Improving Fault Tolerance in AODV
CS 497RHK
Final Report

Matthew J. Miller
mjmille2@uiuc.edu

Jungmin So
jso1@uiuc.edu

***Abstract*— While ad hoc routing research is plentiful, two protocols have as the most popular and widely tested. They are Dynamic Source Routing (DSR) and Ad hoc On demand Distance Vector routing (AODV). In this paper, we look at methods to improve fault tolerance in AODV. This is desirable because in ad hoc settings mobility may cause routes to break frequently and hence alternate paths are needed. We consider two approaches to the problem. The first tries to achieve fault tolerance in an end-to-end fashion with the data source responsible for repairing routes. The second approach tries to achieve the same goal by maintaining multiple routes at intermediate nodes on a path to achieve better results. Simulations show these changes do not greatly affect the performance of AODV. We therefore conclude that while the optimizations may seem useful, they are not worth the added complexity of implementation in the current version of AODV.**

1 Introduction

As wireless systems have become more prevalent, research has exposed many new directions where existing wired network solutions cannot be applied. The many differences include potentially dynamic physical topologies, higher bit error and packet loss rates, the need to conserve energy, possible lack of centralized services and different MAC protocols. A rich field that has emerged is that of ad hoc routing. An ad hoc network is one in which mobile hosts (MHs) communicate with each other over wireless links without the benefit of a centralized authority. Routing is important in such networks because paths can change quickly, so efficient route discovery and maintenance is necessary. However, because bandwidth is scarce and MAC competition relatively high, the amount of routing overhead must be minimized as much as possible.

There are two general approaches to the problem. Proactive protocols attempt to maintain routes to destinations regardless of connection status. Reactive protocols try to find routes only when a source needs a path to a specific destination. The major tradeoff in the two approaches is large amount of overhead in proactive protocols versus the performance degradation in packet latency and delivery in reactive protocols. Particularly, when there is high mobility, links can break frequently and reactive protocols are forced to drop many packets due to the lack of a valid route and delay sending packets while a new route is being discovered. In lieu of this tradeoff, simulation studies [1] have shown reactive protocols generally perform better. Furthermore, two reactive protocols in particular have emerged as the standard against which all others are measured. The protocols, Ad hoc On demand Distance Vector Routing (AODV) [2] and Dynamic Source Routing (DSR) [3], share many similarities, particularly in their route discovery mechanism, but also have many glaring differences. These protocols have the most complete specifications and have undergone more extensive analysis and testing than any other. Therefore, it is reasonable to assume

the ultimate ad hoc routing solution will strongly draw on characteristics of these protocols. We propose to enhance the AODV protocol with some favorable aspects of DSR. By extending these protocols, we start with a well–studied foundation.

DSR uses source routing, so all data packets contain the complete path for a packet to use. AODV, however, just uses next hop information to route packets based on the destination in the header. We believe the AODV paradigm is more likely to be implemented widespread because it fits well in the routing model systems currently use. Specifically, operating systems maintain a forwarding table for data packets which, based on the IP destination, send the packet to the appropriate next hop. Such a procedure has been well–studied in wired networks and has led to quick lookups and route aggregation. In contrast, source routing requires parsing through the packet header for the next hop, does not lend itself readily to aggregation and increases the IP overhead linearly with the length of the path. However, performance studies [4] have shown DSR tends to be more efficient than AODV. One of the major factors in DSR's performance is its ability to maintain multiple paths to destinations during the route discovery phase. This is critical in performance because route discovery is very expensive, potentially requiring a network–wide broadcast. During this process, many MHs may reply to the request. DSR lessens the expense of route discovery by storing multiple paths that are learned in this process. Therefore, when a route breaks, DSR could possibly find alternate routes and use those as opposed to initiating a new route request (RREQ) broadcast. AODV, however, must waste this extra information learned from the route replies (RREP). Therefore, when a link breaks, AODV *must* initiate a new RREQ broadcast.

We propose to augment AODV to take advantage of the multiple paths that can be learned during a route discovery. Two approaches are possible. One is to have the source node maintain more information on the paths to destinations, so that when a link error is detected, the source can use that information to route the packet via another node which has a route to the ultimate destination. This makes end hosts responsible for maintaining routes to the destination. Thus, if a hosts determines an error has occurred it can decide whether to try to find another route to the destination or just give up on the communication. The other approach is to have intermediate routers maintain more information on the route to the destination, so that when a link error occurs, the router would be able to route the packet through an alternate route.

The rest of the paper is divided as follows. In Section 2, we will discuss previous work that has been done in ad hoc routing and fault tolerance in this environment. In Section 3, we describe the design of our protocols. Specifically, Section 3.1 discusses a source oriented recovery approach, while Section 3.2 presents an approach which makes use of intermediate nodes. Section 4 presents the simulation results obtained by the protocols when compared to AODV. In Section 5 we will conclude the paper. Finally, the work division for the project is presented in Section 6.

2 Related Work

As mentioned, there has been a multitude of research done in the area of ad hoc routing [5]. Many protocols have been proposed that address different problems, such as stability and mobility. Many protocols are fairly complex, but they do not discuss how to handle common irregularities such as control packets being lost or how to function when a node's routing state is lost either through timeouts or system reboots. DSR and AODV, however, have been rigorously tested and deal with these issues. Furthermore, both of them have existing implementations, whereas most others have just been tested under the simplifying assumptions of a simulator. Therefore, by extending these protocols, they can more readily be integrated into real-world systems.

The tradeoffs in reactive and proactive protocols have typically been addressed by creating hybrid protocols which attempt extract the benefits of each. One of the best performing protocols in the hybrid category is Adaptive Distance Vector (ADV) [6] which takes a proactive protocol and makes it more reactive by only allowing routing information to be exchanged for active destinations. It also greatly de-

creases the amount of proactive routing overhead by varying the frequency of the routing updates based on network mobility and node queue lengths. Additionally, it allows for partial updates of a subset of active destinations. Simulations show ADV outperforms not only proactive protocols, but AODV and DSR as well, in many important metrics. However, the approach requires a massive amount of state to be maintained and does mention that convergence in learning of active receivers could be slow if packets (or state) is lost. Additionally, all of the results presented are for high mobility scenarios.

There have been previous attempts to add fault tolerance to AODV. In [7], the authors attempt to decrease the latency and packet loss in AODV during a route error (RERR) by allowing nodes to promiscuously listen to RREPs and passively keep track of routes to the destination. When a link break does occur, the node on the primary route broadcasts the data packet with a TTL of one. If one of the neighbors has a passive route to the destination, it can then unicast the packet. Concurrently, the primary node sends a RERR back to the source, who then initiates a new route discovery. The simulations demonstrate this approach suffers when the number of connections and mobility is high because many neighbors will try to broadcast data packets and the collisions will significantly decrease performance. There are a few more problems with the approach. Additional header processing and implementation issues arise to allow intermediate nodes to forward some broadcast data packets from an alternate routing table. Additionally, they require nodes do not forward packets which are duplicates. While the is easy in simulation, it becomes non-trivial in a real-world implementation. Additionally, the amount of overhead for RREQs is not decreased. In contrast, our approach will not require any special processing for data packets, nor does it require nodes to promiscuously listen to unicast packets. Additionally, their approach does not take advantage of multiple paths.

In [8], the authors do address the issue of how more information can be learned from RREPs to maintain multiple routes to a destination. They allow for intermediate nodes to learn of link and node disjoint paths between a given source and destination. They maintain multiple next hops for a given destination that can be used to forward packets when a link break is detected. They show in mobility, their protocol is able to reduce the packet delay drastically and also reduce the routing load. However, the paper leaves some unresolved issues such as how the alternate paths are exercised. Since the entries for a given destination will time out eventually, the alternate paths will not always exist because routing entries are refreshed based on how recently they have been used for data forwarding. Additionally, they require the destination to generate an arbitrary number of RREPs for RREQs arriving from different neighbors. Also, the protocol requires intermediate nodes to reply with a maximum hop count to a destination, then only accept alternate routes that are shorter for a given destination and sequence number. However, this requires an intermediate node to wait before forwarding an RREP. Typically, the route with the shortest hop count would be received first at an intermediate node, so this node would have to collect alternate paths with longer hop counts before forwarding the RREP because once the packet is forwarded, later arriving, longer hop paths cannot be accepted.

