Control Message Reduction Techniques in Backward Learning Ad Hoc Routing Protocols

Navodaya Garepalli Kartik Gopalan Ping Yang
Computer Science, Binghamton University (State University of New York)
Contact: kartik@cs.binghamton.edu

***Abstract*—Most existing wireless ad hoc routing protocols rely upon the use of backward learning technique with explicit control messages to route packets. In this paper we propose a set of techniques that can be applied in a backward learning routing algorithm in order to minimize or even eliminate explicit control messages for route discovery, setup, and maintenance, while minimally using implicit data-like control messages that need no special processing. We also show that such an algorithm does not need to prevent routing loops at all costs, such as by means of extensive network-wide spanning trees in traditional LAN bridges, or destination sequence numbers in AODV, or source-routing in DSR. In fact, we prove that transient loops can be safely allowed to occur when a simple route refresh mechanism is coupled with the use of packet identification field to effectively bound the lifetime of such transient loops without negatively impacting the network performance. Results demonstrate that even a routing algorithm without explicit control messages can perform competitively in comparison to AODV and DSR protocols while significantly reducing the protocol complexity.**

I. Introduction

Existing ad hoc routing protocols [3], [5], [9], [7], [4], [6], [8], [2] set up and maintain loop-free routes by depending upon explicit control message exchanges using an extensive control infrastructure. The use of explicit control messages enables these protocols to optimize the network performance. For instance, the AODV [9] protocol guarantees loop freedom through use of destination sequence numbers and relies upon explicit control messages such as RREQ, RREP, and RERR messages to construct and maintain routes. Similarly, DSR [7] uses explicit control messages, such as RREQ and RREP packets, to perform source routing. While protocols-specific optimizations, such as piggybacking, can reduce the overhead of explicit control messages, significant protocol complexity remains in place due to the need to maintain this control infrastructure and process the explicit control information. In this paper, we investigate whether one can design an ad hoc routing protocol that incurs very low control message overhead and at the same time maintains competitive network performance. We propose a set of generic mechanisms that can be applied to *backward learning* ad hoc routing algorithms in order to greatly reduce protocol complexity. These techniques help to eliminate the use of *explicit* control messages as well as the need to maintain an overarching control infrastructure for route set up, maintenance, or loop prevention, and at the same time maintain a high network throughput. The key idea is to combine the on-demand nature of ad hoc routing protocols with the simplicity of *backward learning* transparent bridges. While most existing ad hoc protocols employ the basic principles of backward learning, we exploit this principle without having to construct and maintain network-wide spanning trees as in wired LANs, or use destination sequence numbers as in AODV [9], or resort to source routing as in DSR [7]. Our results show that, even without explicit control messages, a protocol can deliver highly competitive network performance, if not better, in comparison to the popular AODV and DSR protocols. Using extensive simulations, we investigate the advantages and disadvantages of eliminating control message overhead in order to achieve greater protocol simplicity. For clarity of exposition, we describe these techniques as embodied in a proof-of-concept routing protocol called the Explicit Control Message Free (ECMF) routing protocol. However, the proposed techniques can be readily adapted even to existing backward learning protocols, such as AODV, to eliminate control message processing.

Just as in wired bridging and many wireless ad hoc protocols, nodes in ECMF protocol use backward learning to set up and adapt routes to different destinations on-the-fly. However, unlike conventional wireless ad hoc protocols, ECMF nodes implicitly learn routes from data packets themselves, rather than through explicit control messages. This works well for most network communication sessions which involve bidirectional exchange of data, such as while using TCP. In the case of unidirectional traffic flow, ECMF selectively uses minimal number of *implicit* (or data-like) control messages that do not require any special processing in the network interior.

