The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

IMPLEMENTATION OF AN EFFICIENT WIRELESS ROUTING PROTOCOL BASED ON SMALL END-TO-END SEQUENCE NUMBERS

A Thesis in

Computer Science and Engineering

by

Kiran A. Kashalkar

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

August 2009

The thesis of Kiran A. Kashalkar was reviewed and approved by the following:

John Metzner
Professor of Computer Science and Engineering
Thesis Co-Advisor

George Kesidis
Professor of Computer Science and Engineering
Thesis Co-Advisor

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School

ABSTRACT

Mobile Ad-Hoc Networks (MANETs) have evolved tremendously over the past few years. Today MANETs contain not only simple mobile routers but also those with the capability to perform complex functions like encryption/decryption. Even today, the most common form of a mobile ad-hoc network still remains as a network including a base station or a network of base stations which are responsible for reliable data communication to other devices in the network. Such a network may also employ helper nodes which may be specially designed to relay data alone or those nodes in the network that have their own data to send but have extra resources to help as relays. Energy is an important resource in a wireless ad-hoc network. Wireless routing protocols try to minimize wasteful operations so that energy is conserved wherever possible. The ability of such a routing protocol to change routes dynamically plays a big role in how efficient the protocol is.

In the protocol described here, routes change dynamically with feedback responses. The specific problem dealt with is communication to a base station or a network of multiple base stations, with the possible assistance of helper nodes. At any time, the route can change based on window information and sender desires. No special route establishment is needed. No end-to-end successfully acknowledged packets need to be re-sent.

It will be assumed that the network is of the wideband, low utilization type. This is a common situation with low energy, short range transmissions such as in Ultra-Wideband transmission or Wideband Aloha. Collisions would be a rarity, and the Automatic Repeat Request (ARQ) techniques employed can handle collisions when they occur.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am most grateful and indebted to my thesis advisers, Dr. George Kesidis and Dr. Metzner, for the guidance, patience and encouragement they have shown during my presence here at Penn State. I am especially grateful for the independence they have given me with my work and work style over the years. I thank my committee for their insightful commentary on my work.

I would also like to thank the Penn State University for giving me the push that I required as I had come to a stagnating period in my life and making me believe in myself and making me realize my worthiness.

I would like to dedicate this work to my family who stood by me in these times, offering all they could to help me realize the dream of getting a masters.

Finally, I would like to dedicate this work to the new addition to my family – my wife: Sara. She encouraged me a lot to complete this thesis.

CHAPTER 1

INTRODUCTION

With the growing popularity of Mobile Ad-hoc Networks (MANETs), several ad-hoc routing protocols have been developed [**1**]. The protocols can be categorized according to the way they operate as being proactive or reactive. Proactive protocols maintain fresh lists of destinations and their routes by periodically distributing routing tables through the network, whereas reactive protocols flood the network on demand when they have data to send. Proactive protocols try to achieve smaller latency time by keeping the network ready to send data whenever it is present by keeping all nodes informed about updates in the network topology. Reactive protocols try to minimize the amount of information passed in the network maintenance plane by not sending routing information when data does not need to be sent.

Due to their high overhead, proactive protocols haven't become as popular as reactive protocols yet. This is, however, changing with the higher performance and processing powers of the network devices that are now being manufactured. The most popular of reactive protocols are Ad-hoc On-demand Distance Vector (AODV) [**2**] and Dynamic Source Routing (DSR) [**3**]. Both try to achieve a low overhead and yet are able to react very quickly to changes in the network. On a broader perspective, both have two main mechanisms that help them to function – route discovery and route maintenance.

Most ad-hoc routing protocols along with the above two mentioned try to set up a route prior to some data transfer. Whenever a link fails, the route maintenance mechanism takes over. However, in most protocols, route maintenance is a subset of the route establishment mechanism. In MANETs, routes often need to be changed due to the dynamic nature of the network. It is generally difficult to change a route without interrupting reliable data transfer. Also, since route maintenance is almost always as expensive as route establishment, these protocols are hardly as efficient as desired in MANETs. Adequate sequence numbering is a problem as data might continue flowing from older routes causing sequence number ambiguity. Sending data over multiple routes for better assurance of delivery (reliable transfer) aggravates this problem with a great deal of out-of-order and multiple packet reception.

When order is preserved, sequence numbers of small size have been adequate to prevent ambiguity, as shown by the High-Level Data Link Control (HDLC) protocol [**4**] and early selective repeat protocols [**5**]. With the introduction of Internet datagram communication that allows out-of-order reception and long data paths, the Transmission Control Protocol (TCP) was introduced [**6**] at a higher layer to solve the problems of out-of-order transmission. TCP solved the problem of sequence number ambiguity in a brute force manner by using a large 32 bit sequence number. However, wireless communication often operates best with relatively smaller packets and having smaller sized sequence numbers would greatly increase efficiency thus.

In [**14**], Dr. Metzner defines the problem to solve relative to a special network topology where several nodes in a network exist and can collaborate to achieve transmission to a base station. This is a very common network scenario and can be augmented by the fact that the base station is capable of reaching all desired nodes in one transmission once it receives the data to be transmitted from the sender node. Such a topology is very popular in sensor and peer-to-peer (P2P) networks as well. Such networks tend to be small and dynamic. Certain nodes may be dedicated originators of data while some may be dedicated relay nodes. Nodes can also change their role based on how much surcharge energy they have to assist other nodes in sending data. The protocol as formulated in [**14**] is described in more detail in Section 3.3 of this thesis.

The problem we thus formulate in this thesis is to implement the aforementioned protocol for this network type but not limited to the same for sending packets efficiently end-to-end. The use of different acknowledgement strategies can then be studied in conjunction with the protocol in simulations to determine if it would further improve efficiency. Implementation strategy and decisions will be discussed in Section 3.4.

The implementation details of the protocol will be discussed in Chapter **4** and finally we will close with future work and possible extensions in Chapter **5**.

CHAPTER 2

BACKGROUND

Wireless ad-hoc networks have been around for several years from their inception in the 1970s. The earliest wireless ad hoc networks were the packet radio networks (PRNETs), sponsored by Defense Advanced Research Projects Agency (DARPA) [**9**] after the ALOHANet project [**10**]. Initially, several researchers tried to leverage techniques applicable to wired networks into wireless networks. Although they were successful, they were hit with several problems due to the nature of the physical medium over which wireless networks operated. Lately, this has changed and research in wireless networks has evolved as an almost exclusive branch in the networking field. This has triggered the evolution of standards such as 802.11 [**22**], Wi-Fi Protected Access (WPA) [**23**] and routing protocols such as AODV and DSR. Apart from the popular routing protocols, researchers have been developing newer protocols, trying to improve wireless standards on the efficiency, security and speed. These three factors are not mutually exclusive and hence researchers have been trying to reach an optimized algorithm for routing in the wireless world. Researchers have often been challenged by the dynamic nature of this physical medium as the wireless channel is a very fast-evolving technology. Given this, most algorithmic standards have been developed at the network or higher layers for routing.

MANETs were developed by the global research community and military as a way to simplify initialization and use of wireless networks. This brought distributed algorithms to the wireless world as routing now became a responsibility of all nodes participating in the wireless network. Several routing protocols have been developed for MANETs of which we will discuss a few in the following paragraphs.

DSR is one such protocol that has been actively developed by companies and the academic community alike. DSR runs as a distributed algorithm in a wireless network with the source determining the complete source that a packet traverses to reach its destination. Although very easy to conceptualize, DSR is a very complex routing protocol and it does not deal well with several issues such as packet loss, node crashes, congestion and long routes. DSR needs flooding so that nodes have complete information about the topography of the network and sources are able to determine the route when sending a packet. Thus there is a large startup overhead as well. Also, when completely aware, DSR sends entire path in the packet yielding to low efficiency. These overheads also affect the scalability of DSR. Researchers have actively tried to overcome these problems in DSR, e.g. “flow-state transmission” in DSR [**24**] eliminates the need for packets to contain the entire path once a flow of data has been established in a well-connected network.

To overcome the inefficiencies of DSR, another protocol known as AODV was developed. AODV used sequence numbers to build loop free routes. The key difference from DSR is that source-filled path is not required yielding very high data efficiency. Path discovery and routing table population was distributed in forward and reverse path

setup mechanisms. However, AODV does not consider individual node resources in setup and does not use end-to-end sequence numbers. Several protocols tried to overcome these shortcomings of AODV by adding performance metrics in the protocol that determine which nodes are selected in routing. Better Approach To Mobile Adhoc Networking (B.A.T.M.A.N.) [**25**] is one such protocol. However, B.A.T.M.A.N. does not use end-to-end sequence numbers. Implicit Source routing tries to preserve the advantages of source routing while avoiding the associated per-packet overhead in most cases [**26**].

Most protocols try to find the shortest path between source and destination. However, this has shown to be not enough for efficient and reliable data transfer when it comes to wireless ad-hoc networks [**27**].

In the protocol described in this thesis, the use of end-to-end sequence numbers along with the use of a performance metric that will be based on the resource availability of participating nodes has been conceptualized [**34**]. Based on whether a node participates as a helper node and the amount of resources it has, it can distribute its resources dynamically to result in highly efficient network-wide transfers. Intermediary nodes can vary their function by acting as fast nodes for a few flows or allowing several flows through them slowly to keep the routes at maximum throughput. Such a protocol also facilitates load balancing in the network.

Work was undertaken as part of this thesis to implement the protocol so that it could be analyzed for its performance and compared against the other protocols. There were two ways to do this:

i. Implement the protocol and deploy it along with other protocols in similar hardware network configurations to get results and compare them.

ii. Implement the protocol in a simulator that already had implementations of other wireless ad-hoc protocols so that it could be compared with already present test cases.

As with most academic projects, the second option was chosen. There was a wide variety of simulator options available for this; the most popular ones being NS-2, OMNet++ and OPNET IT Guru Academic Edition [**28**].

All of these are discrete event simulation systems. Of these OMNet++ and NS-2 are both free and popular amongst the developer community. Both have implementations of AODV and DSR. However, OMNet++ was a budding system at the time of implementation and NS-2 was a well established leader with contributions from several reputed researchers. Hence, NS-2 was chosen for implementing the protocol.

The following chapter will describe the protocol in detail.

CHAPTER 3

PROTOCOL DESCRIPTION

3.1 Network Topology

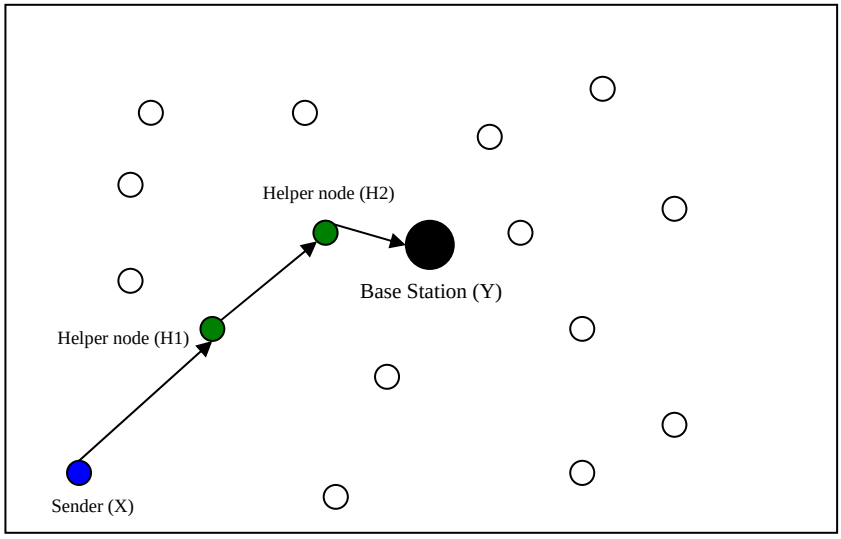


Fig. 1: Typical network topology

As shown in Fig. 1, the typical network topology consists of several nodes in a given area. The nodes can themselves be originators of data (depicted as white or blue in the figure) or helper nodes (depicted as solid green in the figure). Originator nodes that have extra resources such as extra energy and/or extra bandwidth may opt to act as helper

nodes to help other originator nodes get their data to the base station. The base station is a special type of node (depicted as solid black in the figure) that typically has resources enough to broadcast or multicast data to nodes in the entire network. Please note we do not make any special assumptions for the purpose of such a base station, but such topologies are used massively in sensor and mobile networks, and also some P2P networks.

