumption. This is done to explore the possibility of increasing the Hello_Interval timer in order to save energy, while maintaining packet loss at an acceptable rate. Another aspect of our work that differentiates it from previous work is the fact that our NS-3 simulation of the OLSR protocol has been closely modeled after the physical platform developed in [4], and experiments on both platforms were design for cross-validation. With respect to network density, our physical test bed makes use of only six nodes, however, our simulation platform is scalable. In regards to mobility, we do not model the gradual movement of nodes or different node speeds. Instead, we perform instantaneous topology changes every two minutes. Future work will explore different network densities as well as different node velocities that will trigger topology changes.

Chapter 4

Experimental Methodology

This chapter describes our approach to understanding how the Hello_Validity and Hello_Interval parameters, which control neighbor discovery in a MANET, affect network performance in terms of packet loss and energy consumption. In Section 4.1 we introduce the NS-3 discrete-event network simulator and justify its use in this research. Next, the NS-3 simulation platform that we built to extend the work described in [4] is described in Section 4.2; and the method used to validate the simulator, using the physical platform employed in [4], is presented in Section 4.3. Finally, in Section 4.4 we explain our experimental setup.

4.1 NS-3

NS-2, the predecessor of NS-3, was first released in 1996. By 2008, NS-2 was undoubtedly the most widely used network simulator [2],[5]. NS-2 was developed using two languages, C++ and oTcl. C++ is used to implement the simulator core and the models included in NS-2, while oTcl is used to setup and control different aspects of simulation experiments. One of the main reasons for employing two languages was to avoid recompilation of the simulator core and models every time the simulation setup needed to be changed. In 1996, when NS-2 was conceived, recompilation was an expensive operation and, thus, justified the two-language approach. However, today this means simulation performance is compromised for the sake of avoidance of recompilation [3]. This resulted in a major rewrite of NS-2, producing a completely new simulator, NS-3.

4.1.1 Performance

Although NS-3 is relatively new (2006) and completely different from NS-2, we believe its fame will precede it. Based on its fame and the results of the study performed by Weingartner, et al. [3], we realized that NS-3 was an attractive option for our work and made a decision to use it to further study neighbor discovery in MANETs. Weingartner, et al. evaluated five network simulators in terms of computation time and memory footprint: NS-2, NS-3, OMNET++, SimPy, and JiST. To do this, the authors implemented the same simulation using each of the five simulators and conducted two sets of experiments. The first set of experiments varied the number of nodes in the simulated network from as few as four to as many as 3,025; and the second varied the probability of packet drops in a network with a fixed node count of 3,025. Two metrics were recorded for both sets of experiments, i.e., execution time and memory usage. With regards to execution time, Jist performed the best for both sets of experiments, with NS-3 coming in a very close (within 25 seconds) second. However, in terms of memory usage, Jist performed the worst for both sets of experiments, while NS-3 performed the best and OMNET++ came in a very close second (within 15 seconds). In general, NS-3 had the best overall performance.

4.1.2 Key abstractions

Now we introduce the key abstractions used in NS-3. A node class object in NS-3 represents a network device that acts as a source/sink of data. Similar to a real network device, a simulated NS-3 network device can be associated with network interfaces, applications, protocol stacks, and routing protocols such as OLSR. An application class object represents a user program that can be installed in a node class object to perform a specific task. Some of the built-in applications in NS-3 are V4Ping, UdpEcho, and UdpClientServer. In case the required program is not readily available, an NS-3 user is expected to create a subclass of the application class and specialize it accordingly. To represent the medium through which data flows in a simulation, NS-3 provides a channel class. A channel class object can

represent anything from an Ethernet wire to radio waves. A node is usually attached to a channel through a network interface device. In NS-3 the class NetDevice is the abstraction to represent a network interface. Built-in specializations of the NetDevice class include the CsmaNetDevice (similar to an Ethernet interface) and the WifiNetDevice (wireless interface). Finally, notice that in a simulation composed of multiple nodes, common actions such as installing and configuring applications and NetDevices, and attaching interfaces to channels, can be tedious tasks if performed manually. Thus, for most models NS-3 provides Helper classes that are used to automate such tasks.

4.2 Simulation Platform

Our simulation platform was developed using NS-3, and it was modeled after the physical platform developed for the research described in [4]. The physical platform is an OLSR network composed of six devices. Each device is equipped with an external wireless transceiver and runs olsrd, an OLSR implementation. The gnu ping application is used to generate traffic. A novel contribution introduced by this platform is the simulation of topology changes afforded through the use of iptable firewalls. To explain this, consider a MANET that consists of three nodes: A, B, and C. In a MANET, if all three nodes are within transmission range, the topology formed by these nodes is a fully-connected graph (a triangle in this case). If a topology changes to one in which there is no link between nodes B and C, one possibility is to have node B move in such a way that node A is still within its range but C is not. Essentially, this does not allow packets from C to arrive at B and vice versa. The physical platform constructed in [4] accomplishes a similar behavior by placing nodes in transmission range of one another and then using iptables to drop link-layer packets that originate from C (B) and are being sent to B (C). This effectively enables the setup of any topology without actually moving nodes. Notice that this means that the simulated movement of devices happens in an abrupt manner, i.e., the gradual movement of a device from position X to position Y is not modeled by this technique but, instead, the change

from one topology to another happens instantly.

Next we describe the NS-3 simulation platform that we developed, which is composed of six NS-3 nodes and the following eight components:

- NS-3 ConstantPositionMobility Model
- NS-3 YansWifiPhy Model
- NS-3 YansWifiChannel Model
- NS-3 AdhocWifiMac Model
- NS-3 OLSR Model
- NS-3 Ping Model
- NS-3 WifiRadioEnergy Model
- Neighbor Set Tracking

4.2.1 NS-3 nodes

As one might expect, the six objects of the node class correspond to the six network devices in the physical platform, and the seven models listed above are installed within these nodes. Given this foundation, a simulation can be setup to use any number of nodes. This feature of our simulation platform, along with its capability to model node mobility, make it attractive for future research, e.g., to study the scalability of our findings with respect to network density and node mobility. A comma-delimited input file that specifies the number of nodes and a sequence of the positions of every node in the network defines a simulation. The first line is a number that specifies the number of nodes. The rest of the lines in the file have the format: time, node number, x coordinate, y coordinate. Each of these lines are parsed by the simulator and used to schedule the movement of a node. For example, the line 120, 5, 1, 2 initiates the scheduling of an event that causes Node 5 to move to coordinate (1,2) in simulation space at the 120th second of simulated time.

4.2.2 NS-3 ConstantPositionMobility model

Recall that the physical platform makes abrupt topology changes. In order to mimic the behavior of the physical platform as closely as possible, we made use of NS-3s ConstantPositionMobility model. This model allows us to perform instantaneous topology transitions. For example, if the location of a node, which currently is at position (2,5), should be changed to position (3,4) at second 120 of simulated time, this will happen exactly at that time – the node will not be located at positions in between (2,5) and (3,4) prior to being located at position (3,4). Although this allows simulations to act like the physical platform, it does not allow us to observe the effects of propagation delay as nodes gradually move further and further apart. Propagation delay is the time that it takes for the first bit of a packet to propagate from one device to another. Again, future research could easily change the ConstantPositionMobility model to one that models gradual movement from one position to another based on the estimated speed at which the device moves.

4.2.3 NS-3 Wi-Fi models

In Section 4.1 the NetDevice and Helper classes were introduced. Our simulation platform makes use of a specialization of the NetDevice called WifiNetDevice and a specialization of the Helper class called WifiHelper. The configuration and installation of the WifiNetDevice objects at each simulation node is done through the WifiHelper class. The install method of the WifiHelper class takes as parameters WifiPhyHelper, WifiMacHelper, and NodeContainer objects. When the WifiHelper class executes the install method, it installs WifiNetDevice objects at each node in the NodeContainer object. Each of the WifiNetDevice objects is configured according to physical (phy) and mac layer model objects. These objects are managed and configured through WifiPhyHelper and WifiMacHelper objects.

In our simulation, the phy model used is the YansWifiPhy model, and the WifiPhyHelper model used is the YansWifiPhyHelper. The YansWifiPhy model is an 802.11 physical layer model. This model relies on the PropagationDelay and PropagationLoss models

that are part of the YansWifiChannel model, which is a specialization of NS-3s Channel class. Both the YansWifiPhy and YansWifiChannel models resulted from the work performed in [5]. As previously mentioned, the propagation delay is the time that it takes for the first bit of a packet to propagate from one device to another. The PropagationDelay model used by our simulation platform is the ConstantSpeedPropagationDelay model, which is defined as a constant delay of two-thirds the speed of light. Propagation loss relates to the decreased signal strength of a wireless signal. This decrease in signal strength can be due to obstructions between the sending and receiving wireless interfaces. However, it can also occur when there is a clear line of sight between sender and receiver due to the signal dispersion experienced as the distance between sender and receiver increases. Thus, one attribute defined by the YansWifiPhy model is the energy detection threshold. In order for the YansWifiphy model to detect a signal, the energy of the arriving signal should be higher than this threshold. Since we are not modeling the gradual movement of nodes, we configured the YansWifiChannel model to use the RangePropagationLoss model as the propagation loss model. The RangePropagationLoss model allows the user to define a threshold a distance (or range) so that any node at a distance less than or equal to the threshold receives wireless signals at full strength, and any node at a distance larger than the threshold does not receive the signal. Devices in the physical platform are roughly one foot apart. Thus, devices in the physical testbed receive signals very close to 100% signal strength. By using the RangePropagationLoss model we mimic very closely the behavior of the physical platform. Notice that using this model also enabled us to know exactly how far apart we needed to place nodes in order to place them out of range. As a result, this mechanism enabled us to easily construct topologies with various characteristics. After the YansWifiChannel model is configured with the PropagationDelay and PropagationLoss models, it is set as the channel for the YansWifiPhy model.

Since we are simulating a MANET, the mac model used is the AdhocWifiMac model. The corresponding WifiMacHelper class used was the NqosWifiMacHelper class. The NqosWifiMacHelper class sets up the AdhocWifiMac as a mac model that does not model

quality of service (QoS). Quality of service allows packets to be classified in such a way that QoS is delivered according to the priorities associated with different classes of traffic. Since the physical platform after which our simulation platform is based does not use QoS, we do not model it in our simulation platform.

4.2.4 NS-3 OLSR and ping models

Recall from Chapter 1 that each node in a network has a protocol stack (TCP/IP stack). So far, we have seen how the physical layer and part of the data-link (mac) layer are configured in our NS-3 simulation platform. NS-3 provides data-link, network, and transport layer modeling through the InternetStackHelper. In essence, a call to the install method of the InternetStackHelper aggregates models to each network node that provide modeling of: Address Resolution Protocol (data-link layer), Internet Protocol (network layer), and TCP/UDP (transport layer). However, before calling the install method of the InternetStackHelper class, the routing protocol to use in the simulation needs to be set. In our case, we used an instance of the OlsrHelper class to configure the InternetStackHelper to use NS-3s OLSR model. The OLSR model in NS-3 was developed at the University of Murcia (Spain) by Francisco J. Ros for NS-2 and was ported to NS-3 by Gustavo Carneiro at INESC Porto (Portugal) [6]. At this point, all that is left is to add an application layer model that serves as source and sink of data. For this, we used a gnu ping model called V4Ping.

4.2.5 NS-3 WifiRadioEnergy model

Since we are also interested in observing the energy consumption of network nodes, we take advantage of the NS-3 WifiRadioEnergy model, which reports the total energy consumption of the node in which it is installed. This model is installed in every node of the network.

4.2.6 Neighbor set tracking

Finally, as will be described in Chapter 5, we needed a mechanism to monitor the neighbor sets of every node. Thus, we added functionality to the OLSR model in NS-3 that allows us to monitor the neighbor set throughout the life of an experiment. In essence, any time a symmetric neighbor is added or removed from the neighbor set, we record the event in a file. This file is then processed to generate plots that allow us to observe the life of each nodes neighbor set throughout the experiment.

4.3 Validation

An important aspect of any simulation model is the level of certainty with which it models reality. To this end, verification and validation of our simulation model is necessary. Simulation model verification intends to ensure that the implementation of the model is correct [7]. With verification in mind, the development team was divided and assigned to different tasks. Part of the team focused on the physical platform (which started earlier than the simulation platform), while the rest focused on the simulation platform. Prior to initiating the simulation model development, a code walkthrough of the physical platform was performed by the team in charge of the simulation model. Then, during the development of the simulation model, the team in charge of the physical platform reciprocally performed code walkthroughs of the simulation model. Simulation model validation involves substantiating that the implementation of the model provides satisfactory accuracy corresponding to the intended application of the model [7]. Thus, it is necessary to define the intended application of our simulation model. As mentioned in Chapter 1, we are interested in understanding how the neighbor discovery process affects the performance of MANETs in terms of energy and packet loss. And, using this knowledge, we will attempt to identify possible optimizations. Thus, at a minimum, we should be able to observe similar trends between the physical and simulation platforms when varying the parameters that control neighbor discovery. In [7] the author describes a series of validation techniques of

which we used two, *Face Validity* and *Sensitivity Analysis*. In *Face Validity* individuals that are knowledgeable about the real system are asked if they believe that the input-ouput relationships are reasonable [7]. In experimenting with the physical platform we became knowledgeable about the system, thus, we were able to apply this technique. In *Sensitivity Analysis* internal parameters of a model are varied and the ouput is observed. The output relationships produced by the model should also be produced by the real system. This technique can be used either quantitatively by only looking at the direction of outputs, or quantitatively by observing both the direction and precise magnitude of outputs [7]. We took the quantitative approach and observed the direction of outputs or trends. The validation performed, which is described below, was done in a subjective manner and is preliminary. Future research will provide an objective validation. Recall from Chapter 2 that the Hello_Validity and Hello_Interval parameters directly control the behavior of neighbor discovery in OLSR. Figure 4.3 shows a series of plots that compare the output of the physical and simulation platforms. The plots show data for experiments in which the Hello_Validity was varied using the set $\{\delta$ 1.1δ, 1.25δ, 1.5δ, 2δ, 2.25δ, 2.5δ, 3δ, 3.25δ, 3.5δ, 4δ, 6δ, $8\delta\}$ seconds, and fixed the Hello_Interval parameter at 2, 4, and 8 seconds. Data shown corresponds to independent Experiment Set 1. The left column of plots shows the average packet loss period, while the right column shows overall packet loss. As can be seen, both the physical and simulation platforms provide us with comparable trends. For similar plots corresponding to independent Experiments Sets 2-4 and experiments in Experiment Sets 1-4 with Hello_Intervals of 16 and 32 seconds, please see Appendices A and B.

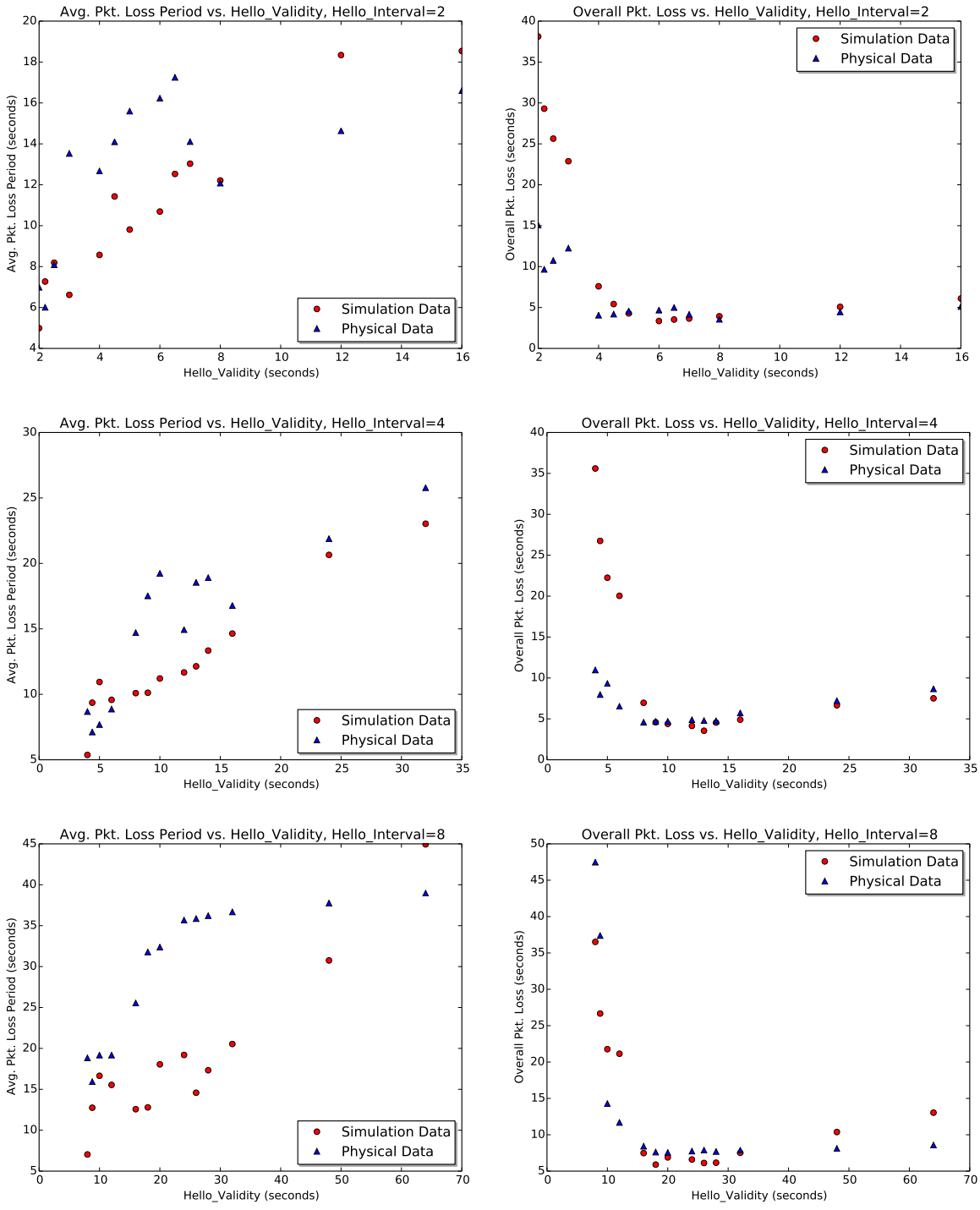


Figure 4.1: Plots comparing simulation and physical platform results from independent Experiment Set 1.