At the same time, unlike wired bridges, ECMF also does not go out of its way to prevent routing loops at all cost but limits their performance impact by design, if and when such loops arise. Hence ECMF eliminates the overhead of constructing and maintaining an extensive infrastructure for loop prevention, such as network-wide spanning trees in wired-LAN bridges, destination sequence numbers in AODV, or source-routing in DSR. *We prove that any transient loops, which may potentially arise, do not last beyond a bounded small duration.* Moreover, packets do not traverse around transient loops more than once and thus do not lead to exponential packet storms. Additional features of ECMF include no need for promiscuous mode operation of wireless interfaces, no *Hello* messages between neighbors, and elimination of ARP (Address Resolution Protocol) request/reply messages. Thus, ECMF is completely decentralized, self-starting, and automatically adapts to the widely varying network conditions.

II. ECMF Protocol Overview

ECMF nodes have two identities. First is at link level, such as layer-2 MAC address, which is visible only to immediate neighbors of the node. Second is at network level, such as

layer-3 IP address, which a node uses to identify and communicate with any other node (not necessarily a neighbor). A node does not need to have any prior knowledge of the identity, connectivity, or location, of other nodes in the network except, of course, the layer-3 IP address of the nodes with which it explicitly wishes to communicate. A node dynamically learns and unlearns the layer-2 MAC address identity of its neighbors only when it needs to communicate with/via those nodes. ECMF operates on top of link layer protocols, such as the 802.11 family of protocols, which support link-level acknowledgment between neighbors to avoid unidirectional links. A network interface is not required to operate in 'promiscuous' mode. Every ECMF node makes completely local routing decisions that require no coordination with other nodes. ECMF is positioned as a thin protocol layer in the networking stack between the Layer-3 (IP layer) and the Layer-2 (MAC layer). Although forwarding is based on destination IP address, the ECMF routing is slightly different from traditional IP routing. The next-hop address for a destination IP in the routing table is the layer-2 MAC address of the next-hop rather than its IP address. ECMF nodes do not care about the IP-to-MAC address mappings of other nodes, except as needed to make forwarding decisions.

Routing Table and Data Packets: Each ECMF node maintains its forwarding knowledge in a *soft-state* routing table. Each routing table entry contains the following basic information: **(1)** Destination IP Address (D), **(2)** Next Hop MAC Address (m), **(3)** A list of alternative next hop MAC addresses, **(4)** Valid flag (v), and **(5)** Expiration time (e). Only routing entries that are *valid* can be used for packet forwarding. Routing entries can be invalid during the times when a node is recovering after a failure or mobility of a neighbor. A routing entry gets *refreshed* each time a packet from the corresponding destination D is received from the next-hop MAC address m. *In other words, reverse traffic from the destination D updates its forward routing entries at intermediate nodes.* This is unlike in the case of AODV, where forward traffic towards D is used to refresh the forward routing entries. Expiration Time field indicates the amount of time left before an *un-refreshed* routing entry can be purged. We represent the relevant header fields of a packet by the quadruple $[s_m, d_m, S_{IP}, D_{IP}]$. Here s_m and d_m represent the source and destination MAC addresses of immediate neighbors exchanging the packet and S_{IP} and D_{IP} represent the original source and final destination IP addresses. A $*$ represents a broadcast MAC or IP address.

III. Route Discovery

Figure 1 illustrates the basic route setup mechanism in ECMF. Source S initiates communication with destination D for the first time by flooding its first data packet with header $[S_m, *, S_{IP}, D_{IP}]$. This data packet travels towards D, setting up routing table entries for S at intermediate nodes. Any intermediate node X, that receives the packet from S, processes the packet in two phases: the *learning* phase and the *forwarding* phase. Note that flooding a potentially large data packet could be more expensive than flooding a smaller

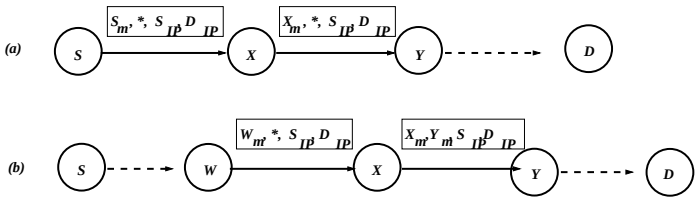

Fig. 1. Illustration of route discovery mechanism.

control packet. In practice, most communication sessions begin with an initial bidirectional exchange of small packets (such as TCP SYN/ACK) which greatly reduces the flooding cost.