3.2 Assumptions about the system

Specific to the description of the protocol described in the next section, the following assumptions are made [**14**]:

Assume station X wishes to communicate to the base station Y. Let X have an address IDX, Y have an address IDY. Helper nodes may exist in between that help X to transmit data to Y. Let each such helper node be H#, where # is the hop index from the sender X. In Fig. 1 above, there are 2 helper nodes, H1 and H2, that help X to get its data to station Y. Routes can change anytime based on several conditions and no special route establishment needs to be done prior to data transfer. Ranges are generally short and energy is the premium resource. Collisions between sending nodes can be minimized by using ultra wide-band, low energy transmissions without careful coordination and the use of protocols such as Multiple Access with Collision Avoidance for Wireless (MACAW) [**7**] [**8**]. Limited collision avoidance can be employed. For example, if a nearby sender is

heard sending a train of pulses, in ultra-wideband case, the transmitted timing can be adjusted to avoid overlap. In a multi-band network, frequencies overheard from neighbors can be avoided for sending data.

3.3 The Protocol

3.3.1 Assumptions about the protocol

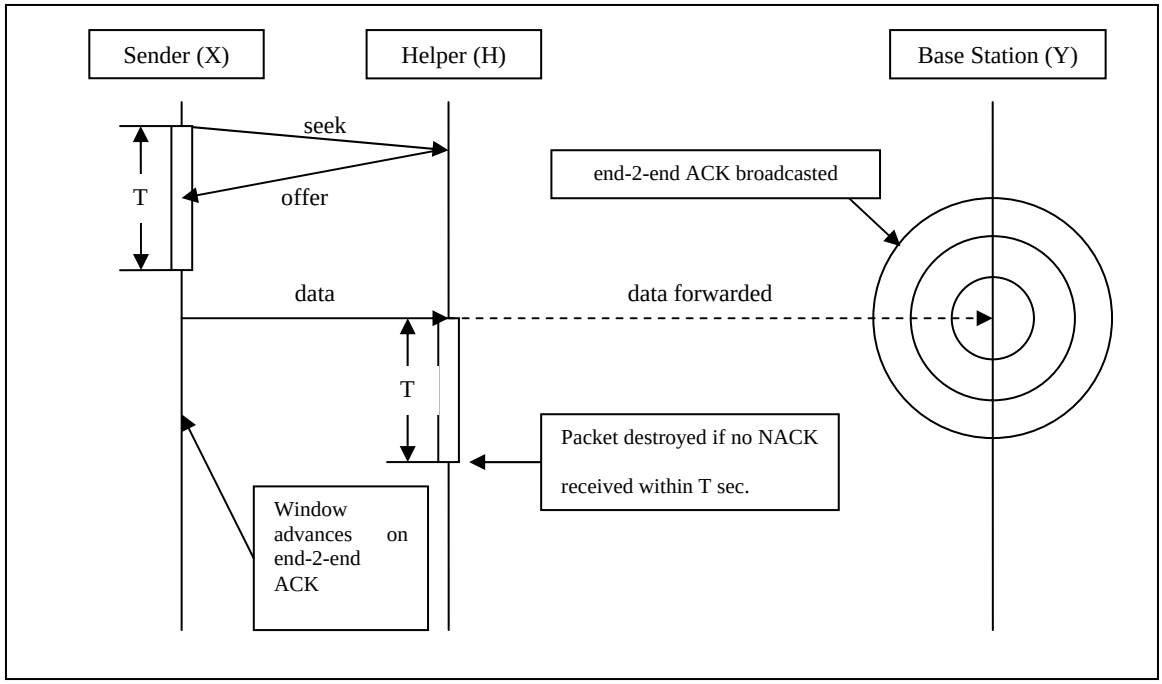


Fig. 2: Message Diagram

X initially sends a “seek” packet indicating its own ID: IDX; and the ID of the destination node: IDY. If Y responds directly favorably, X can send packets directly to Y.

However, in the other case, if X does not hear a direct reply from Y, it may still hear responses from other helper nodes that are willing to forward the data to Y. These packets are known as “offer” packets and along with the replier’s ID, they consist of other information that may help the seeker to make a decision so as to which helper node to forward data through to the destination. This information may include no. of hops to destination information (if the helper node has this information) and/or the available bandwidth that the helper node has to spare for this flow of data. The seeker on receiving this information picks up the most favorable helper node and starts sending its packets to this node by including the helper node’s ID in the packet. An efficient way to include the relay ID would be to use multicast addresses. However, this would involve hierarchical addressing that can be done only in the case of static networks. We can imagine having a static network of helpers and a base station, but this is not included in this work. Another option would be to add the addresses and send the data. Only the chosen helper node knows both addresses and can subtract the same from the packet before sending it further. X can send a new “seek” packet seeking Y at anytime during the transmission to continue with its transmission on another route. Alternatively, X can split its traffic, sending some packets on one route and some on the other. Since X has control over the sequence numbers, it has complete discretion to do this at anytime without affecting the flow of traffic. Multiple packets at the destination with the same sequence number will simply be discarded. In the next section we will prove a limit on the sequence number range that we will use to avoid ambiguity at the destination.

The helper node specifications in the offer will be a “window length” that specifies the number of packets that the helper node is willing to handle in some specific time T. T is a system constant. The window length is a system variable and can change with improvement in the helper node algorithm or hardware. With a static hardware configuration, it varies with no. of hops to the destination and most importantly with the helper node’s available capacity and/or energy reserves.

For the sake of completeness, we should also define acknowledgement packets of 2 kinds here. We use end-to-end acknowledgements with hop-by-hop NACKs (negative acknowledgements) as we assume that the base station (destination) is capable of broadcasting to all nodes in one transmission.

Also, apart from data itself, data packets must include the originator ID, destination ID, relay ID (if any) and a sequence number. Although this is a premium to pay for this protocol, it’s a very small one considering that the nodes themselves need not hold a lot of routing data.

3.3.2 Description

There are 3 parts that the protocol we describe employs:

i. A small sequence number based on a window limitation of a maximum of W packets in T seconds.

ii. A discarding policy that an intermediate station cannot send any packet out more than T seconds after it has been received, which means all packets T seconds old or older are discarded.

iii. A limit on the number of hops.

Suppose the source has a maximum window of W packets allowed to be transmitted in time T, starting from the oldest unacknowledged packet. We wish to relate this to the minimum number of bits in the sequence number that prevents ambiguity.

The source may switch routes or may even send identical copies over different routes. On each route, assume hop-by-hop NACKs are used, with end-to-end cumulative acknowledgments from Y. Hop-by-hop ACKs are optional. The ambiguity problem has a well known solution for cases where all packets follow the same route and arrive in the order transmitted. But in the route switching and parallelism environment out-of-order arrivals violate the assumptions of these relationships. The method is designed for systems where the number of hops is small and the distances are short, which is appropriate for wireless communication with base stations.

Assume that no route has more than H hops. Also assume that the source will not inject more than W different packets into the network in any T-second interval. This corresponds to a maximum packet rate of W/T packets per second.

Every packet received by a non-source node that is not the destination (intermediate node) is allowed to be held for at most T seconds starting from its most recent reception of the packet. If a hop-by-hop NACK is received, it can retransmit if T seconds have not expired. This is a simple way of applying a time-to-live feature that helps solve the ambiguity problem. The value T may be a standard system parameter or negotiated. In our case, we will assume T is a system parameter based on the speed permissible by the underlying wireless protocol. The sender can use a smaller window than W, and a lower packet rate, with lesser concern about ambiguity problems, since fewer packets could be delivered in T seconds. But with changing conditions and routes, the source may wish to change to a faster rate, where as many as W packets actually would be delivered in T seconds. The required minimum number of bits in the sequence number depends logarithmically on the maximum number H of hops but H would almost always be small in wireless communication to a base station.

The source knows an intermediate node h hops from source can not be holding a copy of the packet for retransmission at hT seconds after the source last injected. To allow for either go-back or selective repeat transmission, assume the receiver is allowed to accept any new packet in W_R positions starting from first packet not yet correctly received. $W_R = 1$ is the go-back protocol.

3.3.3 Theorem

Assume propagation time is negligible compared to packet time. The number of bits required in the sequence number is at least $\log_2 |HW + W_R|$.

3.3.4 Proof

Consider first H = 2. Figure 1 shows the receiver viewpoint, knowing the sender's window ambiguity limit is W.

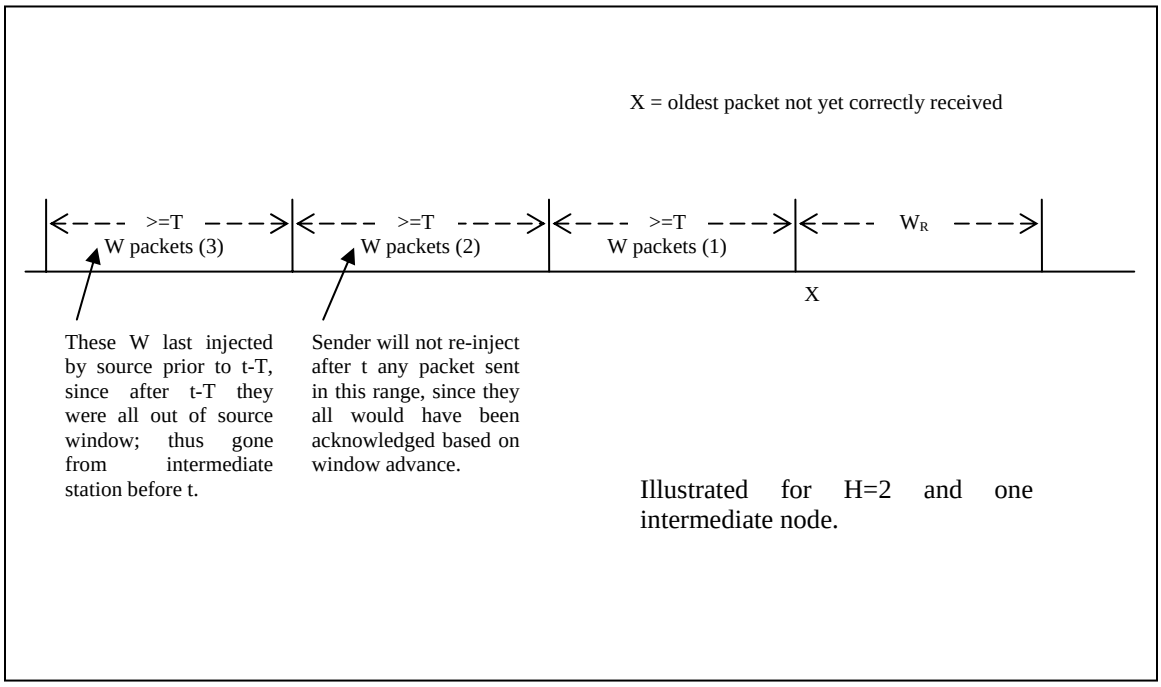


Fig. 3: Receiver's perspective. H=2 [**14**].

Refer Fig. 2. Here, H = 2. For this example, after time t, the only prior packets that could be received at the destination from any path are the 2W most recently sent packets. This is in addition to W_R possible new packets.

If H = 3, the source will not re-inject after t-2T any packet prior to interval 3 since a full window of packets has been sent in interval 3, before t-2T. The second intermediate node could receive this injection up to t-T, and the destination up to t. An interval 3 packet as late as t-T, which could arrive at the next intermediate node as late as t, and the destination as late as t+T, so interval 3 must be included in the possible received range. Thus the widest range of reception is $3W + W_R$.