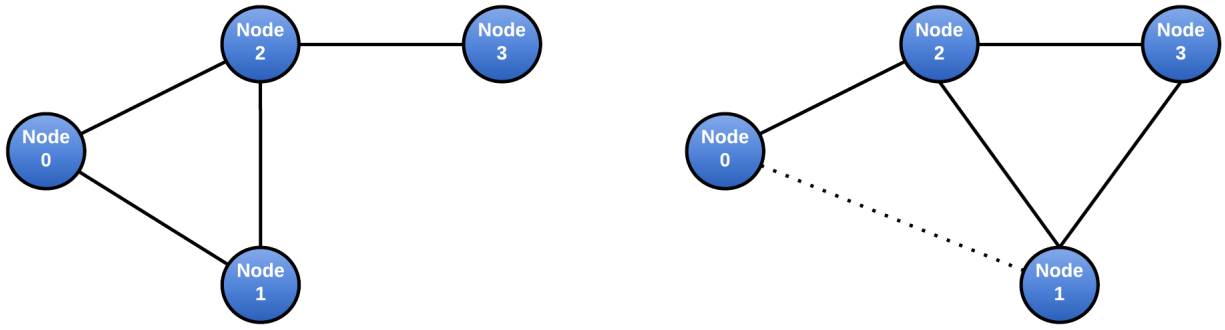
4.4 Experiments

The neighbor discovery process is directly controlled by the Hello_Interval and the Hello_Validity parameters. Thus, to gain a deeper understanding of how these parameters affect neighbor discovery and as a consequence network performance, we performed a parameter sensitivity study. However, before we present the range of parameters used in the experiments it is necessary to provide more details about the experimental setup. The V4Ping model is used as the source of data traffic in the experiments. During an experiment, every node establishes a ping session with every other node in the network. Ping is an application that is used to determine if a remote node is reachable, and if it is, ping reports the round-trip time of the message. To determine the reachability of a node X, a node Y can use ping to send an Internet Control Message Protocol (ICMP) echo request destined for X. If node X receives the ICMP echo request, it responds by sending an ICMP echo reply message to Y. When node Y receives the ICMP echo reply message, connectivity is implied and the round-trip time is reported. Node Y reports round-trip time as the time elapsed between the transmission of an ICMP echo request and the receipt of the corresponding ICMP echo reply. To match an echo request message to its corresponding echo reply message, ping assigns a unique sequence number to each request and the same sequence number to the reply. However, in the context of simulated experiments, the V4Ping model was used only as a source of data traffic.

Throughout an experiment the network topology was changed at a specified frequency. We call this frequency the topology-change interval (TCI). A predefined set of nine topologies was cycled through until 160 topology changes occurred. For each experiment, four independent experiments were performed; each uses a different set of topologies. The topology sets used in our experiments are the same as those used in the physical experiments performed in [4]. The topology set used for independent Experiment number 1 was designed so that each topology moves only one node and that change maintains the connectivity of each node in the network to every other node in the network. The topology sets for Ex-

periment Sets 2-4 were generated randomly with the constraint that all of the nodes in the network are connected at all times. Thus, when a node moves, although one (or more) of its connections to the nodes in the network may disappear, it is not entirely disconnected from the network and it has a way of reaching every other node in the network.

We now proceed to describe the parameter space that was explored in our experiments. To ensure the results observed are not dependent on the TCI used, we performed experiments with TCIs of 120 and 512 seconds. In both instances the Hello_Interval (δ) parameter set was 2, 4, 8, 16, 32. For each Hello_Interval value, multiple experiments were performed by varying the Hello_Validity (τ) parameter from the set $\{1\delta$, 1.1δ, 1.25δ, 1.5δ, 2δ, 2.25δ, 2.5δ, 3δ, 3.25δ, 3.5δ, 4δ, 6δ, 8δ, 16δ, 32δ, 64δ, $128\delta\}$. Note that if τ is greater than TCI, it is guaranteed that nodes will continue to observe the previous local neighborhood for some time after the topology change is made. During this time new links and neighbors will be discovered as well. Figure 4.2 illustrates this assuming $\tau = 130$ seconds and TCI = 120 seconds. Figure 4.2a displays the initial topology after all links and neighbors have been identified. Figure 4.2b shows the topology change made after 120 seconds of simulation time. As can be seen, this change moves Node 1 out of Node 0s transmission range and into the transmission range of Node 3. However, because τ = TCI + 10 seconds, both Node 0 and Node 1 continue to believe they are within the transmission range of each other (illustrated by the dashed line between them) for 10 seconds, i.e., until τ expires. This in turn leads to a delay in recalculating the routing tables of Node 0 and Node 1. And, thus, instead of sending data to Node 2 and having it relay it to Node 0, Node 1 incorrectly attempts to send data directly to Node 1 several times during this period. Accordingly, in order to reduce this effect, all of our experiments use τ values less than TCI. For example, when performing experiments with TCI = 120 seconds, the largest τ value for a δ value of 2 seconds is 32δ. However, the largest τ value that we can use for a δ of 4 seconds that we can use is 16δ. Figure 4.3 shows the configuration of the 116 experiments that were conducted.



(a) Before topology change.

(b) After topology change.

Figure 4.2: Effect of having a τ greater than TCI.

(a) Experiments TCI = 120 Seconds.

τ	$\delta = 2$	$\delta = 4$	$\delta = 8$	$\delta = 16$	$\delta = 32$
(1.0δ)	✓	✓	✓	✓	✓
(1.1δ)	✓	✓	✓	✓	✓
(1.25δ)	✓	✓	✓	✓	✓
(1.5δ)	✓	✓	✓	✓	✓
(2.0δ)	✓	✓	✓	✓	✓
(2.25δ)	✓	✓	✓	✓	✓
(2.5δ)	✓	✓	✓	✓	✓
(3.0δ)	✓	✓	✓	✓	✓
(3.25δ)	✓	✓	✓	✓	✓
(3.5δ)	✓	✓	✓	✓	✓
(4.0δ)	✓	✓	✓	✓	X
(6.0δ)	✓	✓	✓	✓	X
(8.0δ)	✓	✓	✓	X	X
(16.0δ)	✓	✓	X	X	X
(32.0δ)	✓	X	X	X	X

(b) Experiments TCI = 512 Seconds.

τ	$\delta = 2$	$\delta = 4$	$\delta = 8$	$\delta = 16$	$\delta = 32$
(1.0δ)	✓	✓	✓	✓	✓
(1.1δ)	✓	✓	✓	✓	✓
(1.25δ)	✓	✓	✓	✓	✓
(1.5δ)	✓	✓	✓	✓	✓
(2.0δ)	✓	✓	✓	✓	✓
(2.25δ)	✓	✓	✓	✓	✓
(2.5δ)	✓	✓	✓	✓	✓
(3.0δ)	✓	✓	✓	✓	✓
(3.25δ)	✓	✓	✓	✓	✓
(3.5δ)	✓	✓	✓	✓	✓
(4.0δ)	✓	✓	✓	✓	✓
(6.0δ)	✓	✓	✓	✓	✓
(8.0δ)	✓	✓	✓	✓	✓
(16.0δ)	✓	✓	✓	✓	X
(32.0δ)	✓	✓	✓	X	X
(64.0δ)	✓	✓	X	X	X
(128.0δ)	✓	X	X	X	X

Figure 4.3: Experiments conducted.

Chapter 5

Results

This chapter presents the findings of our parameter sensitivity study, which was performed as described in Chapter 4. The results of our experiments quantify the relationship between the Hello_Interval, δ, and the Hello_Validity, τ, parameters of a MANET neighbor discovery protocol, and the effect they have on MANET performance, in particular, in terms of packet loss.

Our results indicate that τ can be fine-tuned to increase network performance in terms of packet loss. In Sections 5.1 and 5.2 we show how the size of τ, with respect to δ, affects the average packet loss period and the overall packet loss rate, respectively. Then, in Section 5.3 we explain why the performance of a MANET varies depending on the value chosen for this parameter. Here, we identify and explain a phenomenon that we call *"neighbor flapping"*, which explains why the neighbor discovery parameter configuration affects network performance. Finally, in Section 5.4 ewe present preliminary results that show the trade-off between energy consumption and network performance in terms of overall packet loss.

5.1 Average Packet Loss Period

Recall from Section 4.4 that a model of the ping application was used as a source of data traffic. Also remember that every pair of Internet Control Message Protocol (ICMP) echo request and matching echo reply messages is assigned the same sequence number. Whenever an ICMP echo reply is not received for a corresponding echo request, packet loss occurs. When packet loss occurs, the packet loss period is measured from the time of the receipt

of the last successfully received ICMP echo reply to the time at which the next ICMP echo reply is received minus its corresponding round-trip time. The average packet loss period is computed by dividing the cumulative sum of all packet loss periods over the number of packet loss periods that occurred during a simulation. Recall, also from Section 4.4, that during an experiment, the topology of the network is changed every topology change interval (TCI). In an ideal MANET, packet loss occurs only when the topology is changed. In this case, the average packet loss period gives an indication of the time that it takes for the MANET to recompute routing tables after a topology change is occurs. However, as we will see in Section 5.3, if δ and τ are not carefully chosen, packet loss can occur in-between topology changes.

Our main observation with respect to average packet loss period is that as τ increases in size, the average packet loss period increases. Remember that once a Hello_Message is received, τ specifies the amount of time that the information provided by the Hello_Message is considered valid; thus, τ is never smaller than δ. This means that the larger τ, the longer it will take for nodes to realize that neighbors have gone out of range. As a consequence, the length of time during which a node may continue to send data to a node that is out of range increases with τ. Accordingly, as τ increases, i.e., the longer this behavior lasts, so do the packet loss periods. Figure 5.1 illustrates this behavior for both the simulation and physical platforms. The three plots in each row show, for TCI = 120 seconds, δ(Hello_Interval) $\in$ {2, 4, 8} seconds, and τ (Hello_Validity) $\in$ {1δ, 1.1δ, 1.25δ, 1.5δ, 2δ, 2.25δ, 2.5δ, 3δ, 3.25δ, 3.5δ, 4δ, 6δ, 8δ} seconds, the average packet loss period experienced during all four sets of independent experiments. Recall that four independent experiments are run for each parameter configuration (i.e., a pair [δ,τ]) and that each independent experiment uses a different set of nine topologies. During an experiment the topology is changed every TCI by cycling from topology1 to topology9, and back to topology1 repeatedly until 160 topology changes occur. In addition, during an experiment each of the six nodes establishes a ping session that sends data every second to every other node. The left column of the plots is based on data from simulations, while the right column is based on data collected from

experiments conducted on the physical platform. Based on this information one might assume that setting τ as close as possible to δ would decrease the average packet loss period and, thus, provide best performance. However, as is shown in Section 5.2, this is not the case.

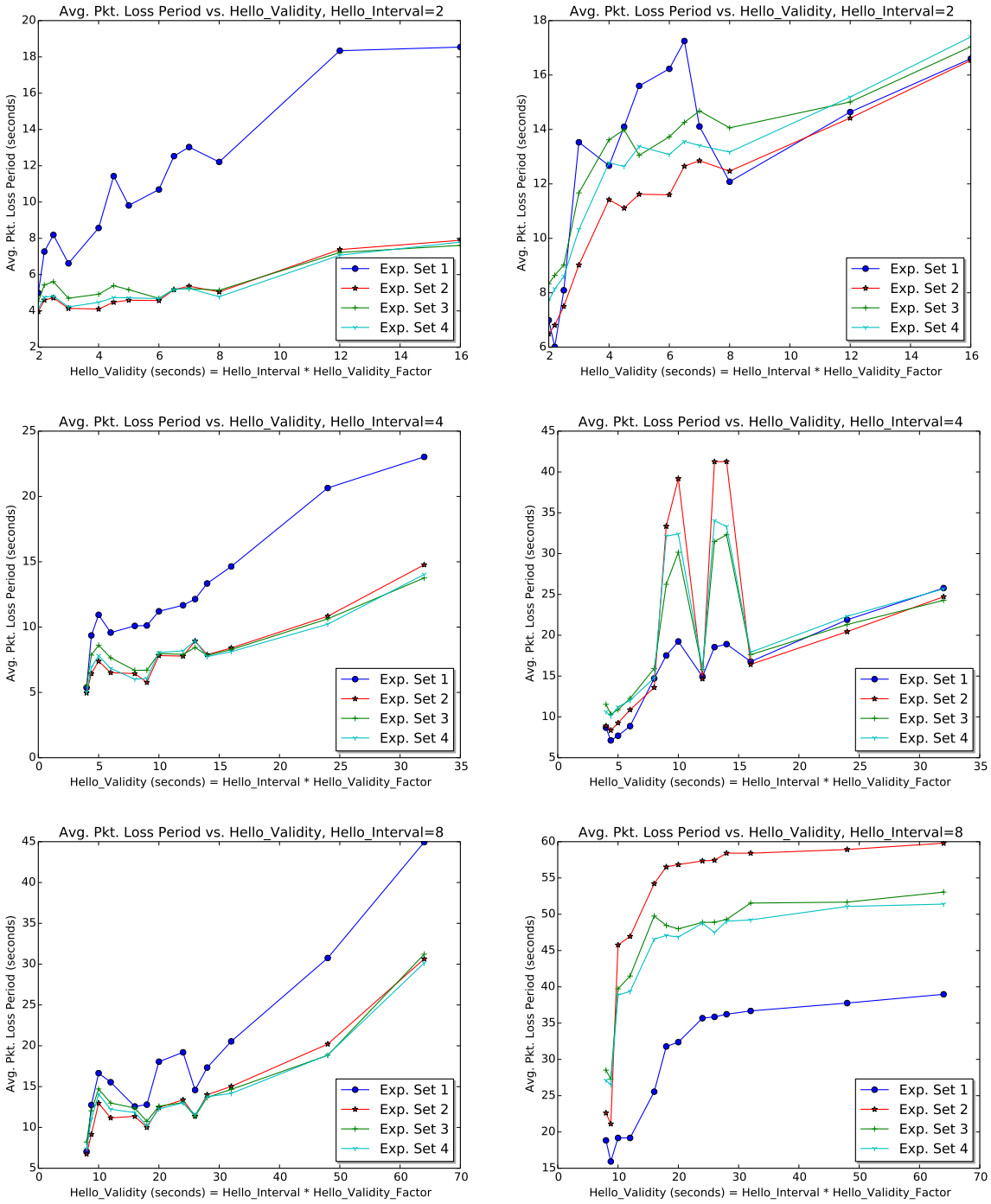


Figure 5.1: Average packet loss period increases as the Hello_Validity parameter(δ) becomes larger than the Hello_Interval parameter. The left column of plots corresponds to simulation data, while the right column corresponds to data generated by experiments conducted on the physical platform. The set from which the Hello_Validity parameter is taken is:$\{1\delta$, 1.1δ, 1.25δ, 1.5δ, 2δ, 2.25δ, 2.5δ, 3δ, 3.25δ, 3.5δ, 4δ, 6δ, $8\delta\}$.

5.2 Overall Packet Loss Percentage

Figure 5.2 illustrates how the values of the τ and δ parameters affect MANET performance in terms of packet loss percentage, i.e., the percentage of the total number of packets sent by all nodes in the network during an experiment that did not arrive at their destinations. The experiments used to generate the data depicted in Figure 5.2 are the same experiments described previously in Section 5.2. Based on this experimental data, we observe that, out of all the parameter configurations experimented with, when $\tau = \delta$, the packet loss percentage is the largest. Notice the shape of the plots in Figure 5.2, which suggest an optimum parameter configuration in terms of packet loss percentage. For the most part, these experiments resulted in packet loss percentages that are at their minimum when $\tau = 2\delta$. To observe the benefit of setting τ in this way, the overall packet loss percentage for experiments in which $\tau = \delta$ (worst-case performance), where $\delta \in \{2, 4, 8, 16, 32\}$, were averaged over all four independent experiments and used to calculate the change in the percentage of packets lost during an experiment with $\tau = \delta$ and the percentage lost in related experiments with $\tau \in \{1.1\delta, 1.25\delta, 1.5\delta, 2\delta\}$. These data are shown in Tables 5.1 and 5.2 for simulation and physical experiments, respectively. Simulation platform data show that by setting $\tau = 2\delta$, the overall packet loss percentage is decreased anywhere from 10.85% to 86.11%, and the percentage decreases monotonically as the value of δ increases. In the case of the physical platform, the data shows a 0.07% to 48.85% reduction in overall packet loss. Similar to the data generated by simulations, the largest reduction is at $\delta = 2$ and the smallest reductions are at $\delta = 16$ and $\delta = 32$, but the reduction does not decrease monotonically as the value of δ increases. Although the overall packet loss percentage reaches its absolute minimum somewhere in between 2δ and 4δ, the overall packet loss percentages yielded by these configurations are only marginally better than that provided by 2δ – they differ by at most 2% (this data is provided in Appendix B).