The protocol described in Section 3.1 was influenced by the idea of virtual links used in overlay networks. This concept is used for multicast [9, 10] and resilience when BGP routers fail [11]. For resilient overlay networks, sources keep track of multiple application layer paths to reach a destination in the event a large performance decrease is detected at the network layer path. We believe this idea is applicable to ad hoc networks and hope our method leads to further ideas which can apply virtual links to ad hoc networks.

3 Protocol Design

3.1 Source Oriented Recovery

The basic idea of this approach is the route discovery process is expensive due to its broadcast nature. However, it will also usually provide nodes extra information about the topology aside from the shortest

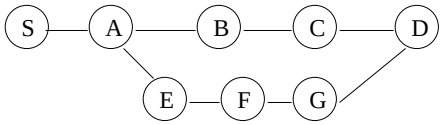

Figure 1: An Example Topology

path to the destination. In particular, multiple paths may be learned for a given destination. While DSR does cache multiple routes, AODV has no such mechanism due to its next-hop nature. In Figure 1, **S** can learn of two routes to **D**. However, all **S** will know about the routes is they use **A** as a next hop and **S** will not be able to control which path is used. In DSR, the entire paths would be learned and placed in each data packet. Therefore, **S** could control which path is being used and have more information about the path the data will travel on. Furthermore, this approach allows intermediate nodes to only maintain a small amount of state for each source that uses it. In DSR, the intermediate routers do not have to maintain any state per source since the data packet carries all the routing information (optimizations such as route caching may require intermediate nodes to maintain some state per source). In AODV, each intermediate node maintains at most one entry per flow (source-destination pair). Previous approaches to fault tolerance in AODV [7, 8] require either intermediate nodes to keep track of potentially multiple entries per flow or require nodes not on the primary path to keep track of flow entries. In contrast, our approach places the burden for maintaining extra state on the source of the flow. As in AODV, each intermediate node on the primary path only maintains at most one entry per flow. The source will maintain multiple routes learned in the route discovery process and be responsible for using alternate paths when an error is detected.

To achieve source oriented fault tolerance, we first observe that while source routing each data packet is expensive, we can add route information to control packets at little expense. Additionally, the routing daemon can then potentially emulate source routing when necessary by having control packets which give the entire path along which the control packet should be sent. Therefore, when the RREQ is broadcast, each node is required to add its address to the packet. If the RREQ reaches the destination, the path information is then attached to the RREP being sent back to the source. Nodes which receive the RREP will only cache the path information if the destination is one which they are actively sending to. If intermediate nodes respond to the RREQ with cached information, the path information is not included in the RREP since the intermediate nodes would not have path information to the destination unless they are also sending to that specific destination. An optimization to the protocol would be to allow intermediate nodes to include the path information if they are sending to the requested destination. In our simulations, the number of flows was never greater than the number of nodes so the situation where more than one node was sending to the same destination did not arise frequently.

In AODV, each node will only respond at most once to a specific RREQ. This is desirable because it avoids the RREP storm problem that may occur in DSR. We modified the protocol to behave according to the DSR specification. The protocol still allows intermediate nodes to reply at most once to a specific RREQ, but the destination may respond multiple times. In DSR, there is no limit to how many times the destination may respond. However, simulations showed that the destination tends to receive many RREQs and hence generates a plethora of RREPs. Furthermore, the performance gain from multiple RREQs plateaued relatively early. This means most of the RREPs would just create extra overhead in the system with no gain. Therefore, we capped the number of RREPs a destination may generate at three. This technique is also used in [8]. All the RREPs generated by the destination are routed back via the shortest path the destination has to the source.

When a source receives RREPs, it will behave like AODV in that the shortest path will be used. However, the k shortest routes obtained are also cached with the path information (if present) and all the information necessary to add a forwarding table entry. Alternatively, many other factors could be considered aside from shortest path, such as the link-disjointness

of the path or any number of other metrics.

One aspect of AODV we have neglected to mention thus far is its use of sequence numbers. These are used to provide loop freedom in many cases. When nodes update information about a given destination, they can only do so if the sequence number is equal to or larger than the last known sequence number for the destination. This serves to give some ordering to the events that occur in the system. Whenever a destination generates a new RREP, it will increment its sequence number. In our protocol, we follow this same methodology for providing some amount of loop freedom. When a source overhears information about a destination via some control packet, it *must* replace all existing information about the destination if the sequence number in the control packet is larger than what was previously known. While this may cause us to lose path information frequently, it is more important to have loop freedom and some measure of a route's "freshness" in our protocol.

Now we have stored additional information during the route discovery process, we must address how to use this information when the primary path detects an error. When a link break occurs, a RERR is generated with the broken link specified. When a node receives the RERR and it makes one of its destinations become unreachable, the node will check for an alternate path it may know of to the destination. If one exists, it will sent a PROBE message to the destination along that path. The PROBE message is essentially source routed along the path to the destination. Since we are not using HELLO messages in our implementation, a node may not have a routing entry for the neighbor specified in the source route. If this occurs, the node broadcasts the PROBE packet with a TTL of one. Only the node whose address is next on the source route will process the PROBE Each node that processes the PROBE message sets up a reverse pointer to the source as is done when processing a RREQ. If the destination receives the PROBE, it responds with a RREP to the source. This RREP is handled as in AODV and sets up a new path between the source and destination without having to do a broadcast route discovery. After alternate paths have been used a specified number of times, a new route discovery process will be initiated instead of the PROBE process to allow the node to discover a fresh path. Additionally, a timer is associated with each PROBE message sent. If the node has not received a RREP when this timer is fired, it will initiate a route discovery process. An additional optimization would be to piggyback data packets on these PROBE packets to allow data to reach the destination while the path is being set up.

Referring back to Figure 1, we will demonstrate how the protocol would behave. First, **S** broadcasts a RREQ searched for **D**. **D** will receive this RREQ along two separate paths and hence send one RREP with the path **S-A-B-C-D** and one with the path **S-A-E-F-G-D**. **S** will choose to use the first path since it has one less hop. Now, assume the link between **B** and **C** breaks. **B** will send a RERR indicating **D** is unreachable because the **B-C** link broke. When **S** gets the RERR, it will search its route cache for a path which does not have the **B-C** link and then place the **S-A-E-F-G-D** path in a PROBE message. When the PROBE message reaches **D**, a RREP will be sent back to **S** and the new path will be established. Intermediate nodes cannot respond to this PROBE message with a cached route to **D** because it probably wouldn't know if its path to **D** includes the **B-C** link since the path information would not be cached at most intermediate nodes.

To summarize, the major changes made to the packets of AODV are:

- Add path information and path length in RREQ and RREP packets.
- Include the broken link information in RERR packets.
- Add the PROBE packet to the protocol. This packet contains a path and path length as well as the source and destination nodes, their respective sequence numbers and the hop count.

The major changes made to the protocol logic was the extra processing to cache path information for a given node's destinations and the sending and handling of probe packets.

3.2 Intermediate Router Recovery

In AODV, the source node performs a route discovery whenever it does not have a route toward the destination, or the route that it was maintaining fails. When performing a route request, the source broadcasts an route request (RREQ) packet, so that the destination or an intermediate node that has a route to the destination can reply to the source with route reply (RREP) packet. Broadcasting a packet causes contention in the network, and is very expensive in terms of overhead. So we want to avoid broadcasting as much as possible.

One observation we make is that the source can learn multiple paths to the destination when performing route discovery. But in original AODV, only one path is maintained at the source and at each router. So if the route breaks, due to mobility or other reasons, the source would have to perform route discovery again. For example, suppose we have a set of nodes placed as in Fig. 2. Node S wants to send packets to node D. During route discovery, node S can learn two paths to D, S-A-B-C-D and S-A-E-F-D. But in original AODV, A only maintains one route for D, so only path S-A-B-C-D can be used for delivering packets toward B. If the link between B and C breaks, S would have to perform route discovery again, because it no longer knows a path to reach D. But if A has maintained both A-E and A-B as routes toward D, S might be able to use S-A-E-F-D instead of performing an expensive route discovery.

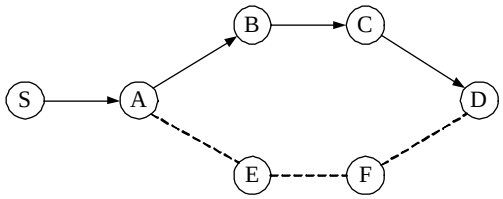

Figure 2: An example scenario in AODV. Each node only maintains one route for node D. The solid line indicates the route from node S to node D.