A. Learning Phase

In the learning phase, node X learns about the direction in which node S is reachable. First consider the case in Figure 1(a) where X is an immediate neighbor of S, but is not aware of this fact and does not have any prior routing information for S in its routing table. When X receives the first broadcast packet with header $[S_m, *, S_{IP}, D_{IP}]$, it immediately infers that a node with layer-3 IP address S_{IP} can be reached in the direction of an immediate neighbor with layer-2 MAC address S_m. The node X then enters this newly learned forwarding information in its routing table as a valid routing entry for S_{IP}. Note that X does not care that S_{IP} and S_m are the same neighbor's IP and MAC addresses respectively. All X cares is that S_{IP} is reachable in the direction of S_m.

Consider the second case in Figure 1(b) when X is not an immediate neighbor of S and does not have any prior routing entry for S_{IP}. Rather X receives a packet with header $[W_m, *, S_{IP}, D_{IP}]$, forwarded from another neighbor W, which does not know the next hop to D. Here X will learn that S_{IP} is reachable in the direction of W_m.

Finally consider the third case, again in Figure 1(b), when node X already has a valid routing table entry for S. Assume that a non-duplicate (i.e. first-time) packet with header $[W_m, *, S_{IP}, D_{IP}]$ is received from a neighbor W, which X already knows is a valid next hop for S_{IP}. In this case the only new information that node X can learn from the packet is that S_{IP} is still reachable via W_m and can refresh its routing table entry for S by resetting the expiration time field. On the other hand, if the current next hop for S_{IP} is not W_m then X can infer that either the earlier known route to S_{IP} has failed, or S has moved to a new location, or simply a better route to S may have appeared depending on the specific link/path quality metric of interest. In this case, X can choose to *re-learn* the new next-hop to S_{IP}, as described later in Section IV.

Another function of the learning phase is to learn alternative routes. Suppose an intermediate node X receives multiple copies of the same packet, which originated from S, from different neighbors. The first copy of the packet will be used by X to learn the *primary* next hop towards S. Any subsequent copy of the packet will be recognized by X as a duplicate (using mechanism is described later in Section V). Before dropping the duplicates, X will extract the source MAC address from the duplicates as *alternative* next hop towards S. This alternative next hop can be used to replace the the primary

next hop, in case the primary link fails or the alternative has better link quality metric (such as bandwidth or noise).

B. Forwarding Phase

In the forwarding phase, node X first checks if the packet is destined to itself, i.e. $D_{IP} = X_{IP}$. If so, the packet is delivered to the IP layer (layer-3) within X. If not, then X determines the next hop for destination D. If no valid routing entry exists for D_{IP} then, as shown in Figure 1(a), X re-broadcasts the data packet to its neighbors with a modified header $[X_m, *, S_{IP}, D_{IP}]$. If X already has a valid routing entry for D_{IP} with the next hop neighbor Y then, as shown in Figure 1(b), X forwards the data packet to Y with a modified header $[X_m, Y_m, S_{IP}, D_{IP}]$. *Hence, if parts of the network already have valid routes to D then those routes are reused.*

If, for some reason, the transmission to next hop node Y fails, then X invalidates the routing entry and attempts to send the data packet to one of its neighbors Z in the *alternative* next hop list of routing entry for D with a modified header $[X_m, Z_m, S_{IP}, D_{IP}]$. If transmissions to none of the alternative next hops succeed, then X simply re-broadcasts the packet with a modified header $[X_m, *, S_{IP}, D_{IP}]$.

To complete the picture, by the time the packet from S reaches D, the intermediate nodes in the network (including D) learn about the route to reach S. Similarly, when the receiving user-level application at node D responds back with its own data to its peer application at node S, the intermediate nodes (including S) learn about how to reach D as well.