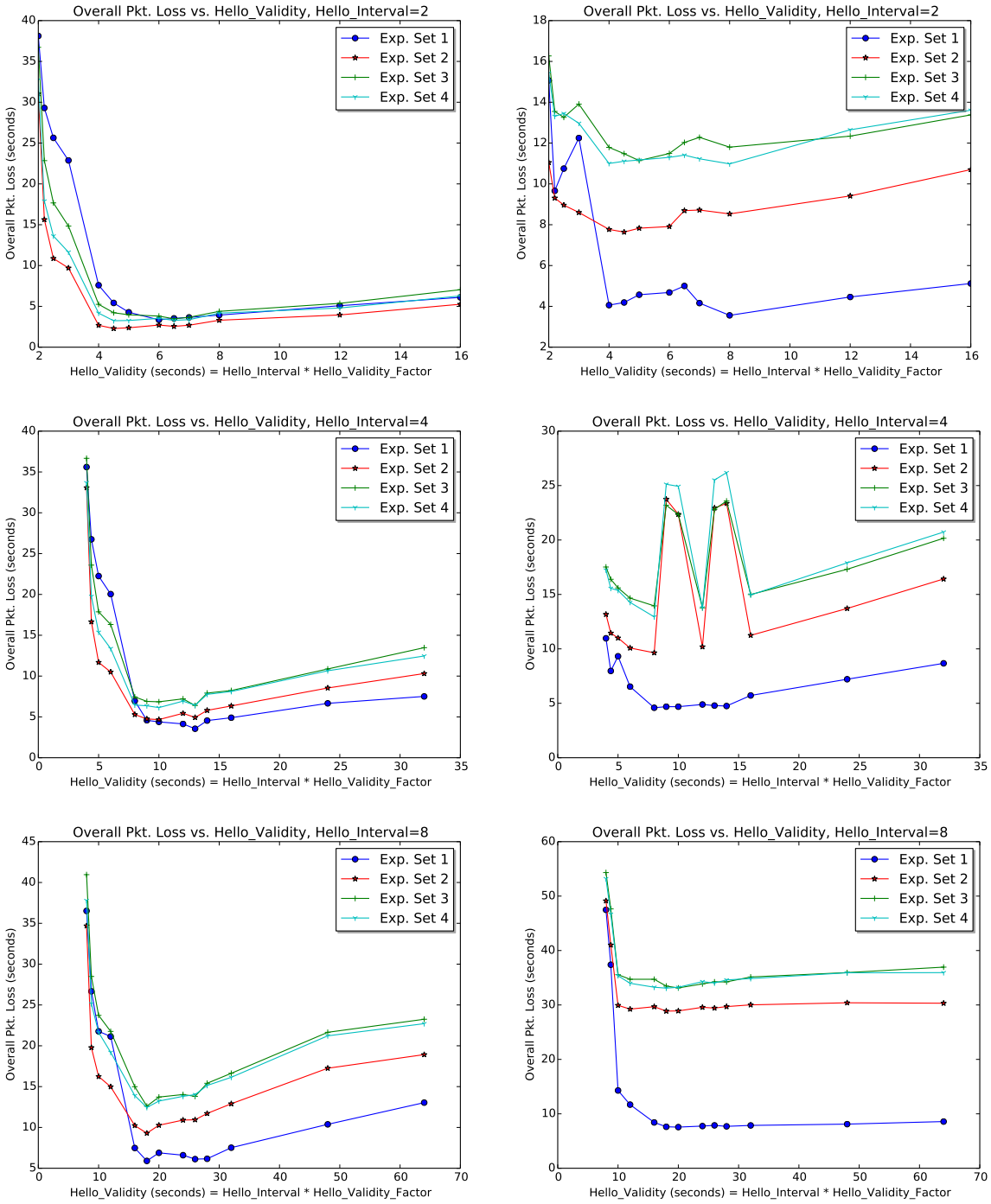


Figure 5.2: MANET performance in terms of overall packet loss percentage as Hello_Validity increases. The left column of plots corresponds to simulation data, while the right column corresponds to data from experiments conducted on the physical platform. The set from which Hello_Validity is taken is: $\{1\delta$, 1.1δ, 1.25δ, 1.5δ, 2δ, 2.25δ, 2.5δ, 3δ, 3.25δ, 3.5δ, 4δ, 6δ, $8\delta\}$

Table 5.1: Percent change using 1δ as a reference point (simulation experiments).

τ (sec)	δ=2	δ=4	δ=8	δ=16	δ=32
(1.1δ)	-38.97	-37.88	-33.46	-20.47	-10.85
(1.25δ)	-52.00	-51.95	-44.59	-31.67	-18.00
(1.5δ)	-58.24	-56.94	-48.75	-36.46	-19.99
(2δ)	-86.11	-81.25	-69.85	-47.86	-28.87

Table 5.2: Percent change using 1δ as a reference point (physical experiments).

τ (sec)	δ=2	δ=4	δ=8	δ=16	δ=32
(1.1δ)	-20.53	-14.15	-15.67	-0.07	-0.29
(1.25δ)	-19.73	-13.33	-44.25	-5.02	-1.48
(1.5δ)	-17.83	-24.37	-47.02	-4.58	-3.27
(2δ)	-39.75	-32.57	-48.85	-4.61	-6.94

5.3 Neighbor Flapping

Next, we explain why the overall packet loss percentage is largest when $\tau = \delta$. It was observed that when τ is close to δ, Hello_Messages fail to refresh neighbor information before the time for validation (τ) the time between Hello_Messages – expires. Such a situation causes nodes to remove neighbor nodes from their neighbor sets and, thus, consider them out of range when in fact they are not. As a consequence, stale (incorrect) routing information is stored in routing tables for a period of time, leading to packet loss during this period. Since, in this situation, neighbor nodes are placed in and evicted from neighbor sets, we call this behavior *neighbor flapping*.

To verify why this behavior occurs, we conducted experiments in which no topology changes are made and during these experiments we monitored the neighbor set of each node

and recorded any changes. Since the topology is fixed in these experiments, any changes to the neighbor sets are due to *neighbor flapping*. Data traffic is generated as previously discussed. Figure 5.3 illustrates the topology that was used in these experiments, in which δ(Hello_Interval) $\in$ {2, 4,8} seconds and τ (Hello_Validity) $\in$ {1δ, 1.1δ, 1.25δ, 1.5δ, 2δ, 3δ, 4δ, 6δ} seconds.

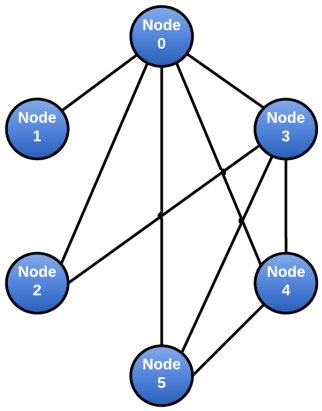


Figure 5.3: Fixed topology used in experiments that record the lives of neighbor sets.

The plots in Figure 5.4 display the number of state changes (per neighbor node) in the neighbor set of Node 0 as τ increases. Note that Node 0s neighbor set should contain a symmetric neighbor entry for Nodes 1-5. Recall from Section 2.3 that when computing a nodes routing table, symmetric neighbor entries in the nodes neighbor set are used to add routes to direct (one-hop) neighbors. A state change is recorded every time a symmetric node is added or removed from a neighbor set. Also, note that since there are no topology changes, one would think that neighbor sets would remain static. However, as can be seen from the experimental data collected using both the simulation and physical platforms, this is not the case.

Referring to the plots in Figure 5.4, observe that analogous to the behavior of overall packet loss percentage, the highest number of state changes occurs when $\tau = \delta$ and it is significantly less when $\tau = 2\delta$. Tables 5.3 and 5.4 show the percent change from the number of state changes obtained when $\tau = \delta$ to the number obtained when $\tau = 2\delta$ for simulation

and physical platforms, respectively. These tables show that for both the simulation and physical platforms, there is a reduction in the number of state changes of at least 86%. This behavior indicates that Hello_Messages are not refreshing neighbor information before the Hello_Validity (τ) interval expires. As a result, symmetric neighbors are evicted from neighbor sets and associated packet losses occur due to misrouting. Figure 5.5 illustrates such a scenario. In Figure 5.5a Node 1 sends data to Node 3 through Node 0. However, in Figure 5.5b *neighbor flapping* occurs and Node 1 evicts Node 0 from its symmetric neighbor set. After a small amount of time, Node 2 also evicts Node 4 from its symmetric neighbor set due to *neighbor flapping*. However, by the time this happens, Node 1 has recomputed its routing table and has added a route to Node 3 through Node 2. When Node 1 forwards data destined to Node 3 to Node 2, Node 2 is unable to forward it to Node 4 for delivery since it's routing table indicates that it is out of range, even though it is not. This results in packet loss at Node 1.

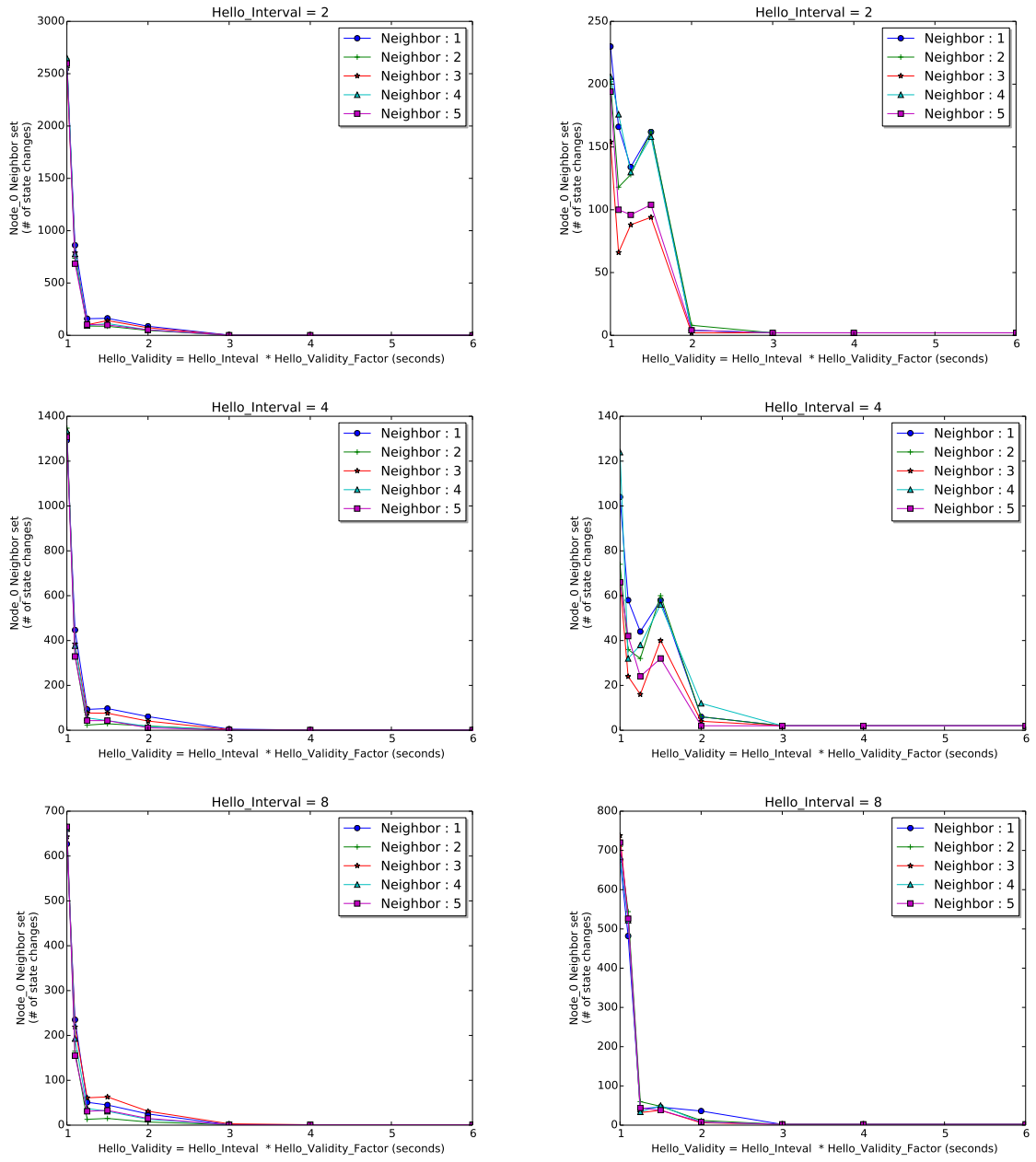


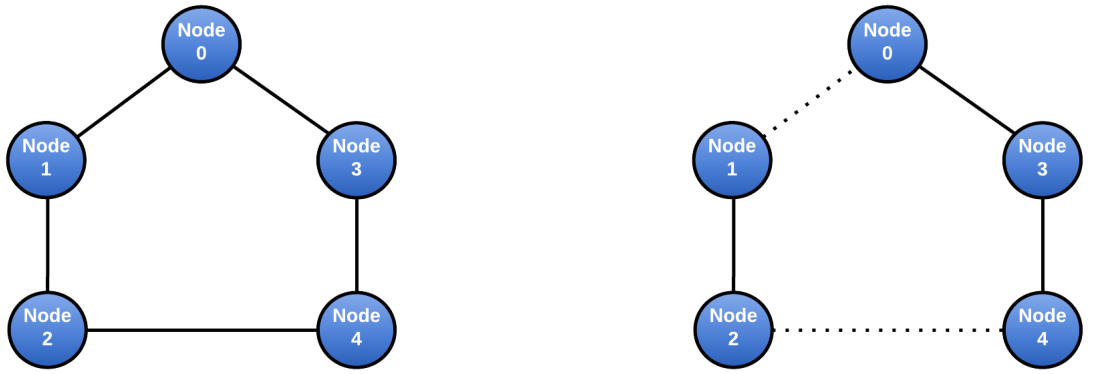
Figure 5.4: Plots showing the number of neighbor set state changes due to *neighbor flapping*. The left column of plots corresponds to simulation data, while the right column corresponds to data generated by experiments conducted on the physical platform. The set from which Hello_Validity is taken is: $\{1\delta, 1.1\delta, 1.25\delta, 1.5\delta, 2\delta, 3\delta, 4\delta, 6\delta\}$.

Table 5.3: Percent change in number of state changes from $\tau = \delta$ to $\tau = 2\delta$, Node 0's neighbor set (simulation experiments)

δ (sec)	Node 1	Node 2	Node 3	Node 4	Node 5
2	-97	-98	-97	-98	-98
4	-86	-94	-89	-95	-97
8	-89	-96	-86	-93	-90

Table 5.4: Percent change in number of state changes from $\tau = \delta$ to $\tau = 2\delta$, Node 0's neighbor set (physical experiments)

δ (sec)	Node 1	Node 2	Node 3	Node 4	Node 5
2	-98	-96	-99	-98	-98
4	-94	-92	-94	-90	-97
8	-95	-98	-99	-99	-99



(a) Before *neighbor flapping* occurs.

(b) After *neighbor flapping* occurs.

Figure 5.5: Possible misrouting scenario due to *neighbor flapping*.

To further understand the effect of *neighbor flapping*, we examine the percentage of time that nodes that should be considered symmetric neighbors do not have symmetric-neighbor

entries in Node 0s neighbor set. Note that this is done using the same experiments that were described above, i.e, those that were used to measure the number of state changes. As expected, the data from both the simulations and physical platform experiments, which are shown in the plots of Figure 5.6, show that when $\tau = \delta$ the percentage of time during which this scenario exists is the largest. The data from both types of experiments also show that when $\tau = 2\delta$, this percentage of time decreases significantly. In the case of the simulations, this percentage decreases monotonically as τ increases; but this is not the case for the data generated by the physical experiments. Tables 5.5 and 5.6 show the change in the percentage of time that nodes that should be in Node 0's symmetric neighbor set are not. This is calculated from the number of nodes in Node 0'symmetric neighbor set when $\tau = \delta$ to the number when $\tau = 2\delta$ for simulation and physical platforms, respectively. It was observed that there is at least an 89% decrease whether we examine the data associated with simulations or experiments generated on the physical platform. These experimental data clearly indicate that even when there are no topology changes taking place during an experiment, valid entries are evicted from neighbor sets when they should remain resident, which could be the primary reason why the overall packet loss is higher when $\tau = \delta$.