So we want to let nodes maintain multiple routes for each destination, as in Fig. 3. In Fig. 3, the route A-E is maintained as a secondary route toward D. If the primary route works ok, everything is fine and the secondary route is not used. If the primary route breaks and a node detects that it cannot forward a packet through the route, it tries to forward the packet to the secondary route if it has one.

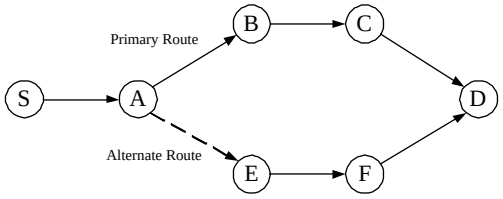

Figure 3: An example scenario in AODVM-R. Node A maintains two routes for node D. The route from A to E is maintained as a secondary route toward D.

The proposed protocol, AODVM-R, modifies the original AODV protocol to allow each node to maintain multiple routes for each destination. A similar approach is taken in AOMDV [8], but our scheme has several differences from AOMDV. First, in AODVM-R, the primary route and the secondary routes are separated. The primary route is always selected, unless the node detects that this route is broken. This is important because if a node has multiple routes for a destination, all the routes have different hop counts toward the destinations. When node C replies to node B with its route information, C should tell its neighbor what is C's distance (hop counts) from the destination. In AOMDV, the maximum hop count of the routes is chosen to be the "advertised hop count". Also, when a node receives a route with the same destination sequence number, the route is accepted only if it has a lower hop count than the advertised hop count. But as mentioned in section 2, an intermediate node needs to wait for some time to gather multiple RREPs, before it sends out its RREP. In our scheme, hop count of the primary route is the advertised hop count. An intermediate node does not need to wait, but can forward the first RREP packet that arrives. All the subsequent RREPs only contribute to establishing alternate routes at the node, and are not forwarded. Second, keeping the alternate routes fresh is not addressed in AOMDV. We address this issue by

introducing REFRESH messages. In Fig. 3, if the primary path S-A-B-C-D is used for long time, eventually the routes at E and F will experience timeout and be discarded from the routing table. So by the time the primary route breaks, the alternate paths might not be available for use. To avoid this problem, we need some ways to refresh these routes. Using REFRESH messages to keep the alternate routes fresh is described later in this section.

Now we described our protocol, *AODV Multipath - Router Approach* (AODVM-R), in detail. Referring again to Fig. 3, suppose node S wants to send packets to node D. Since S does not know how to forward packets to D, it goes through a route discovery process. S broadcasts RREQ packets, and the RREQ packet is forwarded by the intermediate node until it reaches D. Now D will receive RREQ packets from both its neighbors, C and F. D sets up a reverse path to both neighbors. Now since D is the destination, it sends RREP packet back toward S. It sends RREP packets to all the reverse paths it has. Intermediate nodes also forward RREP packets to all the reverse paths they have. Now node A will receive two RREP packets, with the same sequence number, one from B and one from E. If RREP from B came first, B becomes the next hop in the primary route of A. This RREP packet is forwarded toward S. The RREP from E comes later, and E becomes the next hop in a secondary route of A. This RREP packet is not forwarded, because A has already forwarded an RREP packet to S.

If the node density is high, there can be a lot of multiple paths that can be learned from route discovery. If each node maintains all the routes it has learned, the storage overhead will become too high. Also, it is not likely to benefit from having so many alternate routes. So we only maintain up to k routes per destination at each node. All the routes after kth route are discarded.

When maintaining multiple routes, it is important that we make sure there is no loop in the paths. If each node maintains multiple reverse paths toward the source during route request, it is possible that a loop is formed inside a path to the source. To avoid loops, we have RREQ packets include the whole path information from source to the router, so that the router will be able to figure out if this route is making a loop or not. So when forwarding an RREQ packet, each router appends its address to the RREQ packet. When a node receives an RREQ packet, it checks if the reverse path can potentially form a loop. If the path included in the RREQ packet includes the node itself, the node simply ignores the RREQ packet. If the reverse path is not making any loop, the node can maintain the reverse path, even if it already has another reverse path toward the original source.

Now we have made the routers to maintain multiple routes if it learns of multiple routes during route discovery. As long as the primary route is working, the alternate routes are not used. If a link breaks, nodes at the edge of the broken link detect the link breakage. This is done either by link layer detection or by absence of hello messages. The nodes discard any route that uses the broken link. If any destination becomes unreachable because of the broken link, the node sends out route error (RERR) messages, so that other nodes may discard the paths that have been broken. In Fig. 3, suppose the link between B and C fails. on detecting the link breakage, B will find out that D is no longer reachable from B. So B sends out an RERR to its neighbors saying it is no longer able to reach D. When A receives this packet, A knows that D is no longer reachable through B. In AODV, after the primary route is broken, A does not have a route to D, so it propagates the RERR message toward S. Then S will start a route discovery process to find a new route for D. But in AODVM-R, A has another route for D. So A replaces the broken primary route with the alternate route, and E becomes the next hop in a route for D. If this alternate route is not broken, then D is still reachable from A and so A does not have any unreachable destination. So A stops forwarding RERR messages. Since A does not get an RERR message from B, it does not even know if a route had been broken, and it can continue to send packets without initiating another route discovery process.

One important issue to consider in AODVM-R is the timeout associated with alternate routes. Suppose a primary path has been used for a long time and then a link on the path breaks. Then it is likely that the nodes on the alternate paths have not been able

to exercise the route to the destination and would have the entry timed out by the time of link breakage. For example, in Fig. 3, E maintains a route to D after S performs a route discovery, but if the primary path S-A-B-C-D works for a long time, the route entry in E's route table will timeout because it did not have a chance to be exercised for a long time.

To avoid this alternate path timeout problem, we use REFRESH packets to refresh the alternate routes, while the primary route is being used. Sending refresh messages on the alternate routes is called *refresh event*, and the event can be initiated by the source or any intermediate routers. When a node learns a new route the a destination, it starts an *active route timer* associated with the entry. The timer is set to a predefined fixed value. If the route is not exercised for the certain amount of time, the timer expires and the route will be discarded from the route table. In AODVM-R, we maintain an additional timer, called the *refresh timer*. The timer is also initiated when a new route is learned at the router. The refresh timer is set to a half of the active route timeout interval. So for example, if the active route timeout value is 10 seconds, the refresh timer is set to 5 seconds. As the primary route is used, the active route timer is reset, but the refresh timer stays the same. After 5 seconds, the refresh timer of a route will timeout. If a packet uses the route after the refresh timer has been timed out, the node initiates a *refresh event*. In Fig. 3, node A forwards all the packets destined for D through B, because it is the next hop on the primary route. After 5 seconds, the refresh timer of the entry will timeout. If a packet destined for D after 5 seconds, A forwards the packet to B, but sees that the refresh timer has expired and initiates a refresh event.

In the refresh event, the node will transmit a REFRESH packet to the alternate routes it is maintaining. So node A transmits a REFRESH packet to E. The REFRESH packet will refresh the routes on the alternate paths, until it reaches the destination. When forwarding REFRESH packets, if a node sees an link error on the alternate paths, the node discards the route entry from the route table. For alternate routes, the node doesn't generate RERR packets, because it still has the primary route. This REFRESH packets are main sources of overhead in AODVM-R. So the period of sending REFRESH packets on alternate routes should be carefully chosen not to put too much overhead to the traffic while enough to avoid unused alternate paths from timing out. Note that refresh events are different from proactive message exchange, since it is performed only when the primary path is in use.

The major characteristics of AODVM-R can be summarized as follows.

- When performing route discovery, the source and intermediate nodes maintain multiple routes to the destination.
- If a node detects link error, it tries to repair the path using alternate routes instead of generating error messages. If it can use the alternate routes, the route is repaired without initiating another route discovery. If the node has no alternate route, it performs the same way as the original AODV.
- To exercise the alternate routes, a node periodically initiates refresh events, on the path that is being used. In the refresh event, a node transmits REFRESH packets to the alternate routes, so that the alternate routes on the path to the destination can be exercised.

