12-1-2005

A Combined Routing Method for Ad hoc Wireless Networks

Zhenhui Jiang
Dartmouth College

A Combined Routing Method for Ad hoc Wireless Networks

Dartmouth Computer Science Technical Report
TR2005-566

Zhenhui Jiang

Dartmouth College, New Hampshire

December, 2005

Abstract

To make ad hoc wireless networks adaptive to different mobility and traffic patterns, we studied in this thesis an approach to swap from one protocol to another protocol dynamically, while routing continues. By the insertion of a new layer, we were able to make each node in the ad hoc wireless network notify each other about the protocol swap. To ensure that routing works efficiently after the protocol swap, we initialized the destination routing protocol's data structures and reused the previous routing information to build the new routing table. We also tested our approach under different network topologies and traffic patterns in static networks to learn whether the swap is fast and whether the swap incurs too much overload. We found that the swap latency is related to the destination protocol and the topology of the network. We also found that the control packet ratio after swap is close to the protocol running without swap, which means our method does not incur too many control packets for swap.

Contents

Table Index

Figure Index

Chapter 1 Introduction

A mobile ad hoc network (MANET) is a collection of moving computers connected by wireless links [Toh2002]. By routing packets cooperatively among the nodes, these nodes can communicate with each other without any infrastructure. Thus, ad hoc networks are often proposed for use in emergent situations, such as disaster environments and military conflicts. It is important that the ad hoc networks should react to network topological changes and traffic demands quickly and efficiently, and respect the inherent bandwidth and energy constraints [RFC2501]. Several projects compare the performance of different ad hoc routing algorithms [Gray2004, Broch1998, LeeW2000]. They all found that each routing algorithm can outperform the others in certain conditions, depending on the workload, terrain, network characteristics, or node mobility pattern.

Gray et al. compared four different routing algorithms: AODV [RFC3561], ODMRP [Lee2000], APRL [Karp1998] and STARA [Gupta1997, Gupta2000]. The authors found that under different wireless network conditions the relative performance was not the same. For example, ODMRP's message delivery ratio is better than AODV outdoors, while AODV has a higher message delivery ratio indoors [Gray2004]. Broch et al. compared DSDV, TORA, DSR and AODV. They found that DSDV's routing overhead was almost constant with respect to mobility rate while TORA, DSR and AODV's routing overhead dropped as the mobility rate dropped [Broch1998]. Lee et al. compared ODMRP, AMRoute [Bommaiah1998], CAMP [Garcia1999], AMRIS [Wu1998], and flooding. They found that "in a mobile scenario, mesh-based protocols (ODMRP) outperformed tree-based protocols (AODV)", but they also pointed out that ODMRP

showed "a trend of rapidly increasing overhead as the number of senders increased" [LeeW2000].

Ad hoc wireless networks routing protocols are usually divided into two groups: Proactive (Table Driven) and Reactive (On-Demand) routing [Royer1999]. Proactive routing protocols compute the routes in advance while reactive routing protocols compute the routes only when necessary. Both have advantages and disadvantages. Thus several hybrid routing protocols have been proposed to combine both proactive and reactive routing modes [Haas2001, Navid2001, Rama2003].

The zone routing protocol (ZRP) [Haas2001] divides the network into overlapping, variable-size zones. Routing within a zone uses proactive algorithms and routing between zones uses reactive algorithms. There are some other hybrid routing algorithms that combine proactive and reactive routing algorithms, such as HARP [Navid2001] and SHARP [Rama2003]. To reduce overhead, these hybrid methods group nearby nodes and use proactive routing algorithms within groups and use reactive routing algorithms between groups. Chen et al. proposed adaptive routing using clusters, which improves throughput by up to 80% [Chen2003]. Belding-Royer et al. proposed hierarchical protocols to reduce the overhead and gain more scalability [Belding2003]. However, since the technique uses higher-level topological information, the route to a destination might not be optimal, and the extra topological information itself requires more memory. Hoebeke et al. proposed an adaptive multi-mode routing algorithm [Hoebeke2004]. The implementation added a statistical component at the network layer: they collected non-local statistics through periodic broadcasting a "hello" message to neighbors. Their method improved efficiency via adaptation to different protocols. To achieve this

efficiency, however, they introduced many more components for the routing algorithm, which increased the complexity of the algorithm.

A common aspect of previous efforts is to allow the routing algorithm to adapt by combining multiple protocols because it is hard to come up with a routing protocol that is best for all situations. Our approach is to dynamically select one of three existing routing protocols rather than to create a new adaptive routing algorithm. We aim to achieve better performance by dynamically switching to the best protocol according to current wireless network conditions. In this thesis we focus on the mechanism for switching protocols, rather than the policy for choosing when to switch. Specifically, we develop and evaluate a mechanism for a network of nodes to switch to a new routing protocol.

To simplify our combined method, we assume that we already know these existing protocols' characteristics, and that some mechanism exists to choose the best routing protocol based on the current network traffic pattern. We could use, for example, Hoebeke's method to gather statistics about current network traffic, identify the traffic pattern, and then select a proactive or reactive protocol accordingly.

In ad hoc networks, each node acts both as a host and a router. We thus use the term "node" instead of "host" or "router". We also use the two terms "routing algorithm" and "routing protocol" interchangeably. In Chapter 2, we introduce three different routing algorithms, AODV, ODMRP, and APRL. We describe the differences among these three protocols and compare their performance. We also introduce SWAN, a simulator on which our experiments run. In Chapter 3, we describe the related work in more detail. In Chapter 4, we propose a method to switch among the three routing algorithms and discuss the implementation issues of this approach. In Chapter 5, we show the

performance of this approach and discuss the advantages. In Chapter 6, we summarize and draw conclusions.

Chapter 2 Background

In this chapter, we describe briefly the simulator on which our routing protocol runs. We also describe the three protocols that we combined: AODV, ODMRP, and APRL.

2.1 PLATFORM

We ran our routing protocols on the Dartmouth Simulator for Wireless Ad hoc Networks (SWAN) [SWAN]. SWAN is built on the parallel discrete event driven simulator DaSSF [DaSSF], which is a C++ implementation of the Scalable Simulation Framework [SSF]. DaSSF is particularly optimized for high performance when simulating large telecommunication systems [Liu2001]: DaSSF is able to simulate a network model that contains thousands of nodes. SWAN implements two layers of 802.11 protocols; a "pseudo-protocol-session" for the physical layer, and a protocol session for the MAC layer. SWAN also includes IP and ARP layers ported from the SSFNet [SSF] simulator code. A convenient feature of SWAN is that we can dynamically configure the protocol stack using the DML language [SSF]. The protocol stack of the whole system, illustrated in Figure 1, is composed of five layers. Our routing protocols AODV, ODMRP, APRL are above the UDP layer.

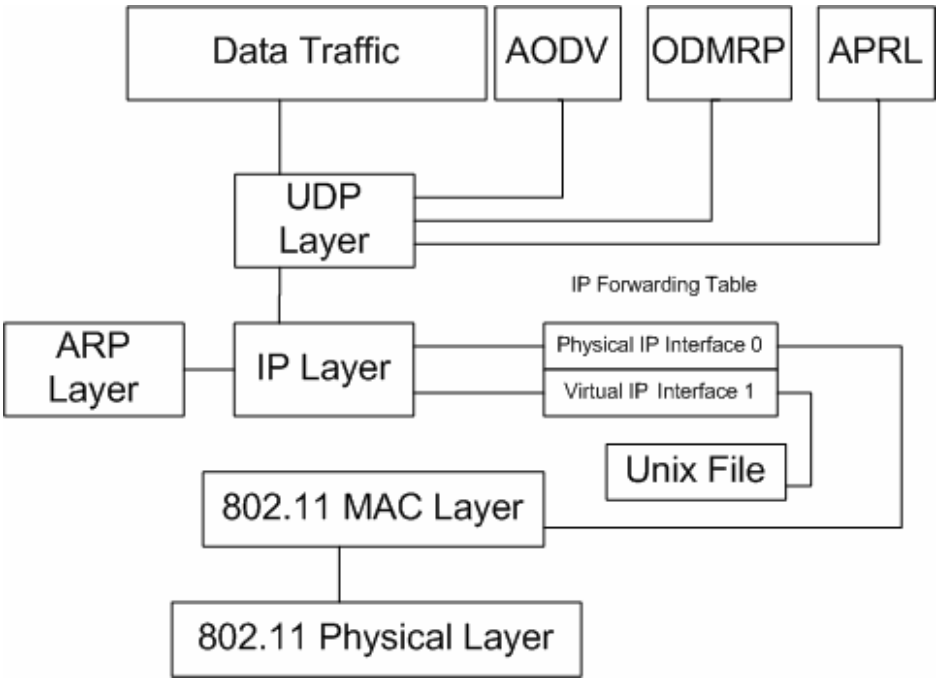

Figure 1. SWAN System Architecture

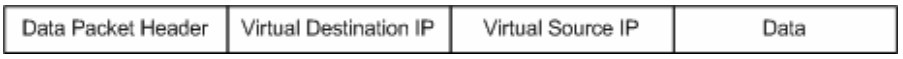

Figure 2. Data from UDP

We used existing implementations of AODV, ODMRP and APRL from the Dartmouth ActComm project [ActComm]. All three routing protocols are implemented in user space on Linux, and they use an IP tunnel and UDP sockets to perform their routing. An "IP tunnel" is a virtual network device that connects a UNIX device file and a network interface. Each node has a virtual IP address associated with the tunnel network interface, and a physical IP address associated with the real network interface in the node's IP forwarding table (Figure 1). At first, the application sends packets using the virtual IP address of the desired destination node (Figure 2). Then the packets are

forwarded to the UNIX device file through the IP tunnel. After that, the routing engine converts the virtual IP address to a physical IP destination address, and finds the physical IP address of the next hop according to its routing table and pushes the packets down to the IP layer. These packets with a physical IP address are forwarded to the real network interface instead of the virtual network interface. The original virtual-addressed packet is thus wrapped in an IP packet addressed to the physical IP address of the next hop in the IP layer, in effect, tunneling the virtual network into the physical network. When a packet arrives, the simulator notifies the routing engine about this event and then the routing engine unwraps the packet and checks the virtual address to see whether the packet has reached the destination or needs to be forwarded again. Finally, when a packet arrives at the destination, the simulator notifies the routing engine and the routing engine writes the packet to the UNIX device file for delivery to the application.

2.2 Classification of protocols

Ad hoc network-routing protocols are usually categorized into two groups: Proactive (Table Driven) and Reactive (On-Demand) routing.

2.2.1 Proactive routing

In this method, all the routes are computed in advance and each node maintains a routing table containing information about the best route to any node in the wireless network. The obvious advantage is that the route is already known when packets need to be sent. The disadvantage is that nodes need to update their tables periodically. Therefore, the nodes consume some network bandwidth exchanging routing information even when no data needs to be sent.

2.2.2 Reactive routing

In this method, the route to any destination is constructed only when necessary, then cached in the routing table. The advantage is avoiding proactive routing information exchange. The disadvantage is an increased, possibly large latency at the beginning of the transmission.

Earlier Dartmouth College research implemented three protocols on SWAN: AODV, ODMRP, and APRL [ActComm, SWAN]. APRL is a proactive protocol, while ODMRP is reactive, and AODV has both proactive and reactive characteristics. We selected these three routing algorithms because:

- they have already been correctly implemented,
- they represent different wireless routing styles, and
- research on their performance under different scenarios shows that each has strengths in different conditions.

2.3 Three protocols

We summarize each of the three protocols according to the ActComm implementation here. (See Appendix A for more details.)

2.3.1 Ad hoc On-demand Distance Vector Routing (AODV)

The Actcomm AODV implementation is an extension from the originally proposed AODV, adding the broadcast HELLO message. This implementation is capable of both unicast and broadcast routing. There are four types of control packets: RREQ, RERR, HELLO, and RREP. The first three are sent by broadcast, while RREP is by unicast.

AODV periodically generates HELLO messages to discover newly arrived or

departed neighbors. HELLO messages are not flooded or forwarded. When an originator needs to send a packet to a destination, AODV checks the local routing table to see whether there is an active route to the destination. If the originator knows an active route, then it sends the packet to the next hop according to this route's gateway IP address saved in the routing table. If not, AODV broadcasts a RREQ packet and sets a timer. The neighbors rebroadcast the RREQ in a flood search for the destination. When the timer times out, another RREQ will be sent out. This procedure repeats until the originator broadcasts RREQ_RETRIES times or the originator receives a RREP from the destination.

When a node receives a RREQ, but is not the destination, it examines its routing table to see whether there is an active route to the requested destination. If there is one, the node sends back a RREP to the last hop. Otherwise, it caches the RREQ's sender address in the routing table for backward learning and also adds the sender address to the precursor list. Then it re-broadcasts the RREQ. If the node is the destination, it will update its routing table, and send a RREP back to the originator via the last hop. When a RREP is received, AODV looks up the reverse path in the routing table, and forwards the RREP to the originator along the path.

If any link breakage is detected, AODV checks all the destinations affected and marks these routes invalid. In addition, AODV broadcasts RERR packets according to the precursor lists.

2.3.2 On-Demand Multicast Routing Protocol (ODMRP)

ODMRP is a multicast on-demand routing protocol for ad hoc networks. There are two types of control packets: Join Query and Join Reply [Bae2000]. Join Query is sent by

broadcast and Join Reply is sent by unicast. Both Join Query and Join Reply contain the originator and multicast group ID addresses.

ODMRP uses multicast groups to keep member information. In this thesis, we examine ODMRP only when used for unicast messaging. For each known node M in the whole network, ODMRP maintains a multicast group for that M, where the multicast group ID is M's IP address. Each ODMRP node has two data structures in addition to the routing table: a multicast group table and a message cache. The multicast group table contains all the multicast groups for a node; the message cache is used to detect routing loops. The multicast group table contains expiration time and information about whether it knows a route to M or it should receive data originated from M.