By a repeat of this argument for each additional hop, the general range of possible receptions is $HW + W_R$. This range cannot exceed 2^b, where b is the sequence number bit size. Hence proved.

Since $W_R \leq W$, a bound for any amount of selective repeat is $2^b \geq (H+1)W$. Doubling H adds at most 1 bit to the required sequence number size. Also, a large number of hops are not desirable, and one might use a lower W and lower bit rate for such routes. Applying the method to communication to a base station or a network of cooperating base stations allows very efficient, adaptive communication. If a route gives fast rates and acknowledgments, stick to that route. As soon as a route starts to give problems, seek an alternate route. For fault tolerance, hold onto two routes. With

cooperating base stations having a common destination address, different routes might go to different base stations, often without intermediate stations.

The next chapter will go into details of the implementation. We will discuss the different headers we used for differentiating packets, the implementation details of the protocol agent and the algorithms with the help of flowcharts of the implementation.

CHAPTER 4

PROTOCOL IMPLEMENTATION

4.1 Implementation decisions.

We studied two simulators before implementation of the protocol – OMNet++ [**15**] and Network Simulator 2 (NS-2) [**11**]. Both are discrete event simulation systems. While OMNet++ is rapidly gathering momentum, at the time of implementation of this protocol, it was neither the most popular nor did it have a huge community of developers to ask about technical difficulties with.

We ultimately chose NS-2 was as the simulation tool for implementing both our protocol and test scripts to evaluate the performance of the protocol. NS-2 is a discrete event simulator targeted at networking research. NS-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. NS-2 was chosen as it allows one to both implement their own protocols and write scripts to test these protocols. NS-2 is written in C++ and object oriented version of Tcl called OTcl. NS-2 also allows tracing of events based on time for exhaustive evaluation. With graph plotting additions like “trace graph” [**12**], one can convert traces to graphs. NS-2 also comes with an optional add-on Network Animator (Nam) [**13**], which is a Tcl/TK based animation tool for viewing network simulation

traces and real world packet traces. It supports topology layout, packet level animation, and various data inspection tools.

NS-2 was especially advantageous as it already has several implemented routing algorithms, which can be used for studying the implementation techniques and modified to accommodate the requirements of our protocol. The Ad-hoc On-demand Distance Vector (AODV) [**2**] and Dynamic Source Routing (DSR) [**3**] protocol implementation were the primary ones studied among other protocols to be viably modified to implement our protocol. DSR was chosen because of the fact that it already had headers that could be easily modified and ported to the implementation of our protocol. AODV and other protocols implemented in NS-2 were not suitable to be modified. However, several more items required modification in the current DSR implementation to get our protocol up and working.

Scripts for running simulations in NS-2 can be written in OTcl, which is an extended version of Tcl with support for objects.

In the following sections, we will define the different kinds of packets we will require to be part of our protocol. We will then go on to details of the implementation and experiences in implementing the protocol. Finally we will discuss about the scenarios that can be used to evaluate our protocol.

4.2 Packet Formats.

As our protocol is generic, we define all packets as a custom header followed by the actual data (see Fig. 4).

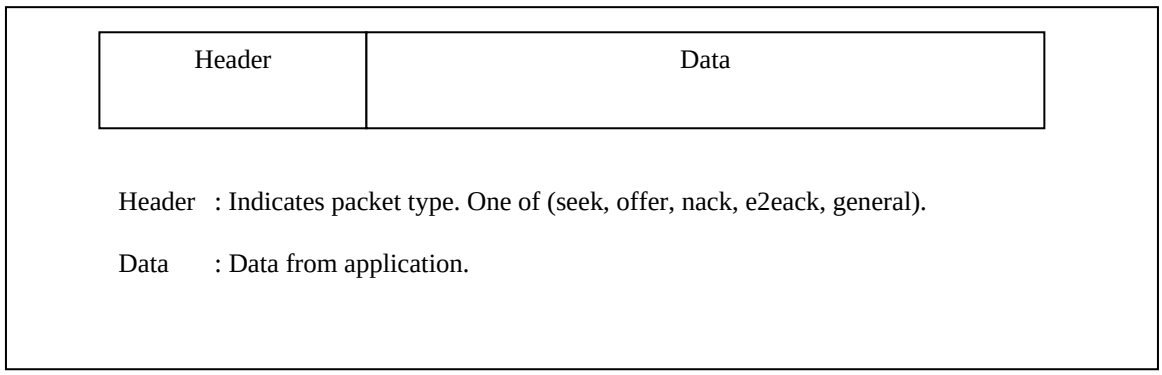


Fig. 4: Generic packet format

There are 5 different kinds of headers we use:

i. seek

ii. offer

iii. nack

iv. e2eack

v. general

Our current implementation doesn't send data along with seek, offer, nack, e2eack type headers. That is application data is sent only with the general header as the packet

header. The topic of chaining headers is a subject of future work that can benefit this protocol.

4.2.1 The “seek” header.

The seek packet is the first packet sent to initiate any transmission by the sender. The sender includes its own ID and the ID of the node it seeks in the header for such kind of a packet. A receiver that has enough resources may decide to forward this packet to help the sender and so it may require retransmitting this packet with the sender ID as its own ID. However, for the destination to know which node this packet originated from, the intermediate node has to copy the original sender’s ID in the header as well. Thus there is an additional field of “original_seeker_id” in the seek packet header. The original sender replicates its ID in both the seeker_id and original_seeker_id fields and intermediate nodes only update the seeker_id field if they need to send this packet forward. We use an internal field known as “header_type” to indicate the packet type to our protocol handlers. This value is set to “seek_valid” for all seek packets. The header format for this type of packet is shown in Fig. 5.

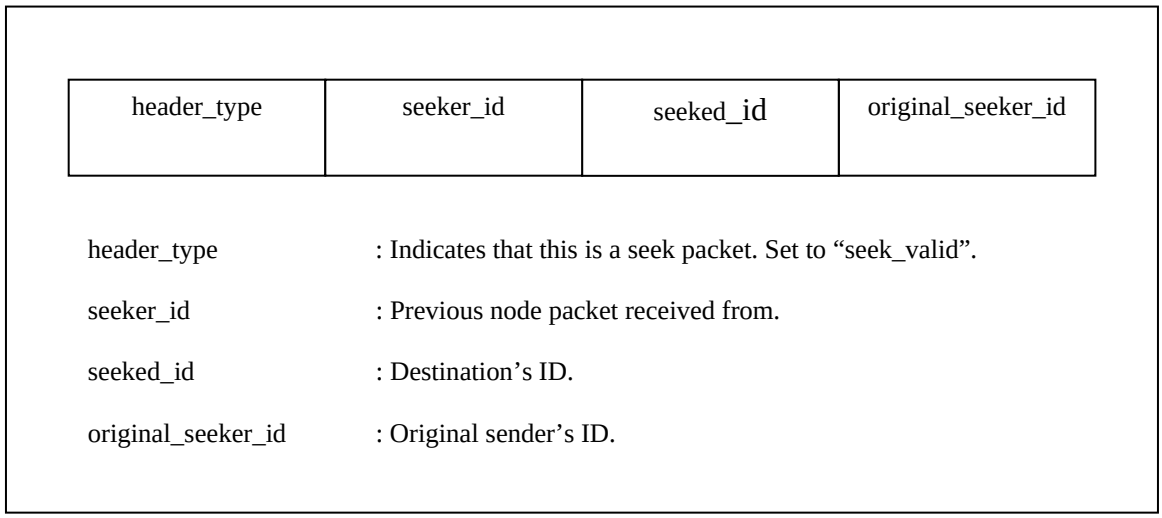
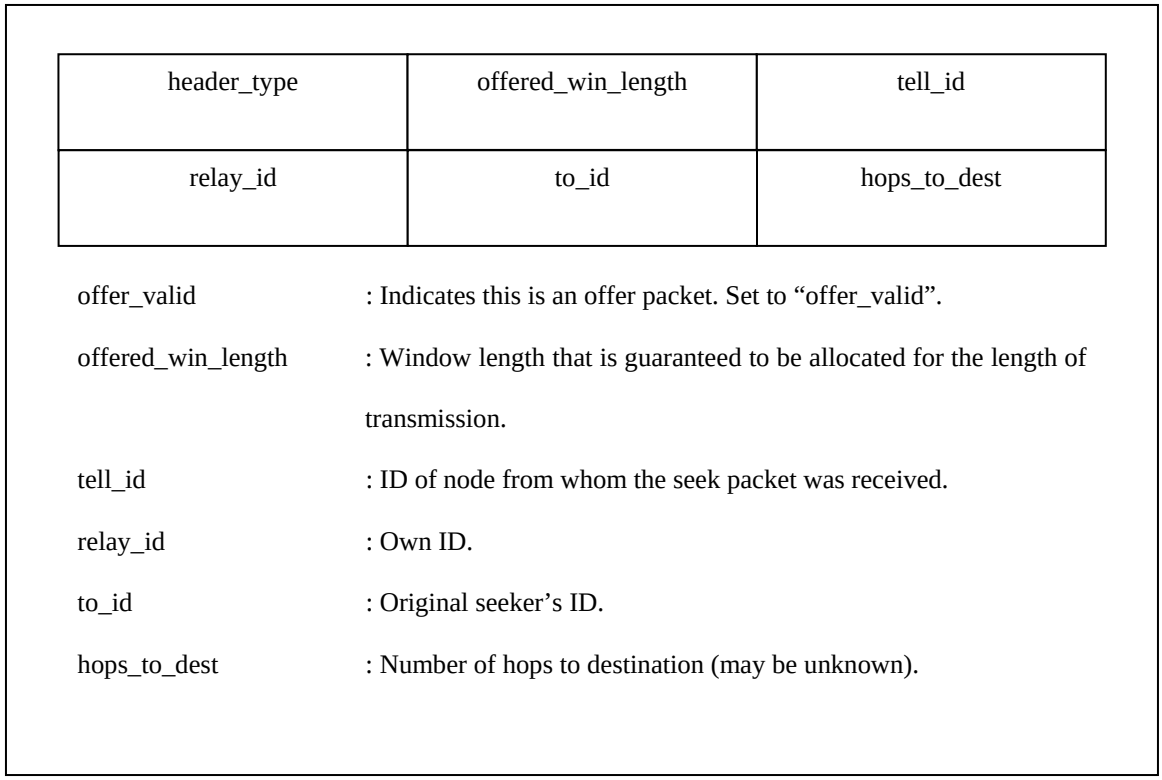


Fig. 5: The seek header

4.2.2 The “offer” header.

The offer packet is the packet is sent by prospective relay nodes that have enough resources to forward data to the seeked_id. However, the decision of which relay to use is solely left to the sender of data and hence all offer packets have information that help the sender determine which node it should use as a relay. The information may thus include no. of hops to destination, available power that the relay can allocate for this transmission, etc. We simplify power availability and other factors in one simple parameter: window size. The window size is thus the window that the relay can allocate for this transmission. The relay also sends the no. of hops to destination from its previous knowledge. As the offer itself may respond to two kinds of seek packets: one directly received from originator or one relayed on by other intermediate nodes, we require three

identifiers in the offer packet: the ID of the node that the seek packet came from, the ID of the node that the seek packet originated from and its own ID. We should also note at this point that the offer packet may also propagate all the way back to the originator and the intermediary ID values may change. The offer header thus looks like Fig. 6.



header_type	offered_win_length	tell_id
relay_id	to_id	hops_to_dest

offer_valid : Indicates this is an offer packet. Set to “offer_valid”.

offered_win_length : Window length that is guaranteed to be allocated for the length of transmission.

tell_id : ID of node from whom the seek packet was received.

relay_id : Own ID.

to_id : Original seeker’s ID.

hops_to_dest : Number of hops to destination (may be unknown).