4 Experimentation and Results

4.1 Source Oriented Approach (AODVM)

To test our protocols, we implemented them in *ns*-2 [12]. The mobility model used was the random waypoint whereby a mobile node chooses a random destination point and a speed up to a maximum specified value. Once the node has reached the destination, it will pause for a specified amount of time and then repeat the process. The radio interface uses the 802.11 MAC protocol and has a data rate of 2 Mbps and a range of $250m$.

Each simulation consisted of 25 nodes in a $750m \times 150m$ area. The traffic was generated such that a

specified number of senders choose a random destination and send *constant bit rate traffic* (CBR) beginning at a randomly selected time. The CBR traffic was set to a rate of 4 packets per second, where each packet was 1000 bytes. A specified number of sources pick a random destination and begins sending data at some random time uniformly distributed between the interval [0, 150] seconds. When we refer the the number of connections in these tests, we are referring to the *maximum* number of connections possible in the given scenario (rather than the exact number of connections).

Each simulation was ran for 300 seconds. Each point in the graphs represents the average over ten simulation runs with the specified input parameters. The input parameters were varied to determine the effects of increasing the amount of traffic (Section 4.1.1) and of increasing mobility (Section 4.1.2 and Section 4.1.3). We then compared AODV and our protocol (referred to as AODVM since it is the Modified version) with respect to the following metrics.

Average Packet Delay This measures the delay experienced by each data packet which successfully reaches its destination. It is calculated by recording the time a data packet is sent from its source and when it is received by the destination. Packets dropped in transit, for whatever reason, are not considered in this metric.

Packet Delivery Fraction This metric calculates how many data packets generated by a source actually reach the destination. Packets can be dropped due to an unreachable route or a full queue, for example. It is calculated by dividing the number of packets received by their intended destinations by the total number of packets generated in the system.

Route Discoveries per Second This is an indication of how many route discoveries the protocol induces. To calculate this, we simply record the number of times a route discovery is initiated divided by the total time of the simulation during which data is being sent.

Overhead to Data Packet Ratio This calculates how many control packets are generated as a function of the number of data packets. Control packets include RREQs, RREPs, RERRs and PROBEs. An alternative approach would be to measure the ratio by bytes instead of packets. However, since the MAC and energy cost of sending a packet generally outweighs the cost of a few extra bytes, we chose to look at the packet ratio.

In the simulations, link layer loss detection was used, local repair was not used and ARP was done statically. When ARP was done dynamically, there were excessive drops due to multiple RREPs being generated by the destinations. To correct the problem would require tweaking some kind of delay between RREP responses. Since this process was time consuming and requires investigating *ns*-2's MAC layer interactions, we chose to avoid the issue at this time. For each run, AODV and AODVM were seeded with the same value for the simulator's random number generator. Each run had a different random topology, node movement and traffic pattern. However, the input files used for a particular run were identical for both protocols.

4.1.1 Varied Loads

The first testing we did evaluates how the protocols perform as the offered load in the network increases. For these tests, we varied the number of connections in the system from 5 to 25 in increments of five. All the load tests had a pause time of zero seconds and a maximum speed of $20\frac{m}{s}$.

The packet delay of the protocols, shown in Figure 4, remains relatively low when the number of connections is low and there is no significant difference in AODV and AODVM. When the load increase to 20 connections, the contention at the MAC layer delays packets. The packets have to wait longer at each hop along a path. At 20 connections, AODVM has a slight reduction in the delay. This may be due to the fact that when routes are broken, the PROBE packet is unicast directly to the destination, rather than waiting for the route discovery broadcasts to

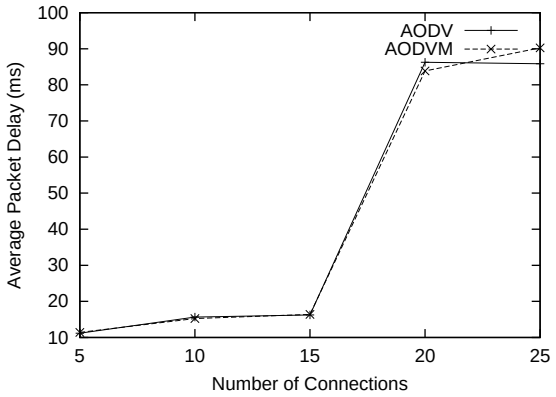

Figure 4: Packet Delay vs. the Number of Connections

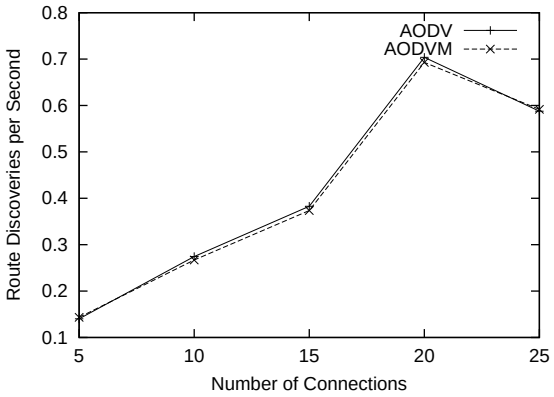

Figure 6: Route Discoveries per Second vs. the Number of Connections

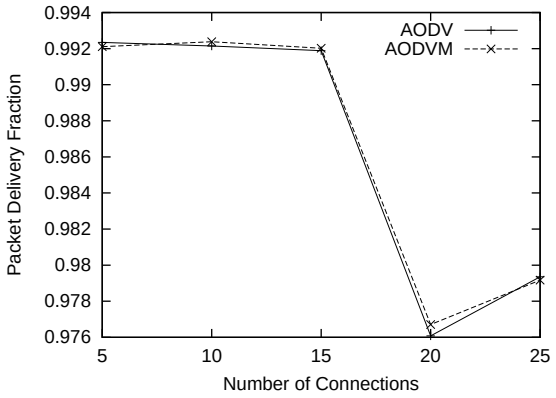

Figure 5: Packet Delivery Ratio vs. the Number of Connections

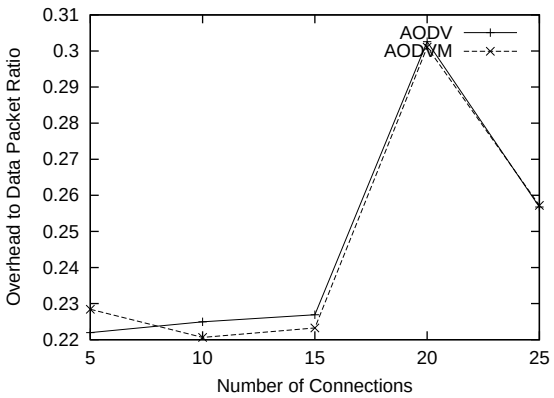

Figure 7: Overhead to Data Ratio vs. the Number of Connections

complete. On the other hand, when the number of connections grows a little larger to 25, AODV tends to have less delay. With the increased number of connections comes more cached routes at intermediate nodes and hence RREPs are generated closer to the source. In this case, AODV's caching seems to be a quicker recovery mechanism than waiting for the PROBE packet to reach the destination and trigger a RREP.

Figure 5 shows the decrease in how many packets are delivered as the number of connections increases. The decrease is due to packets being dropped because of increased collisions at the MAC layer. AODVM does slightly better with respect to this metric for some of load values. This is probably a result of AODVM reducing some of the control overhead and thereby inducing less collisions for data packets. There is no apparent reason for the slight increase in the packet delivery fraction when the load increases from 20 to 25 connections. The deviation

is small, less than 0.005, so it may be a statistical anomaly.

One goal of AODVM was to reduce the number of route discoveries which are expensive because they are broadcast. However, they are desirable because the broadcasting can give a fairly accurate picture of the network's current topology. As is seen in Figure 6, AODVM does appear to slightly reduce the number of broadcast route discoveries for each of the various loads. This is due to the fact that AODVM is sometimes able to set up an alternate route when an error is detected by unicasting the PROBE rather than broadcasting a RREQ.

The amount of overhead packets generated by the protocols should be correlated to the number of route discoveries. As shown in Figure 7, AODVM is able to reduce the amount of overhead packets most noticeably at intermediate load (10 to 15 connections). The overhead reduction comes from less RREQ packets. However, AODVM will also increase the number of RREP packets in the system and introduce some PROBE packets to the system. It should be noted this metric would be further reduced by AODVM if the protocols did not use an expanding ring search during route discovery. This optimization allows nodes to try searching their nearby area before doing network-wide broadcasts. At a low load, AODV performs better than AODVM. This is may be, because of the limited number of flows, AODVM is not able to keep its path cache very accurate. Therefore, most of the alternate paths it tries will be invalid, thereby causing the PROBE procedure and a RREQ procedure to take place.