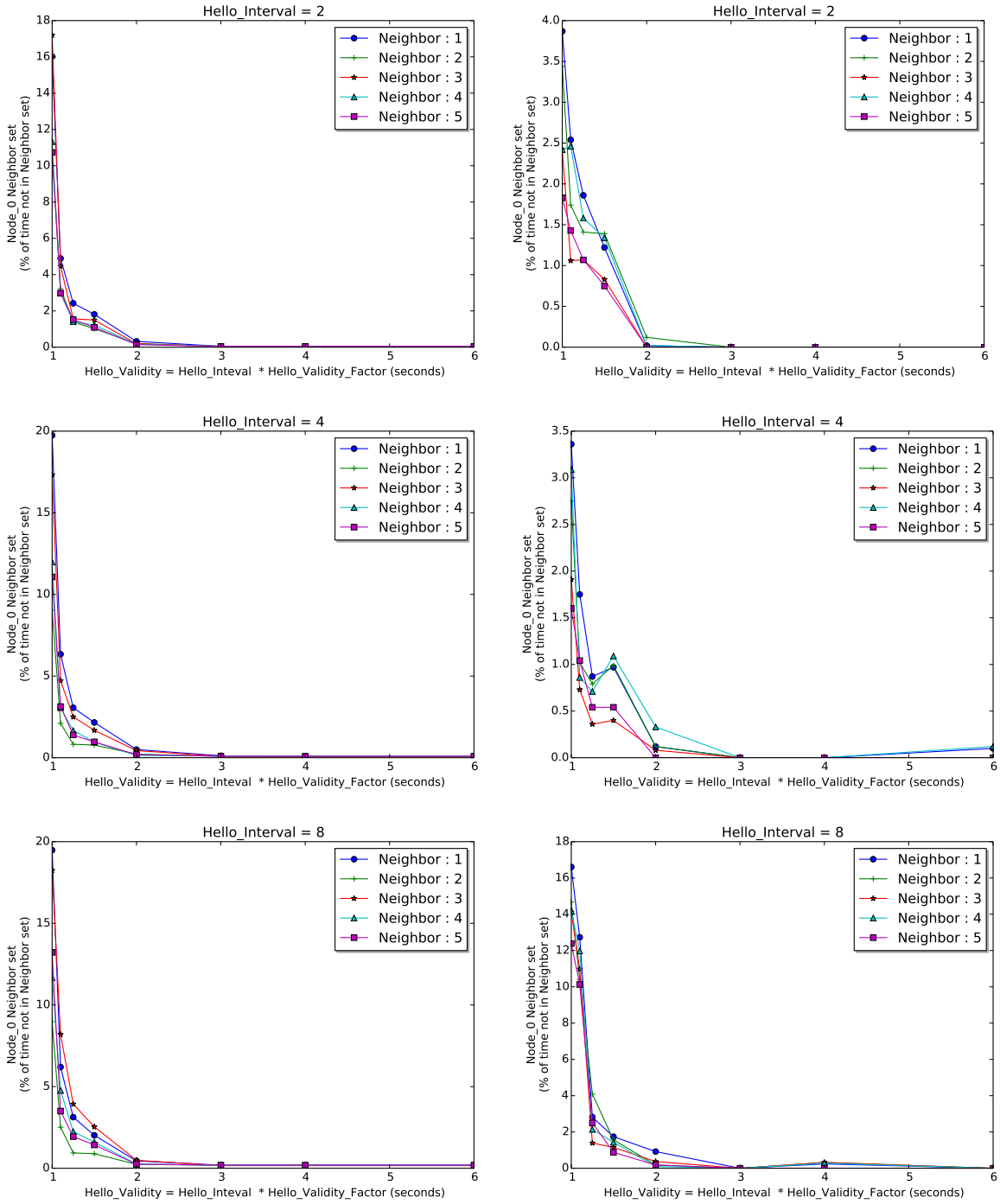


Figure 5.6: Plots showing the percentage of time nodes that should be in the symmetric neighbor set of Node_0 are not due to *neighbor flapping*. The left column of plots corresponds to simulation data, while the right column corresponds to data generated by experiments conducted on the physical platform. The set from which Hello_Validity is taken is: $\{1\delta, 1.1\delta, 1.25\delta, 1.5\delta, 2\delta, 3\delta, 4\delta, 6\delta\}$.

Table 5.5: Change in percentage of time not in neighbor set from $\tau = \delta$ to $\tau = 2\delta$, Node 0's neighbor set (simulation experiments)

δ (sec)	Node 1	Node 2	Node 3	Node 4	Node 5
2	-98	-98	-99	-99	-99
4	-92	-90	-91	-95	-94
8	-93	-90	-94	-95	-93

Table 5.6: Change in percentage of time not in neighbor set from $\tau = \delta$ to $\tau = 2\delta$, Node 0's neighbor set (physical experiments)

δ (sec)	Node 1	Node 2	Node 3	Node 4	Node 5
2	-99	-97	-100	-99	-100
4	-96	-96	-96	-89	-100
8	-95	-99	-97	-99	-99

Our experiments show that *neighbor flapping* indeed occurs; however, they do not identify the cause of this phenomenon. In the case of $\tau = \delta$ or $\tau = 1.1\delta$, *neighbor flapping* is caused by the closeness of the values of the neighbor discovery parameters. To see this, assume that two nodes A and B, which are within transmission range of each other, have discovered each other as symmetric neighbors. In order for A to continue to classify B as a symmetric neighbor, A expects B to refresh its neighbor information with a Hello_Message before the τ interval expires. Now, assume $\tau = \delta = 2$ seconds. This means that: (1) when A receives a Hello_Message from B, let us say at time t, it will assume that the information in it is valid for $\tau = 2$ seconds, i.e., until time t + 2; and (2) B will not transmit another Hello_Message until time t + δ = t + 2 seconds has elapsed. In this case, after two seconds elapse, node A will invalidate the information provided by the previous Hello_Message from B. And, at the same time B will start transmitting a Hello_Message to A. Accordingly, A

removes B from its neighbor set even though it is still within transmission range.

The above-stated argument is valid for $\tau = \delta$ and $\tau = 1.1\delta$. However, we still see neighbor flapping for $\tau = 1.25\delta$ and $\tau = 1.5\delta$. In these cases, *neighbor flapping* is caused by contention on the transmission medium. To show this, we performed experiments with no topology changes as before when we computed number of state changes and the percent of time neighbor nodes are out of the neighbor set, but with no generation of data traffic. Figure 5.6 shows plots of the data collected from experiments with and without data traffic. These plots indicate that for $\tau = \delta$ and $\tau = 1.1\delta$, the number of state changes for both types of experiments are close (due to the proximity of parameters). However, for τ values greater or equal to 1.25δ, the experiments with no data traffic show significantly smaller numbers of state changes. Tables 5.7 and 5.8 show the percent change in number of state changes between experiments with data traffic, and those without, for simulation and physical experiments, respectively. We observe that when $\tau \in \{\delta,\ 1.1\delta\}$ there is, on average, a decrease in the number of state changes of 12%. We also note that when $\tau \in \{1.25\delta,\ 1.5\delta\}$ there is, on average, a decrease of 71%. Thus, when τ is at least 1.25δ, we attribute neighbor flapping to contention on the transmission medium.

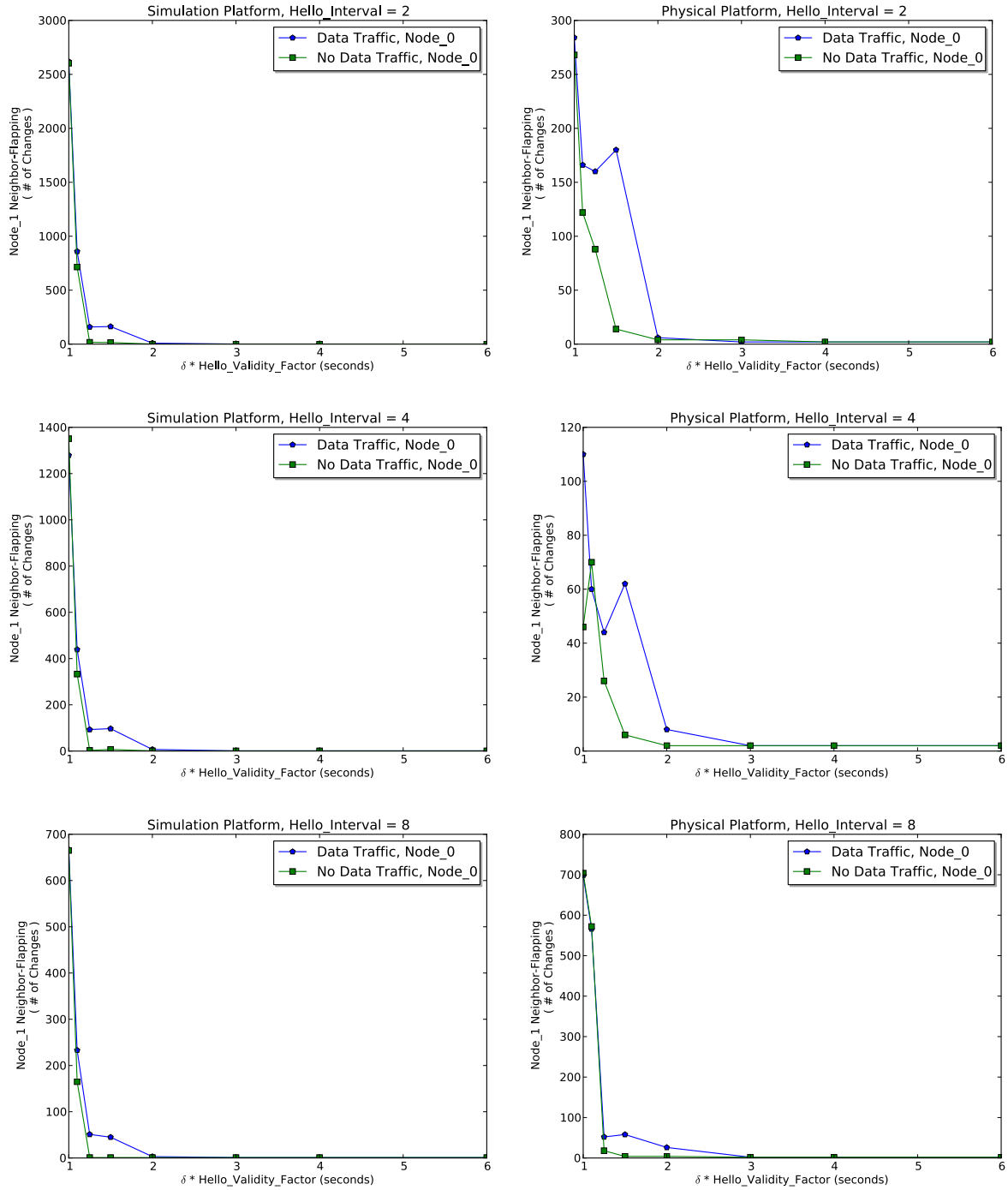


Figure 5.7: Plots comparing the number of state changes during experiments with data traffic and those with out data traffic. The left column of plots corresponds to simulation data, while the right column corresponds to data generated by experiments conducted on the physical platform. The set from which Hello_Validity is taken is: {1, 1.1, 1.25, 1.5, 2, 3, 4, 6}.

Table 5.7: Percent change in the number of state changes from experiments with data traffic and those without, Node 1's neighbor set (simulation)

τ (sec)	$\delta = 2$	$\delta = 4$	$\delta = 8$
(δ)	-0.61	-5.63	-0.30
(1.1δ)	-16.80	-24.15	-29.18
(1.25δ)	-89.31	-96.77	-98.04
(1.5δ)	-90.80	-92.78	-97.78

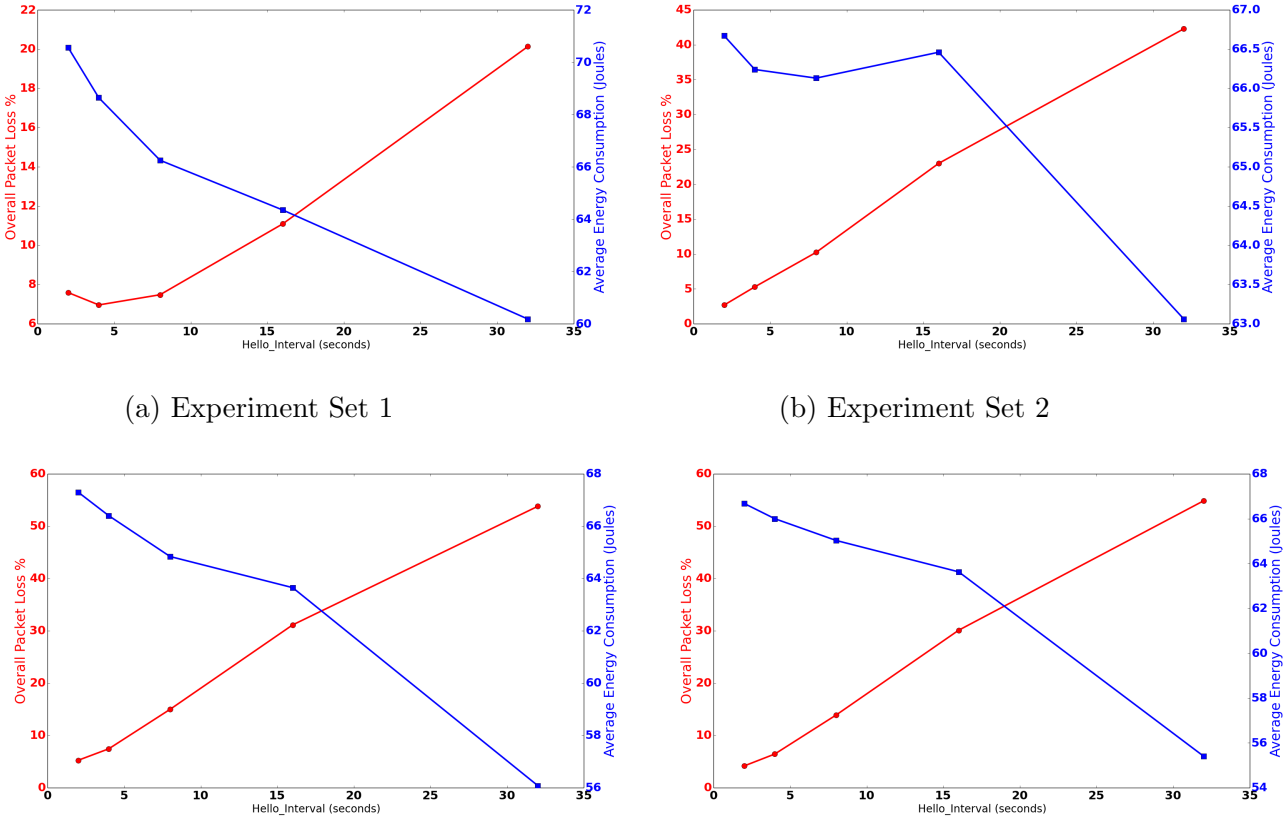
Table 5.8: Percent change in the number of state changes from experiments with data traffic and those without, Node 1's neighbor set (physical)

τ (sec)	$\delta = 2$	$\delta = 4$	$\delta = 8$
(δ)	-5.63	-58.18	0.86
(1.1δ)	-26.51	16.67	1.06
(1.25δ)	-45.00	-40.91	-65.38
(1.5δ)	-92.00	-90.32	-93.10

5.4 Energy Consumption

Recall that one of the objectives of this research is to explore the possibility of reducing the energy consumption of a MANET by reducing the frequency at which Hello_Messages are sent, while maintaining packet loss at an acceptable rate. To understand how decreasing δ affects the overall packet loss and energy consumption, we conducted experiments that varied δ from the set: {2,4,8,16,32} seconds. Since we previously observed that the overall packet loss percentage is optimal at $\tau = 2\delta$, for these experiments τ was fixed at 2δ. Again, for each parameter configuration, four independent experiments that use different sets of topologies were run and topology changes were performed every 120 seconds until

160 topology changes occurred. Again, data traffic was provided by a model of the ping application which sends an ICMP echo request every second; every node established a ping session to every other node in the network. Figure 5.8 shows the results of these experiments. As shown, network performance, in terms of overall packet loss percentage, decreases as the frequency at which Hello_Messages are transmitted decreases. From the plots, it can be seen that the average energy consumption of the MANET is at its highest when $\delta = 2$, and at its lowest when $\delta = 32$. However, the overall packet loss percentage is at its lowest when $\delta = 2$, and at its highest when $\delta = 32$. Thus, the performance objectives of lowest energy consumption and lowest percentage of packet loss are incompatible. However, tradeoffs between these two objectives are possible. To obtain a perspective on how much network performance is being traded off for energy savings, we use the overall packet loss percentage and energy consumption data obtained from the four independent experiments with $\delta = 2$ and average those values to obtain reference values. Then for each of the four independent experiments with $\delta = 4$, 8, 16, and 32, we average the overall packet loss and energy consumption data and express the difference between these values and the reference values as a percentage change. Table 5.11 presents this information. As can be seen, decreasing the frequency at which Hello_Messages are transmitted does decrease the average energy consumption. However, these relatively small reductions in the percentage of average energy consumption are accompanied by large increases in overall packet loss. For example, by increasing δ from two to four seconds, the overall packet loss is increased by 46.42% and the average energy consumption is decreased by 1.43%. For reference, the values for each of the points in Figure 5.8 are shown in Tables 5.9 and 5.10.



(a) Experiment Set 1

(b) Experiment Set 2

(c) Experiment Set 3

(d) Experiment Set 4

Figure 5.8: Plots showing the tradeoff between packet loss percentage and energy as the Hello_Interval increases.

Table 5.9: Overall packet loss percentage increases as the Hello_Interval increases.