When an originator needs to send a packet to a destination but this destination is not in the multicast group or the multicast group has expired, it broadcasts a Join Query with the data packet attached. This is the main difference between ODMRP and AODV when ODMRP is used only for unicast.

When a node receives a Join Query from neighbor N, this node keeps the sender address and sequence number in the message cache. If there is a duplicate source address and a sequence number in message cache, this node will discard the packet. Otherwise, it checks the multicast group table. If it is not the multicast receiver of this multicast group (that is, its IP address does not match the group ID), it stores the upstream node (N) into the routing table for backward learning and rebroadcasts the Join Query (with the data packet attached) to its neighbors. If it is a multicast receiver, it receives the data and then sends a Join Reply packet back to the upstream node from which it receives the Join Query.

When a node receives a Join Reply, it checks the Join Reply's multicast group ID field to see whether this ID is in its multicast group table. If the node is a multicast receiver, it adds this multicast group to its multicast group table. If it is not the multicast receiver (the originator), it checks the routing table to see if there is a route to the multicast receiver. If so, it forwards the Join Reply to next hop according to the routing table. This procedure is repeated until the Join Reply reaches the originator.

A major difference with AODV is that the Join Request is not initiated until a node has some data to send to a multicast group and has no route to the multicast group. Also, even if an intermediate node already knows the route to the multicast group, it does not stop the route query procedure. Instead, it still broadcasts Join Query packets until the Join Query reaches a multicast receiver, because the Join Query has data attached.

Although ODMRP is a multicast protocol, we use it only as a unicast protocol [Gerla2000].

2.3.3 Any Path Routing without Loops (APRL)

APRL is a unicast, proactive routing protocol for ad hoc networks [Karp1998]. There are two types of control packets: Beacon and PDVN[1]. A node periodically broadcasts beacons to its neighbors; each beacon contains the route information known by the sender. PDVN is used to confirm the routes that the node receives in beacons.

Upon startup, each node broadcasts a beacon message to its neighbors so that each node's routing table only contains the destinations of its neighbors. After initializing the routing table with only its neighbors' information, each node broadcasts its own routing table to its neighbors periodically.

[1] ActComm implementation termed this packet PSVN, while [Karp1998] used PDVN.

On receipt of new routes in the beacon message, each node checks all the new routes using a control packet: namely, Ping Destination Via Neighbor (PDVN), to confirm the routes and to avoid loops.

A PDVN packet has two states: forward or backward. The forward state means the PDVN is in the procedure of being forwarded from originator O to destination D. During this procedure, each intermediate node keeps O and the immediate sender of the packet in the routing table for backward learning. When the PDVN reaches D, node D changes the PDVN to the backward state, and sends this PDVN back to its last hop. Similarly, the backward PDVN is sent back to the originator hop by hop based on each node's routing table.

Once a route is confirmed by receiving a backward PDVN, this route is added to the node's routing table.

Since APRL is a proactive routing protocol, any data packet to be sent checks the routing table. If there is no route to the required destination, the data packet is discarded; unlike AODV there is no route-request mechanism.

Chapter 3 Implementation

In this chapter, we discuss our method to combine the three different protocols. Simply speaking, we insert a new layer between the routing protocols and the UDP layer (or equivalent layer on some other infrastructure). We call this new layer the Protocol-Swap Layer. Thus, the change of the packets is transparent to the lower layer (in our case, the UDP layer).

3.1 The combined method

Because we insert a new layer (the protocol-swap layer) between the routing layer and UDP layer, any control packet generated by the routing protocols is intercepted by the protocol-swap layer where the packet is wrapped with additional information; namely, the protocol type and the epoch number (Figure 3). These two extra fields specify the current protocol type and the freshness of the protocol respectively.

For any received control packet, we first check the additional information at the protocol-swap layer, and then forward the control packet to the appropriate routing protocol (subject to some details discussed below).

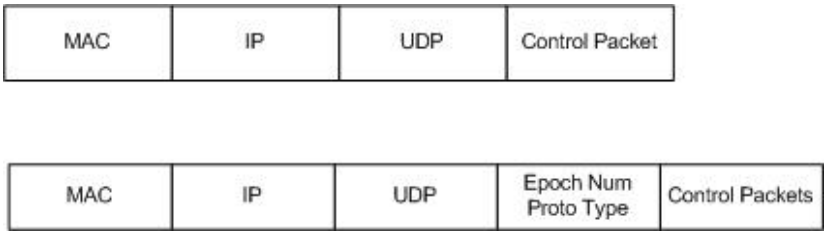

Figure 3. Packet Format, the top is the old format, the bottom is the new format with the protocol-swap layer header.

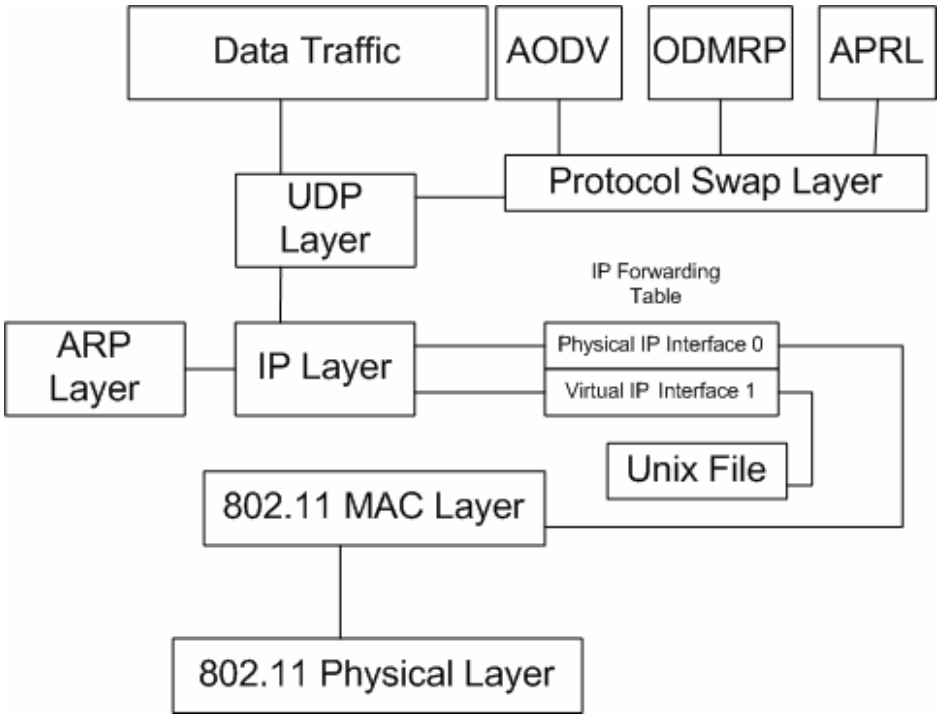

Figure 4. New System Architecture

Because of the insertion of the protocol-swap layer (Figure 4), the protocol type and epoch number are transparent to the routing protocol layer. The advantage of this layer is that we only have to change the interface with the new protocol-swap layer and can reuse the routing part of the existing routing protocol codes. We also encapsulate all the code

for swapping protocols in the protocol-swap layer.

Another advantage is that the combined method brings little overhead because:

1. The two fields are only 4 bytes each, which is small yet enough to prevent wraparound ambiguity.[2] Since a control packet is composed of a MAC header, an IP header, a UDP header, and a control packet body, the extra two fields do not use much extra bandwidth.
2. Only control packets are wrapped with the protocol type and epoch number while data packets remain the same as before.
3. During run time, only the routing table of current protocol is maintained, and the other combined protocols' routing tables are empty. So the combined method does not use extra memory for additional routing tables.
4. So far, our implementation of the protocol-swap layer does not set up a virtual connection to other protocol-swap layers, which means this method does not invent any new control packets.
5. We could add a network traffic monitor component in this layer. For example, if a node detects that the ratio of route requests is higher than normal, it might decide whether to swap to another routing protocol. This topic is beyond the scope of this thesis, but we will discuss this topic in the chapter on future work.

3.2 The problems we need to solve

To implement the combined method, which can swap from one protocol to another,

[2] In our implementation, we did not handle epoch number rollover, but it is easy to fix.

there are three problems to solve:

1. Who determines when to swap, and how?
2. How is the swap decision communicated to all nodes?
3. How does each node adjust its internal tables to make the swap?

3.2.1 Who decides?

A full solution to this problem is outside the scope of this thesis. We simply assume that one “master node” can initiate a protocol swap and notify all the nodes about the swap while regular nodes can only follow the protocol swap. It is beyond the scope of this thesis to determine when a swap should occur or to which protocol.

3.2.2 How is swap communicated?

The master node communicates its decision to its neighbors through the protocol type and epoch number. After these neighbors change to the new protocol, all their future control packets will carry this news to their neighbors via the protocol type field and epoch number, thus diffusing the news. The master node increments the epoch number and changes the protocol type every time it decides to swap.

We do not add the protocol-swap layer header to data packets, however, because data packets do not need to know which routing protocol is used to find a path to the destination. That is, even if two nodes are using different routing protocols, they can still send data packets to each other, and the network can continue forwarding packets even while a swap is in progress.

The mechanism for protocol swap requires each node to record its own notion of the

current local protocol type and epoch number. It then compares the protocol-swap layer header of incoming packets to determine whether a new epoch has occurred and thus it is time to switch to a new protocol. There are just two cases to consider.

Case 1 The received protocol number is the same as the local protocol number.

- ***Case 1.1*** The received epoch number is lower than its local epoch number; the node will discard the packet.
- ***Case 1.2*** The epoch number is equal; process the packet.
- ***Case 1.3*** The received epoch number is larger than the local epoch number; the node will update its local epoch number to be the received epoch number, and process the packet.

Case 2 The received protocol number is different from the local protocol number.

- ***Case 2.1*** The received epoch number is lower than or equal to its local epoch number; the node will discard the packet.
- ***Case 2.2*** The received epoch number is larger than the local epoch number; the node will update its local epoch number to be the received epoch number, swap to the received protocol, then process the packet.

3.2.3 How to swap?

To swap, we need to initialize the new protocol's routing table and other data structures by using those of the current protocol.

The primary goal when changing protocols is to build the routing table for the new protocols and to initialize it as much as possible using information in the routing table for the old protocol. We illustrate six different cases for the swap in Table 1.

To perform the swap, we must not only change the routing table, but each protocol's special associated data structures as well. We discuss each such data structure in turn.

AODV Precursor List. This data structure contains all the upstream nodes that use the node itself towards the same destinations. If the node determines that any one of its links is broken, as a hint it sends a RERR packet to those neighbors who are in its precursor list. When we swap to AODV, it is safe to leave the precursor list empty, because this data structure will be rebuilt when nodes later send out RREQ.

AODV Packet Queue. The source node queues any data packets that are yet to be sent in per-destination packet queues. When we swap from AODV to another protocol, we discard the packets in these queues and they are lost. We assume that some other mechanism (such as TCP) will realize that these packets did not reach their destinations and will resend those data again.

The packet queue is AODV's unique data structure; other protocols do not have a queue for data packets. If we swap to AODV, we can simply create empty packet queues.

AODV RREQ Packet Cache. This data structure is used to store recently received RREQ packets to avoid loops. It may be created as empty when we swap to AODV, and may be discarded when we swap from AODV.

ODMRP Message Cache. This data structure is used to store recently received Join Query packets to avoid loops. It may be created as empty when we swap to ODMRP, and may be discarded when we swap from ODMRP.

ODMRP Multicast Group Table. This data structure is used to maintain a list of multicast groups in which this node is a member and is checked when receiving a Join Query. If this node is in the multicast group, then it should accept the Join Query packet

and send back a Join Reply. ODMRP has to rebuild the multicast group table via Join Queries when we swap to ODMRP. This data structure may be created as empty when we swap to ODMRP, and may be discarded when we swap from ODMRP.

APRL has no additional data structures, so there is nothing extra to do when swapping to or from APRL.

From AODV to ODMRP
1. Create ODMRP's routing table based on AODV's routing table. 2. Discard AODV's routing table. 3. Initialize message cache and multicast group table to be empty. 4. Update the protocol type.
From AODV to APRL
1. Create APRL's routing table based on AODV's routing table. 2. Discard AODV's routing table. 3. Update the protocol type. 4. Broadcast APRL's beacon message.
From ODMRP to AODV
1. Create AODV's routing table based on ODMRP's routing table. 2. Discard ODMRP's routing table. 3. Initialize precursor lists, and packet queues. 4. Update the protocol type. 5. Broadcast AODV's hello message
From ODMRP to APRL
1. Create APRL's routing table based on ODMRP's routing table. 2. Discard ODMRP's routing table. 3. Update the protocol type. 4. Broadcast APRL's beacon message.
From APRL to ODMRP
1. Create ODMRP's routing table based on APRL's routing table. 2. Discard APRL's routing table. 3. Initialize message cache and multicast group table to be empty. 4. Update the protocol type.
From APRL to AODV
1. Create AODV's routing table based on APRL's routing table. 2. Discard APRL's routing table. 3. Initialize precursor lists, and packet queues. 4. Update the protocol type. 5. Broadcast AODV's hello message.

Table 1. Types of Swaps

3.3 Reuse the prior routing table entries

To take advantage of the prior protocol's routing information, we reuse the entries in the prior routing table. However, the entries in the routing tables of AODV, ODMRP, and APRL are different (Table 2), which complicates our effort to copy the entries between routing protocols. We copy any similar fields of two entries and choose a reasonable value for the fields that are different.

It is important to note that AODV, ODMRP and APRL all have two key fields for routing: the destination IP address and the next-hop IP address. These two fields determine the next hop for forwarding packets to the destination. Since all these routing protocols use these two fields to determine any route, it is correct to copy these two IP addresses from the prior routing table entry to the new routing table entry. The other fields are used to determine the current status of the routes. AODV, ODMRP, and APRL keep different status of the routes for routing, so it might not be correct to reuse them in the new protocol. But we can carefully select a valid default value, as shown in Tables 3, 4, and 5; the tables also comment on the correctness and the drawbacks of these default values.