Fig. 6: The offer packet

4.2.3 The “nack” header.

A nack (negative acknowledgement) is sent on a hop-by-hop basis in our protocol so that incorrect packets do not propagate in the network. Upon receiving a nack, the

previous node resends the packet if it still has it in its buffer/window. If not, it may either send back this nack to the previous node it received the packet from or drop the nack. Our implementation decides to drop packets that aren't in the buffer. Extensions may be written later that will have a more complete nack solution. The most important field to note in a nack packet is the "nacked_pkt_seq_num". This field identifies the packet that was received in error and that should be resent by the previous node(s). The nack header thus looks like Fig. 7.

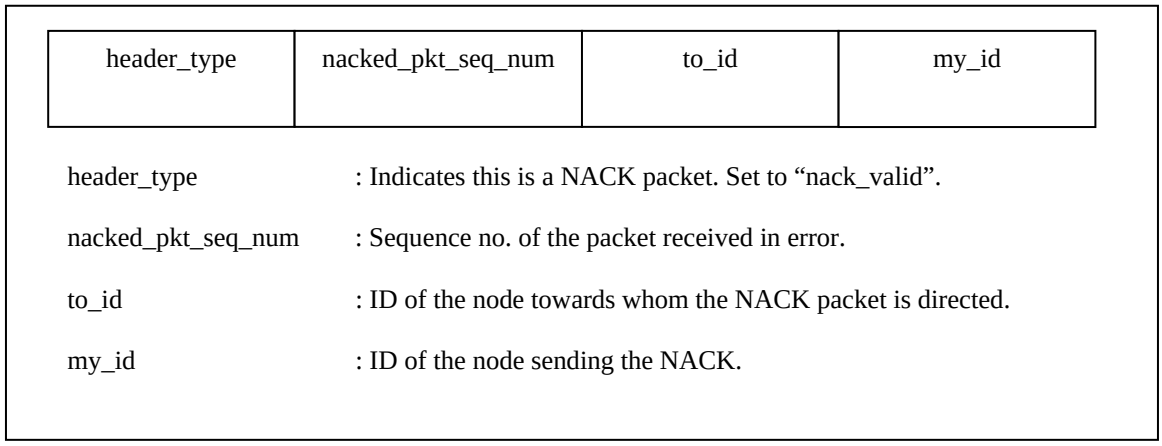


Fig. 7: The nack packet

4.2.4 The "e2eack" header.

An end-to-end ACK (acknowledgment) packet is sent by the base station once it receives a packet destined for it. In our implementation we assume that the base station has enough power capability to send the e2eack packet directly to the originator. This is

true in real world scenarios wherein base stations are usually dedicated antennas with large resources available to handle thousands of connections if not millions. The e2eack packet acknowledges a packet by having the sequence number of the same in its header. The e2eack is thus as Fig. 8.

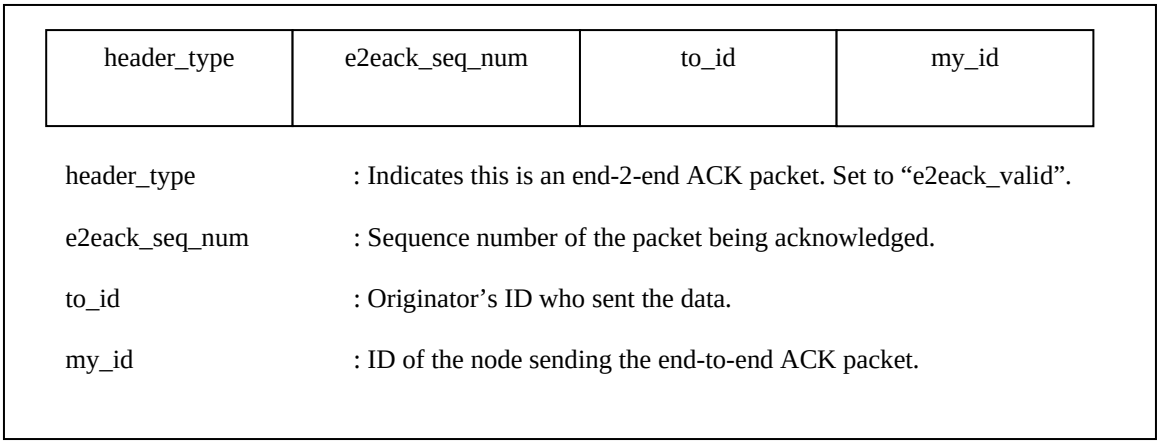


Fig. 8: The e2eack packet

4.2.5 The “general” header.

The general header is pre-pended to all data packets and amongst other information contains the end-to-end sequence number of the packet that is being transmitted. The other important field in this header is the selected window length. This is to incorporate a three way handshake after an offer is received and a relay node is selected by the originator of data. The general header also contains ID’s of the originator, the intermediary relay node and the destination node. The general header thus looks like Fig. 9.

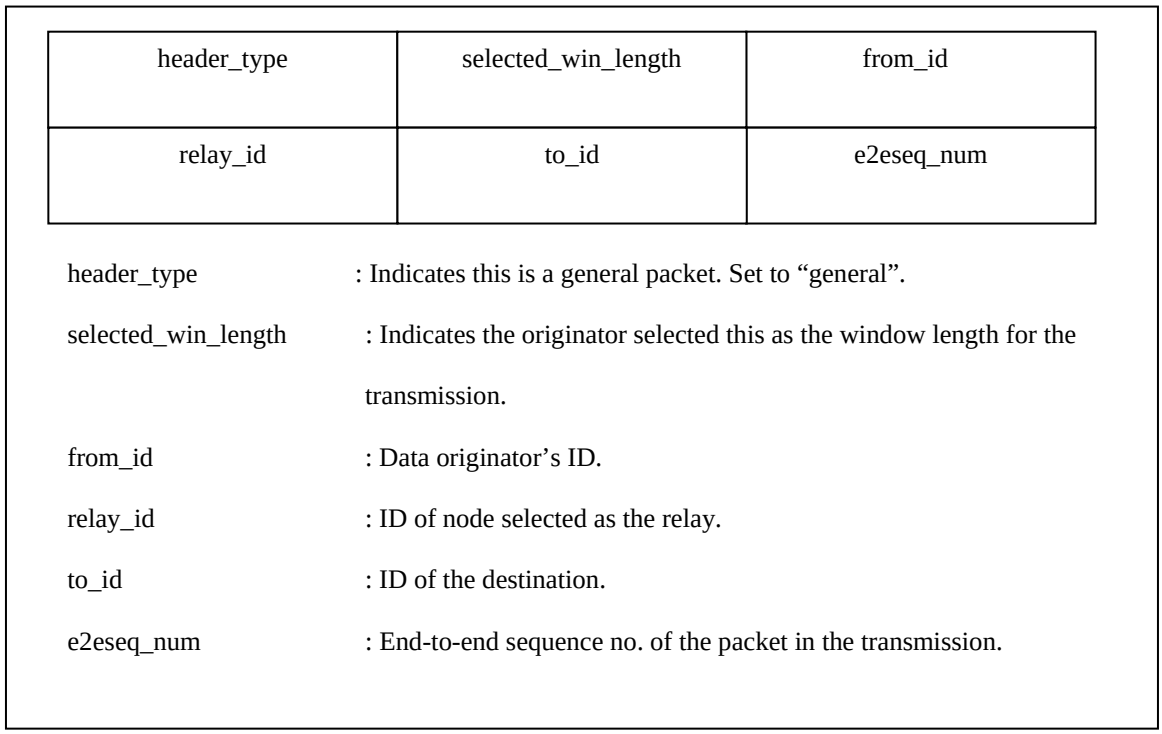


Fig. 9: The general header

4.3 Protocol implementation in NS-2

Our protocol is designed to fit at the Network layer in the Open Systems Interconnection (OSI) Reference Model [**16**]. It is implemented as an object that runs as the routing agent in a network node.

4.3.1 Node implementation

Before we go into the details of the routing agent, let's first understand how a node looks like in NS-2.

A node in NS-2 is implemented as a composition of several objects that make it emulate a real-world network object. At a minimum the node consists of an address or ID to uniquely distinguish it from other objects in NS-2 and a list of neighbors. The other objects in a node are a list of agents, a node type identifier, an address classifier and a port classifier. Fig. 10 shows the schematic of a NS-2 unicast node. Our protocol is implemented as an agent that resides inside every node of our specific routing agent type. Once we define a node with our protocol identifier in the test scripts, it uses our routing agent as the routing protocol. Although several protocol agents can be active at the same time within a node, we do not test interworking with other protocols and that may be a topic of future study.

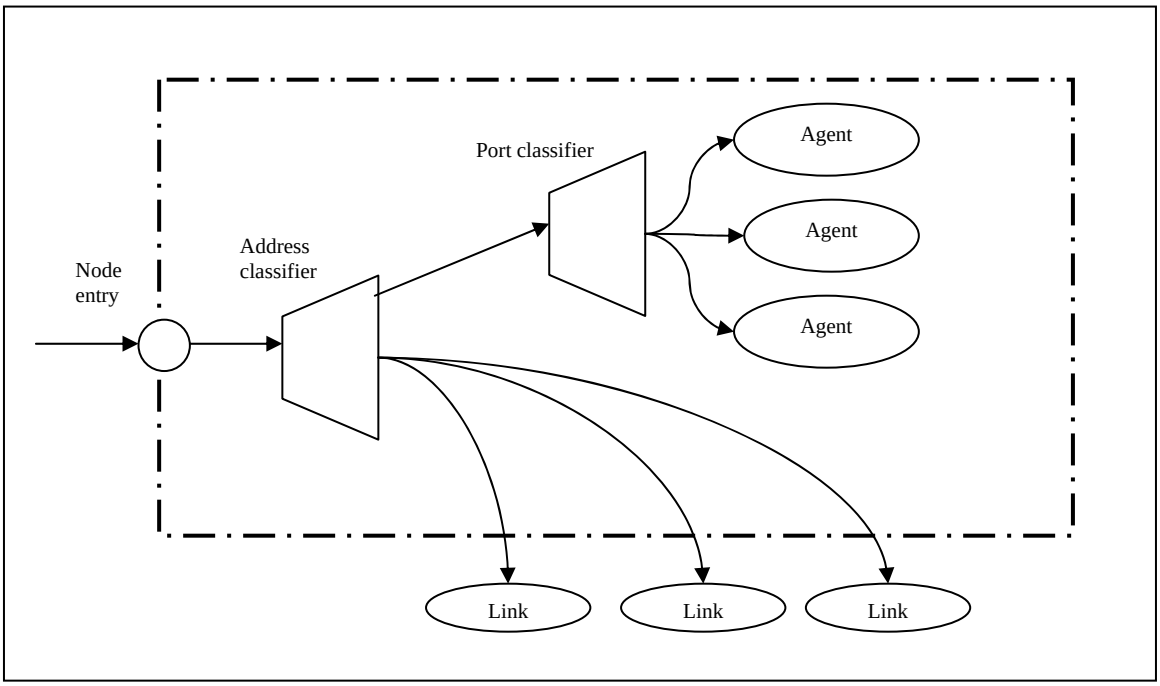


Fig. 10: NS-2 unicast node [**17**]

4.3.2 Routing Agent implementation

The routing agent resides in every node and acts as a super-packet handler through which all packets are passed through. The agent itself interfaces with the NS-2 logging system, the Media Access Control (MAC) layer, the link layer and the interface queue used for internal event prioritization. The log interface is used for gathering log traces that can be used for analysis of test scenarios. Having interfaces to the MAC and link layer as pointers allows a routing agent to use different MAC layer protocols and link layer characteristics. Although we have stated that our protocol would be ideal for a wideband ALOHA type MAC channel, at the time of our protocol implementation a

stable implementation of such a MAC layer protocol does not exist in NS-2 and so we use the default MAC implementation as provided by NS-2, which has been provided by developers at Carnegie Melon University (CMU). The implementation of wideband ALOHA falls outside the scope of this thesis work. The agent also has offsets to the different headers inside of a packet. For our protocol, these are the offsets to the link layer header, the MAC layer header, the IP layer header and our protocol header (srheader). We have named the agent as E2ESeqNumAgent. Table 1 below shows the fields in E2ESeqNumAgent.