	$\delta = 2$	$\delta = 4$	$\delta = 8$	$\delta = 16$	$\delta = 32$
Experiment set 1	7.59	6.96	7.48	11.09	20.13
Experiment set 2	2.68	5.29	10.23	22.99	42.29
Experiment set 3	5.25	7.45	14.99	31.13	53.80
Experiment set 4	4.17	6.45	13.90	30.11	54.87

Table 5.10: Energy (Joules) slightly decreases as the Hello_Interval increases.

	$\delta = 2$	$\delta = 4$	$\delta = 8$	$\delta = 16$	$\delta = 32$
Experiment set 1	70.56	68.64	66.25	64.35	60.18
Experiment set 2	66.67	66.24	66.13	66.46	63.06
Experiment set 3	67.30	66.40	64.84	63.65	56.08
Experiment set 4	66.68	66.00	65.03	63.63	55.40

Table 5.11: Average percent change using δ 2 as the reference point.

	$\delta = 4$	$\delta = 8$	$\delta = 16$	$\delta = 32$
Overall Packet Loss	46.42	174.78	479.74	945.95
Average Energy Consumption	-1.43	-3.26	-4.78	-13.42

Chapter 6

Conclusions and Future Work

The goal of the research described in this thesis is to improve the performance of Mobile Ad Hoc Networks (MANETs) that use soft-state signaling routing protocols, specifically OLSR. However, this work should be applicable to other network protocols that rely on the soft-state signaling technique such as RSVP, PIM, and IGMP [12]. In particular, using simulations and previous work based on experiments conducted on a physical test bed [4], we studied the behavior of the neighbor-discovery algorithm used by OLSR, which relies on soft-state signaling to identify and explore ways to optimize the neighbor discovery process.

Candidate optimizations include the actual and relative settings of the refresh and expiry timers used by the algorithm. Previous studies focused on understanding how the settings of the algorithms Hello_Interval (δ) and TC_Interval refresh timers affect network performance in terms of protocol message overhead and network throughput. Since the Hello_Validity (τ) and TC_Validity expiry timers were neglected in these studies, the first objective of this thesis is to investigate how the setting of the Hello_Validity timer, relative to the setting of the Hello_Interval timer, affects network performance in terms of overall network packet loss. The second objective of the thesis is to explore how the setting of the Hello_Interval timer affects energy consumption and to quantify the tradeoff between energy consumption and packet loss.

To accomplish these objectives, NS-3, a popular discrete-event network simulator, was instrumented to simulate an OLSR network. OLSR is a popular soft-state signaling routing protocol. The simulator was designed to closely model the physical platform developed in [4]. Experiments were designed to enable cross-validation of both platforms. The setting of the Hello_Interval timer determines the Hello_Message interval, i.e., the frequency

with which Hello_Messages are transmitted by each node in the network. Intuitively, we know that small frequencies translate to larger protocol message overhead, more network traffic, and a greater potential for network contention. However, the small frequencies also translate to more current routing information in the network since nodes are made aware more quickly when neighbor nodes move out of range. Analogously, the setting of the Hello_Validity timer determines the Hello_Validity interval, i.e., the life span during which neighbor state information resident in nodesis considered valid. Thus, it does not make sense to set this timer to be less than the setting of the Hello_Interval timer, otherwise the information will expire before it is refreshed. Intuitively, smaller life spans translate to a greater potential for the expiration of valid routing information, which can result in packet loss; while larger life spans translate to a greater potential for using stale routing information, which can also result in packet loss.

6.1 Conclusions

The question we wanted to answer was: How should these timers be set relative to one another to minimize packet loss? Results obtained from our simulations indicate that the relationship between the settings of these timers do, indeed, impact network performance in terms of the percentage of overall packet loss. In particular, we discovered that: (1) For Hello_Validity values smaller than 2δ, as the Hello_Validity approaches the value of the Hello_Interval, the percentage of packet loss increases due to the proximity of these parameters and collisions; and (2) setting the Hello_Validity timer to twice the value of the Hello_Interval timer results in a configuration with little packet loss. Note that although the OLSR Request For Comments (RFC3626) suggests that the Hello_Validity timer should be set to 3δ, the literature does not provide support for this suggestion.

We attribute this result to a phenomenon that we call *neighbor flapping*, where neighbor information for a node that is within range is repeatedly placed in and then evicted from neighbor set of other nodes. Our simulation data show that this occurs when the

Hello_Validity interval is less than twice the Hello_Message interval. In this case, the simulations confirm that the smaller the Hello_Validity interval, the greater is the potential for the expiration of valid neighbor information, which, in turn, can result in greater packet loss. This is because as the Hello_Validity interval decreases, there is a greater potential for a refreshing Hello_Message to be received after the targeted routing information expires. As shown by our simulation data, this delay is due to both the relative settings of the two timers, i.e., they are set to values that are too close to one another, and contention in the network. With regards to energy consumption, our preliminary experimental results show that energy can be saved by decreasing the frequency at which Hello_Messages are sent, however, this comes at a high price in terms of packet loss.

6.2 Future Work

In terms of the study presented in the thesis, further investigation is warranted to confirm that the relationship between the Hello_Interval and Hello_Validity timers holds for larger node densities. In addition, the simulator should be extended to model gradual movement of nodes in order to confirm this relationship with propagation delay and propagation loss modeling. In terms of energy consumption, further experimentation on the physical platform is required to validate that the observation made using simulation data holds on the physical platform. This will require that the physical platform be instrumented to measure energy consumption. Furthermore, an objective validation methodology should be employed to improve the credibility of the results of the subjective validation methodology used in this work.

The experiments conducted as part of this thesis found that packet loss increases as the Hello_Interval timer setting increases. This is due to the fact that Hello_Messages form the basis for topology discovery, making it impractical to rely on a decreased frequency of Hello_Messages for energy savings. Nonetheless, it would be interesting to study the effect that other refresh timers, such as the TC, Multiple Interface Declaration (MID), and Host

Network Association (HNA) timers, have on network performance, in particular, energy consumption and packet loss.

References

[1] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," url: http://www.ietf.org/rfc/rfc3626.txt., 2003.

[2] T. Issariyakul and E. Hossain, Introduction to Network Simulator NS2, Springer Science & Business Media, 2008, pp. vii.

[3] E. Weingartner ,H. vom Lehn, and K. Wehrle "A Performance Comparison of Recent Network Simulators." IEEE International Conference on Communications (ICC), 2009, pp. 1-5.

[4] J. McCartney "Neighbor Discovery Message Hold Times for Mobile Ad Hoc Networks," Master's Thesis, The University of Texas at El Paso, 2014.

[5] M. Lacage and T. Henderson, "Yet Another Network Simulator" Proceeding from the workshop on ns-2: the IP network simulator, ACM, 2006, pp. 1-10.

[6] nsnam.org "Optimized Link State Routing (OLSR)" http://www.nsnam.org/docs/models/html/olsr.html, 2011.

[7] R. Sargent, "Verification and Validation of Simulation Models" Simulation Research Group, Department of Electrical Engineering and Computer Science, Syracuse University,New York, 1998, pp. 1-10.

[8] S. Demers and L. Kant "MANETs: Performance Analysis and Management" Military Communications Conference (MILCOM), IEEE, 2006, pp. 1-7.

[9] Y. Huang, S. Bhatti, and D. Parker, "Tuning OLSR," Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), IEEE, 2006, pp. 1-5.

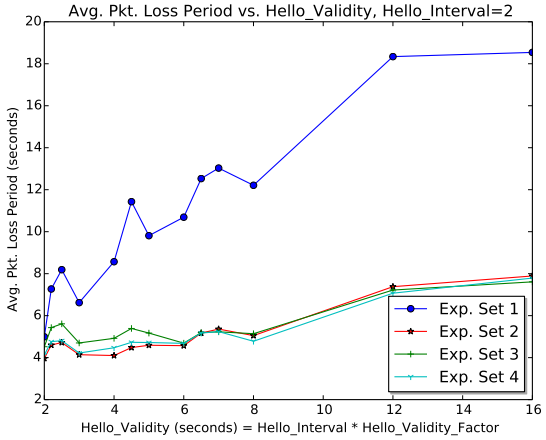
[10] C. Gomez, "Improving Performance of a Real Ad-hoc Network by Tuning OLSR Parameters" Symposium on Computers and Communications (ISCC), 2005, pp. 1-6.

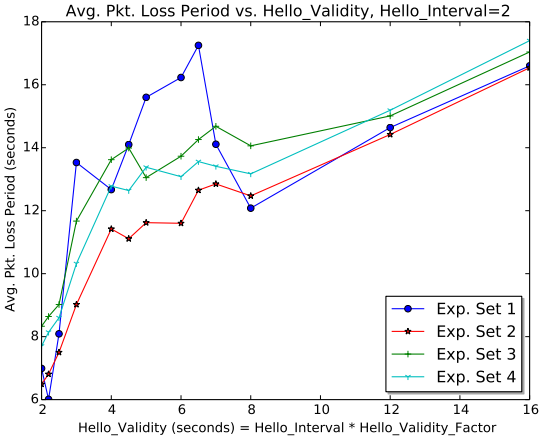
[11] A. Bose, "Impacts of Refresh Interval Parameters on M-OLSR Performance for Wireless Mesh Networks" Region 10 Conference, TENCON, IEEE, 2009, pp. 1-5.

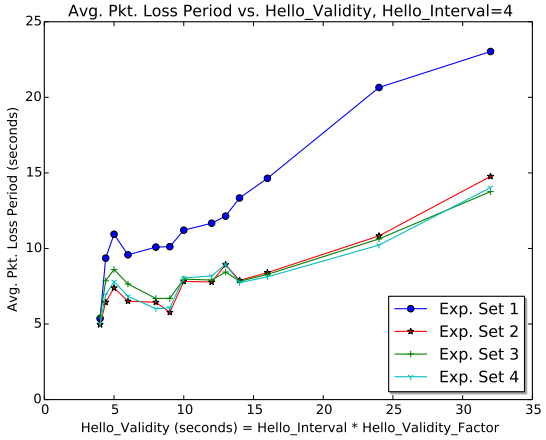
[12] J. Kurose and R.Keith Computer Networking: A Top Down Approach, 5th Edition USA: Addison-Wesley Publishing Company, 2010

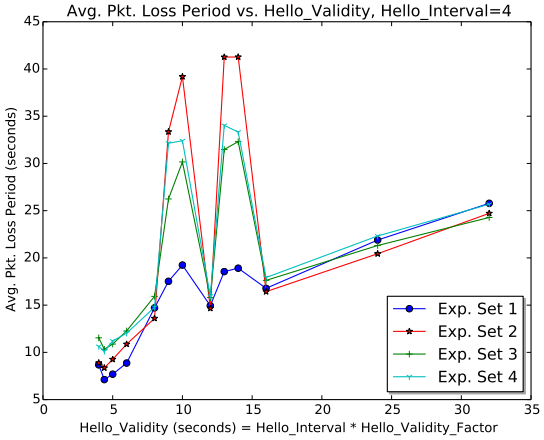
Appendix A

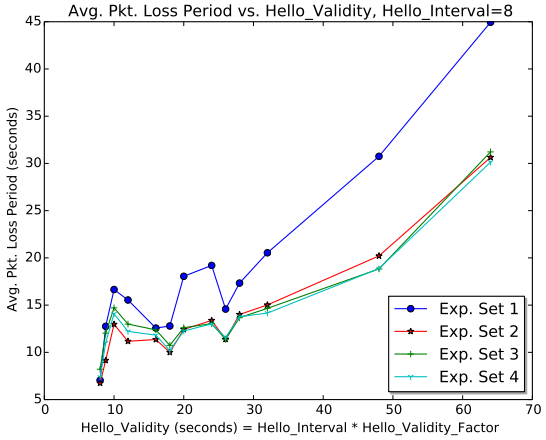
Average Packet Loss Period

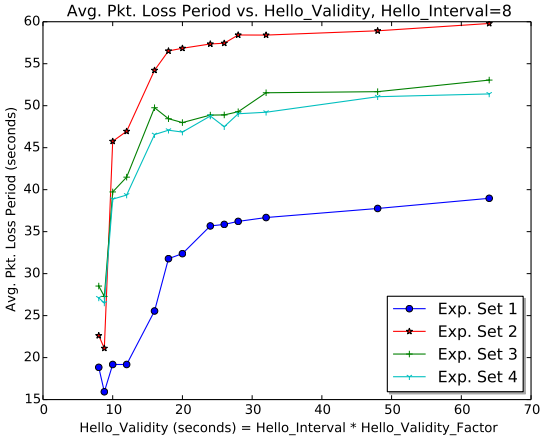












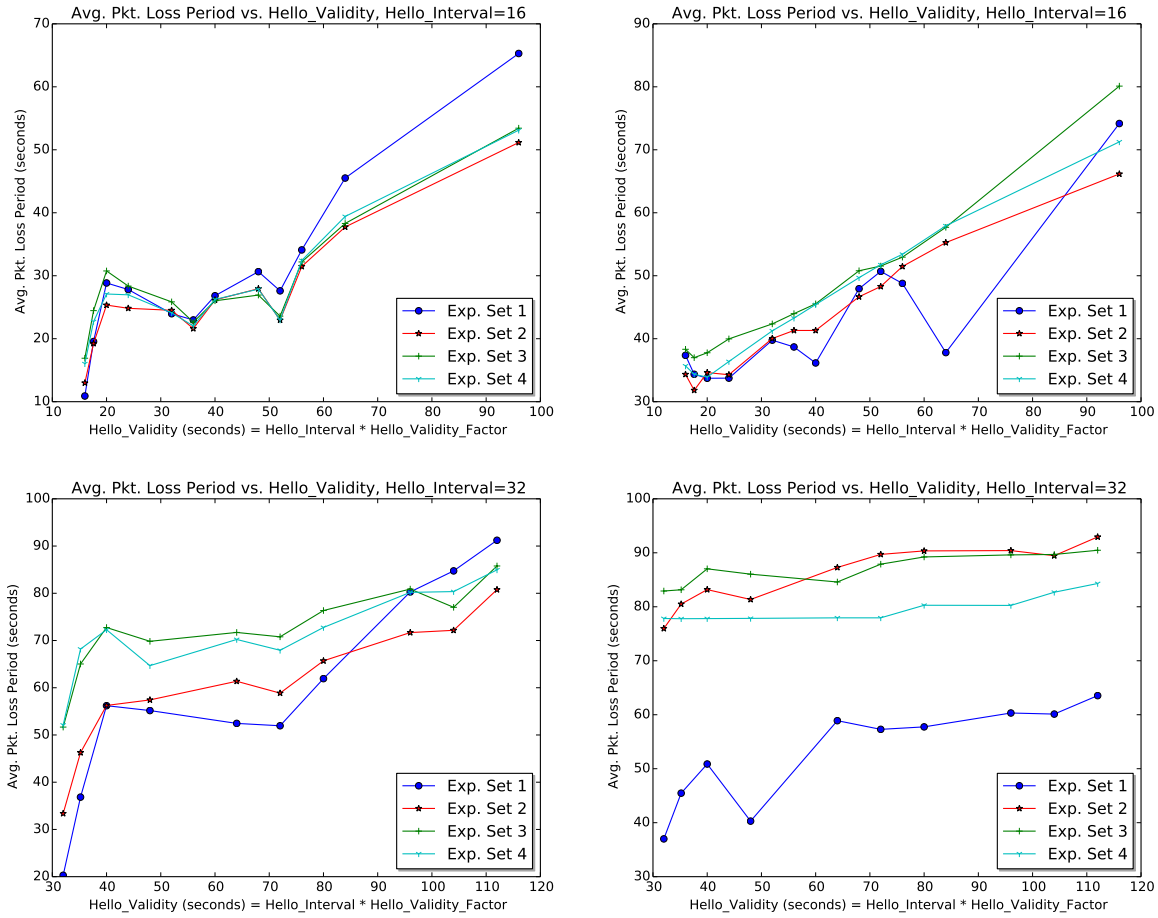


Figure A.1: Average packet loss period increasing as the Hello_Validity parameter becomes larger than the Hello_Interval parameter. The left column of plots corresponds to simulation data, while the right column corresponds to data from the physical platform. The set of values used for Hello_Validity is:$\{1\delta,\ 1.1\delta,\ 1.25\delta,\ 1.5\delta,\ 2\delta,\ 2.25\delta,\ 2.5\delta,\ 3\delta,\ 3.25\delta,\ 3.5\delta,\ 4\delta,\ 6\delta,\ 8\delta\}$. Topology Change Interval = 120 seconds.