Another advantage of reuse the prior routing table entries is we can keep our route consistent, after the swap, we does still use the prior route, so the data packets do not loop back the originator.

Routing table entry of AODV

Type	Variable name	Comment
in_addr	dstIP	Destination IP address
in_addr	nextHop	Next hop IP address
unsigned int	dstSeqNum	Sequence number
unsigned int	hopCount	Number of hops to destination
timeval	exprTime	Route expiration time
AodvPrecursorList	precursors	Upstream nodes to the originator
int	act_inv	Active or Invalid

Routing table entry of ODMRP

Type	Variable name	Comment
in_addr	dstIP	Destination IP address
in_addr	nextHop	Next hop IP Address
unsigned int	hopCount	Number of hops to destination
timeval	exprTime	Route expiration time

Routing table entry of APRL

Type	Variable name	Comment
in_addr	dstIP	Destination IP address
in_addr	nextHop	Next hop IP address
int	conf	Confirmed, Unconfirmed, Expiring
int	dyn_perm	Dynamic or Permanent
timeval	tLastConf	Time marked as Confirmed
int	act_alt	Active or Alternate

Table 2. Routing Table Entries

From AODV to ODMRP

AODV	ODMRP	Verification	Drawbacks
dstIP	dstIP	Same field	
nextHop	nextHop	Same field	
dstSeqNum		No such field	
hopCount	hopCount	Same field	
exprTime	exprTime = current time + ODMRP active interval time [3]	We assume this new route can still be used	If the mobility is high, the possibility of link break increases
precursors		No such field	
act_inv		No such field	

From ODMRP to AODV

ODMRP	AODV	Verification	Drawbacks
dstIP	dstIP	Same field	
nextHop	nextHop	Same field	
	dstSeqNum = ++ local dstSeqNum [4]		
hopCount	hopCount	Same field	
exprTime	exprTime = current time + AODV active interval time [5]	We assume this new route can still be used	If the mobility is high, the possibility of link break increases
	Precursors = {}	No such field.	Can not tell precursors about link breakage.
	act_inv = Active	The route will be verified later	It might be inactive.

Table 3. Swaps between AODV and ODMRP

[3] ODMRP active interval time is ODMRP's route life time.
[4] AODV's local destination sequence number is used as RREQ designation sequence number.
[5] AODV active interval time is AODV's route life time.

From AODV to APRL

AODV	APRL	Verification	Drawbacks
dstIP	dstIP		
nextHop	nextHop		
	conf = unconfirmed	Verify new routes before use.	
	dyn_perm = dyn	All routes are dynamic	
	tLastConf = current time	Any value will do, since conf = unconfirmed	
	act_alt = Active		
dstSeqNum		No such field	
hopCount		No such field	
exprTime		No such field	
precursors act_inv		No such field No such field	

From APRL to AODV

APRL	AODV	Verification	Drawbacks
dstIP	dstIP	Same field	
nextHop	nextHop	Same field	
	dstSeqNum = ++ local dstSeqNum		
	hopCount = default max hops	If there is a shorter path, the route is replaced .	
	exprTime = current time + AODV active interval time.	We assume this new route can still be used.	If the mobility is high, the possibility of link break increases.
	precursors = {}	No such field. Set it empty.	Can not tell precursor the link breakage.
	act_inv = Active	The route will be verified later.	It might be inactive.
conf		No such field	
dyn_perm		No such field	
tLastConf		No such field	
act_alt		No such field	

Table 4. Swaps between AODV and APRL

From APRL to ODMRP

APRL	ODMRP	Verification	Drawbacks
dstIP	dstIP	Same field	
nextHop	nextHop	Same field	
	hopCount = default max hops	Any new route will replace this route	
	exprTime = current time + ODMRP active interval time.	We assume this new route can still be used.	If the mobility is high, the possibility of link break increases.
conf		No such field	
dyn_perm		No such field	
tLastConf		No such field	
act_alt		No such field	

From ODMRP to APRL

ODMRP	APRL	Verification	Drawbacks
dstIP	dstIP	Same field	
nextHop	nextHop	Same field	
	conf = unconfirmed	Verify new routes before use	
	dyn_perm = dyn tLastConf = current time	All routes are dynamic Any value will do, since conf=unconfirmed	
	act_alt = Active		
hopCount		No such field	
exprTime		No such field	

Table 5. Swaps between ODMRP and APRL

Chapter 4 Experiments and Discussion

In this chapter, we describe our experiments based on the SWAN simulator and we compare performance of our protocol-swap layer with performance without the new layer. Our goal is to measure the overhead (in time and traffic) due to a protocol swap.

4.1 Experiments

We configured the SWAN simulator using dml files. We chose a static network, which means all the nodes were preset to a certain position and would not move during the experiments. The effective transmission distance was 73m. We ran the protocol for 200 seconds and the swap occurred at 100 seconds. (See Appendix B for configuration details and dml files.)

We selected two types of network topology: line (Figure 5) or lattice (Figure 6). We selected two different numbers of nodes: 9 nodes and 49 nodes. We selected two traffic speeds: 1 data packet originated per node per second or 1 data packet originated per node every 5 seconds. The data packet's destination is randomly selected from the rest nodes.

We ran each combination 5 times, each time with a different random seed for SWAN. We report the average results across the 5 runs.

Figure 5. Line, 9 nodes

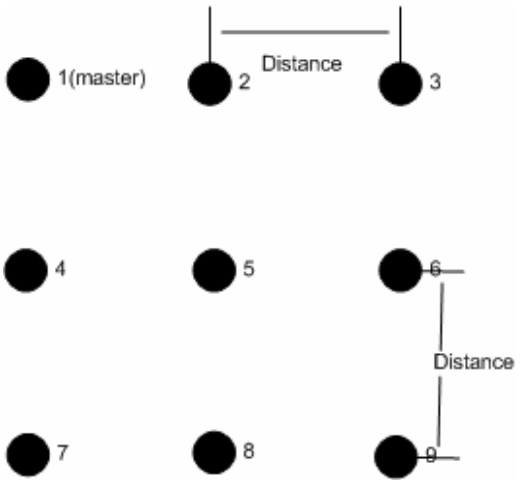

Figure 6. Lattice, 9 nodes

	AODV	ODMRP	APRL
AODV	**AODV**	**AODV to ODMRP**	**AODV to APRL**
ODMRP	**ODMRP to AODV**	**ODMRP**	**ODMRP to APRL**
APRL	**APRL to AODV**	**APRL to ODMRP**	**APRL**

Table 6. Swap Matrix

4.1.1 Metrics

We compare the performance of our combined method with plain AODV, APRL, and ODMRP. We used two metrics: the time to complete a protocol swap in stationary networks, and the ratio of control packets per data packet sent from UDP layer.

Metric 1: Time to complete a protocol swap in stationary networks.

The swap time starts when the master node decides to swap, and ends when all the nodes in the network have updated their local protocol number and local epoch number. The metric is thus the End time – Start time. This metric measures the swap latency.

There are six types of swaps to consider (Table 6).

Metric 2: Ratio of unicast and multicast control packets per data packet sent from UDP layer.

AODV has four control packets: HELLO, RREQ, RREP, RERR. RREP is unicast and the rest are multicast.

APRL has two control packets: BEACON, PDVN. PDVN is unicast and BEACON is multicast.

ODMRP has two control packets: Join Query, Join Reply. Join Reply is unicast and Join Query is multicast.

In both cases, the control-packets ratio is the number of (unicast and multicast) control packets divided by the number of data packets sent from UDP layer.

We compare the control-packets ratios of three interval: the interval after swap of the destination protocol, the first half interval of the destination protocol and the second half interval of the destination protocol. For example, in Figure 7, we compare the ratio of:

- The destination protocol after the swap.
- The destination protocol within first 100 seconds.
- The destination protocol within second 100 seconds.

This metric measures the efficiency of the destination protocol after swap.

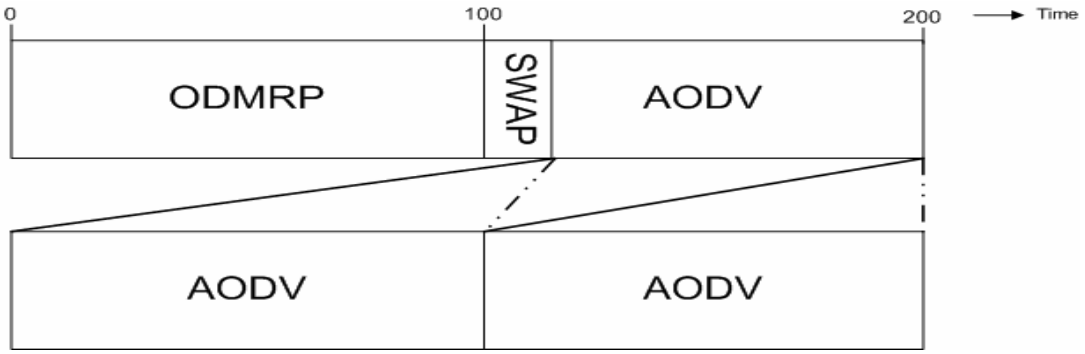

Figure 7. Control Packets Ratio

4.1.2 Environment

We chose four configurations as shown in Table 7. Referring back to Figure 5 and Figure 6, and recalling the effective communication distance (73m), several nodes are in range of each node, including the master node. Although all nodes were connected directly or indirectly, we can see in Configuration 3 that all nodes were connected within the transmission range of each other, but in other configurations multi-hop communication was required. (See Appendix B for dml file configurations. See Appendix C and D for more detailed data about the environment.)

Configuration	Topology	# Nodes	Distance	Neighbors to the master node
1	Line	9	20m	3
2	Line	49	20m	3
3	Square	9	25m	8
4	Square	49	30m	8

Table 7. Distance of the Experiments

4.2 Experiment Results

We used the metrics we defined above to measure the efficiency of the swap. We use max swap latency to the same destination protocol from two sources. For example, in Figure 9, we use the max swap latency from ODMRP to AODV and APRL to AODV in order to get swap latency for AODV. (In Figure 8, we show swap latency dependent on different source and destination protocols).

4.2.1 Swap Latency

Association with the connectivity of the networks. Table 8 and Figure 9 both show that swap latency is associated with the network connectivity for each type of swap. The highest connectivity (Configuration 3) has the best swap latency and lowest connectivity (Configuration 2) has the worst swap latency.

Configuration	Topology	# Nodes	Distance	Max Swap Latency(s)
1	Line	9	20m	10.004
2	Line	49	20m	35.048
3	Square	9	25m	1.813
4	Square	49	30m	6.011