Table 1: Routing Agent fields.

Field	Description
Logtarget	The target to log statistics at
Offset_mac	The offset of the mac header in the packet
Offset_ll	The offset of the link-layer header in the packet
Offset_ip	The offset of the ip header in the packet
Offset_sr	The offset of the header to identify our protocol is being used in the packet
net_id	Net layer address (IP layer address)
mac_id	Mac layer address (unique ID if used)
ra_addr_	Unique ID of this node
Ll	Object of link layer protocol being used
Mac	Object of MAC protocol being used
Ifq	Queue object of the queue in use at link layer
MobileNode * node	Flag to determine this is a mobile node. To be used in test scripts
port_dmux	This node's port demuxifier that determines where the packet goes to within after processing
route_table	Small single deep cache that keeps reachability data to a node. Currently setup via scripts in ObjectTcl.
send_window[MAX_WINDOWS]	Our small receive/send window
send_buf_timer	Timer object that does processing every interval – the interval is very very small for realtime simulation, but also depends on the PC's capability right now
Link	NS specific
agent_head	NS specific

In NS-2, every packet received at a node results in a call to the "recv" method of the agent. The packet can then be analyzed and handled appropriately to implement one's routing protocol. We thus overload the default recv method for an agent such that our recv method handles routing for nodes that are defined to use our protocol as the routing protocol. This definition as mentioned earlier is made in test scripts that generate network scenarios for simulation. The overloaded recv() function extracts the generic "srheader" from the packet, analyzes it and hands it down to one of the packet handlers. Every header-type has its own packet handler, viz. seekPktHandling, offerPktHandling,

nackPktHandling, nackPktHandling, e2eackPktHandling, handleGeneral. We will discuss these in detail in the following sections. Fig. 11 and Fig. 12 show the implementation of the overloaded recv() method using a flowchart representation.

4.3.3 Overloaded recv method

As shown in the flowchart in Fig. 11 and Fig. 12, the recv method contains the main decision making logic for the packet routing within a node. Based on the combination of flags in the header, the recv method makes the following decisions:

i. If the packet has been originated by an application tied to this node using our protocol?

ii. If the packet is a broadcast packet?

iii. If the node is a designated receiver of the packet (as a helper or intermediary node)?

iv. If the node is the final designated receiver of the packet?

v. What specific type of header the packet has or needs and which packet handling method must it be handed to?

The packet is either dropped, consumed by the application designated to receive the packet or handed down to one of the specific header packet handlers for further processing.

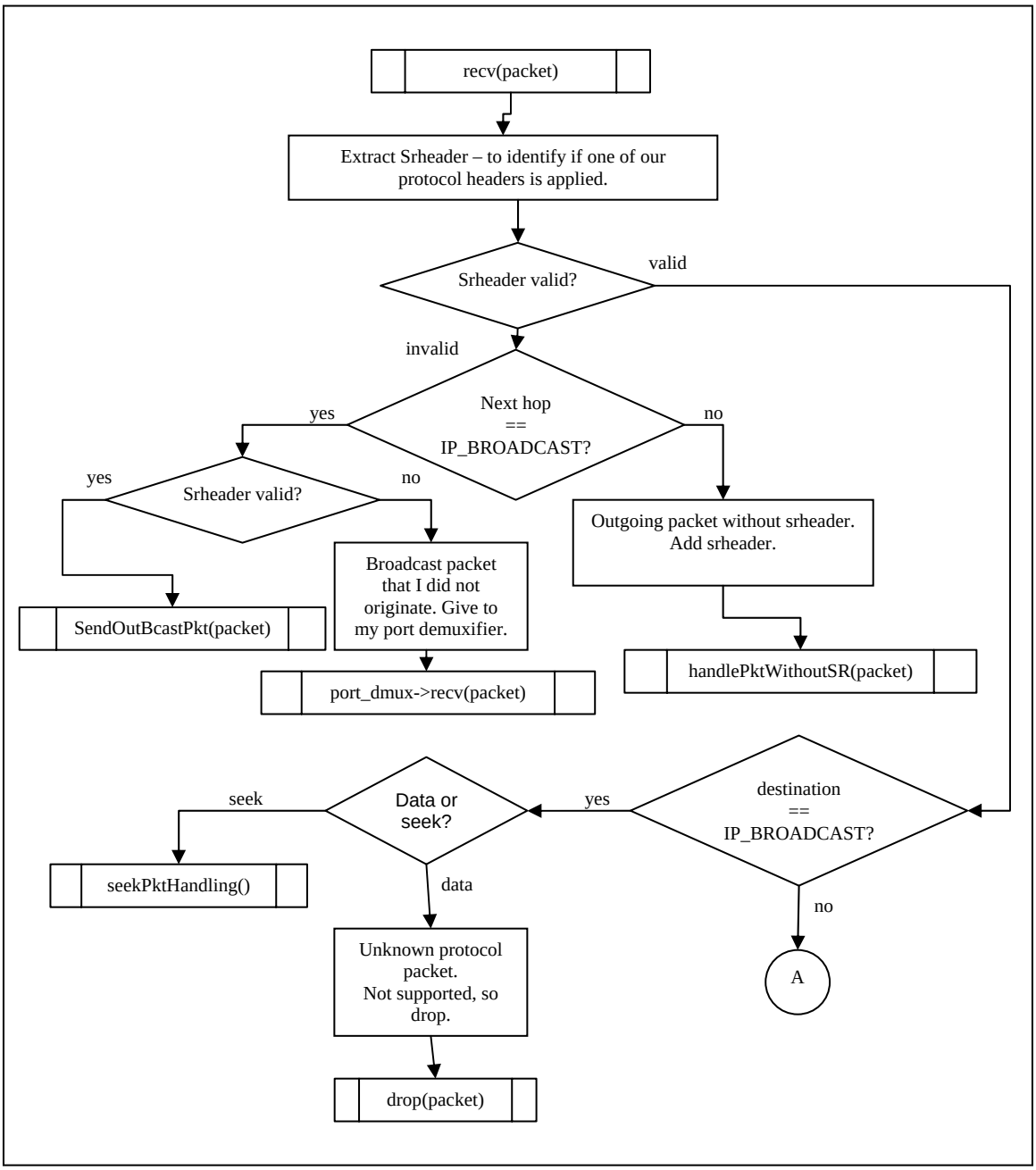


Fig. 11: Overloaded recv method flowchart

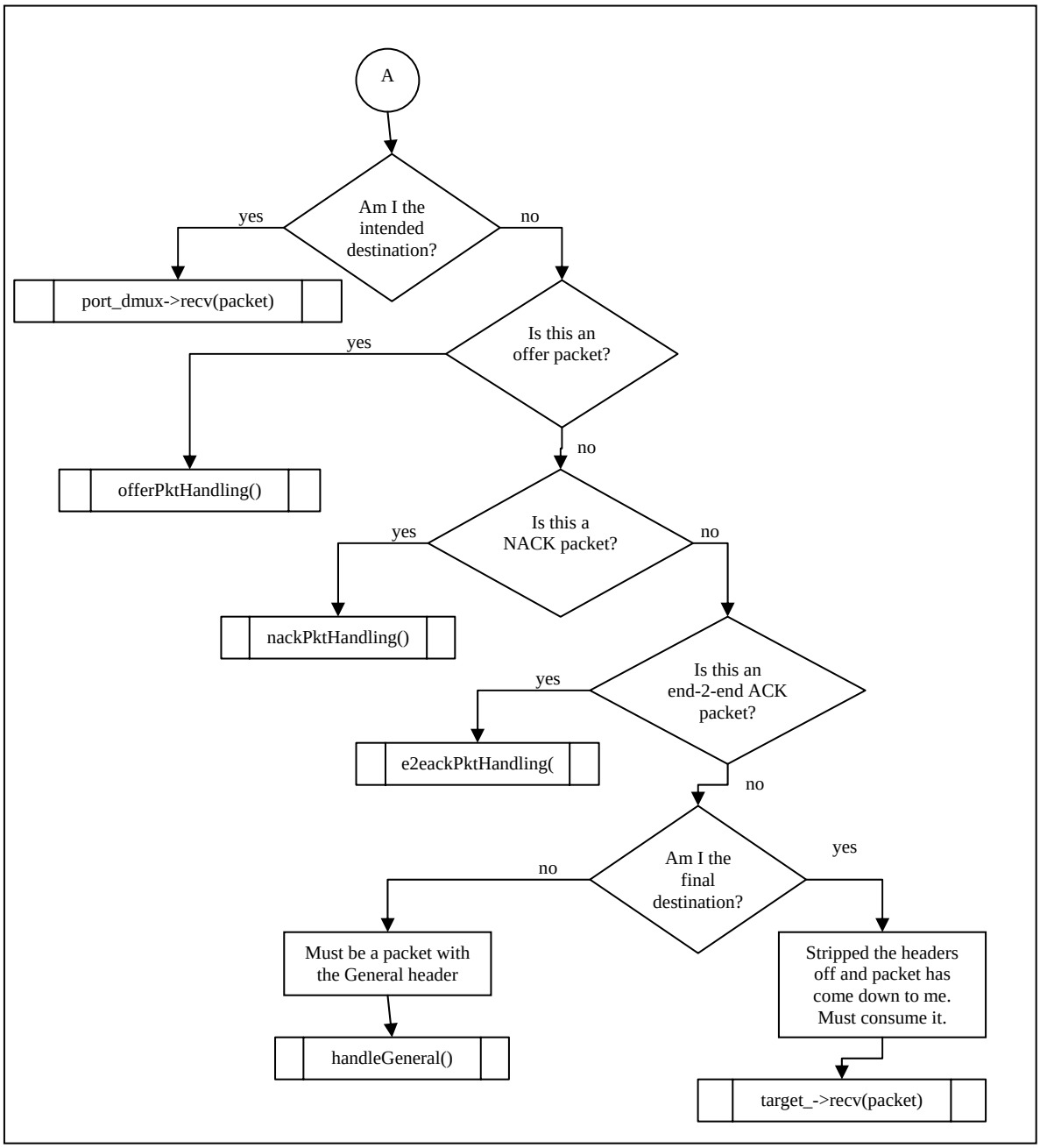


Fig. 12: Overloaded recv method flowchart (contd.)

4.3.4 seek header packet handling

The seekPktHandling method is responsible for handling packets with the seek header. Seek packet handling consists of adding the sender node's information in the local cache and looking through the local cache for determining whether the seeked ID (final destination) is reachable by this node or not. Based on reachability to the final destination and the currently available resources, the node decides how many resources it can allocate the seeker and sends these as part of the offer header in its reply. The processing applied by seekPktHandling method can be seen in Fig. 13.