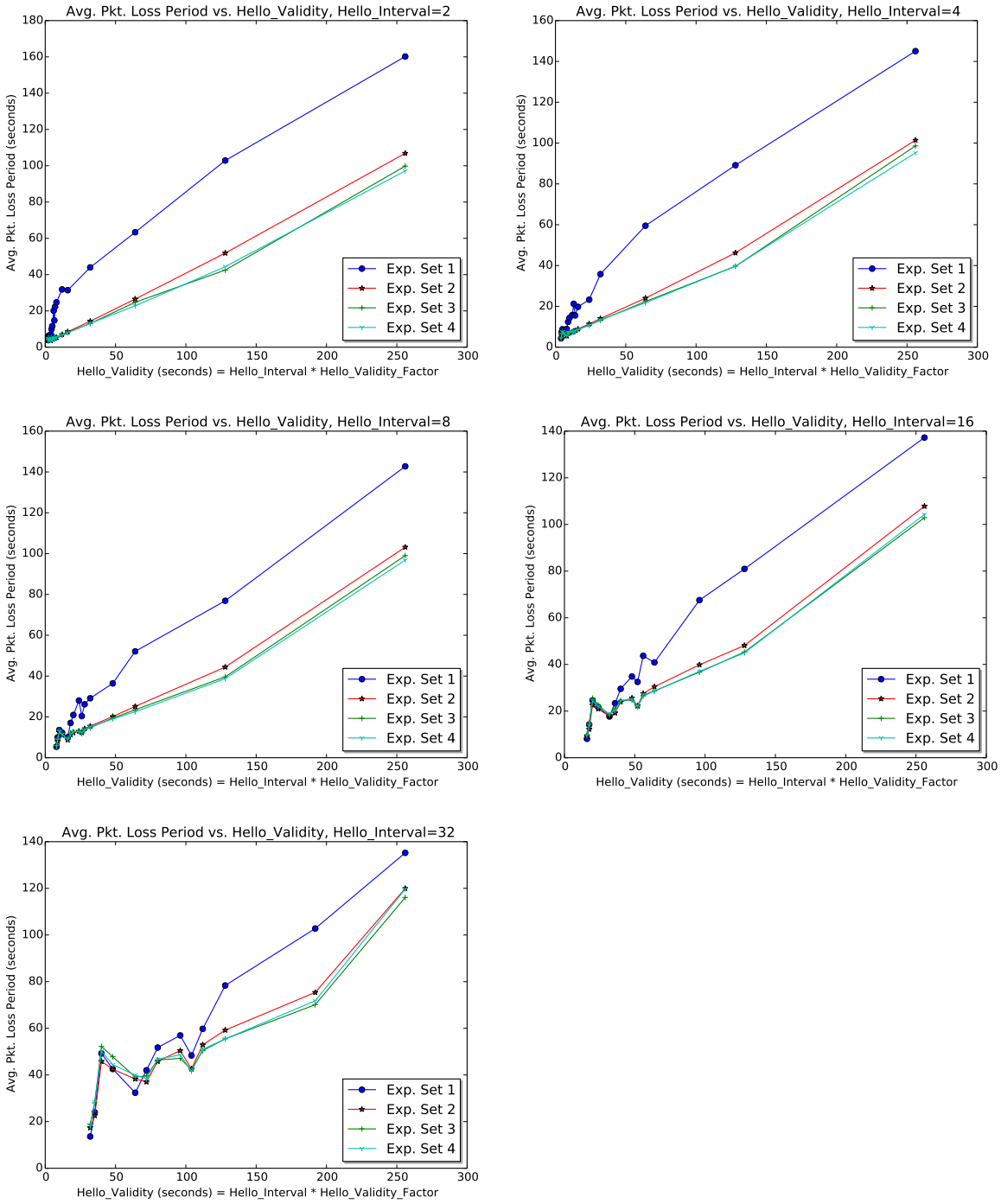


Figure A.2: Average packet loss period increasing as the Hello_Validity parameter becomes larger than the Hello_Interval parameter. Plots correspond to simulation data. The set of values used for Hello_Validity is:{1δ, 1.1δ, 1.25δ, 1.5δ, 2δ, 2.25δ, 2.5δ, 3δ, 3.25δ, 3.5δ, 4δ, 6δ, 8δ}. Topology Change Interval = 512 seconds.

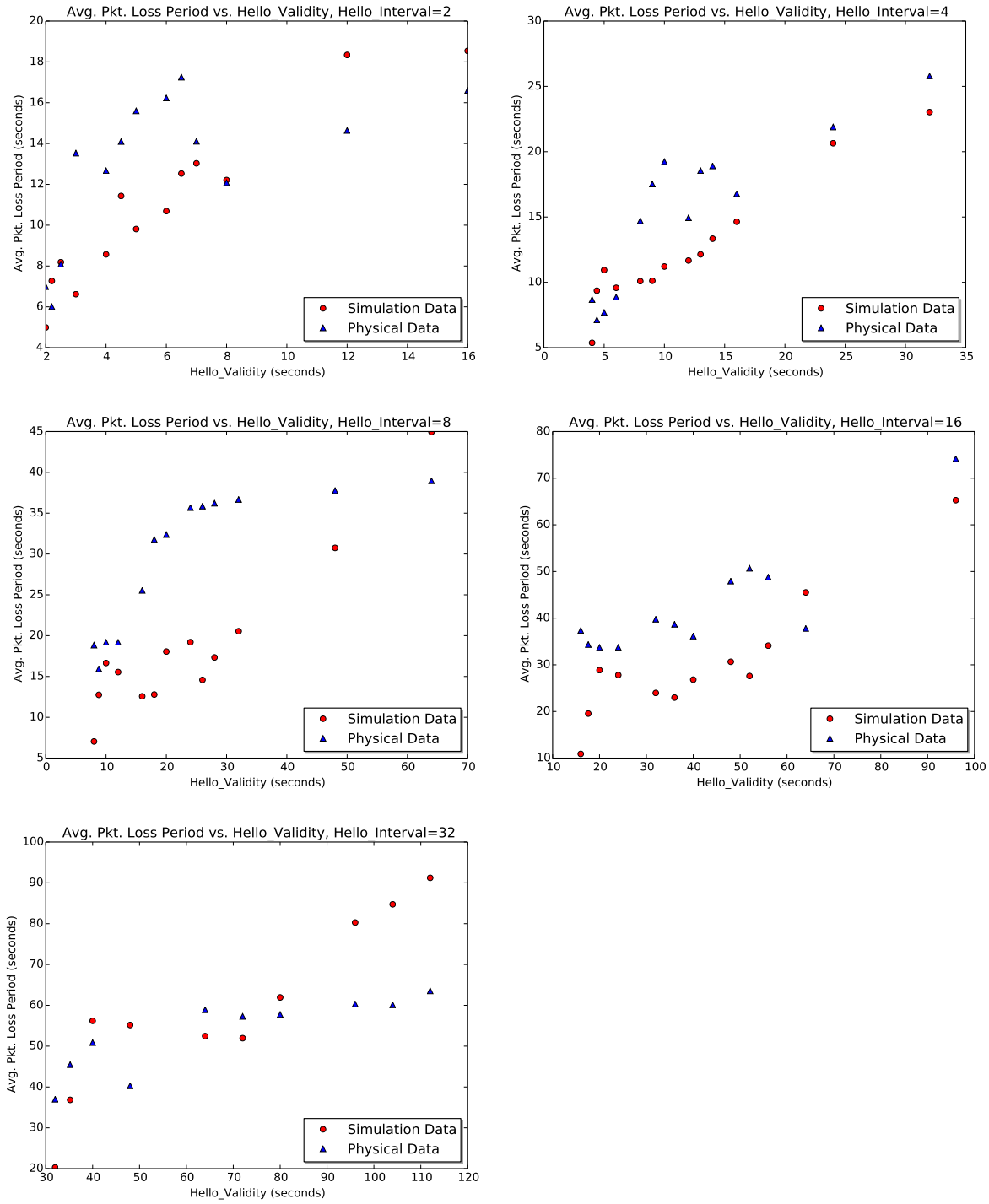


Figure A.3: Comparison of average packet loss period trend shown by Experiment Set 1 data generated using simulation and physical platforms.

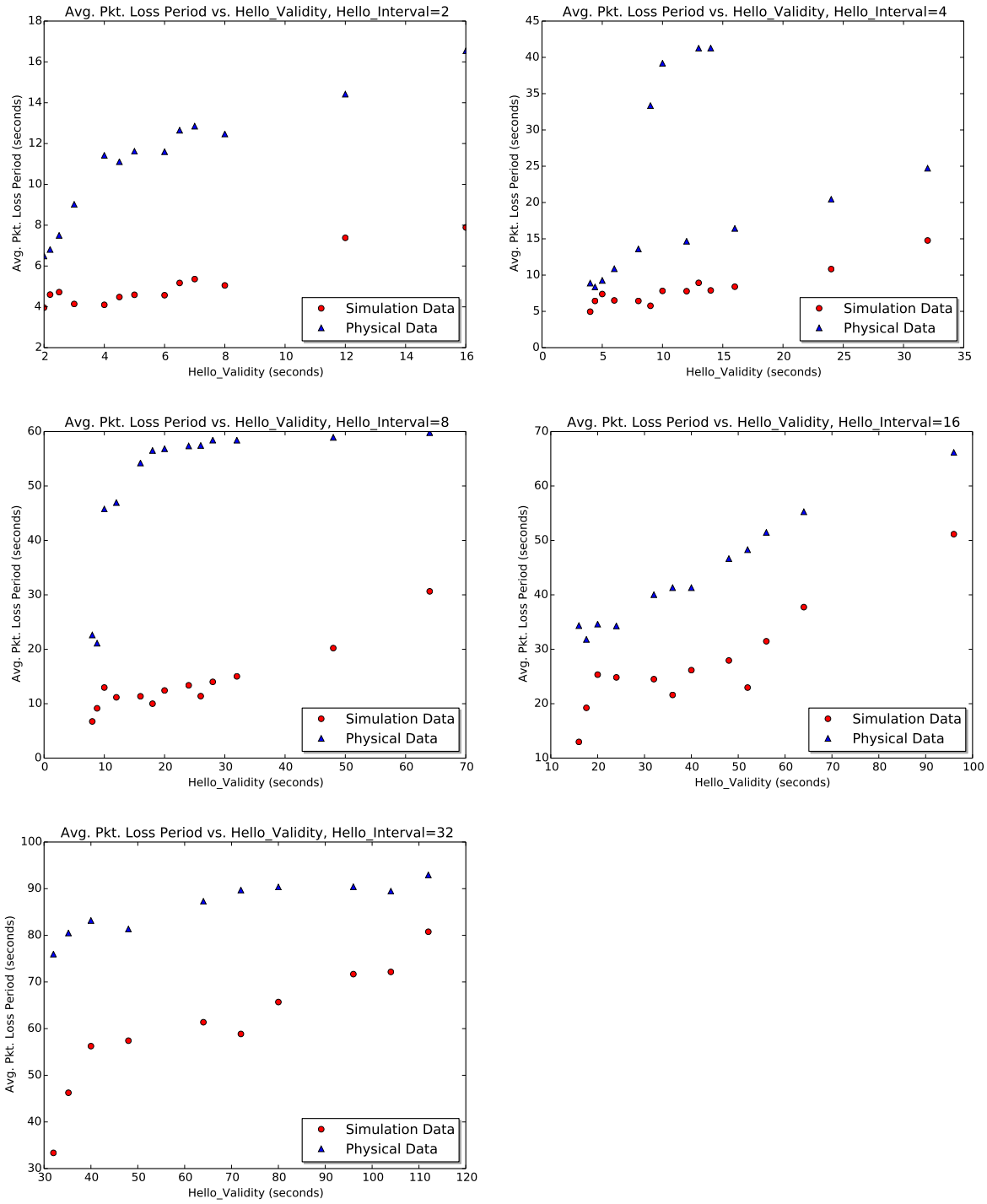


Figure A.4: Comparison of average packet loss period trend shown by Experiment Set 2 data generated using simulation and physical platforms.

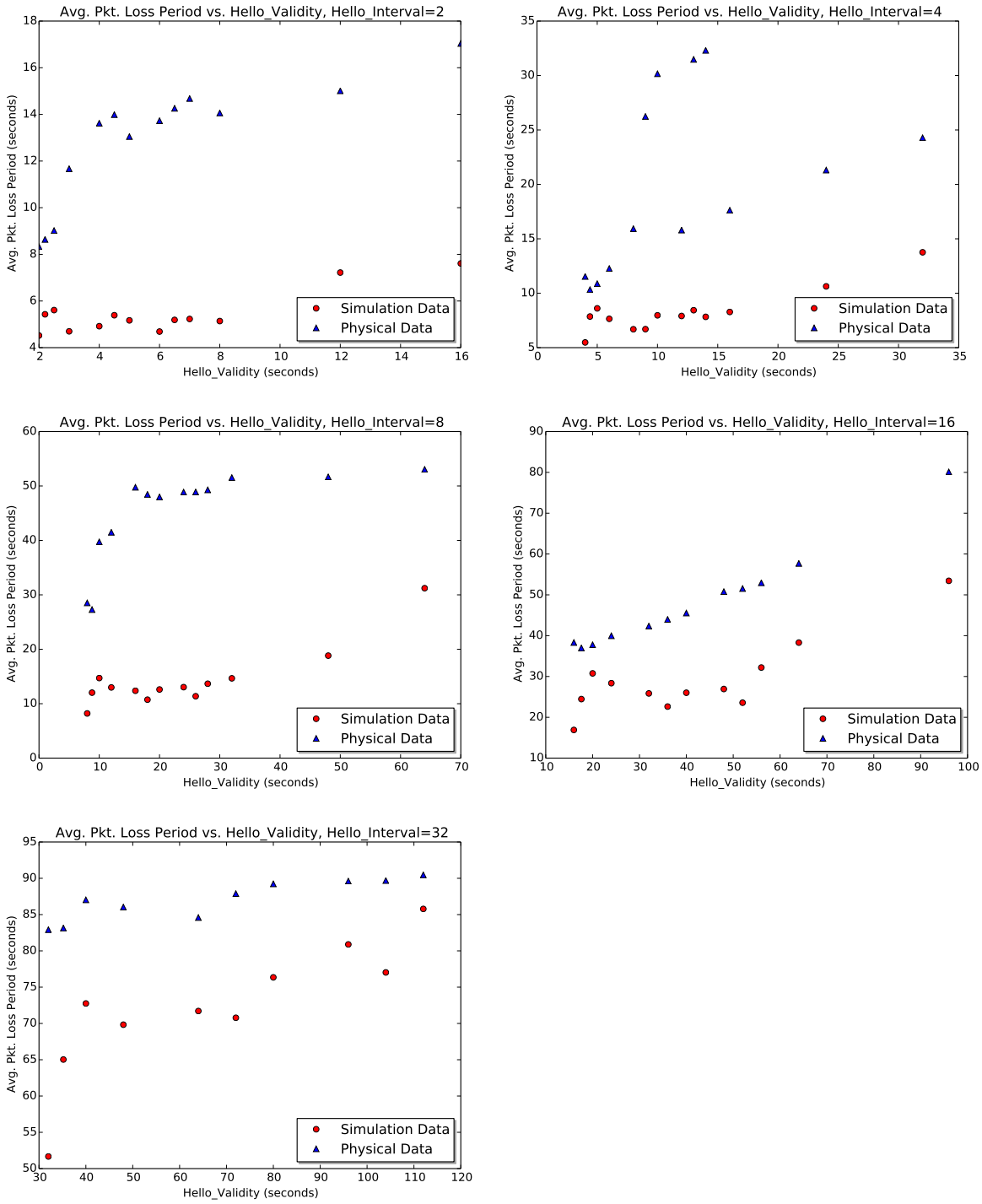


Figure A.5: Comparison of average packet loss period trend shown by Experiment Set 3 data generated using simulation and physical platforms.