Table 8. Association with Network Connectivity

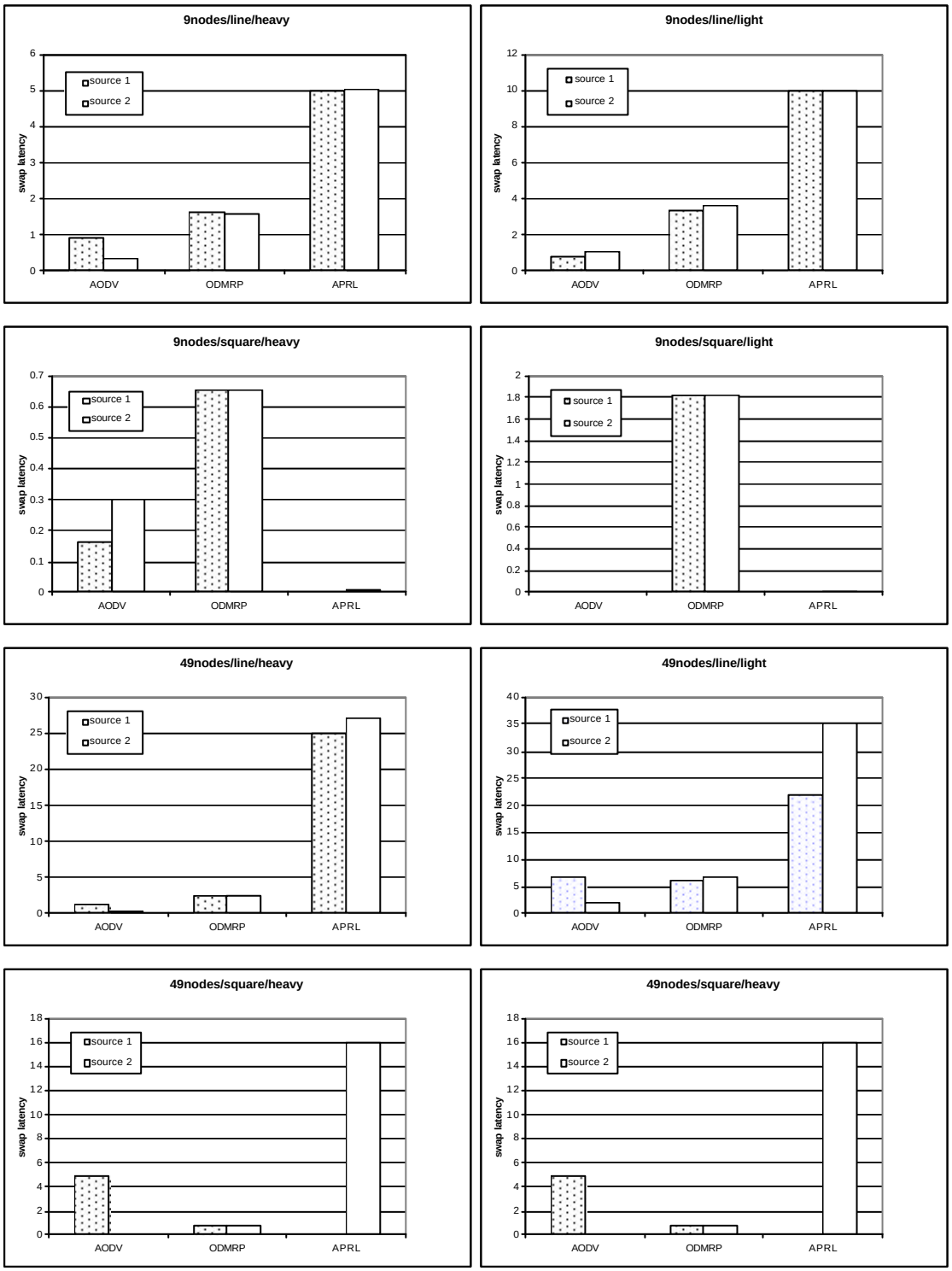

Figure 8. Swap Latency dependent on the source and destination protocols

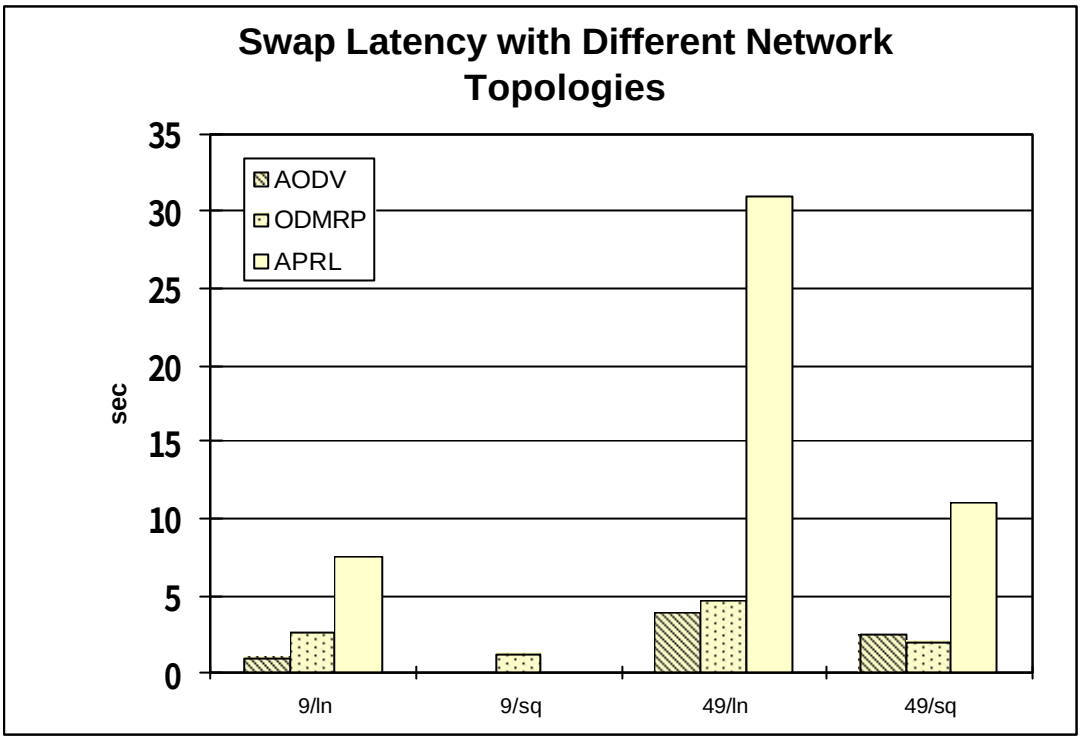

Figure 9. Swap Latency with Different Network Topologies; ln = line and sq = square lattice

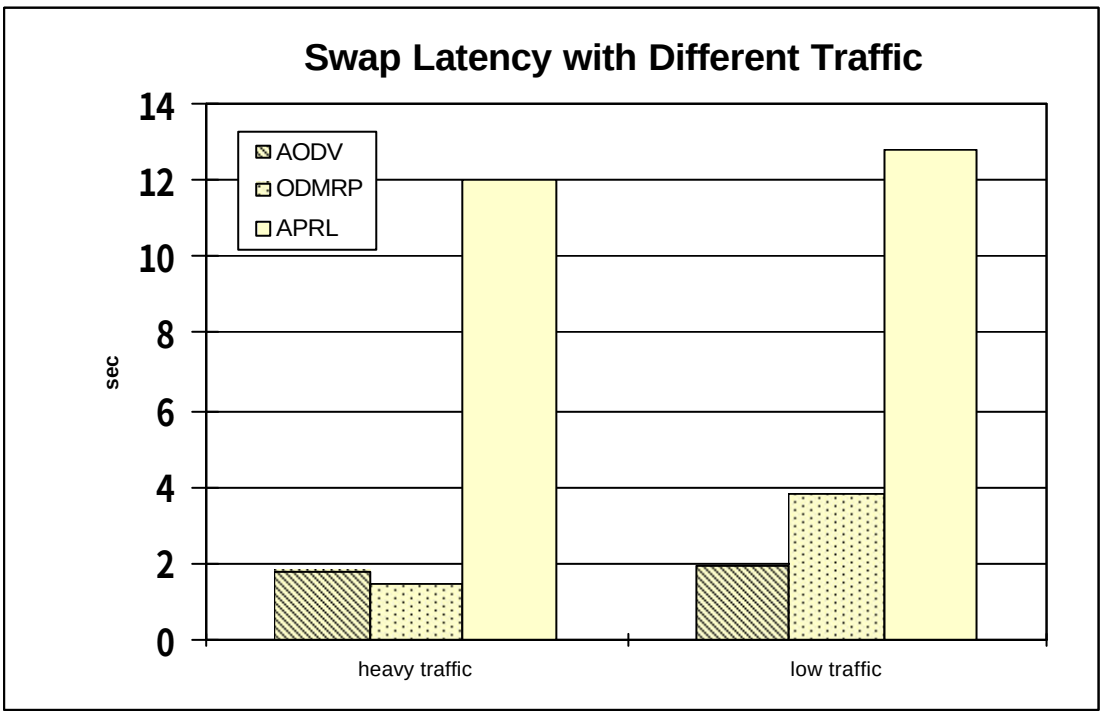

Figure 10. Swap Latency with Different Traffic

Association with the network traffic. Figure 10 shows that for AODV and APRL, the swap latency were similar with heavy and low traffic workloads. For ODMRP, the heavy traffic's swap latency is nearly twice as fast than low traffic's; because ODMRP is a pure reactive routing protocol, it only sends Join Query when it needs to. So a busier traffic pattern generates more control traffic and thus spreads the news about the swap. But AODV and APRL both periodically broadcast message to its neighbors.

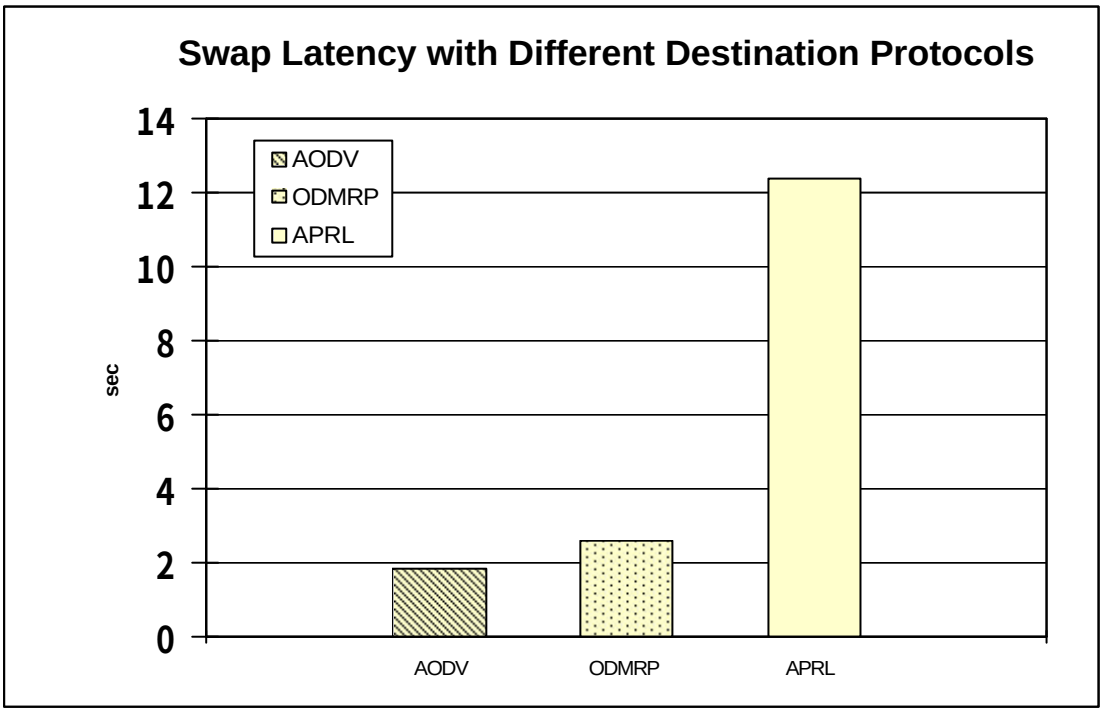

Figure 11. Swap Latency with Different Destination Protocols

Association with the destination protocol. Figure 11 shows that AODV and ODMRP completed the swap fast, while APRL was relatively slow. After swap, ODMRP needs to broadcast Join Query packets to maintain its multicast group membership information. Similarly, AODV needs to broadcast RREQ if there is no route to the destination in the routing table after swap. But APRL drops the packets if it can not find route information in its routing table. Those route query broadcasting packets makes AODV and ODMRP

fast.

In Table 9, we can see that in high connectivity configurations (row 3 and 4), AODV and APRL swapped quickly. This is because they both periodically broadcasting message to neighbors.

We can also see in Table 9 that AODV and ODMRP completed their swaps in short time even in the low connectivity networks. But APRL's swap time was highly related to the network topology. For example, AODV swap latency range was from 0.001 to 6.665 seconds and ODMRP from 0.652 to 6.635 seconds. APRL's swap latency range was from 0.009 to 35.048 seconds, however, because APRL only broadcasted beacons to its neighbors periodically. So the swap latency depends on the period of these broadcasts rather than on the traffic patterns as in AODV or ODMRP.

Exp No.[6]	To AODV(s)	To ODMRP (s)	To APRL (s)
1	0.928	1.645	5.032
2	1.048	3.595	10.004
3	0.098	0.652	0.009
4	0.001	1.813	0.014
5	1.235	2.593	27.043
6	6.655	6.635	35.048
7	4.890	0.849	16.032
8	0.095	3.162	6.010

Table 9. Swap Latency with the Destination Protocol

4.2.2 The control packets ratio.

In Figure 12Figure 13Figure 14, we show the control packets ratio of eight types of network configurations. For each configuration and each type of swap, we can see that:

- The control packets ratio after swap was not the largest one of each group in most

[6] Exp No. is corresponding to Experiments number in Appendix C. Exp 1 and Exp 2 are using configuration 1. Exp 3 and Exp 4 are using configuration 2, so on and so forth.

cases.

- Even if the control packets ratio after swap was the largest one of each group, it was very close to the rest.

After the swap, in most cases, the destination routing protocol performed much as it would without a swap. Thus the swap did not add too many control packets to the new routing protocol.