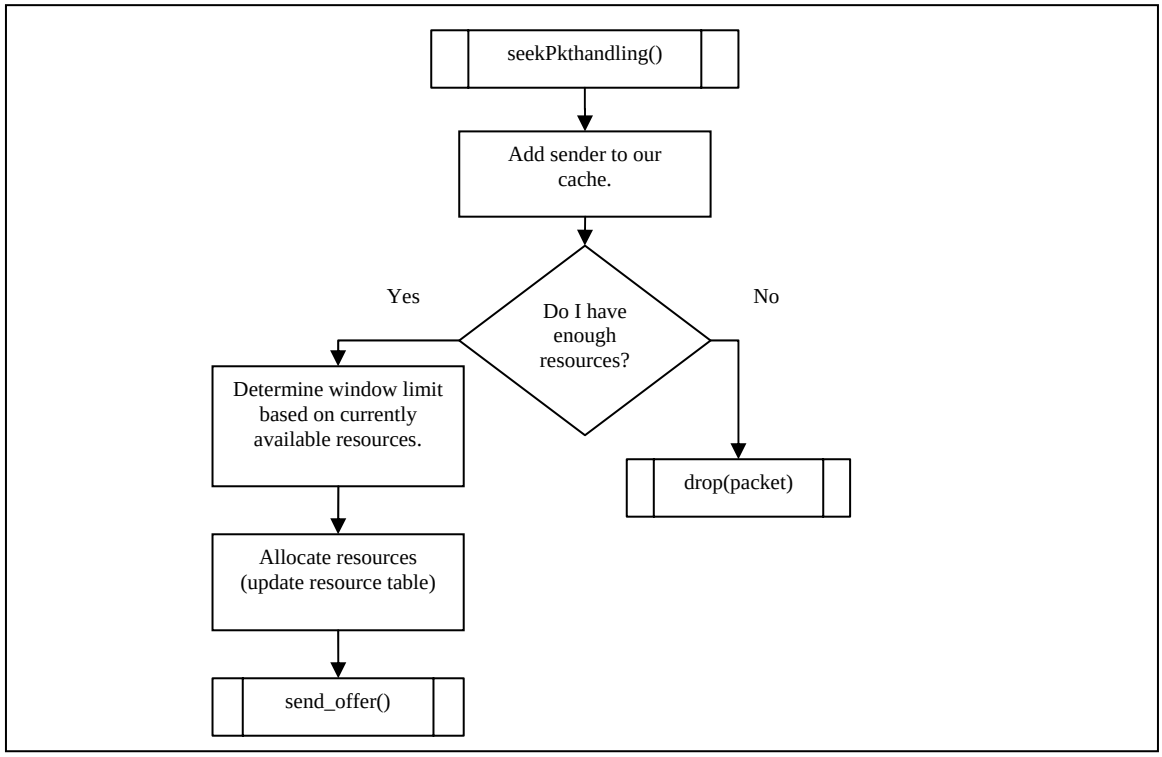


Fig. 13: seek header packet handling

4.3.5 offer header packet handling

The offerPktHandling() method is responsible for handling of offer packets that have been sent by a helper node. Offer packet handling consists of allocating resources on the sender's side and thus form a way of stateless handshaking with the helper node. Please note that security is not considered to be one of the factors of this protocol and hence we do not attempt to add three-way handshaking and stateful handshaking in the protocol. The processing applied by offerPktHandling can be seen in Fig. 14.

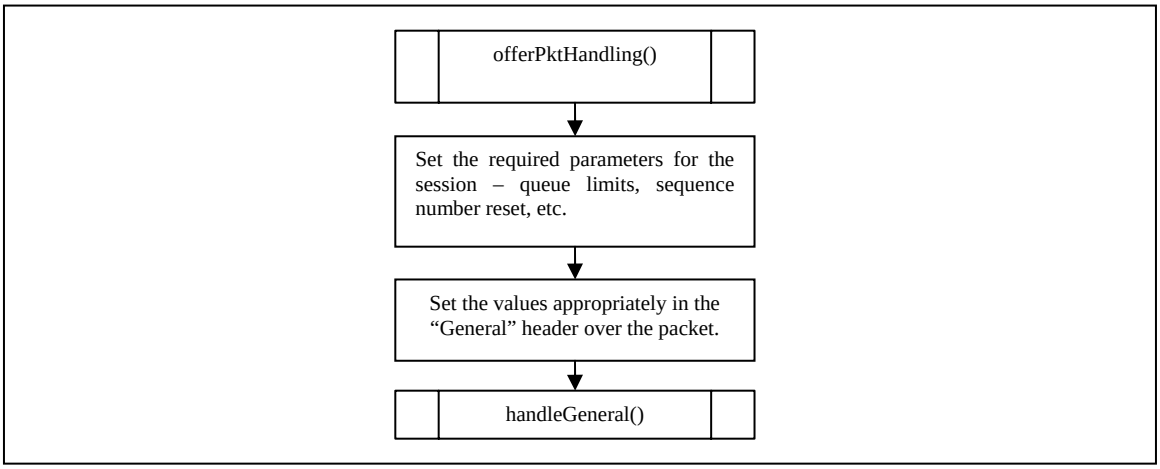


Fig. 14: offer header packet handling

4.3.6 general header packet handling

The handleGeneral() method is responsible for handling packets with the general header. A general header packet designates a packet in the middle of a flow of data between end-to-end nodes. The helper nodes change the selected_win_len and relay_id fields in the packet header and send it ahead in the network to the next node (relay_id). The processing applied by handleGeneral() method can be seen in Fig. 15.

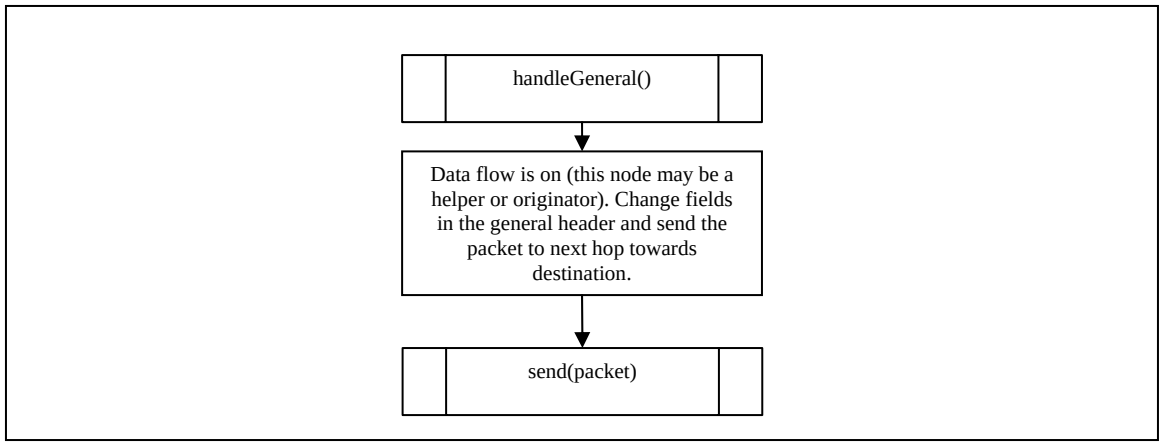


Fig. 15: general header packet handling

4.3.7 nack header packet handling

The nackPktHandling() method is responsible for resetting the window parameters of last packet sent and resending the packet to the intermediary node that sent out the nack. The behavior is different on whether ARQ(1) or ARQ(n) is used, which is

determined by the application in the OTcl test scripts. Fig. 16 shows the generic processing that nackPktHandling() applies.

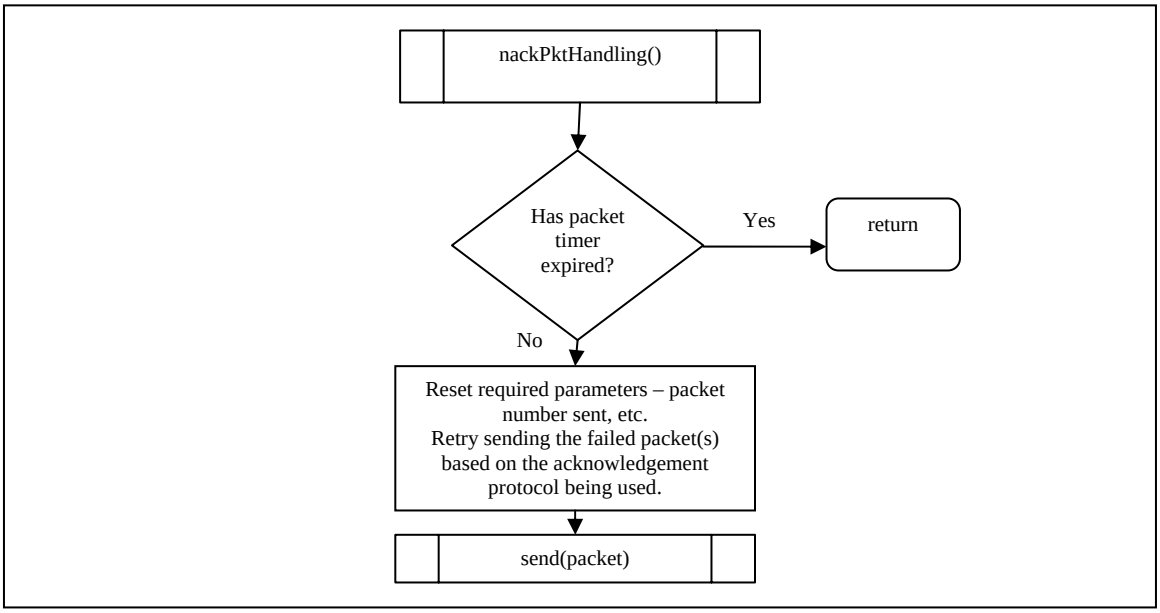


Fig. 16: nack header packet handling

4.3.8 end-2-end ack header packet handling

The e2eackPktHandling() method, which is responsible for end-2-end acknowledgements for window advancements has been implemented to be triggered from the higher layer test scripts. It has been done so to keep the protocol simple and as it integrates well with the current mechanism of signaling nodes with the initial parameters, such as total resources available and the reachability matrix. When the application signals

an end-2-end ack, a call is sent to the node's recv() method that falls down to the e2eackPktHandling() method which applies the necessary processing as show in Fig. 17.

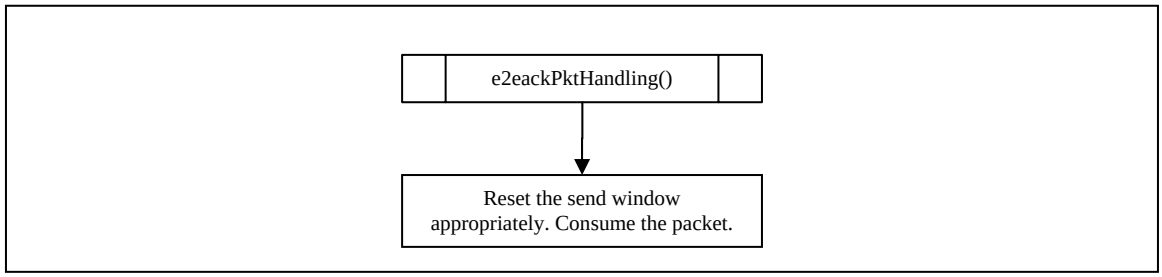


Fig. 17: end-2-end acknowledgement header packet handling

4.4 How to implement test scripts in OTcl for the protocol

It is important to note that the applications coded in OTcl need to perform certain shared responsibilities. These responsibilities include:

i. Creation of nodes and initialization of topography

Topography needs to be set by the test scripts in OTcl. NS-2 reads topology related commands in the OTcl scripts and parses them to form a network of nodes. One can additionally add movement data in the scripts as well if the network is non-static.

ii. Initialization of reachability matrix

NS-2 provides a General Operations Director (GOD) object that is aware of the whole network topology. The test scripts have to use the GOD

object to specify initial node distances for helper nodes to the base station node. This is done by the following simple example statement:

```
$ns_ at 0.0 "$god_ set-dist 4 10 2"
```

This loads the GOD object with the knowledge that the shortest path between node 4 and node 10 changed to 2 hops at time 0. As is evident, one can change the distances during run-time as well based on the time.

iii. Initiating the end-2-end acknowledgement calls

It is the responsibility of the OTcl scripts to initiate end-2-end acknowledgement calls so that the initiator of data transmission can advance the window accordingly for continuing transmission.

iv. Attaching application specific streams to the routing agent

A data stream, e.g. Constant Bit Rate (CBR) stream can be attached to the routing agent using the following commands:

```
set e2e_(0) [new Agent/E2ESeqNumAgent]
$ns_ attach-agent $node_(2) $e2e_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(3) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.25
$cbr_(0) set random_ 1
```

```
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $e2e_(0)
$ns_ connect $e2e_(0) $null_(0)
$ns_ at 10.000 "$cbr_(0) start"
```

This code-block creates an agent of our routing protocol (E2ESeqNumAgent) and attaches it to node 2. It then creates a null sink agent and attaches it to node 3. It then creates a CBR data stream and attaches it to our agent viz. e2e_(0). Finally it starts the flow at time 10.000.