From these results, we conclude AODVM is able to decrease the overhead of a system at intermediate loads. When the load is high, both protocols have about the same about of overhead. When the overhead is decreased, this leads to less contention at the MAC layer and hence more data packets getting through. Overall, though, the difference in the protocols does not appear to be significant as the load changes.

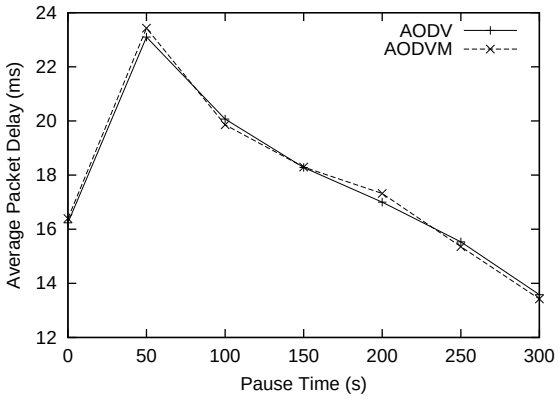

Figure 8: Packet Delay vs. Pause Time

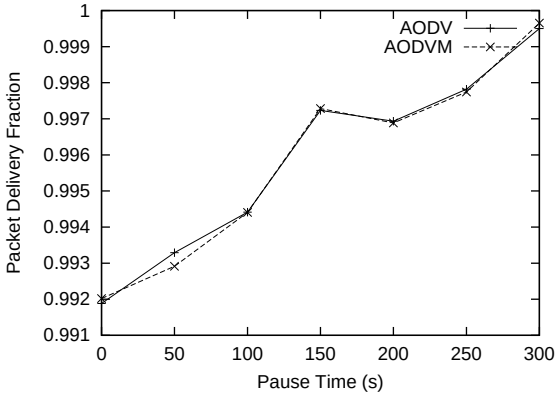

Figure 9: Packet Delivery Ratio vs. Pause Time

4.1.2 Varied Pause Times

To determine the effects of mobility on our protocol, we first considered the pause times of the nodes. Recall that this is how long a node stays at a given point once it arrives there. For these scenarios, the pause time was varied from 0 to 300 seconds in increments of 50. The maximum speed was set to $20\frac{m}{s}$ and the number of connections to 15 for all tests.

In Figure 8, we show how the pause times affect the delay packets experience. Because the load is relatively low, the primary contributor to increased

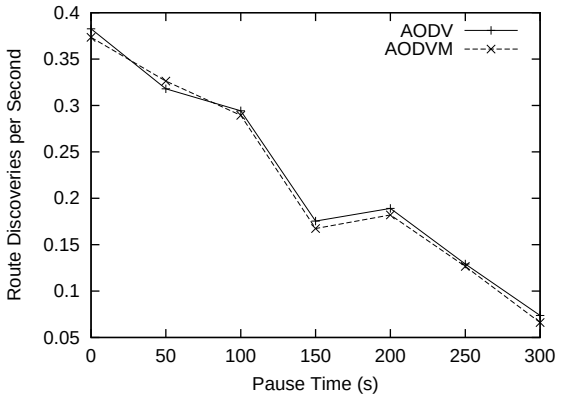

Figure 10: Route Discoveries per Second vs. Pause Time

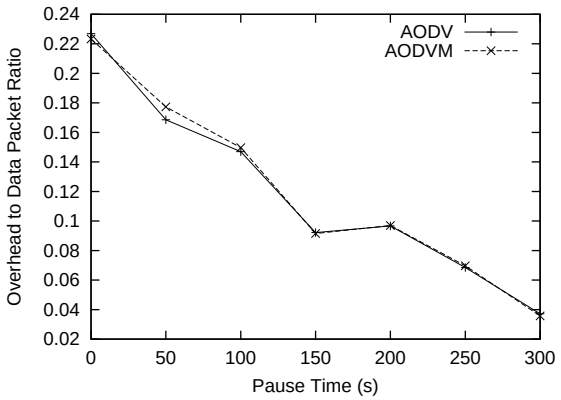

Figure 11: Overhead to Data Ratio vs. Pause Time

packet delay is packets which wait at the source while new paths are discovered. Our expectation was AODVM would be able to decrease the packet delay in this situation by quickly setting up an alternate route instead of waiting for a broadcast route discovery to complete. However, the flip side is with increased mobility, the cached paths become less accurate and hence less usable in the event of an error. Both protocols performed about the same in our tests with each doing slightly better for certain pause times. In general, it appears, AODV may do slightly better when the pause time is low and AODVM does slightly better when the pause time is higher. This is to be expected since at high mobility, AODV's immediate route discovery will find paths that are more likely to be usable. AODVM, however, will first attempt an alternate path before resorting to the broadcast.

Figure 9 seems to reinforce this intuition with AODVM showing slightly better packet delivery fractions at high pause times. However, AODV has a better packet delivery fraction at higher mobility. It should be noted all packet delivery for these tests is greater than 99%. Combined with the results in Section 4.1.1, we can see increased load causes a much more drastic reduction in packet delivery than mobility. This is due to the MAC layer contention. The reduction in delivery fraction in the pause time tests is primarily due to link layer loss when neighbors move out of range. AODV appears to generate more stable routes in the face of high mobility. This may be due to its route discovery broadcasts resulting in fresher paths than AODVM's probe mechanism.

Figures 10 and 11 seem to reinforce this idea that AODV does better at low pause times. In Figure 10, AODVM does reduce the number of route discoveries at high pause times since the topology is relatively static and hence the alternate routes are valid more often. At higher mobility, AODV and AODVM have about the same number of route discoveries. However, as shown in Figure 11, the protocols perform almost identically in terms of total overhead at high pause times. The only difference comes when AODV performs slightly better at low pause times. Here again, AODVM adds extra overhead packets and is not able to reduce the number of route discoveries. This results in higher overhead.

From the results, we can conclude AODVM actually degrades performance when the network topology is changing due to increase mobility. However, when the nodes are more static and do not move often, AODVM's alternate paths do provide some benefit.