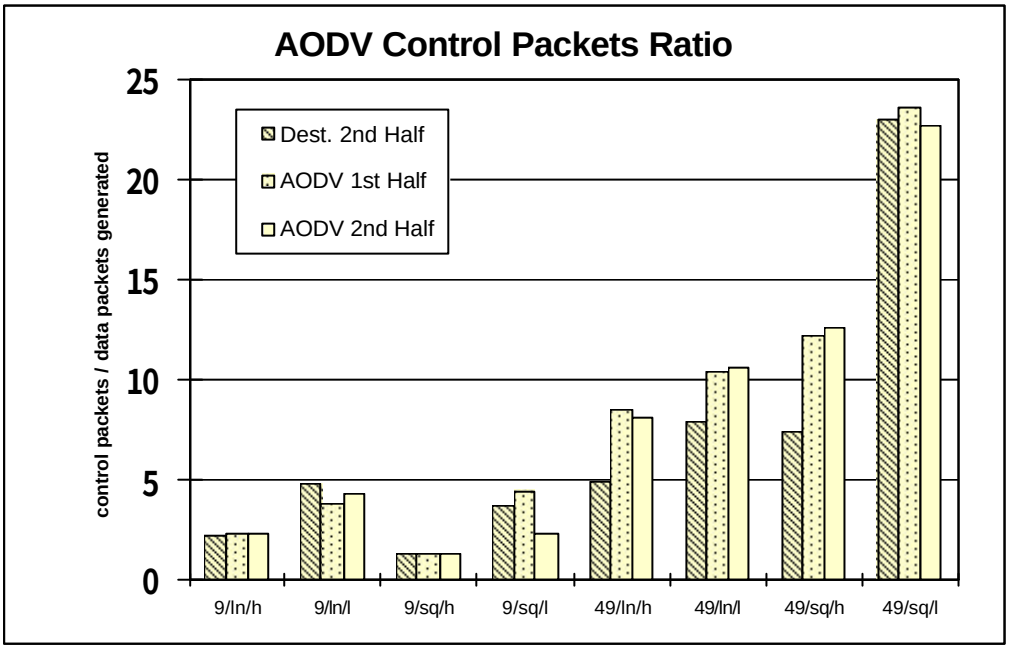

Figure 12. AODV Control Packets Ratio

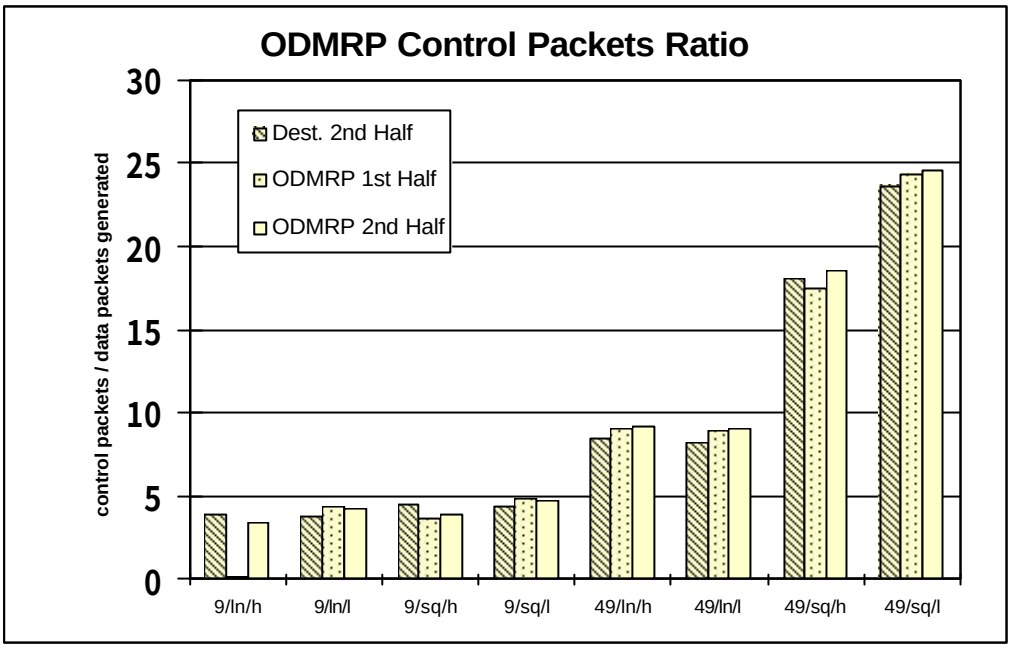

Figure 13. ODMRP Control Packet Ratio

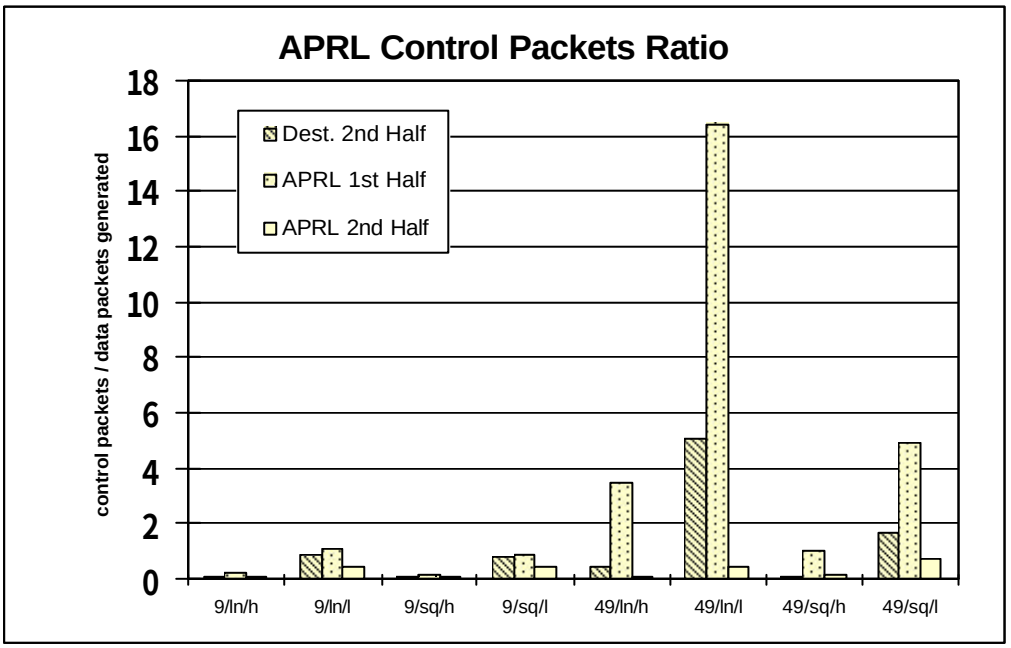

Figure 14. APRL Control Packet Ratio

4. 3 Discussion

4.3.1 Swap Latency is related to network connectivity, network traffic and the characteristics of the destination protocol we swap to.

First, we found that the swap latency depends on the network connectivity, the highly connected networks had a better swap latency, because news of the swap had fewer hops to traverse.

We also found that traffic workload influenced the swap latency of reactive routing protocols like ODMRP.

Last, swap latency also depends on the characteristics of the destination protocol Protocols that are reactive depend on the data traffic to generate control packets and thus propagated the swap; proactive protocols depend on periodic broadcasts to spread the new. If a protocol performs routing without flooding, the swap latency was long (particularly in low connected networks). In this case, we might need to send back an empty message to tell a sender about the update of the routing protocol if we receive any out-dated control packets from the sender.

4.3.2 Swap does not incur too many control packets.

As the results show, the control packets ratio after swap was lower than or close to control packets ratio of running a protocol without a swap.

First, the swap does not require extra control packets to diffuse the swap information or rebuild tables. Second, because we initialize the new routing table using the old routes, we send few route query packets. This situation is true only in static networks or low mobility networks. In high-mobility networks, of course there would be more control

traffic to rebuild a route to the destination, and more lost data packets, but the same would be true if the swap had not occurred. Thus we believe that our method efficiently transfers the network from one routing protocol to another, using no new packet types, reusing routing table information where possible, and not excessively increasing control traffic after a swap.

Chapter 5 Related Work

In this chapter, we discuss some related prior research because their methods and conclusions may help to understand the combined method. We discuss other related work in Chapter 1.

5.1 Some known routing protocol characteristics

Gray et al. described an outdoor experimental evaluation of AODV, ODMRP and APRL using laptops [Gray2004]. They also simulated these three protocols in SWAN and compared the simulation results with the real-world results. The following are some interesting conclusions from [Gray2004]:

"AODV is efficient, but far from effective in all scenarios. In an environment with limited bandwidth or energy resources, AODV is a good choice. In an environment with extremely high mobility, however, AODV may not be able to keep up with topology changes."

"ODMRP is highly effective for some scenarios, but very bad for others. If bandwidth and energy resources are plentiful, data packets are small, and communication reliability is crucial, ODMRP is a good choice."

Their analysis of APRL's performance shows "an unnecessarily large number of messages dropped before leaving their source node".

These conclusions reinforced the value of switching protocols as conditions change,

and could help to decide which protocol to use under which conditions.

5.2 How to know the current traffic pattern

To implement an adaptive routing protocol, it is important to determine when to swap protocols, and how to collect appropriate information to make that decision.

Hoebeke et al. added a monitoring agent, a new component responsible for collecting information about network traffic [Hoebeke2004]. The monitoring agent gathers local statistics from the network layer and non-local statistics from a "Hello" message. Both the local statistics and non-local statistics could contribute to make decisions on when to swap.

5.3 Difference with the current adaptive methods

Hoebeke et al. proposed an adaptive multi-mode routing protocol for ad hoc networks [Hoebeke2004]. Their adaptive method is similar to our combined method: both want to dynamically swap to another protocol based on current network conditions, but there are two differences.

1. New message introduced:

This adaptive method introduced a new type of control packet (Hello) to the existing protocols, which is periodically broadcasted. Our method did not introduce any new message to the existing protocols.

Thus, it is relatively easy to combine more protocols if necessary. On the other hand, we need to design N^2 routing-table converters if we want to combine N protocols.

2. Routing table maintenance:

In their method, different protocols share the same routing table, requiring all

protocols to be reimplemented to suit the new, common routing table format. Each node also has a neighbor table to keep track of connectivity and neighbors' modes (reactive or proactive).

In our method, different protocols each maintain their own routing table, and we translate tables when we swap protocols. Thus, we do not need to change the routing table format and introduce a new neighbor table.

Chapter 6 Summary and Future Work

We developed a method to combine AODV, ODMRP, and APRL in such a way that we can swap from one protocol to another. For each pair of protocols, we identify how to initialize each protocol's data structure from those of the other protocol. We proposed two metrics to measure the performance, designed different network topologies and conditions to test the swaps, and ran simulations using SWAN. The results show that the time to complete a protocol swap depended on the characteristics of the protocol we swap to, the topology of the network, and the traffic of the network.

Our combined method swaps slowly for the less-connected networks and for the destination protocols without flooding like APRL. But a hybrid method would not have this problem because they use different mechanism to swap. Each node in hybrid method can dynamically swap without master node. However, this decision is based on the each node's own network traffic statistics, which may incur some additional control traffic. In our combined method, from a software engineering point of view, we can reuse the source code of the existing protocol by inserting a new layer and without changing existing protocol implementations.

Future Work. For simplicity, we chose to use static networks to run our experiments. However, these static networks could not test whether our method would be efficient for high-mobility networks. To fix the invalid routes requires broadcasting route query packets, and meanwhile we may lose data packets because of the wrong routing

information. We recommend further testing with other mobility models.

Another tradeoff is whether we should let the protocol-swap layer broadcast an empty packet just to notify its neighbors about the swap. This broadcast should decrease the swap completion time by increasing the speed of disseminating news about the swap. This might be helpful for these protocols that do not broadcast periodically. Also, we could add some fields in that empty packet's header to carry statistics besides protocol type and epoch number. For example, if we know the most of received multicast packets are route queries, then we know it might be better to swap. However, these fields add some complexity to the new protocol-swap layer and may add overhead.

Appendix A Handlers

These handlers showed the way our combined method handles different events on SWAN simulator. The followings are the main handlers in our implementation.

Protocol-swap layer Handlers

1)Tunnel Event Handler

a) If local protocol is AODV, call AODV Tunnel Event handler.
b) If local protocol is ODMRP, call ODMRP Tunnel Event handler.
c) If local protocol is APRL, call APRL Tunnel Event handler.

2)Incoming Multicast Handler

a) If the incoming packet's protocol is the same as the node's local protocol.
   i) If the incoming packet's protocol is AODV, based on the packet type call corresponding AODV multicast packet handler.
   ii) If the incoming packet's protocol is ODMRP, based on the packet type call corresponding ODMRP multicast packet handler.
   iii) If the incoming packet's protocol is APRL, based on the packet type call corresponding APRL multicast packet handler.
b) Otherwise, incoming packet's protocol is different from the node's local protocol.
   i) If the incoming packet's epoch number is larger than the node's local epoch number, swap the node's local protocol to the incoming protocol type.
   ii) Otherwise, drop the packet.

3)Incoming Unicast Hanlder

a) If the incoming packet's protocol is the same as the node's local protocol.
   i) If the incoming packet's protocol is AODV, based on the packet type call corresponding AODV unicast packet handler.
   ii) If the incoming packet's protocol is ODMRP, based on the packet type call corresponding ODMRP unicast packet handler.
   iii) If the incoming packet's protocol is APRL, based on the packet type call corresponding APRL unicast packet handler.
b) Otherwise, incoming packet's protocol is different from the node's local protocol.
   i) If the incoming packet's epoch number is larger than the node's local epoch number, swap the node's local protocol to the incoming protocol type.
   ii) Otherwise, drop the packet.

4)Incoming Data Packet Handler

a) If local protocol is AODV, call AODV data packet handler.
b) If local protocol is ODMRP, call ODMRP data packet handler.
c) If local protocol is APRL, call APRL data packet handler.

AODV Handlers

1)Tunnel Event Handler

a) Wrap the data into a data packet.
b) Get physical destination.
c) Look for a route to destination.
    i) If there is an active route to the destination.
        (1) forward the data packet to the next hop according to the routing table.
    ii) Otherwise there is no active route to the destination.
        (1) If a search for the destination is already in progress.
        (2) Do nothing and put and data packet in the packet queue.
        (3) Otherwise send out a Outgoing RREQ packet.

2)Incoming Multicast HELLO Packet Handler

a) Get the sender's IP address.
b) If the hello packet is not Looped back then:
    i) Add the new neighbor to our routing table.
    ii) Schedule a RouteRemovalHandler to invalidate the route when it expires.
    iii) Send any queued packets that we can with the newly added route.
c) Otherwise, drop the packet.

3)Incoming Multicast RERR Packet Handler

a) Get the Sender's IP address.
b) For each of the unreachable detonations listed in the RERR:
    i) If the route's precursor list is not empty.
        (1) Append this destination to the RERR to be broadcast.
        (2) Schedule the RERR to be broadcast.

4)Incoming Multicast RREQ Packet Handler

a) Get the Sender's IP address.
b) If the packet is not looped back then:
c) Keep reverse route.
    i) Check the RREQ cache.
    ii) If we have processed the same RREQ, just ignore this one.
    iii) If We don't have a cache entry for this RREQ.
        (1) Add this RREQ broadcast to the cache table.
        (2) Make sure we have a reverse route to the source of the RREQ (create/update one as necessary).
        (3) Send any queued packets that we can with the newly added route.
d) If we are the destination of the RREQ, then:
    i) Assemble the correct RREP packet to be sent back to the source of the RREQ.
e) Otherwise we are an intermediate node.
    i) If We have a good enough route to the destination of the RREQ, then:
        (1) Schedule an RREP to send back to the source of the RREQ.

ii) Otherwise We are not the destination of the RREQ.
(1) Schedule a rebroadcast of the RREQ.

5)Incoming Unicast RREP Packet Handler

a) Get the sender's IP address.
b) If the packet is not looped back then:
c) Check the routing table.
   i) If we don't have any route to the destination of the RREP at all, create one and set an expiration timer for the new route.
   ii) If we already have one route, updating our route to the destination of the RREQ and send any queued packets that we can with the newly added route.
   iii) If we are not the source of the RREP, find the route to the source of the RREP according to the routing table and forward the RREP towards its source.
   iv) If we are the source of the RREP, send any queued packets that we can with the newly added route.