Detailed scripts used to validate the correctness of implementation of this protocol can be found in Appendix B. In the next chapter, we will use these scripts to compare the routing overhead incurred by DSR, AODV and the protocol implemented here.

CHAPTER 5

ANALYSIS AND FUTURE WORK

5.1 Experiment and Analysis

To validate the implementation of the routing protocol, a scenario was used to compare the normalized routing load vs. pause time for DSR, AODV and the implemented protocol (E2ESeqNumAgent). E2ESeqNumAgent was expected to show higher efficiency than both as no route setup was necessary. Simulations were run in a small 10 node multi-hop scenario. The CMU scenario generator, which is distributed as a part of the NS-2 package, was used to generate a scenario with 10 nodes in a 500m x 500m 2-dimensional area. For generation of traffic, the CMU traffic generator tool was used, which is again distributed as a part of the NS-2 installation. A modification was required to be done to this tool to statically set one node as the receiver for all transmissions as we had to compare results with such a network only. Constant Bit Rate (CBR) traffic was generated for comparison. Varying pause times were used as a basis of comparison. The scripts used to generate the scenario, traffic and trace logs can be found in Appendix B.

Normalized routing load is defined as the number of routing packets transmitted per data packet delivered at the destination. Each hop-wise transmission of a routing packet is counted as one transmission.

The simulation parameters which have been considered for doing the performance comparison of two on-demand routing protocols are given below:

Table 2: Simulation parameters

Protocols	AODV, DSR, E2ESeqNumAgent
Simulation time	100 seconds
No. of nodes	10
Map size	500m x 500m
Traffic type	Constant Bit Rate (CBR)
Packet size	512 bytes
Connection rate	8 packets/sec
Pause time	0, 10, 20, 40, 100
No. of connections	5, 10

Fig. 18 shows the normalized routing load (NRL) vs. pause time for all three routing protocols. Lower routing load is a desirable feature in any protocol as it indicates lesser overhead and thus higher efficiency of the routing protocol. Another factor to consider is a stable NRL as it determines the scalability of the protocol. As seen in Fig. 18, the implemented protocol is superior to both AODV and DSR in such small topographies. DSR's lower NRL than AODV is also due to the small network. With only 10 nodes and fewer connections, source routes are comparatively small in the packets being sent.

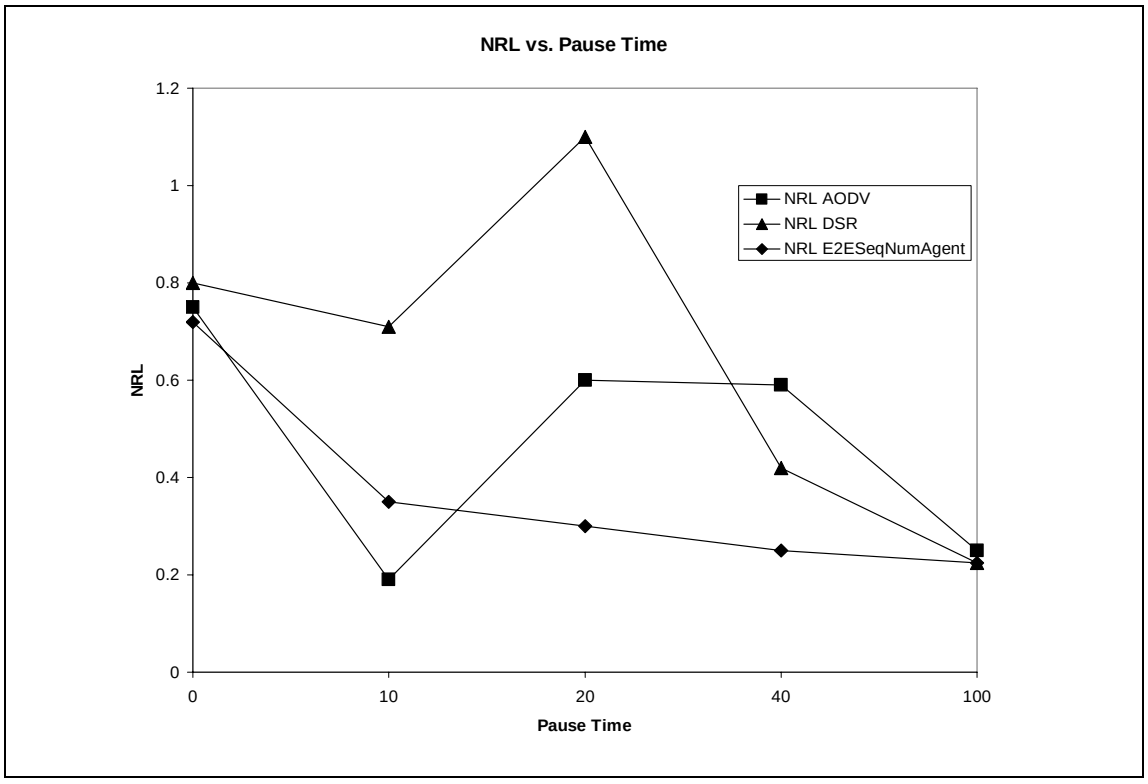


Fig. 18: NRL vs. pause time for 10 nodes

5.2 Future Work

This thesis involved an exhaustive study of the NS-2 simulator in addition to design and implementation of our specific experiment. NS-2, although a well-known and accepted simulator in academia, is a very complex simulator with a number of limitations due to its design and implementation methodologies. The most important of these limitations is the splitting of implementation in C++ and OTcl. OTcl is a home-brewed object-version of Tcl that hasn't gained much popularity. Further work involves implementing OTcl test scripts for performance evaluation of the protocol.

Other limitations include insufficient scalability, lack of emulation support and non-standard logging [**19**]. The implementers of NS-2 and the open-source community have accepted these limitations and have been actively working towards new standards known as NS-3. The NS-3 project was announced in July 2006 as a 4 year program and is in a current state of active development. The first stable release of NS-3 was announced in June 2008 [**18**] but was published with limited functionality. The scripting interface has been changed to Python that is more familiar and accepted by the developer community. Tracing and logging has been changed such that traces can now be seen in Wireshark [**21**], which is a very popular network packet statistics viewing tool in the developer and network administration community [**20**]. It would greatly benefit porting the protocol to NS-3 once NS-3 is complete and released as it would let one easily test different scenarios without worrying about the additional split-object architecture.

Also, the current protocol requires initialization of the reachability matrix through OTcl explicitly. Further work can be done to automate initialization through flooding or some other mechanism. This work falls outside the scope of this thesis, but would be a worthwhile project to follow. Further improvement in the implementation can be brought about by chaining the headers so that all packets can contain data including packets with seek, offer headers. Finally, the use of error-coding as highlighted in [**14**] can be explored in the future.

Security was not considered when designing and implementing this protocol and so adding security constraints would be another area of feature development. This would include overcoming denial of service attacks that could be easily done with the current implementation.

Future work would also involve trying different metrics for window length selection and trying the protocol with a well implemented wideband ALOHA or similar MAC protocol.

REFERENCES

1 "List of ad-hoc Routing Protocols", Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Ad_hoc_routing_protocol_list.

2 C. Perkins, E. Belding-Royer, S. Das, RFC 3561, "Ad hoc On-Demand Distance Vector (AODV) Routing", July 2003.

3 D. Johnson, Y. Hu, D. Maltz, RFC 4728, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4", February 2007.

4 W. Simpson, RFC 1662, "PPP in HDLC-like Framing", July 1994.

5 R.A.Comroe and D.J.Costello, "ARQ schemes for data transmission in mobile radio systems", IEEE J. Select. Areas Commun., 2:472-481, July 1984.

6 Information Sciences Institute, for Defense Advanced Research Projects Agency, RFC 793, "Transmission Control Protocol", September 1981.

7 Vaduvur Bhargavan, Alan Demers, Scott Shenker, Lixia Zhang, "MACAW: A Media Access Protocol for Wireless LAN's", In the Proc. ACM SIGCOMM Conference (SIGCOMM '94), August 1994.

8 R.J. Fontana, "Recent System Applications of Short-Pulse Ultra-Wideband (UWB) Technology", Invited Paper, IEEE Microwave Theory & Tech., Vol. 52, No. 9, September 2004, pp. 2087-2104.

9 Defense Advanced Research Projects Agency, http://www.darpa.mil.

10 Franklin F. Kuo, “The ALOHA system” in Computer Networks, Prentice-Hall, pp. 501–518, 1973.

11 The Network Simulator – ns-2, http://www.isi.edu/nsnam/ns/.

12 Trace Graph - Network Simulator NS-2 trace files analyzer, http://www.tracegraph.com/.

13 Nam: Network Animator, http://www.isi.edu/nsnam/nam/.

14 J. Metzner, “New acknowledgement protocols for dynamic wireless routing and multiple path transmission”, Draft, 2008.

15 OMNet++ Discrete Event Simulation System, http://www.omnetpp.org/index.php/.

16 Hubert Zimmermann, OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection, Invited paper, IEEE Transactions and Communications, Vol. Com-28, No. 4, April 1980.

17 The ns Manual, The Network Simulator ns-2: Documentation, http://www.isi.edu/nsnam/ns/ns-documentation.html.

18 ns-3 News, http://www.nsnam.org/news.html.

19 Tom Henderson and Sumit Roy, “ns-3 Project Plan, http://www.nsnam.org/docs/meetings/snowbird06/ns-3.ppt, 2006 NSF CRI-PI Meeting.

20 Workshop on ns-3, Tutorials, http://www.nsnam.org/workshops/wns3-2009/ns-3-tutorial-part-1.pdf and http://www.nsnam.org/workshops/wns3-2009/ns-3-tutorial-part-2.pdf.

21 Wireshark, http://www.wireshark.org/.

22 IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications http://standards.ieee.org/getieee802/download/802.11-2007.pdf.

23 IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 6: Medium Access Control (MAC) Security Enhancements, http://standards.ieee.org/getieee802/download/802.11i-2004.pdf.

24 Yih-Chun Hu and David B. Johnson, “Implicit Source Routes for On-Demand Ad Hoc Network Routing”, Proceedings of the 2001 ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc 2001), pp. 1-10, ACM, Long Beach, CA, October, 2001.

25 B.A.T.M.A.N. Concept, “B.A.T.M.A.N. Overview”, http://www.open-mesh.net/wiki/BATMANConcept.

26 YihChun Hu and David B. Johnson, “Implicit Source Routes for OnDemand Ad Hoc Network Routing”, In Proceedings of the Second Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2001), 2001.

27 Douglas S. J. De Couto, Daniel Aguayo, Benjamin A. Chambers and Robert Morris, “Performance of Multihop Wireless Networks: Shortest Path is Not Enough”, ACM SIGCOMM Computer Communications Review, Volume 33, Number 1: January 2003.

28 OPNET IT Guru Academic Edition, http://www.opnet.com/university_program/itguru_academic_edition/.

29 Eitan Altman and Tania Jimenez, “NS Simulator for beginners”, Lecture Notes, 2003-2004, December 4 2003.

30 Francisco J. Ros and Pedro M. Ruiz, “Implementing a New Manet Unicast Routing Protocol in NS2”, December 2004.

31 Marc Greis, “Tutorial for the Network Simulator ns, http://www.isi.edu/nsnam/ns/tutorial/index.html.

32 Bryan’s NS-2 DSR FAQ, http://www.geocities.com/b_j_hogan/, September 2007.

33 Rishi Sinha, “Using DSR in ns2”, http://nile.cise.ufl.edu/ee499/rishi-dsr-ns2.ppt.

34 J. Metzner, Personal Communication.

APPENDIX A

NS-2 ADDENDUM

Although NS-2 is a versatile simulator it is not trivially simple enough to start working with. The ns manual [**17**] is an exhaustive set of documents developed by the NS community, but is not kind for beginners. This is perhaps why several people have documented how to get started with NS-2. “NS Simulator for Beginners” [**29**] introduces the NS simulator and systematically ramps up rookies with most of the tools used within NS. However, much has changed in NS-2 since the first version of the ns simulator was released and much of the content in this document appears to be out-dated. Also, the document is not really useful for learning about wireless network simulations.