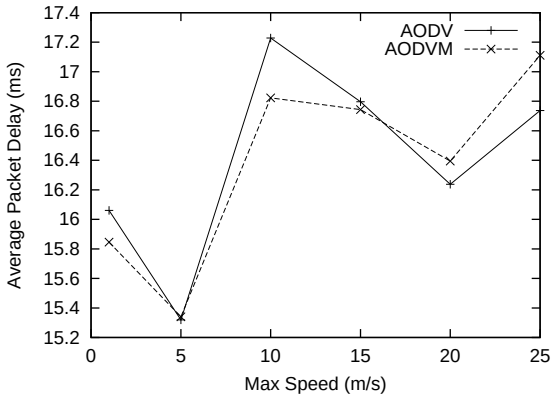

Figure 12: Packet Delay vs. Max Speed

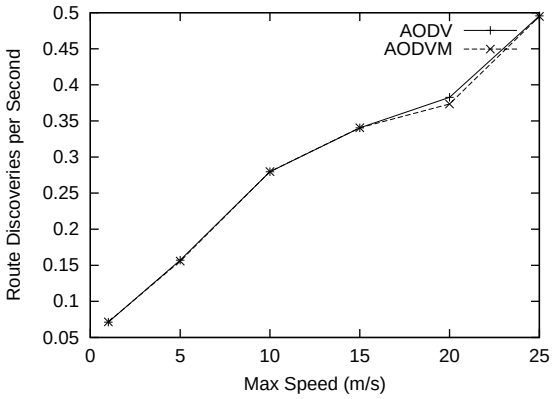

Figure 14: Route Discoveries per Second vs. Max Speed

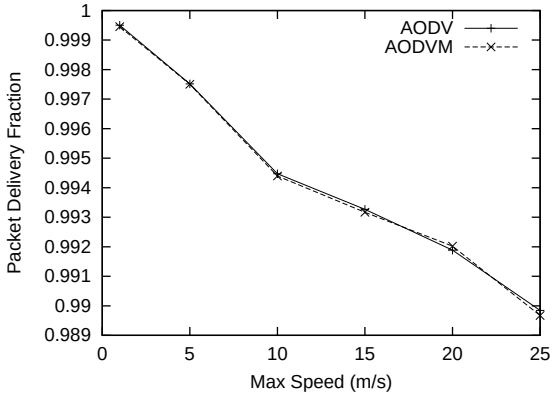

Figure 13: Packet Delivery Ratio vs. Max Speed

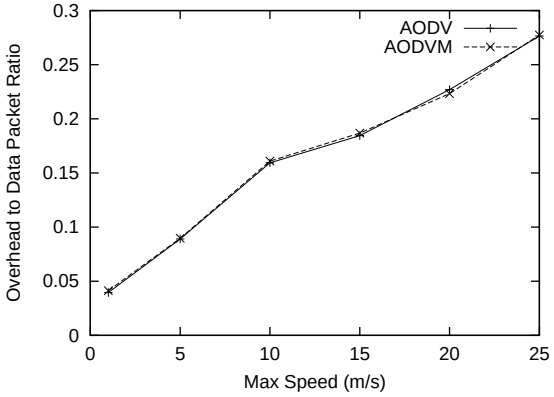

Figure 15: Overhead to Data Ratio vs. Max Speed

4.1.3 Varied Maximum Speeds

Another measure of mobility is increasing the maximum speed at which nodes move when choosing another location. Recall that the speed at which the node moves is chosen from a uniform random distribution from zero up to the maximum speed value. For these scenarios, the maximum speed a node could travel was varied from 1 to $20\frac{m}{s}$ in increments of 5. The pause time was set to zero seconds and the number of connections to 15 for all tests.

In Figure 13, we see virtually no difference in the performance of the protocols in terms of packet delivery. Similarly, Figure 15 shows the protocols both produce the same amount of overhead in the system. In Figure 14, AODVM does slightly reduce the number of route discoveries at $20\frac{m}{s}$. However, the reduction does not appear significant based on the small magnitude of the difference and the general trend of the other results for varying speed.

Only Figure 12 has some interesting results for the speed tests. The packet delay is fairly sporadic, but seems to indicate AODVM will decrease the delay

at lower speeds while AODV has lower delays at increased speeds. However, the difference at any level of speed is probably not significant as evidenced by the small range over which the values are distributed. The deviations are probably just due to sampling error. Notice that the y-axis in Figures 4 and 8 have a much larger range than that of Figure 12.

Overall, these test seem to indicate the maximum speed is not a significant factor in differentiating between the two protocols. The pause time and offered load in the system dominate the maximum speed parameter. Since the tests in this section use zero pause time and an intermediate load, both protocols seem to perform equally bad.

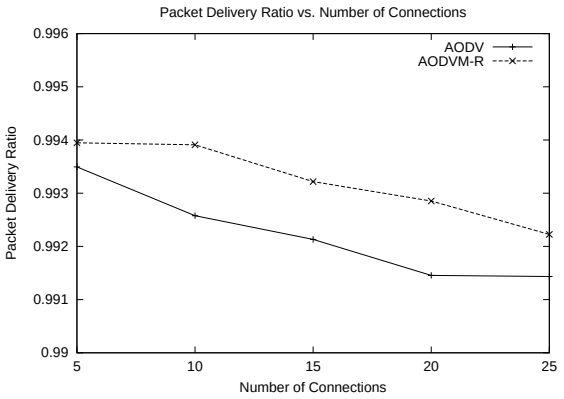

Figure 16: Packet Delivery Ratio vs. Number of Connections

4.2 Intermediate Router Recovery (AODVM-R)

In this section we present simulation results for the AODVM-R protocol. The simulation model and the parameters used in the simulations are the same as those used in the simulations for AODVM (explained in section the previous section), except for one thing. There is no PROBE message in AODVM-R, but AODVM-R has REFRESH messages. So in AODVM-R, RREQ, RREP, RERR, and REFRESH messages are the overhead packets.

The metrics we measure are also the same as in the previous section: average packet delay, packet delivery ratio, route discoveries per second, and overhead to data packet ratio. We vary the number of connections, pause time, and the maximum speed. In the graphs, "AODV" refers to the original AODV protocol, and "AODVM-R" refers to the proposed protocol.

4.2.1 Varied Loads

In these simulations, we vary the number of flows in the network to see the impact of network load on different metrics. The number of flows range from 5 to 25. The maximum speed of mobile nodes are fixed to $20m/s$, and pause time is fixed to 0.

In Fig. 16, we can see that AODVM-R does slightly better than AODV in terms of packet delivery. The number of dropped packets is smaller in AODVM-R. But the as seen in the graphs, the difference is not so significant. In AODV, if the route to the destination breaks, the packets that were being delivered through that route gets dropped because the intermediate nodes do not know how to reach the destination. But once the source performs route discovery, the subsequent packets will be delivered to the destination, unless the destination is totally disconnected from the source. AODVM-R tries to save those packets that were being delivered through the broken route. The intermediate nodes, if they were maintaining alternate routes, will not drop the packet but deliver it to the next hop on the alternate path. If the destination can be reached through the alternate path, the packet can be delivered correctly, even though the delay might be higher. But if the destination becomes totally disconnected from the source, there is no way that AODVM-R can deliver the packets. So in this case, both protocols will drop packets anyway. So the main difference in packet delivery ratio comes from the saved packets that were being delivered through the broken route.

As the network load increases, the packet delivery ratio drops in both protocols, mainly because of increased contention at MAC layer. So the rate at which the packet delivery ratio decreases is similar in two protocols.

Fig. 17 shows the average packet delay of two pro-

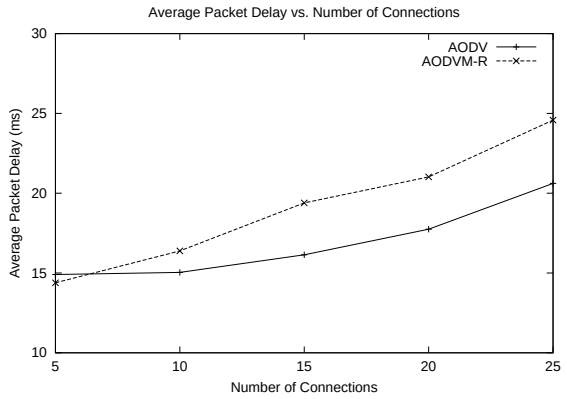

Figure 17: Average Packet Delay vs. Number of Connections

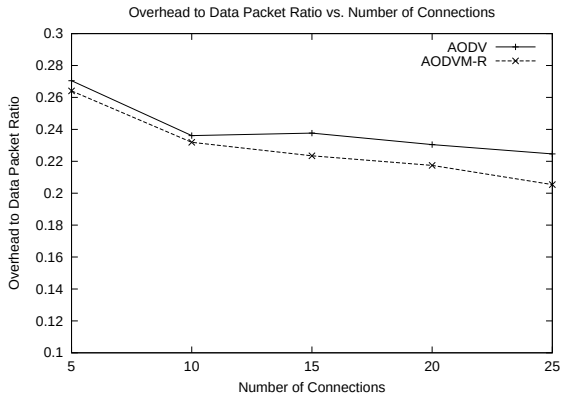

Figure 18: Overhead to Data Packet Ratio vs. Number of Connections

tocols. As the graph shows, the packet delay is higher in AODVM-R. This is due to the following reasons. In AODVM-R, if a route is repaired using alternate routes, it is likely that the repaired route has a longer hop count than the actual optimal route from the source to the destination. Whereas in AODV, a new route discovery is performed every time the route is broken, and the route with shortest hop count is chosen. This is one reason that AODVM-R has longer delay than original AODV. Another reason comes from the delay of repairing the broken route. Every time a route fails, AODV has to pay for the delay of route discovery. In AODVM-R, if the route is repaired successfully using the alternate routes, then the delay for route discovery is saved. But, if the alternate path is also broken so that the source would have to initiate a route discovery anyway, then the delay penalty becomes higher, because the source performs route discovery after the attempt to use alternate routes fails. Also in AODVM-R, the REFRESH messages in AODVM-R also increases contention at the MAC layer and thus affects the packet delay.