6)Incoming Data Packet Handler

a) Get the virtual destination IP address of the packet and convert it to the real one.
b) If we are the destination of the packet, write the packet to the tunnel.
c) If we are an intermediate node, Check the routing table.
   i) If there is a route to the destination. Forward the packet to the gateway.
   ii) else Create a RERR packet with the unreachable destination.

ODMRP Handlers

1)Tunnel Event Handler

a) Wrap the data into a data packet.
b) Get physical destination.
c) Look for the destination of the packet in the multicast group table.
   i) If the multicast group exists and has not expired, broadcast the packet out.
   ii) Otherwise we either don't know anything about the destination, or the multicast group has expired.
      (1) If the multicast group has expired, invalidate this forwarding group.
      (2) Otherwise broadcasts the Join Query packet with the data piggy-backed.

2)Incoming Multicast JOIN QUERY Packet Handler

a) If we are the source of the Join Query, or we have already processed the same Join Query, ignore this one.
b) Otherwise this Join Query is not a duplicate, so add a cache entry for it.
   i) Check the routing table.
      (1) If we don't have a reverse route to the source of the Join Query, create one.
      (2) Otherwise we already have a route, so just update it.
   ii) Check the multicast group.
   iii) If we want to receive packets sent to this multicast group IP, construct a Join Reply.
      (1) If the Join Query has piggy-backed data, send it to the tunnel.
   iv) Otherwise we do not want to receive packets on this multicast group IP.
   v) Schedule a rebroadcast of the Join Query.

3)Incoming Unicast JOIN REPLY Packet Handler

a) If the Join Reply's multicast group IP is not in the multicast group table, add the multicast group of the Join Reply to the table.
b) Otherwise, this IP is in the table.
   i) If it is a the IP is not in the forwarding group, activate the forwarding group.
   ii) Otherwise, update the lifetime of the forwarding group.
c) If we are not the source of the Join Reply, check the routing table.
   i) If we find a route to the source of the Join Reply, schedule a Join Reply to be forwarded towards the source of the Join Query.
d) Otherwise, drop it because we the source of the Join Reply.

4)Incoming Data Packet Handler

a) Get the virtual original source IP of the packet and convert it to the physical one.
b) If we are the source of the packet or we have already processed the same one, ignore this one.
c) Otherwise, this data packet is not a duplicate, so add a cache entry for it.
d) Look for the multicast group in our table.
   i) If we do not know anything about this multicast group, drop the packet.

ii) If we are a member of the multicast group, receive the packet.
iii) If we are part of the forwarding group for this address, forward the packet.
iv) Otherwise if the forwarding group has expired, invalidate it. Since we are not part of the forwarding group for the multicast group, drop the packet.

APRL Handlers

1)Tunnel Event Handler

a) Wrap the data into a data packet.
b) Get physical destination.
c) Look for a route to the destination of the packet in the routing table.
   i) If route exists, forward the packet to the gateway.
   ii) Otherwise, drop the data packet.

2)Incoming Multicast BEACON Packet Handler

a) Get the sender of the beacon.
b) Add the sender in the routing table if there is no such route.
c) Examine all the routes in the beacon we received.
d) If there are any routes, then I want to check whether this route is in the table.
   i) If it is, update the routes.
   ii) Otherwise, add the route as an unconfirmed route and schedule an outgoing PSVN to confirm it.

3)Incoming Unicast PSVN Packet Handler

a) If the packet is going forward.
   i) If the packet originated from this node looped back here, drop it.
   ii) If the packet is not destined for this machine which means that we need to look up this machine's route to the destination.
      (1) If we find in the table, forward the packet to next hop.
      (2) Otherwise there is no route to the destination, so drop it.
      (3) If the packet is destined for this node, reverse it and send it back towards the source.
b) Otherwise the packet is a reverse packet.
   i) If the packet looped back to me on its reverse path so we drop it.
   ii) If this is the node it came from, confirm the route in the routing table
   iii) Otherwise we check routing table.
   iv) If there is a reverse hop, forward the packet to the reverse hop.
   v) If no reverse hop available, drop the packet.

4)Incoming Data Packet Handler

a) Get the virtual destination IP address of the packet and convert it to the real one.
b) If we are the destination of the packet, write the packet to the tunnel.
c) Otherwise drop the packet.

Appendix B Configurations

Arguments used to generate experiments files:

1. stat_preset_9nodes_line_9neighbors.pl
   - network topology: static network, preset node's positions, 9 nodes, in a line with 20 meters interval, node's effective transmission distance 73m, arena size 180m * 180m
   - run time: 200 seconds
   - swap time: 100 seconds
   - traffic speed: 1 data packet to send out per second / 5 data packet to send out per second

2. stat_preset_49nodes_line_9neighbors.pl
   - network topology: static network, preset node's positions, 49 nodes, in a line with 20 meters interval, node's effective transmission distance 73m, arena size 9800m * 9800m
   - run time: 200 seconds
   - swap time: 100 seconds
   - traffic speed: 1 data packet to send out per second / 5 data packet to send out per second

3. stat_preset_9nodes_square_4neighbors.pl
   - network topology: static network, preset node's positions, 9 nodes, in a lattice with 25 meters interval, node's effective transmission distance 73m, arena size 75m * 75m
   - run time: 200 seconds
   - swap time: 100 seconds
   - traffic speed: 1 data packet to send out per second / 5 data packet to send out per second

4. stat_preset_49nodes_square_25neighbors.pl
   - network topology: static network, preset node's positions, 49 nodes, in a lattice with 30 meters interval, node's effective transmission distance 73m, arena size 210m * 210m
   - run time: 200 seconds
   - swap time: 100 seconds
   - traffic speed: 1 data packet to send out per second / 5 data packet to send out per second

Arguments for different protocols:

AODV

MAX_DATA_PACKET_QUEUE_SIZE 10

MAX_AODV_PACKET_SIZE 128

NODE_TRAVERSAL_TIME 40

RREQ_RETRIES 2

ACTIVE_ROUTE_TIMEOUT 3000

DELETE_PERIOD 12000

HELLO_INTERVAL 3000

APRL

BLOCK_SIZE 3048

ROUTE_CAP 7

REPORT_INTERVAL 30000

ODMRP

FWD_GROUP_LIFE 6000

REFRESH_INTERVAL 3000

REV_ROUTE_LIFE 2000

Appendix C Experiments[7]

All the tables are generated from the log files of the simulator experiments. These data are averages of five experiments results with the same configuration except the simulator's random seed.

No.	Data Packet Rate (sec/packet)	Topology	# Nodes	First Swap Time (sec)	Runtime (sec)
1	**1**	**Line Distance: 20m**	**9**	**100**	**200**

Type	1st Half Mcast[8]	1st Half Ucast	1st Half Total Control Packets	1st Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio[9]
AODV	2060	2071	4130	2.31	8287	1787	4.64
ODMRP	4821	733	5554	3.22	11456	1723	6.65
APRL	175	265	440	0.25	620	1787	0.35

[7] These Tables are based on five run of same configurations, the data is the average of five runs.
[8] First half is the time interval of 0 to100 seconds and second half is the time interval of 100 to 200 seconds.
[9] Ratio is the Total Control Total Packets divided by Total Data Packets generated from UDP layer.

Type	2nd Half Mcast	2nd Half Ucast	2nd Half Total Control Packets	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	2152	2004	4156	2.33	8287	1787	4.64
ODMRP	5116	786	5902	3.43	11456	1723	6.65
APRL	180	0	180	0.10	620	1787	0.35

Type of Swap	Swap Latency (s)	Mcast 2nd Half	Ucast 2nd Half	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV ODMRP	1.645652	7014	0	3.93	11145	1787	6.24
AODV APRL	5.002703	178	0	0.10	2071	1787	1.16
ODMRP AODV	0.928241	1646	1560	1.86	8760	1723	5.09
ODMRP APRL	5.032506	176	112	0.17	5842	1723	3.39
APRL AODV	0.332115	2412	2107	2.53	2372	1787	1.33
APRL ODMRP	1.600228	7033	0	3.93	11551	1787	6.47

Table 10. 9nodes_line_heavy

No.	Data Packet Rate (sec/packet)	Topology	# Nodes	First Swap Time (sec)	Runtime (sec)
2	**5**	**Line Distance: 20m**	**9**	**100**	**200**

Type	1st Half Mcast	1st Half Ucast	1st Half Total Control Packets	1st Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	900	620	1520	3.98	3172	382	8.30
ODMRP	1321	206	1527	4.35	3010	351	8.58
APRL	175	247	422	1.10	602	382	1.58

Type	2nd Half Mcast	2nd Half Ucast	2nd Half Total Control Packets	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	935	717	1652	4.32	3172	382	8.30
ODMRP	1293	190	1483	4.23	3010	351	8.58
APRL	180	0	180	0.48	602	382	1.58

Type of Swap	Swap Latency (s)	Mcast 2nd Half	Ucast 2nd Half	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV ODMRP	3.373787	1464	0	3.83	2984	382	7.81
AODV APRL	10.002182	172	0	0.45	620	382	1.63
ODMRP AODV	0.798326	1002	591	4.54	3120	351	8.91
ODMRP APRL	10.004271	173	254	1.22	1954	351	5.58
APRL AODV	1.048517	1161	756	5.01	1003	382	2.62
APRL ODMRP	3.595250	1438	0	3.76	3355	382	8.78

Table 11. 9nodes_line_light

No.	Data Packet Rate (sec/packet)	Topology	# Nodes	First Swap Time (sec)	Runtime (sec)
3	**1**	**Square Distance: 25m**	**9**	**100**	**200**

Type	1st Half Mcast	1st Half Ucast	1st Half Total Control Packets	1st Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	453	1865	2318	1.30	4574	1787	2.56
ODMRP	5655	635	6289	3.65	12912	1723	7.49
APRL	175	172	347	0.20	527	1787	0.30

Type	2nd Half Mcast	2nd Half Ucast	2nd Half Total Control Packets	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	430	1826	2256	1.26	4574	1787	2.56
ODMRP	5954	669	6623	3.84	12912	1723	7.49
APRL	180	0	180	0.10	527	1787	0.30

Type of Swap	Swap Latency (s)	Mcast 2nd Half	Ucast 2nd Half	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV ODMRP	0.652548	8046	0	4.50	10365	1787	5.80
AODV APRL	0.001458	180	0	0.10	1865	1787	1.04
ODMRP AODV	0.164112	825	563	0.81	7678	1723	4.46
ODMRP APRL	0.009996	180	111	0.17	6580	1723	3.82
APRL AODV	0.298631	1808	1304	1.74	1476	1787	0.82
APRL ODMRP	0.652558	8046	0	4.50	11158	1787	6.25

Table 12. 9nodes_square_heavy

No.	Data Packet Rate (sec/packet)	Topology	# Nodes	First Swap Time (sec)	Runtime (sec)
4	**5**	**Square Distance: 25m**	**9**	**100**	**200**

Type	1st Half Mcast	1st Half Ucast	1st Half Total Control Packets	1st Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	336	335	671	1.75	1559	382	4.07
ODMRP	1507	174	1681	4.79	3331	351	9.50
APRL	175	177	352	0.92	532	382	1.40

Type	2nd Half Mcast	2nd Half Ucast	2nd Half Total Control Packets	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	339	549	888	2.32	1559	382	4.07
ODMRP	1478	172	1650	4.71	3331	351	9.50
APRL	180	0	180	0.48	532	382	1.40

Type of Swap	Swap Latency (s)	Mcast 2nd Half	Ucast 2nd Half	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV ODMRP	1.813727	1689	0	4.42	2360	382	6.16
AODV APRL	0.001101	180	0	0.47	335	382	0.87
ODMRP AODV	0.001694	935	309	3.55	2925	351	8.36
ODMRP APRL	0.014353	180	218	1.14	2079	351	5.94
APRL AODV	0.001641	931	525	3.81	702	382	1.83
APRL ODMRP	1.813726	1688	0	4.41	3143	382	8.22

Table 13. 9nodes_square_light

No.	Data Packet Rate (sec/packet)	Topology	# Nodes	First Swap Time (sec)	Runtime (sec)
5	**1**	**Line Distance: 20m**	**49**	**100**	**200**

Type	1st Half Mcast	1st Half Ucast	1st Half Total Control Packets	1st Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	70585	4396	74981	7.77	153682	9654	15.92
ODMRP	80128	5004	85132	8.99	172239	9476	18.18
APRL	955	32486	33441	3.46	34818	9654	3.61

Type	2nd Half Mcast	2nd Half Ucast	2nd Half Total Control Packets	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	74297	4403	78701	8.15	153682	9654	15.92
ODMRP	82020	5087	87107	9.19	172239	9476	18.18
APRL	980	397	1377	0.15	34818	9654	3.61

Type of Swap	Swap Latency (s)	Mcast 2nd Half	Ucast 2nd Half	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV ODMRP	2.476105	87188	0	9.03	162184	9654	16.80
AODV APRL	25.014661	853	0	0.09	4396	9654	0.46
ODMRP AODV	1.235202	12523	1272	2.43	98927	5680	17.66
ODMRP APRL	27.043107	865	7599	0.89	93597	9476	9.88
APRL AODV	0.339797	58782	4637	7.33	37123	8656	4.55
APRL ODMRP	2.593889	69105	0	7.98	132523	8656	13.79

Table 14. 49nodes_line_heavy

No.	Data Packet Rate (sec/packet)	Topology	# Nodes	First Swap Time (sec)	Runtime (sec)
6	**5**	**Line Distance: 20m**	**49**	**100**	**200**

Type	1st Half Mcast	1st Half Ucast	1st Half Total Control Packets	1st Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	19613	1217	20831	10.40	42022	2002	20.99
ODMRP	16631	895	17526	8.88	35377	1973	17.93
APRL	955	31837	32792	16.38	33773	2002	16.87