Once familiar with the basics of the ns simulator, learning how to implement a new routing protocol in ns-2 is very useful. This task is simplified by Francisco J. Ros and Pedro M. Ruiz in their document “Implementing a New Manet Unicast Routing Protocol in NS2” [**30**]. The authors nicely explain how to implement a routing protocol for MANETs in NS2 without going into the depths of a particular routing protocol.

Marc Greis in [**31**] attempts to make it easier for new ns users to use ns and nam, to create their own simulation scenarios for these tools and to eventually add new functionality to ns. Of most importance for this thesis work were sections IX (Running

Wireless Simulations in ns), X (Creating Wired-cum-Wireless and MobileIP Simulations) and XI (Generating traffic-connection and node-movement files for large wireless scenarios) from this tutorial.

When implementing a new wireless protocol, it is always helpful to study and follow an already implemented protocol. Since the protocol implemented here is similar in some ways to DSR, Bryan's NS-2 DSR FAQ [**32**] was very helpful when learning NS-2 and implementing this protocol. Rishi Sinha's presentation slide deck [**33**] is also useful for understanding how to use other tools provided by ns-2 in tandem with DSR.

Several other resources have been already listed in the body of the thesis that will help new ns users.

APPENDIX B

NS-2 SCRIPTS

B.1 Simulation running script

This is the script used to start a simulation. This script, "compare.tcl" takes 4 command line arguments - scenario file, traffic file, output trace file and routing protocol(1 = DSR, 2 = AODV and 3 = E2ESeqNumAgent). Script usage is:

```
$ ns compare.tcl -scen {scen} -tfc {tfc} -tr {tr} -rpr
{rpr}
```

This script requires a scenario file and a traffic file. These files can be generated using third party tools which are now part of the NS2 installation.

Script listing follows:

```
#
# ======================================================
# Define options
#
# ======================================================
set opt(chan)  Channel/WirelessChannel
set opt(prop)  Propagation/TwoRayGround
set opt(netif) Phy/WirelessPhy
set opt(mac)   Mac/802_11
#set opt(ifq)  Queue/DropTail/PriQueue
set opt(ifq)   CMUPriQueue
set opt(ll)         LL
set opt(ant)        Antenna/OmniAntenna
set opt(x)          500   ;# X dimension of the topography
set opt(y)          500   ;# Y dimension of the topography
```

```
set opt(ifqlen)     50         ;# max packet in ifq
set opt(seed)  0.0
set opt(tr)         dsr-10-0-5.tr    ;# trace file
set opt(adhocRouting)   DSR
#set opt(rpr)  1    ;#1  for  DSR,  2  for  AODV,  3  for
E2ESeqNumAgent
set opt(nn)              10           ;# how many nodes are
simulated
set opt(scen)       "movement/scen-10-0"
set opt(tfc)        "traffic/cbr-10-5"
set opt(stop)       100.0          ;# simulation time

#
# ======================================================
# Main Program
#
# ======================================================

if { $argc != 8 } {
        puts "Wrong no. of cmdline args."
     puts "Usage: ns compare.tcl -scen <scen> -tfc <tfc> -
tr <tr> -rpr <rpr>"
        exit 0
}


# proc getopt {argc argv} {

        for {set i 0} {$i < $argc} {incr i} {
                set arg [lindex $argv $i]
                if  {[string  range  $arg  0  0]  !=  "-"}
continue
                set name [string range $arg 1 end]
#         puts $name
                set opt($name) [lindex $argv [expr $i+1]]
        }
     set opt(scen) [lindex $argv 1]
     set opt(tfc) [lindex $argv 3]

        if {$opt(rpr) == 1} {
     set opt(adhocRouting)   DSR
     set opt(ifq)   CMUPriQueue
#    set opt(ifq)   Queue/DropTail/PriQueue
        } else if {$opt(rpr) == 2} {
     set opt(adhocRouting)   AODV
     set opt(ifq)   CMUPriQueue
```

```
#    set opt(ifq)   Queue/DropTail/PriQueue
        } else if {$opt(rpr) == 3} {
     set opt(adhocRouting)   E2ESeqNumAgent
     set opt(ifq)   CMUPriQueue
#    set opt(ifq)   Queue/DropTail/PriQueue
        }


#    set val(mov) $opt(scen)
#    set val(traf) $opt(tfc)
#    set opt(trace) $opt(tr)

     puts $opt(scen)
     puts $opt(tfc)
     puts $opt(tr)
# }


# getopt $argc $argv


     puts $opt(adhocRouting)
#    puts $val(mov)
#    puts $val(traf)
#    puts $opt(trace)

# Initialize Global Variables
# create simulator instance
set ns_        [new Simulator]

# set wireless channel, radio-model and topography objects
set wtopo [new Topography]

# create trace object for ns and nam
set tracefd    [open $opt(tr) w]
$ns_ trace-all $tracefd
# use new trace file format
$ns_ use-newtrace

# define topology
$wtopo load_flatgrid $opt(x) $opt(y)

# Create God
set god_ [create-god $opt(nn)]

#set chan_1_ [new $opt(chan)]
```

```
#set chan_2_ [new $opt(chan)]

# define how node should be created
#global node setting
$ns_ node-config -adhocRouting $opt(adhocRouting) \
           -llType $opt(ll) \
           -macType $opt(mac) \
           -ifqType $opt(ifq) \
           -ifqLen $opt(ifqlen) \
           -antType $opt(ant) \
           -propType $opt(prop) \
           -phyType $opt(netif) \
           -channelType $opt(chan) \
           -topoInstance $wtopo \
           -agentTrace ON \
                 -routerTrace ON \
                 -macTrace OFF
#    -channel $chan_1_

#   Create the specified number of nodes [$opt(nn)] and
"attach" them
#  to the channel.
for {set i 0} {$i < $opt(nn) } {incr i} {
     set node_($i) [$ns_ node]
     $node_($i) random-motion 0         ;#  disable  random
motion
}

# Define node movement model
puts "Loading connection pattern..."
source $opt(scen)

# Define traffic model
puts "Loading traffic file..."
source $opt(tfc)

# Define node initial position in nam
for {set i 0} {$i < $opt(nn)} {incr i} {

    # 20 defines the node size in nam, must adjust it
according to your scenario
    # The function must be called after mobility model is
defined
   $ns_ initial_node_pos $node_($i) 20
}
```

```
# Tell nodes when the simulation ends
for {set i 0} {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).000000001 "$node_($i) reset";
}

# tell nam the simulation stop time
#$ns_ at  $opt(stop)     "$ns_ nam-end-wireless $opt(stop)"
$ns_  at   $opt(stop).000000001  "puts  \"NS  EXITING...\"  ;
$ns_ halt"
puts "Starting Simulation..."
$ns_ run
```

B.2 Scenario generator script

The scenario file is generated using the “setdest” tool in the “make-scenario.sh” script. Listing of “make-scenario.sh” follows:

```
#!/bin/bash

dest_dir="movement"

if [ -d $dest_dir ]
then
     # Do nothing
     echo "'$dest_dir' is a directory"
else
     echo "Creating directory $dest_dir";
     mkdir --verbose $dest_dir
fi

setdest_loc="~/ns/ns-allinone-2.33/ns-2.33/indep-utils/cmu-
scen-gen/setdest/setdest";

if [ -x $setdest_loc ]
then
     # Do nothing
     echo "$setdest_loc is executable"
else
```

```
     echo   "$setdest_loc   does   not   exist   or   is   not
executable";
     exit;
fi

# Create the scenarios

for i in 0 10 20 40 100
do
     $setdest_loc -v 1 -n 10 -p $i -M 20 -t 100 -x 500 -y
500 > $dest_dir/scen-10-$i
done

echo ""
echo "Created the following files"
echo ""
ls -la $dest_dir/scen-10*
```

B.3 Traffic generator script

The traffic file is generated using the “cbrgen.tcl” tool in the “make-traffic.sh” script. Listing of “make-traffic.sh” follows:

```
#!/bin/bash

dest_dir="traffic"

if [ -d $dest_dir ]
then
     # Do nothing
     echo "'$dest_dir' is a directory"
else
     echo "Creating directory $dest_dir";
     mkdir --verbose $dest_dir
fi

script_file="~/ns/ns-allinone-2.33/ns-2.33/indep-utils/cmu-
scen-gen/cbrgen.tcl";
```

```
if [ -f $script_file ]
then
     # Do nothing
     echo "$script_file exists"
else
     echo "$script_file does not exist"
     exit;
fi

# Create the scenarios

for i in 5 10
do
     ns $script_file -type cbr -nn 10 -seed 1 -mc $i -rate
8.0 > $dest_dir/cbr-10-$i
done

echo ""
echo "Created the following files"
echo ""
ls -la $dest_dir/cbr-10*
```

B.4 AODV running script

The “run-aodv.sh” script listed as follows will start and run the simulation and generate 2 files with suffix sent and route_pkts to mean sent and routing packets. Each line in the file is “Pause Time”, “CBR Load” and the “Extracted Value”.

```
#!/bin/bash

for i in 5 10;
do
     for j in 0 10 20 40 100
     do
          ns  compare.tcl  -scen  movement/scen-10-$j  -tfc
traffic/cbr-10-$i -tr temptr -rpr 2;
          sent=`grep "^s.*\-Nl AGT.*\-It cbr.*" temptr | wc
-l`;
```

```
          echo "$j $i $sent" >> aodv-sent;
          route_pkts=`grep    "^\(s\|f\).*\-Nl    RTR.*\-It
\(AODV\|message\).*" temptr | wc -l`;
          echo "$j $i $route_pkts" >> aodv-route_pkts;
     done
done
```

B.5 DSR running script

The “run-dsr.sh” script listed as follows will start and run the simulation and generate 2 files with suffix sent and route_pkts to mean sent and routing packets. Each line in the file is “Pause Time”, “CBR Load” and the “Extracted Value”.

```
#!/bin/bash

for i in 5 10;
do
     for j in 0 10 20 40 100
     do
          ns  compare.tcl  -scen  movement/scen-10-$j  -tfc
traffic/cbr-10-$i -tr temptr -rpr 1;
          sent=`grep "^s.*\-Nl AGT.*\-It cbr.*" temptr | wc
-l`;
          echo "$j $i $sent" >> dsr-sent;
          route_pkts=`grep    "^\(s\|f\).*\-Nl    RTR.*\-It
\(DSR\|message\).*" temptr | wc -l`;
          echo "$j $i $route_pkts" >> dsr-route_pkts;
     done
done
```

B.6 E2ESeqNumAgent running script

The “run-e2eseqnumagent.sh” script listed as follows will start and run the simulation and generate 2 files with suffix sent and route_pkts to mean sent and routing packets. Each line in the file is “Pause Time”, “CBR Load” and the “Extracted Value”.

```
#!/bin/bash

for i in 5 10;
do
     for j in 0 10 20 40 100
     do
          ns compare.tcl -scen movement/scen-10-$j -tfc
traffic/cbr-10-$i -tr temptr -rpr 3;
          sent=`grep "^s.*\-Nl AGT.*\-It cbr.*" temptr | wc
-l`;
          echo "$j $i $sent" >> e2eseqnumagent-sent;
          route_pkts=`grep    "^\(s\|f\).*\-Nl    RTR.*\-It
\(E2ESeqNumAgent\|message\).*" temptr | wc -l`;
          echo  "$j  $i  $route_pkts"  >>  e2eseqnumagent-
route_pkts;
     done
done
```