In Fig. 19, it seems that AODVM-R succeeds in reducing the number of route discoveries. But still the reduce in amount overhead is not so significant, as shown in Fig. 18. The main reason we want to avoid route discoveries is because flooding the network with route request messages is a high overhead. But in AODVM-R, the benefit of reduced route discoveries is diminished by the refresh messages. Refresh messages are not broadcast, but they are sent to the alternate routes of the active path periodically. So in a network with low mobility, where the routes are not frequently broken, the overhead of refresh messages will be higher than the overhead of broadcasting the route request. Since the refresh message is a major source of overhead in AODVM-R, precautions must be taken to minimize the overhead. This issue is discussed later in this section.

4.2.2 Varied Maximum Speed

The final set of simulations were done varying maximum speed of the mobile nodes. The mobile nodes may move at a random speed between 0 and the specified maximum speed. The maximum speed is varied from 0 to $25m/s$. Number of flows is fixed to 15, and pause time is 0 in the simulations.

As the pause time increases, the network becomes more static, and topology change occurs less frequently. So as shown in Fig.20, the packet delivery ratio goes up for both protocols, as the pause time increases. AODVM-R performs slightly better than AODV, but the difference is not so significant. Fig.20 shows the packet delay of AODVM-R is worse than AODV. The difference in packet delay is more signif-

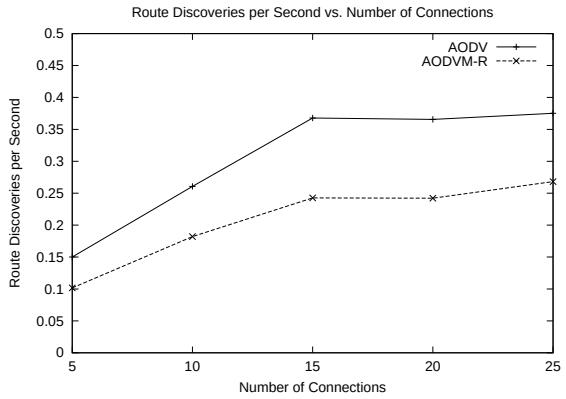

Figure 19: Route Discoveries per Second vs. Number of Connections

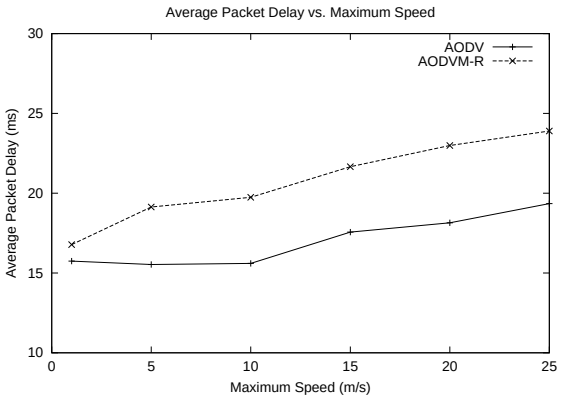

Figure 21: Average Packet Delay vs. Max Speed

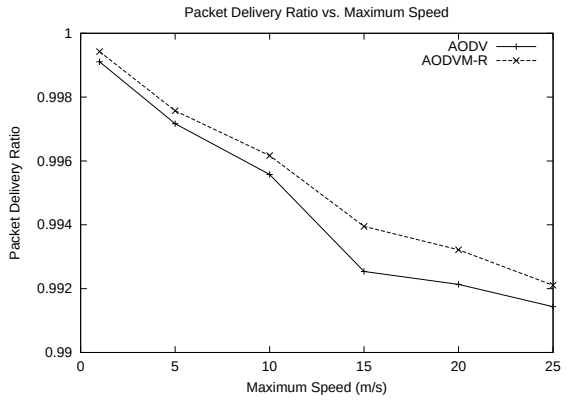

Figure 20: Packet Delivery Ratio vs. Max Speed

icant when the pause time is smaller, because high mobility leads to more frequent route failure, and repaired routes in AODVM-R tend to be suboptimal than rediscovered routes in AODV.

Fig. 22 shows that AODVM-R reduces number of route discoveries using alternate paths. The difference between AODV and AODVM-R is significant with medium and high mobility, but there is not much difference with low mobility. If the mobility is low, topology is not frequently changing, so there is not much need for route discoveries (less than 1 in every 10 seconds). So AODVM-R does not gain much benefit. As the mobility increases, AODVM-R repairs a lot of broken routes using alternate routes, reducing the number of route discoveries.

The amount of overhead is shown in Fig. 23. With low mobility, AODVM-R has higher overhead than AODV. Because refresh messages are sent periodically as long as the route is being used. So AODVM-R performs better than AODV in terms of overhead, only if there are frequent route failures so that the overhead of route recovery or new route discovery dominates the overhead. As the maximum speed increases, AODVM-R has lower overhead than AODV.

4.2.3 Varied Pause Times

Here we vary the pause time of mobile nodes. In the random waypoint model, every time a node moves to a location, it pauses for a specified time before it starts moving toward a new location. The pause times are varied from 0 to 300 seconds. 0 pause time means the nodes are always moving. A pause time of 300 seconds means the nodes only move once and stops for the rest of simulation time, because the simulation time is 300 seconds. In these simulations, the maximum speed of each node is fixed to $20m/s$, and number of flows is 15 in each simulation.

The results of these simulations lead to similar arguments as previous simulations varying maximum

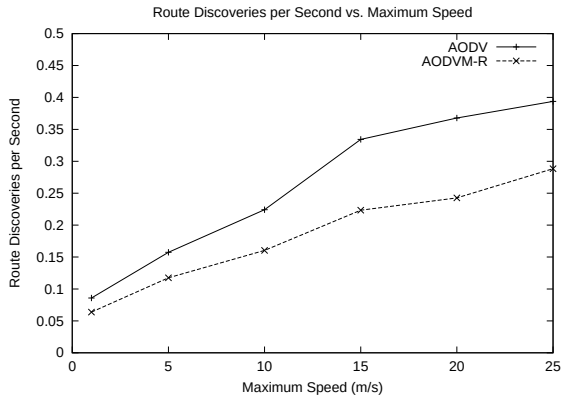

Figure 22: Route Discoveries per Second vs. Max Speed

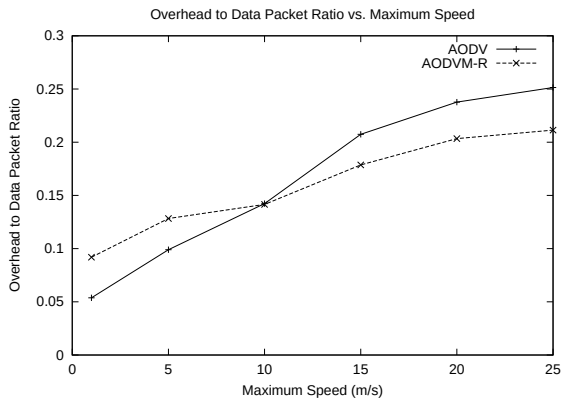

Figure 23: Overhead to Data Packet Ratio vs. Max Speed

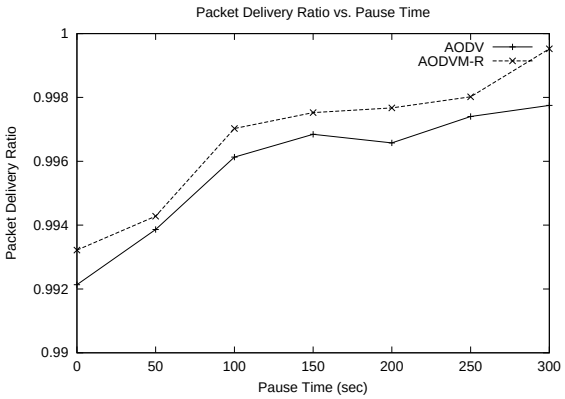

Figure 24: Packet Delivery Ratio vs. Pause Time

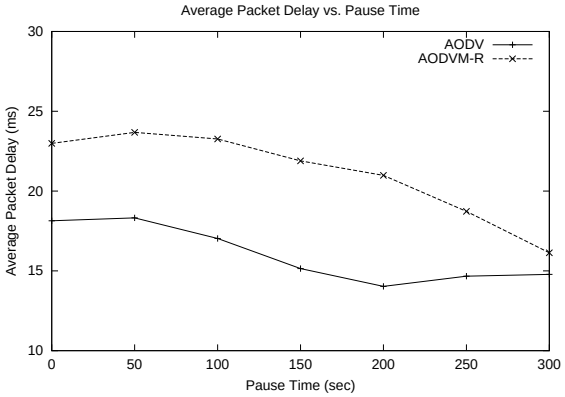

Figure 25: Average Packet Delay vs. Pause Time

speed. Low pause time means high mobility, whereas high pause time means low mobility. Although Fig. 24 shows the packet delivery ratio is slightly higher in AODVM-R, the packet delay is higher in AODVM-R, as shown in Fig. 25. The difference in packet delay is not so significant with longer pause time, but becomes significant as the pause time becomes smaller. As shown in Fig. 26 and 27, AODVM-R succeeds in reducing the number of route discoveries, but still the overhead is not significantly reduced because AODVM-R has refresh message overhead. When mobility is low, the overhead of route discoveries lessens, but the amount of refresh messages sent in AODVM-R stays the same. So AODVM-R performs worse than AODV in terms of overhead, when the mobility is low.