Type	2nd Half Mcast	2nd Half Ucast	2nd Half Total Control Packets	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	19990	1201	21192	10.58	42022	2002	20.99
ODMRP	16849	1002	17852	9.05	35377	1973	17.93
APRL	980	2	982	0.49	33773	2002	16.87

Type of Swap	Swap Latency (s)	Mcast 2nd Half	Ucast 2nd Half	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV ODMRP	6.087929	16404	0	8.19	37246	2002	18.61
AODV APRL	22.016328	858	0	0.43	1217	2002	0.61
ODMRP AODV	6.655081	5797	596	4.80	23918	1331	17.77
ODMRP APRL	35.048456	832	18304	9.70	36661	1973	18.58
APRL AODV	1.950950	19800	1988	10.88	33825	2002	16.90
APRL ODMRP	6.635281	16433	0	8.21	38220	2002	19.09

Table 15. 49nodes_line_light

No.	Data Packet Rate (sec/packet)	Topology	# Nodes	First Swap Time (sec)	Runtime (sec)
7	**1**	**Square Distance: 30m**	**49**	**100**	**200**

Type	1st Half Mcast	1st Half Ucast	1st Half Total Control Packets	1st Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	66151	50914	117065	12.13	238651	9654	24.72
ODMRP	158643	7154	165797	17.50	341040	9476	35.99
APRL	147	9772	9919	1.02	11441	9654	1.18

Type	2nd Half Mcast	2nd Half Ucast	2nd Half Total Control Packets	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	68481	53106	121586	12.59	238651	9654	24.72
ODMRP	167691	7552	175243	18.49	341040	9476	35.99
APRL	147	1375	1522	0.16	11441	9654	1.18

Type of Swap	Swap Latency (s)	Mcast 2nd Half	Ucast 2nd Half	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV ODMRP	0.849875	173533	0	17.97	291296	9654	30.17
AODV APRL	0.021982	196	0	0.02	50914	9654	5.27
ODMRP AODV	4.890444	16228	15843	4.71	197868	6803	30.31
ODMRP APRL	16.032424	177	1797	0.21	167772	9476	17.70
APRL AODV	0.094207	42873	33965	9.97	43737	7706	4.95
APRL ODMRP	0.819795	173703	0	17.99	250542	9654	25.94

Table 16. 49nodes_square_heavy

No.	Data Packet Rate (sec/packet)	Topology	# Nodes	First Swap Time (sec)	Runtime (sec)
8	**5**	**Square Distance: 30m**	**49**	**100**	**200**

Type	1st Half Mcast	1st Half Ucast	1st Half Total Control Packets	1st Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	27558	17407	44965	22.46	90337	2002	45.13
ODMRP	45807	2309	48116	24.39	96698	1973	49.01
APRL	147	9668	9815	4.90	11309	2002	5.65

Type	2nd Half Mcast	2nd Half Ucast	2nd Half Total Control Packets	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV	27373	17999	45372	22.67	90337	2002	45.13
ODMRP	46229	2352	48582	24.62	96698	1973	49.01
APRL	147	1348	1495	0.75	11309	2002	5.65

Type of Swap	Swap Latency (s)	Mcast 2nd Half	Ucast 2nd Half	2nd Half Ratio	Total Control Packets	Total Tunnel Packets	Ratio
AODV ODMRP	2.492797	47502	0	23.72	92807	2002	46.36
AODV APRL	6.010896	188	166	0.18	17573	2002	8.78
ODMRP AODV	0.021537	27283	18386	23.15	93785	1973	47.53
ODMRP APRL	0.040672	196	6185	3.23	54497	1973	27.62
APRL AODV	0.195823	25391	16156	22.95	25824	1810	13.80
APRL ODMRP	3.162364	47117	0	23.53	88664	2002	44.30

Table 17. 49nodes_square_light

Exp No.	AODV			ODMRP			APRL		
	dest	1st half	2nd half	dest	1st half	2nd half	dest	1st half	2nd half
1	2029	2060	2152	7023	4821	5116	177	175	180
2	1082	900	935	1451	1321	1293	172	175	180
3	1317	453	430	8046	5655	5954	180	175	180
4	933	336	339	1688	1507	1478	180	175	180
5	35653	70585	74297	78147	80128	82020	859	955	980
6	12798	19613	19990	16418	16631	16849	845	955	980
7	29551	66151	68481	173618	158643	167691	186	147	147
8	26337	27558	27373	47309	45807	46229	192	147	147

Table 18. Multicast Packet Numbers Second Half

Exp No.	AODV			ODMRP			APRL		
	dest	1st half	2nd half	dest	1st half	2nd half	dest	1st half	2nd half
1	1833	2071	2004	0	733	786	56	265	0
2	674	620	717	0	206	190	127	247	0
3	933	1865	1826	0	635	669	55	172	0
4	417	335	549	0	174	172	109	177	0
5	2955	4396	4403	0	5004	5087	3800	32486	397
6	1292	1217	1201	0	895	1002	9152	31837	2
7	24904	50914	53106	0	7154	7552	899	9772	1375
8	17271	17407	17999	0	2309	2352	3176	9668	1348

Table 19. Unicast Packet Numbers Second Half

Exp No.	AODV			ODMRP			APRL		
	dest	1st half	2nd half	dest	1st half	2nd half	dest	1st half	2nd half
1	2.20	2.31	2.33	3.93	0.14	3.43	0.14	0.25	0.10
2	4.78	3.80	4.32	3.80	4.35	4.23	0.84	1.10	0.48
3	1.28	1.30	1.26	4.50	3.65	3.84	0.14	0.20	0.10
4	3.68	4.42	2.32	4.42	4.79	4.71	0.81	0.92	0.48
5	4.88	8.51	8.15	8.51	8.99	9.19	0.49	3.46	0.15
6	7.84	10.40	10.58	8.20	8.88	9.05	5.07	16.38	0.49
7	7.34	12.13	12.59	17.98	17.50	18.49	0.12	1.02	0.16
8	23.05	23.63	22.67	23.63	24.39	24.62	1.71	4.90	0.75

Table 20. Control Packets Ratio

Appendix D Plots

This Appendix showed the what happened after swap for each type of swap. We compared the network traffic after swap with the destination protocol we swap to. For example, Figure 15 is the swap from ODMRP to AODV (Exp 2[10]. Plots[11]). It is composed of 4 small bar charts. The first tow are unicast and multicast controls packets numbers after the swap (90 sec – 150 sec, and swap time is at 100 sec). The third one is the total control packets. The last one is the total control packets of the destination protocol running 200 seconds. Compared the third and the fourth one, we can see directly the whole network traffic changes because of the swap.

[10] Appendix C has the corresponding data in experiment 2.
[11] Appendix D only shows the plot of experiment 2. This is the average of 5 runs of experiments with same network figuration except the random seed for SWAN.