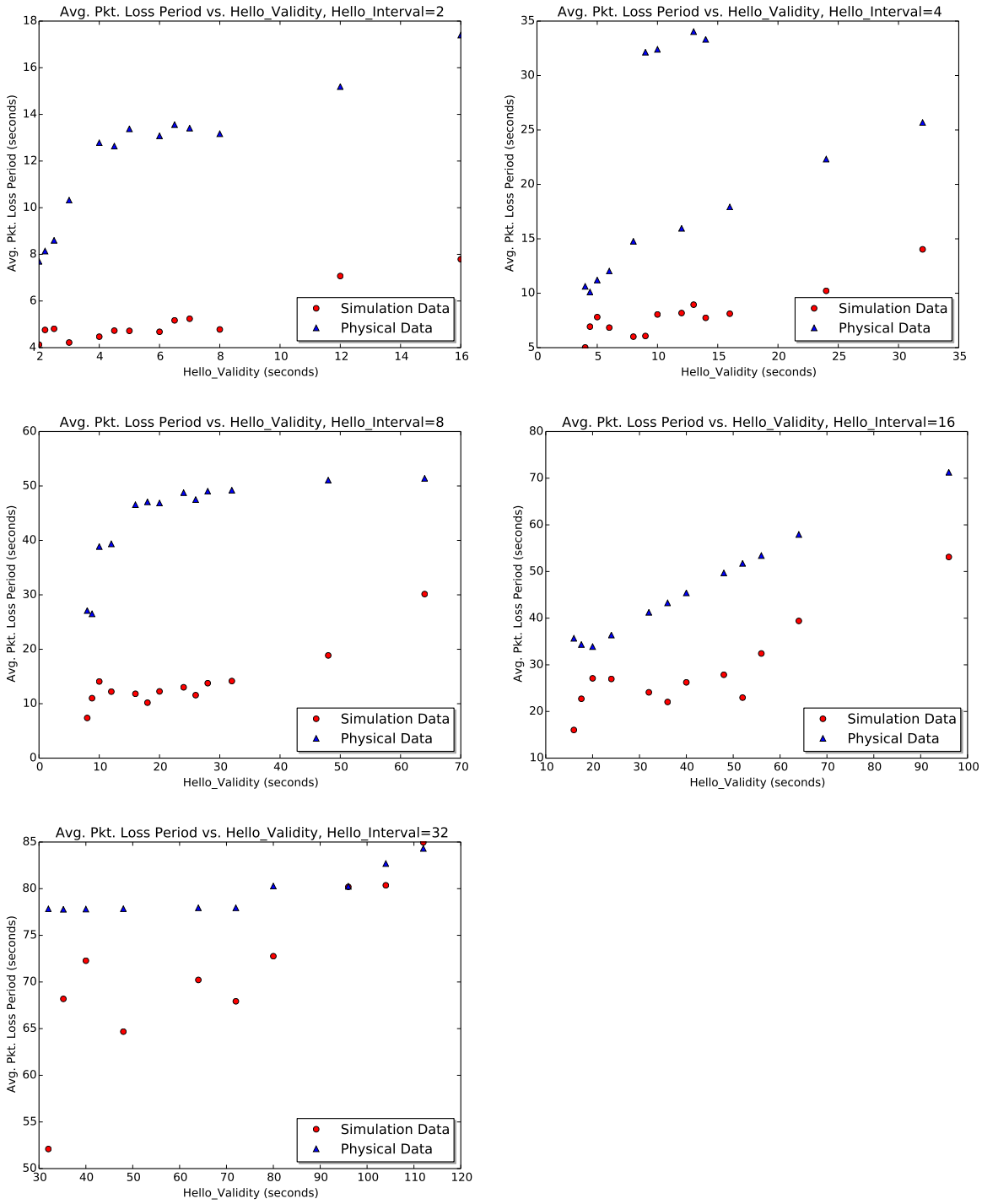
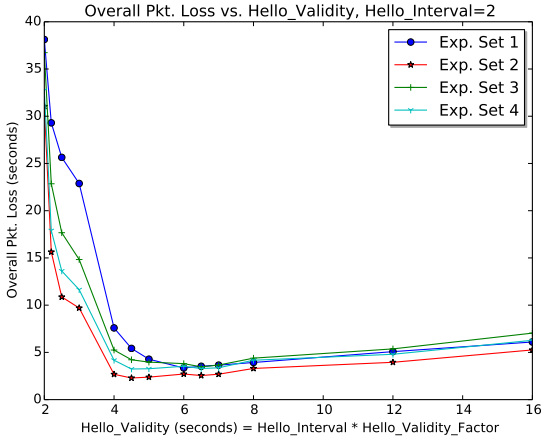
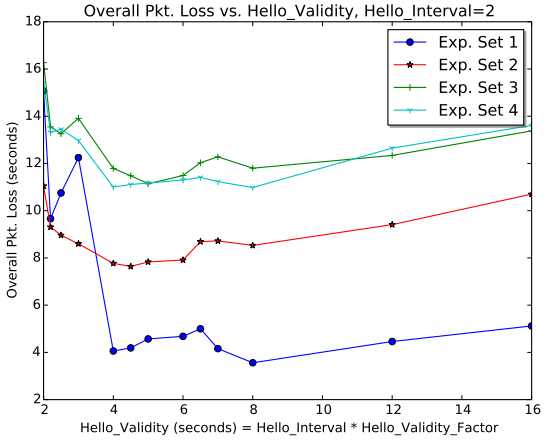


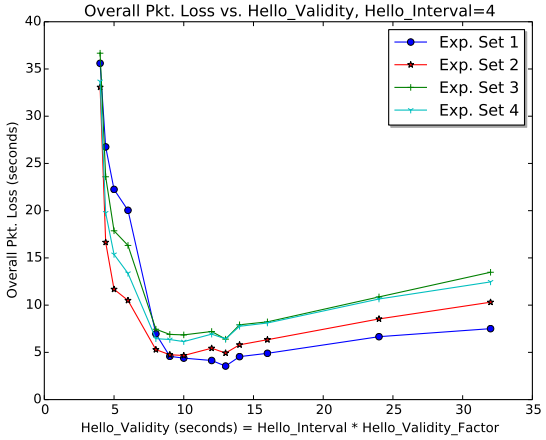
Figure A.6: Comparison of average packet loss period trend shown by Experiment Set 4 data generated using simulation and physical platforms.

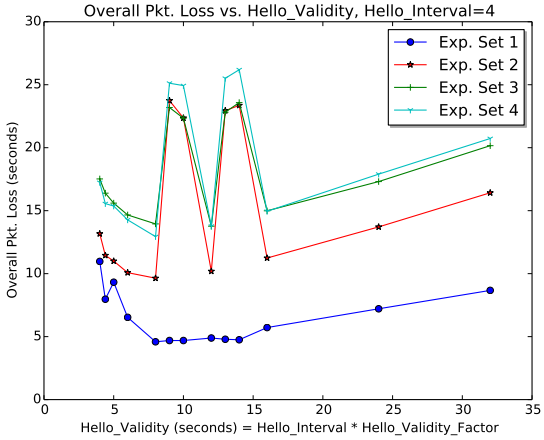
Appendix B

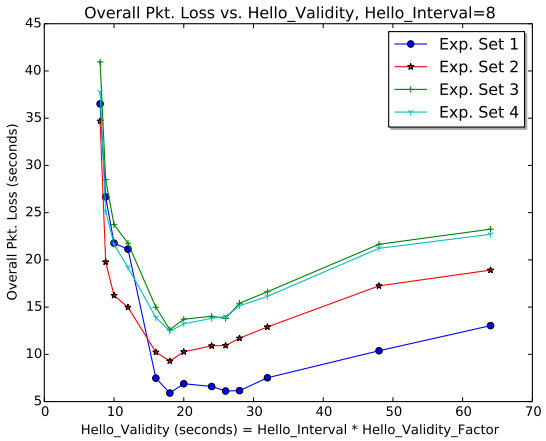
Overall Packet Loss Percentage

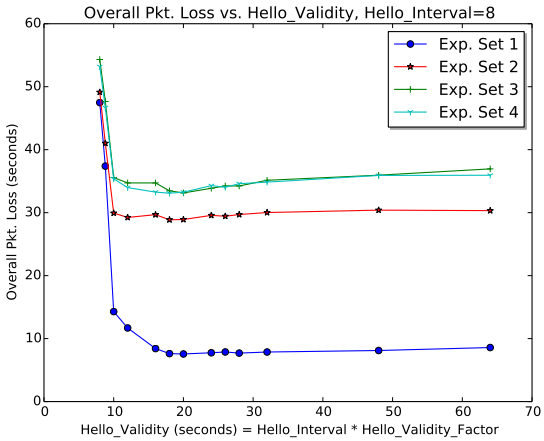












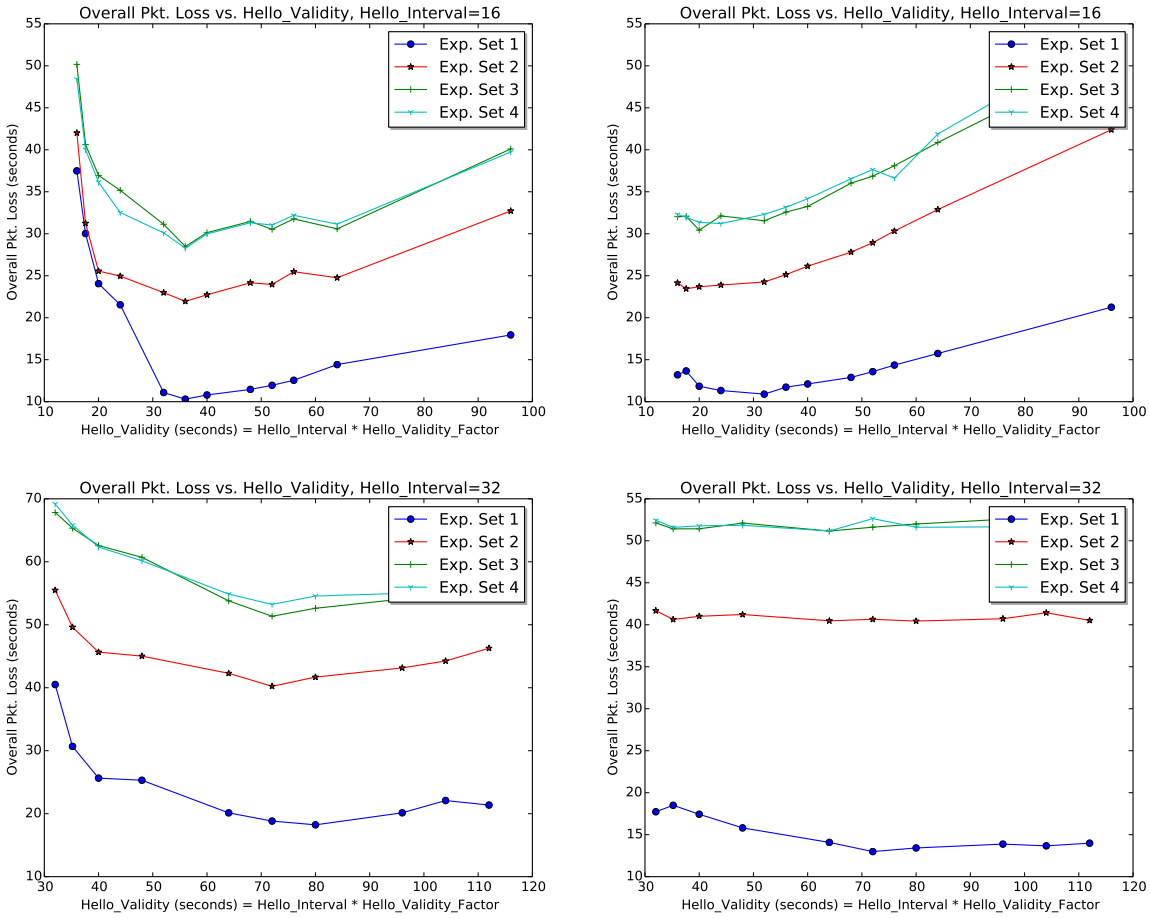


Figure B.1: MANET performance in terms of overall packet loss percentage as the Hello_Validity increases. The left column of plots corresponds to simulation data, while the right column corresponds to data generated using the physical platform. The set of values used for Hello_Validity is: $\{1\delta$, 1.1δ, 1.25δ, 1.5δ, 2δ, 2.25δ, 2.5δ, 3δ, 3.25δ, 3.5δ, 4δ, 6δ, $8\delta\}$. Topology Change Interval = 120 seconds.

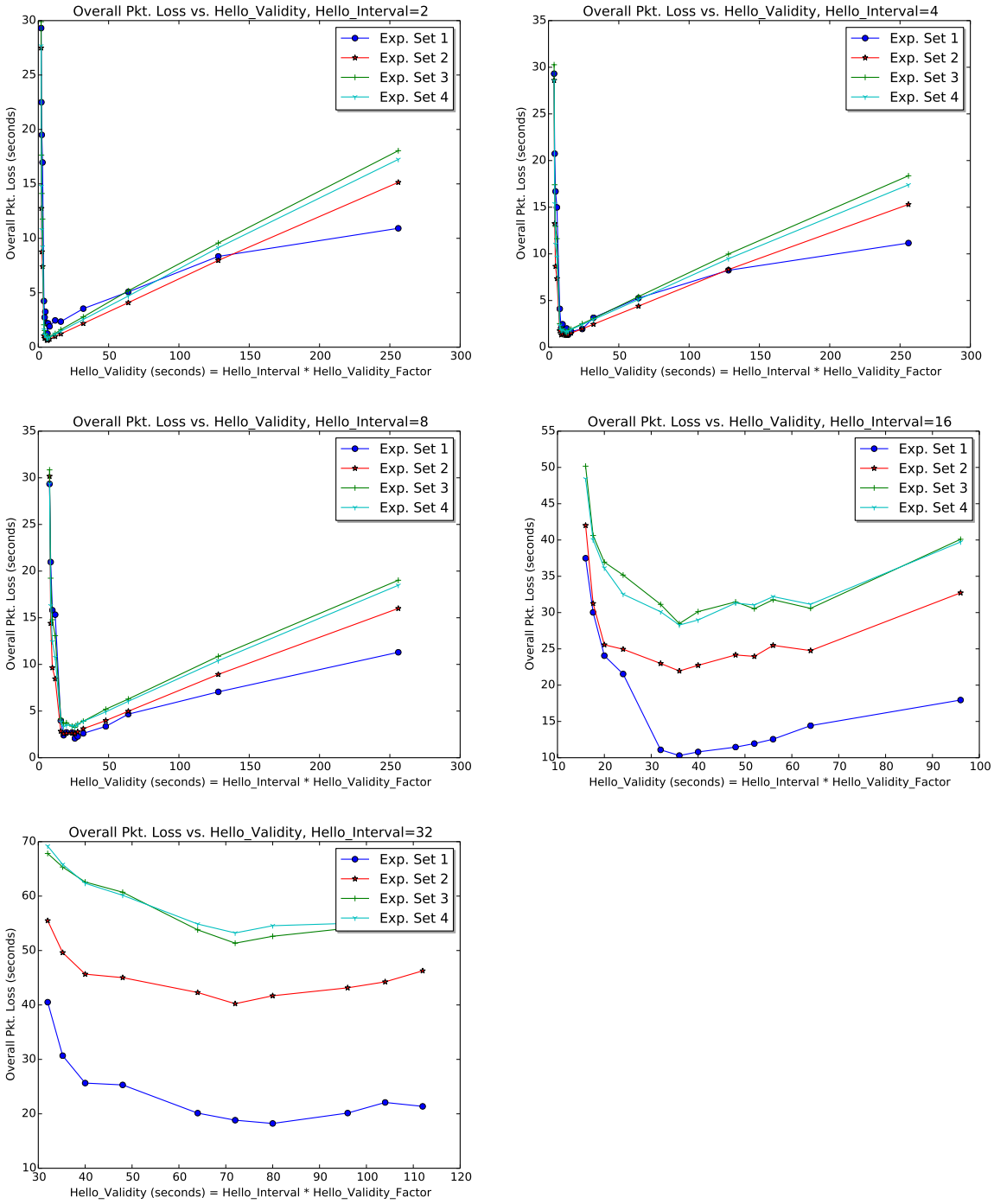


Figure B.2: MANET performance in terms of overall packet loss percentage as the Hello_Validity increases. The left column of plots corresponds to simulation data, while the right column corresponds to data generated using the physical platform. The set of values used for Hello_Validity is: $\{1\delta$, 1.1δ, 1.25δ, 1.5δ, 2δ, 2.25δ, 2.5δ, 3δ, 3.25δ, 3.5δ, 4δ, 6δ, $8\delta\}$. Topology Change Interval $= 512$ seconds.

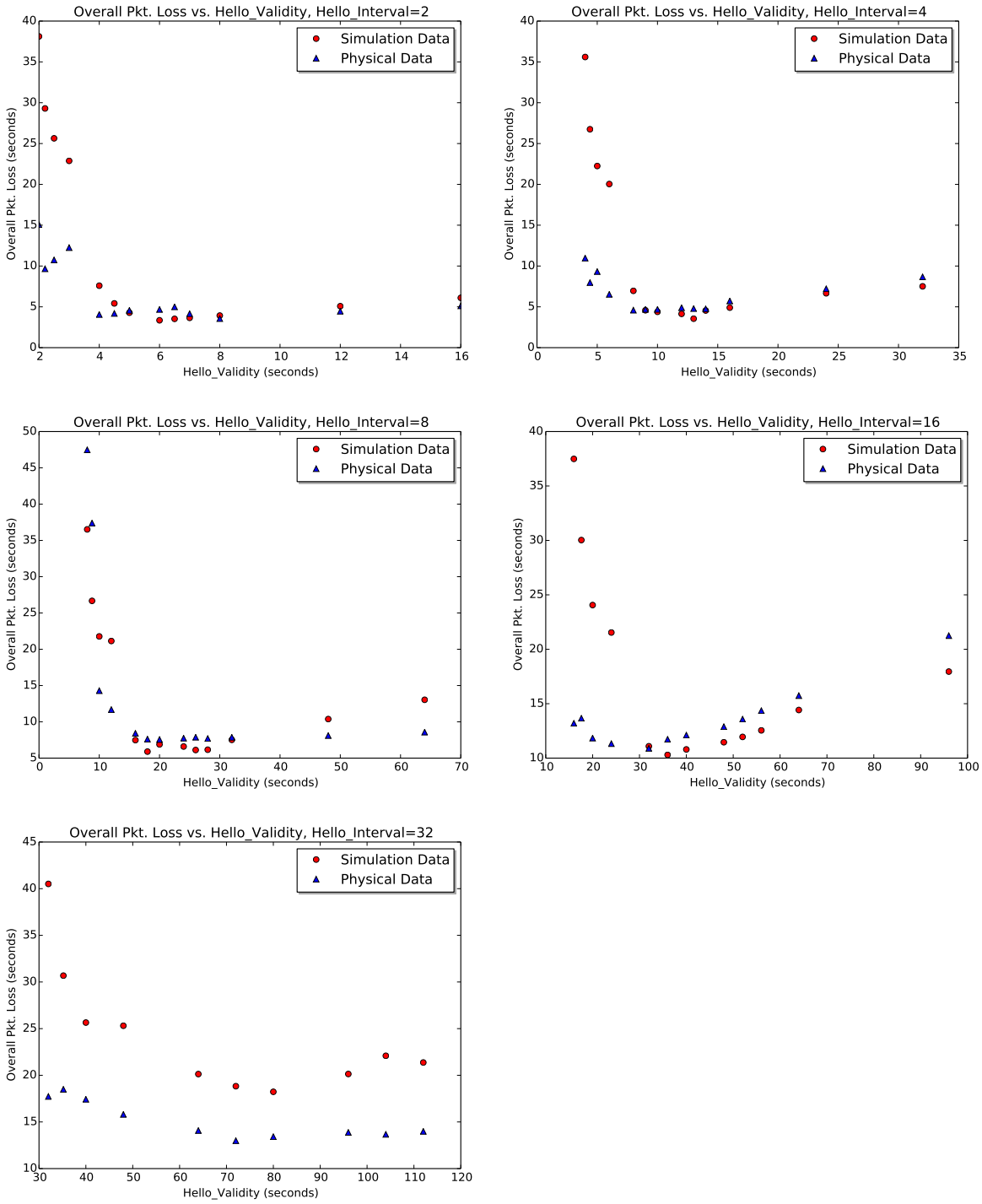


Figure B.3: Comparison of overall packet loss percentage trend shown by Experiment Set 1 data generated using simulation and physical platforms.