In overall, AODVM-R seems to succeed in reducing the number of route discoveries significantly, but at the cost of refresh message overhead. The simulation results have shown that the refresh message overhead in AODVM-R is significantly large that it diminishes the benefit gained from reducing the number of route

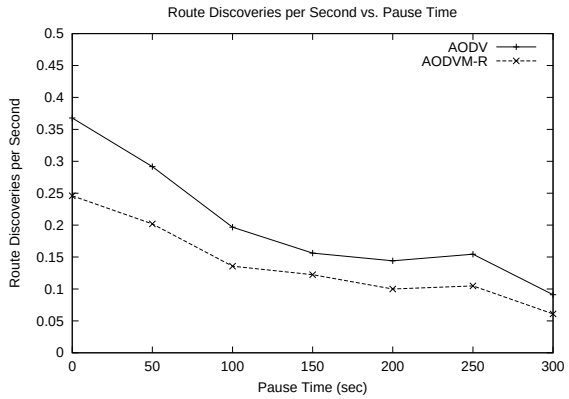

Figure 26: Route Discoveries per Second vs. Pause Time

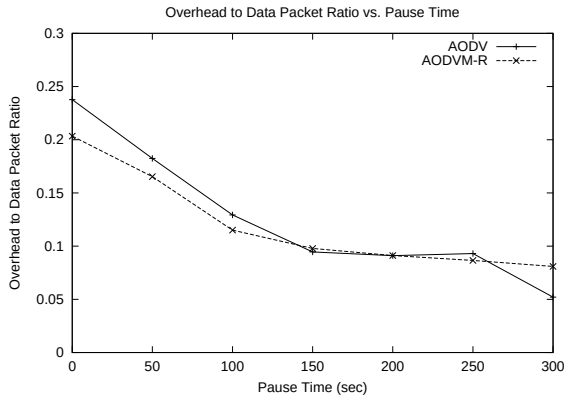

Figure 27: Overhead to Data Packet Ratio vs. Pause Time

discoveries. Also, non-optimal routes and attempt to use broken alternate paths makes the packet delay of AODVM-R higher than that of AODV. In total, AODVM-R does slightly better in terms of packet delivery ratio, but does worse in terms of packet delay.

Since the refresh message overhead is shown to be significant in AODVM-R, we can consider changing the refresh timeout to reduce the overhead. In the basic scheme of AODVM-R, we set the refresh timeout as half of the active route timeout (explained in Section 3.2). This is be a conservative approach. We can set the refresh timeout to be a little smaller than the active route timeout, for example 90%. But then the risk of *false timeout* becomes higher. Suppose the active route timeout is 10 seconds, and we set the refresh timeout to be 9 seconds. Then a packet that uses the primary route after 9 seconds initiates refresh event for alternate routes. So if there is no packet that uses the primary route between 9 and 10 seconds, the alternate routes will timeout, even though they should have been refreshed. Another option is to eliminate the periodic refresh events, and do not apply timeouts to the alternate paths. If we eliminate the refresh events, the total overhead will become much smaller. But since the alternate paths have infinite life time, they are more likely to be stale when they are ready for use. If we use refresh messages, these messages can detect failures in alternate routes and discard the broken routes from the route table.

Also, if we use alternate routes that are maintained at the routers, the repaired route is likely to be suboptimal. If many parts on a path are repaired using alternate routes, the route would be very inefficient in terms of hop counts. This results in higher packet delay. Also, longer hop distance means more number of packet transmissions take place before the packet is delivered to the destination. If average hop distance in AODVM-R is twice of average hop distance in AODV, then AODVM-R must generate twice amount of traffic to achieve the same goodput. So large average distance is one of the factors that diminish the benefit gained from reducing number of route discoveries.

5 Conclusion

For AODVM, we conclude that the added complexity of this protocol and extra control packet overhead are not justified since the impact upon key metrics when compared to AODV is minimal. The major reasons for this are when the topology is changing quickly, sources will rarely have more than one alternate path to a destination and an alternate path that does exist will probably be stale and the PROBE procedure

will fail. Another metric, which we did not have time to test, would be to see how far the lengths of the paths generated by each protocol deviates from the optimal. Intuitively, AODV should do better in this metric because its route discovery procedure would find the path AODVM is probing plus any shorter paths which may exist. The longer paths would also lead to more packet transmissions in the system as a whole since data packets would have to be transmitted over more hops to reach the destination. Another thing to note is if local repair is turned on, AODVM's performance is further hindered since intermediate nodes will try to repair a path first and delay telling the source. This increases the probability the source's alternate paths would be out of date. Overall, it results seem to indicate the major source of AODVM's ineffectiveness is the lack of opportunity to use the PROBE mechanism. When cached routes cannot reply with alternate path information, the RERRs remove all paths with the broken link before probing and the learning of a fresher route causes all known alternate routes to be invalidated, the protocol does not leave much opportunity for the probe procedure. Hence, the number of PROBE packets sent will be dominated by the number of other control packets in the system.

AODVM-R is also not so successful, because although it reduces number of route discoveries, other sources of overhead are introduced and the packet delay becomes higher. To maintain alternate routes, the refresh message overhead must be introduced, regardless of how fast the network is changing. So if network is static, we do not gain any benefit from introducing this new overhead. If there is high mobility in the network so that the route is frequently broken, then AODVM-R reduces the amount of overhead, because it avoids expensive route discoveries using alternate routes. But as the mobility increases, the possibility of alternate routes being stale also increases. So the performance heavily depends on the given topology. The main reason of AODVM-R's higher packet delay is due to long hop counts of repaired routes. Also, if repairing path through alternate routes fails, the delay of route discovery is higher than AODV. As mentioned in the previous section, tuning the refresh timeout value might lower the overhead of AODVM-R. This is left as a future work.

6 Work Division

Matt Miller worked on the source oriented approach described in Section 3.1 and tested in Section 4.1. **Jungmin So** worked on the intermediate router recovery approach described in Section 3.2 and tested in Section 4.2.

References

[1] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols," in *Proc. of IEEE/ACM MOBICOM*, 1998.

[2] C. E. Perkins and E. M. Royer, "Ad-Hoc On Demand Distance Vector Routing," in *Proc. of IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, 1999.

[3] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in *Mobile Computing* (T. Imielinski and H. Korth, eds.), pp. 153–181, 1996.

[4] S. R. Das, C. E. Perkins, and E. M. Royer, "Performance Comparison of Two On-Demand Routing Protocols for Ad hoc Networks," in *Proc. of IEEE INFOCOM*, 2000.

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

[6] R. V. Boppana and S. P. Konduru, "An Adaptive Distance Vector Routing Algorithm for Mobile, Ad Hoc Networks," in *IEEE INFOCOM*, 2001.

[7] S.-J. Lee and M. Gerla, "AODV-BR: Backup Routing in Ad hoc Networks," in *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, 2000.

[8] M. K. Marina and S. R. Das, "On-demand Multipath Distance Vector Routing in Ad Hoc Networks," in *Proc. of IEEE Intl. Conference on Network Protocols (ICNP)*, 2001.

[9] Y. hua Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast," in *Proc. of ACM SIGMETRICS*, 2000.

[10] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. of ACM SIGCOMM*, 2002.

[11] D. Anderson, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient Overlay Networks," in *Proc. of the Symposium on Operating System Principles*, 2001.

[12] The Network Simulator. http://www.isi.edu/nsnam/ns/.