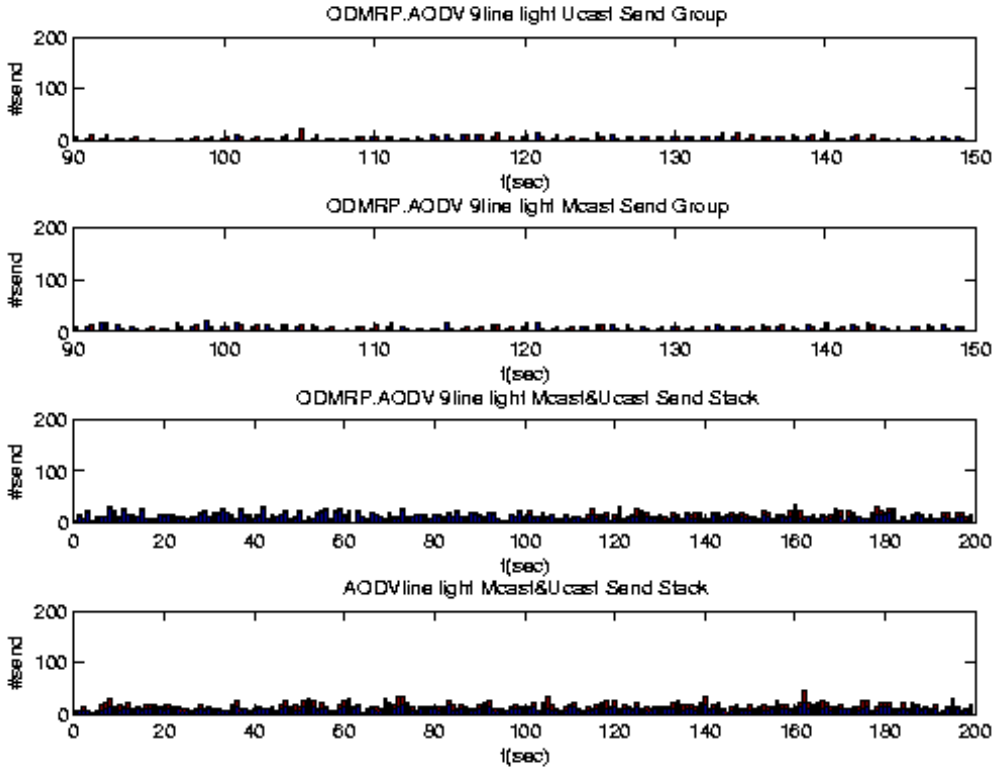

Figure 15. ODMRP to AODV

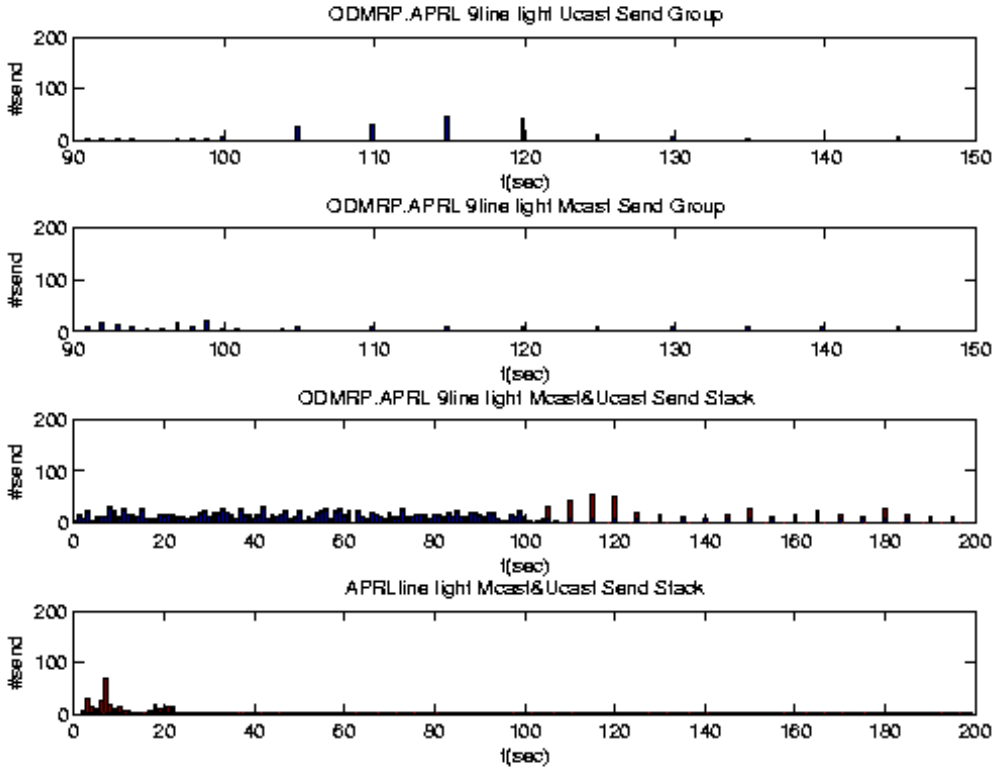

Figure 16. ODMRP to APRL

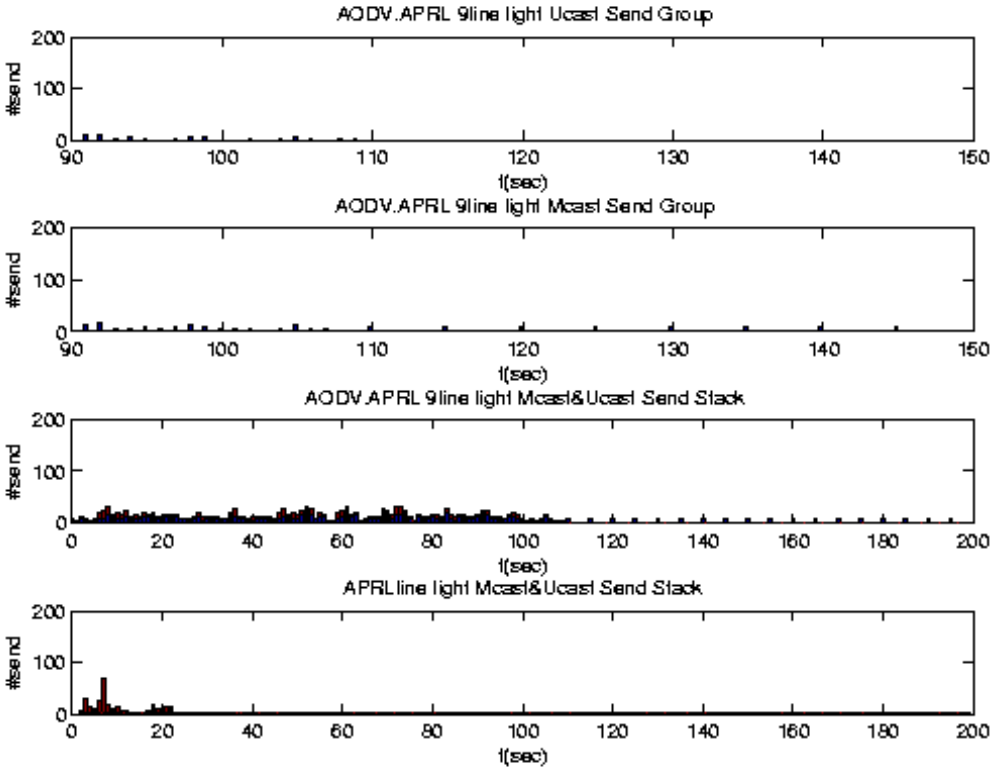

Figure 17. AODV to APRL

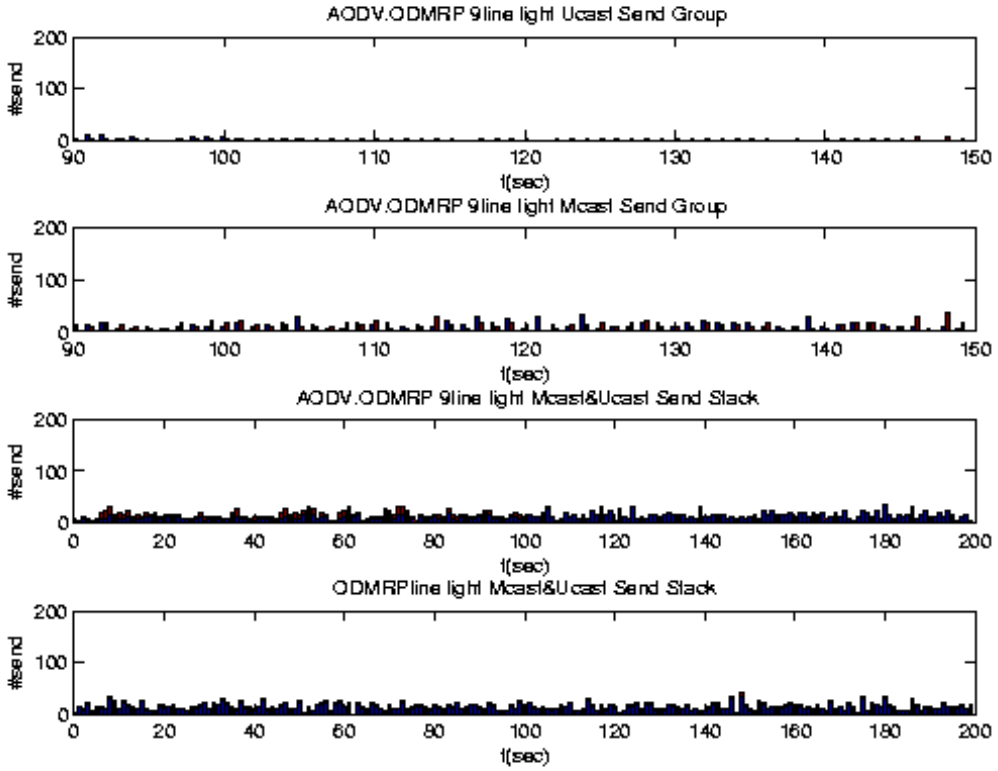

Figure 18. AODV to ODMRP

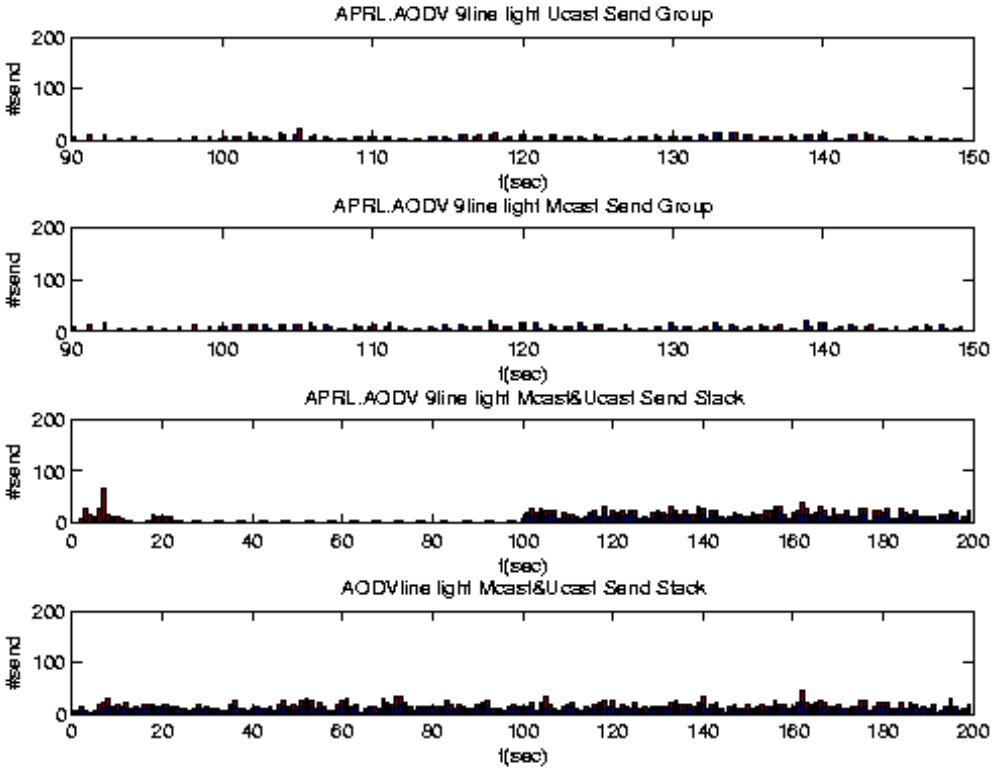

Figure 19. APRL to AODV

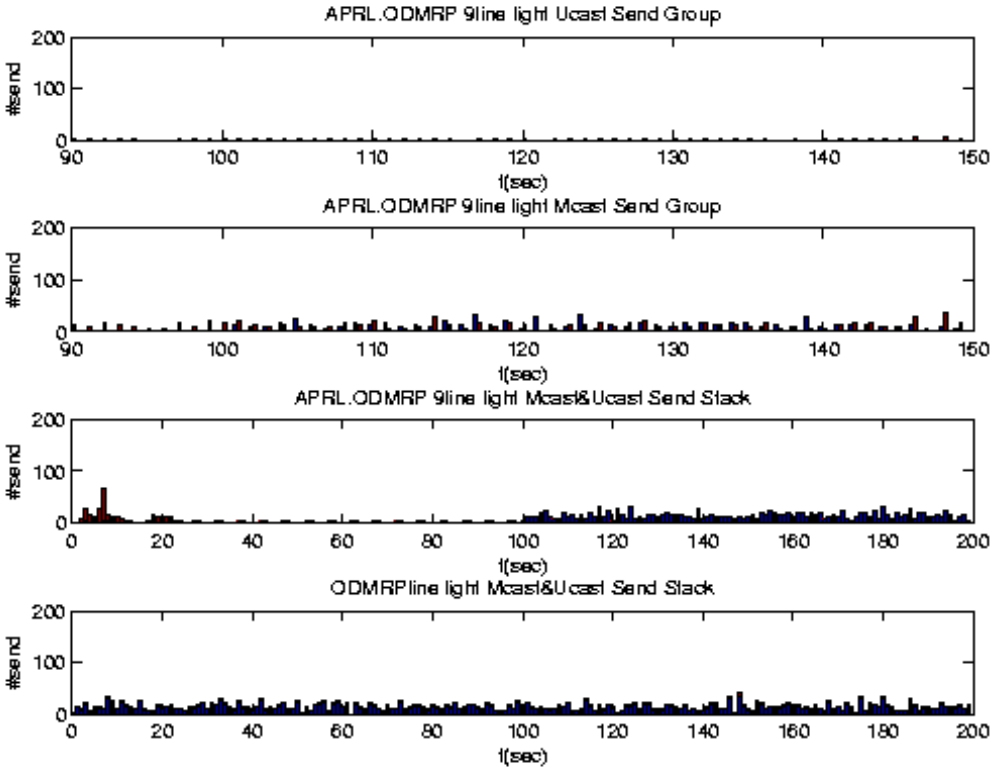

Figure 20. APRL to ODMRP

Reference

[ActComm] ActComm Project. http://actcomm.dartmouth.edu

[Bae2000] S. H. Bae, S. Lee, W. Su, M.Gerla, "The Design, Implementation, and Performance Evaluation of the On-Demand Multicast Routing Protocol in Multihop Wireless Networks", IEEE Network Magazine, vol. 14, no. 1, Jan./Feb. 2000, pp. 70-77.

[Belding2003] E. Belding-Royer, "Multi-level hierarchies for scalable ad hoc routing", Wireless Networks, Vol. 9, Issue 5, 2003, pp. 461-478.

[Bommaiah1998] E. Bommaiah, M. Liu, A. McAuley, and R. Talpade, "AMRoute: Adhoc Multicast Routing Protocol", Internet-Draft, draft-talpade-manetamroute-00.txt, Aug. 1998; Work in progress.

[Broch1998] J. Broch, D. Maltz, D. Johnson, Y. C. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols", Proc. of MobiCom 1998, Oct. 1998, pp. 85-97.

[Chen2003] Y. P. Chen, A. Liestman, "A zonal algorithm for clustering ad hoc networks", International Journal of Foundations of Computer Science, 14(2), 2003, pp. 305-322.

[DaSSF] Dartmouth SSF, http://www.crhc.uiuc.edu/~jasonliu/projects/ssf/

[Garcia1999] J.J. Garcia-Luna-Aceves and E.L. Madruga, "The Core-Assisted Mesh Protocol", IEEE Journal on Selected Areas in Communications, vol. 17, no. 8, Aug. 1999, pp. 1380-1394.

[Gerla2000] S. Bae, S. Lee, M. Gerla, "Unicast Performance Analysis of the ODMRP in a Mobile Ad Hoc Network Testbed", Proceedings of IEEE International Conference on Computer Communications and Networks (ICCCN), Oct. 2000, pp. 148-153.

[Gray2004] R. Gray, D. Kotz, C. Newport, N. Dubrovsky, A. Fiske, J. Liu, C. Masone, S. McGrath, "Outdoor Experimental Comparison of Four Ad Hoc Routing Algorithms", In Proceedings of the Seventh ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM 2004), Venice, Italy, Oct. 2004.

[Gupta1997] P. Gupta and P. Kumar, "A system and traffic dependent adaptive routing algorithm for ad hoc networks", in Proc. IEEE 36th Conf. on Decision and Control, San Diego, CA, Dec. 1997, pp. 270-283.

[Gupta2000] P. Gupta, "Design and Performance Analysis of Wireless Network", Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana-Champaign, Aug. 2000.

[Haas2001] Z. Haas and M. Pearlman, "The Performance of Query Control Schemes for the Zone Routing Protocol", IEEE/ACM Trans. Networking, vol. 9, no. 4, Aug. 2001, pp. 427-438.

[Hoebeke2004] J. Hoebeke, I. Moerman, B. Dhoedt, P. Demeester, " Adaptive Multi-mode Routing in Mobile Ad Hoc Networks", Proceedings of the 9th International Conference on Personal Wireless Communications, Delft, The Netherlands, Sep. 2004, pp. 107-117.

[Karp1998] B. Karp, H. T. Kung, "Dynamic Neighbor Discovery and Loop- Free, Multi-Hop Routing for Wireless Mobile Networks", Harvard University, May 1998. Draft.

[Lee2000] S. Lee, W. Su, M. Gerla, "On-Demand Multicast Routing Protocol in Multihop Wireless Mobile Networks", ACM/Baltzer Mobile Networks and Applications, special issue on Multipoint Communication in Wireless Mobile Networks, 2000.

[LeeW2000] S. Lee, W. Su, J. Hsu, M. Gerla, R. Bagrodia, " A Performance Comparison Study of Ad Hoc Wireless Multicast Protocols", Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), Tel Aviv, Israel, Mar. 2000, pp. 565-574.

[Liu2001] J. Liu, D. Nicol, "DaSSF 3.1 User's Manual", Apr. 2001.

[Liu2004] J. Liu, Y. Yuan, D. Nicol, R. Gray, C. Newport, D. Kotz, L. Felipe Perrone, "Simulation Validation Using Direct Execution of Wireless Ad-Hoc Routing Protocols", 18th Workshop on Parallel and Distributed Simulation (PADS'04), Kufstein, Austria, May 2004, pp. 7-16.

[Navid2001] N. Navid, S. Wu, C. Bonnet, "HARP: Hybrid Ad hoc Routing Protocol", International Symposium on Telecommunications (IST 2001), Teheran, Iran, Sep. 2001.

[Perkins1999] C. Perkins, E. Royer, and S. Das, " Ad hoc on demand distance vector routing", Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, February 1999, pp. 90-100.

[Rama2003] V. Ramasubramanian, Z. Haas, E G. Sirer, "SHARP: A Hybrid Adaptive Routing Protocol for Mobile Ad Hoc Networks", In Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC), Annapolis, Maryland, June 2003. pp. 303 – 314.

[RFC2501] S. Corson, J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations," RFC 2501, Jan. 1999.

[RFC3561] C. Perkins, E. Belding-Royer, S. Das, " Ad hoc On-Demand Distance Vector (AODV) Routing", RFC 3561, July 2003.

[Royer1999] E. Royer and C-K. Toh, "A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks" , IEEE Personal Communications, Vol. 6, April 1999, pp. 46-55.

[SSF] Scalable Simulation Framework. http://www.ssfnet.org/homePage.html

[SWAN] Simulator of Wireless Ad hoc Networks. http://www.eg.bucknell.edu/swan

[Toh2002] C-K. Toh, "Ad Hoc Mobile Wireless Networks: Protocols and Systems", Prentice Hall, 2002.

[Wu1998] C.W. Wu, Y.C. Tay, C-K. Toh, "Ad hoc Multicast Routing protocol utilizing Increasing id-numberS (AMRIS) Functional Specification", Internet-Draft, draft-ietf-manet-amris-spec-00.txt, Nov. 1998, Work in progress.