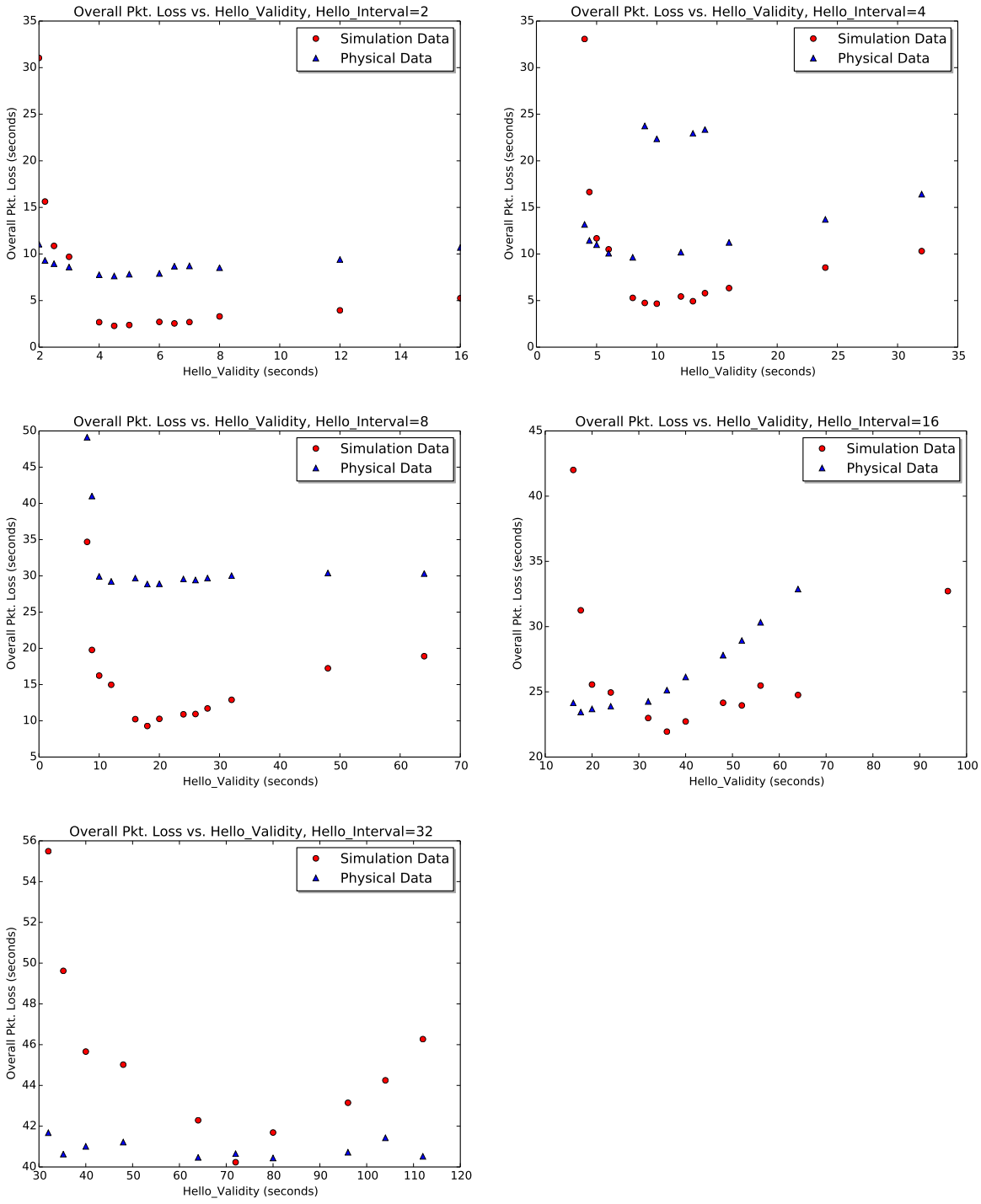


Figure B.4: Comparison of overall packet loss percentage trend shown by Experiment Set 2 data generated using simulation and physical platforms.

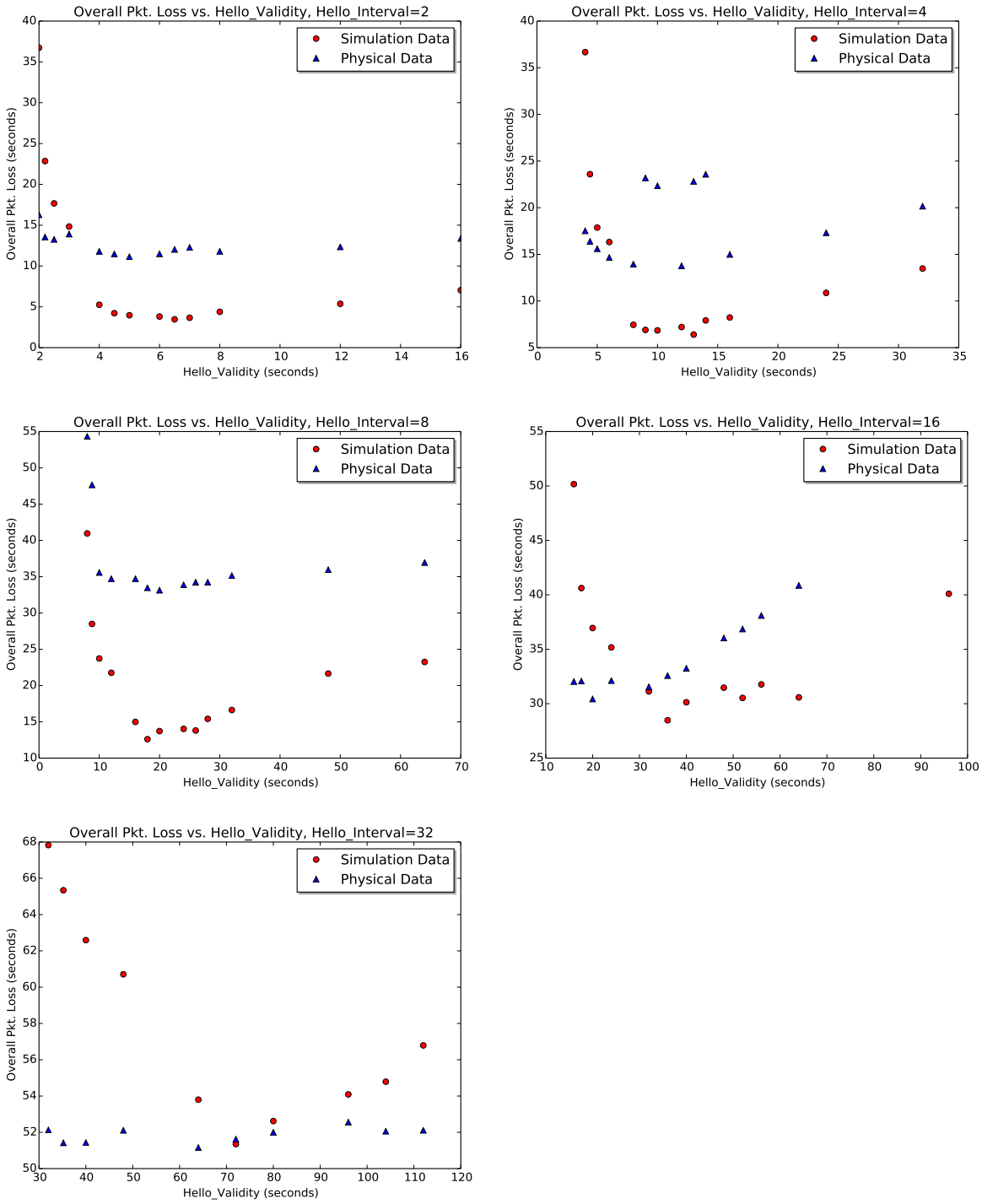


Figure B.5: Comparison of overall packet loss percentage trend shown by Experiment Set 3 data generated using simulation and physical platforms.

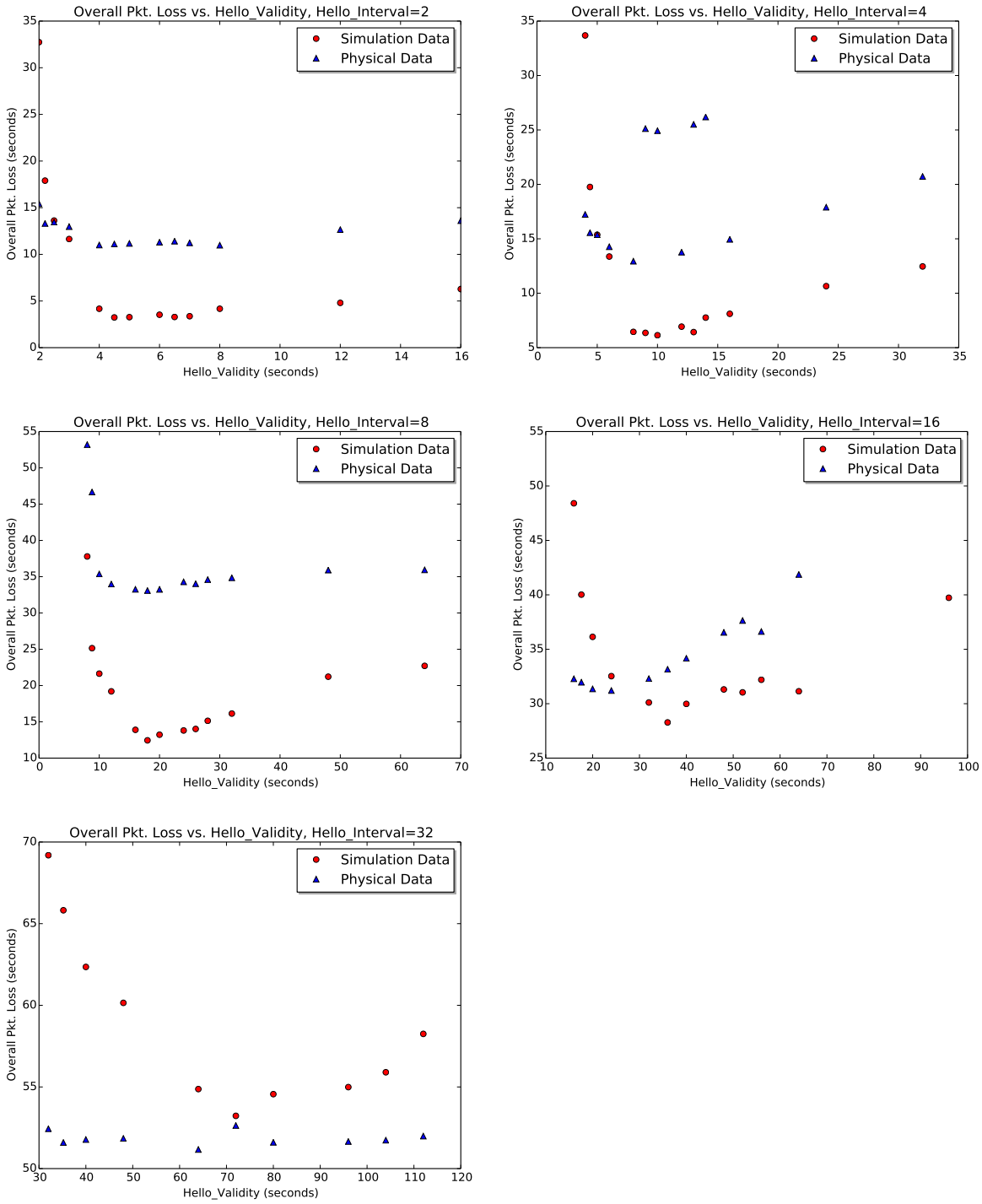


Figure B.6: Comparison of overall packet loss percentage trend shown by Experiment Set 4 data generated using simulation and physical platforms.

Table B.1: Packet loss percentage for $\tau = 2\delta$ vs. τ yielding the lowest (Best) packet loss percentage. $\delta = 2$

τ (sec)	Exp. Set 1	Exp. Set 2	Exp. Set 3	Exp. Set 4
(2δ)	-4.06	7.77	11.79	11.00
(Best)	3.56	7.64	11.14	10.98

Table B.2: Packet loss percentage for $\tau = 2\delta$ vs. τ yielding the lowest (Best) packet loss percentage. $\delta = 4$

τ (sec)	Exp. Set 1	Exp. Set 2	Exp. Set 3	Exp. Set 4
(2δ)	4.59	9.64	13.95	12.94
(Best)	4.59	9.64	13.77	12.94

Table B.3: Packet loss percentage for $\tau = 2\delta$ vs. τ yielding the lowest (Best) packet loss percentage. $\delta = 8$

τ (sec)	Exp. Set 1	Exp. Set 2	Exp. Set 3	Exp. Set 4
(2δ)	8.42	29.68	34.72	33.26
(Best)	7.56	28.87	33.13	33.09

Table B.4: Packet loss percentage for $\tau = 2\delta$ vs. τ yielding the lowest (Best) packet loss percentage. $\delta = 16$

τ (sec)	Exp. Set 1	Exp. Set 2	Exp. Set 3	Exp. Set 4
(2δ)	10.9	24.26	31.55	32.3
(Best)	10.9	23.69	30.43	31.21

Table B.5: Packet loss percentage for $\tau = 2\delta$ vs. τ yielding the lowest (Best) packet loss percentage. $\delta = 32$

τ (sec)	Exp. Set 1	Exp. Set 2	Exp. Set 3	Exp. Set 4
(2δ)	14.08	40.47	51.16	51.17
(Best)	12.99	41.43	51.16	51.17

Figure B.7: Tables illustrating that the difference in packet loss between experiments in which $\tau = 2\delta$ and those in which the lowest packet loss was achieved is negligible (within 2% for both physical and simulation platforms). Data shown in these tables was generated using the physical platform.

Table B.6: Packet loss percentage for $\tau = 2\delta$ vs. τ yielding the lowest (Best) packet loss percentage. $\delta = 2$

τ (sec)	Exp. Set 1	Exp. Set 2	Exp. Set 3	Exp. Set 4
(2δ)	7.59	2.68	5.25	4.17
(Best)	3.35	2.29	3.47	3.24

Table B.7: Packet loss percentage for $\tau = 2\delta$ vs. τ yielding the lowest (Best) packet loss percentage. $\delta = 4$

τ (sec)	Exp. Set 1	Exp. Set 2	Exp. Set 3	Exp. Set 4
(2δ)	6.96	5.29	7.45	6.45
(Best)	3.55	4.67	6.40	6.14

Table B.8: Packet loss percentage for $\tau = 2\delta$ vs. τ yielding the lowest (Best) packet loss percentage. $\delta = 8$

τ (sec)	Exp. Set 1	Exp. Set 2	Exp. Set 3	Exp. Set 4
(2δ)	7.48	10.23	14.99	13.9
(Best)	5.9	9.29	12.61	12.45

Table B.9: Packet loss percentage for $\tau = 2\delta$ vs. τ yielding the lowest (Best) packet loss percentage. $\delta = 16$

τ (sec)	Exp. Set 1	Exp. Set 2	Exp. Set 3	Exp. Set 4
(2δ)	11.09	22.99	31.13	30.11
(Best)	10.3	21.95	28.48	28.28

Table B.10: Packet loss percentage for $\tau = 2\delta$ vs. τ yielding the lowest (Best) packet loss percentage. $\delta = 32$

τ (sec)	Exp. Set 1	Exp. Set 2	Exp. Set 3	Exp. Set 4
(2δ)	20.13	42.29	53.80	54.87
(Best)	18.23	40.23	51.35	53.23

Figure B.8: Tables illustrating that the difference in packet loss between experiments in which $\tau = 2\delta$ and those in which the lowest packet loss was achieved is negligible (within 2% for both physical and simulation platforms). Data shown in these tables was generated using the simulation platform.

Curriculum Vitae

Felipe Jovel was born in Cd. Juarez, Chihuahua, Mexico. The second son of Maria and Carlos Jovel, he graduated from Socorro High School, Socorro, Texas, in the spring of 2007 and entered The University of Texas at El Paso in the fall. While pursuing a Bachelor of Science degree in Computer Science, he worked with Academic Technologies at UTEP, as a software developer from June 2007 to May 2013. In spring 2012, he obtained his Bachelor of Science degree in Computer Science, and in fall of the same year he entered Graduate School at The University of Texas at El Paso to pursue a Master's of Science degree in Computer Science. During this time, he left Academic Technologies to join the Hipersys research group at UTEP. In May 2013 he interned at the Army Research Laboratories, and currently he has a standing job offer from them.

Permanent Address: 10535 Elba Margarita Cir.
El Paso, Texas 79927
or
fjovel@miners.utep.edu