C. Handling Silent Endpoints

Most real-world network sessions, such as TCP, engage in bidirectional exchange of data or control packets at application or transport level. In the unlikely event, where only one of the end-points actively sends traffic, the backward learning mechanism would maintain routing information for only the active sender and not for the passive receiver. This creates an undesirable situation where all data packets to the silent receiver might end up being flooded due to absence of routing information for the receiver. To rectify this situation, the ECMF layer at the receiver node D monitors whether packets from S are being received for some $ACTIVITY\_INTERVAL$ duration (say 80% of maximum route expiration time) without the application at D sending any data back to S. If so, the ECMF layer at D transmits a *dummy IP packet* to S_{IP} with zero byte data payload. As the dummy IP packet from D_{IP} to S_{IP} travels through the network, intermediate nodes learn the routing information for D_{IP}. Thus future data packets from S to D are unicasted rather than flooded through the network. Since the dummy IP packet is not meant for any specific application at S, it is silently discarded by the IP layer at S. ECMF layer at D stops sending dummy IP packets once it observes that S is not sending any more data packets. The dummy IP packet is an *implicit control message* that is forwarded just like any other data packet without any special processing. Further, it is created only under the exceptional circumstance when one of end-points is completely silent for a long time. ECMF layer at S also avoids initial broadcast of too many back-to-back packets from S to D by buffering the packets following the first packet, until either the first data packet or dummy IP packet is received from D.

IV. Route Maintenance

A. Refreshing Soft Routing State Using Reverse Traffic

The route refresh mechanism employed by ECMF is particularly different from the refresh mechanisms employed by other protocols such as AODV. Each routing entry in ECMF routing table has an associated lifetime, which indicates the time duration after which the a non-refreshed routing entry will be deleted. Assume that a node X's routing table contains a valid routing entry for a destination address D_{IP} with next hop as node Y. The lifetime field of this routing entry is refreshed to a maximum lifetime value of MAX_ROUTE_LIFETIME each time a packet *originating* from node D is forwarded by node Y to node X. In other words, X can reaffirm the fact that D is still reachable via its neighbor Y when it receives a packet from Y that has a *source* IP address of D_{IP}. Note that **reverse** traffic originating from D is used to refresh the **forward** routing information towards D. This in more accurate than other routing protocols such as AODV in which packets *destined* to D are used to refresh its routing information.

B. Handling Mobility and Network Failures

A failure is detected whenever a node doesn't receive a link-level acknowledgment for its unicast transmission. Consider Figure 2 in which there is a failure along a route from S to D between neighbors X and Y. When X detects a failure during unicast transmission of a packet to its neighbor Y, it first checks every routing entry that contains Y_m as the next hop address. If such a routing entry contains a list of alternative next hops, then the best alternative, as per the link metric used, is assigned as the primary next hop. Otherwise, the corresponding routing entry is marked as invalid. For example, in Figure 2, X has four other neighbors L, M, N, and O, besides Y. Of these four, L happens to be the best alternative next hop towards destination D. Upon failure of communication with Y, X first assigns L as the primary next hop and unicasts the packet for D to L. This is shown in case (a). If the communication with L fails as well and there are no more alternative next hops, then X floods the packet to all its neighbors hoping that at least one of them can reach D. This is shown in case (b). The scope of this flood is naturally limited by the packet's TTL value when it is received by X. Further, nodes around the area that are affected by the failure can quickly re-learn about new routes to the destination D as soon as packets from D flow through those nodes.

V. Damping Duplicates Without A Spanning Tree

ECMF nodes do not construct or maintain any spanning trees. Instead they damp the propagation of duplicate packets by using the TTL and 16-bit Identification fields, which are carried inside the IP header of every packet. Every ECMF node remembers the Identification values of last k packets seen from source node S. A packet is dropped only if its value is among the last k remembered values or is smaller than

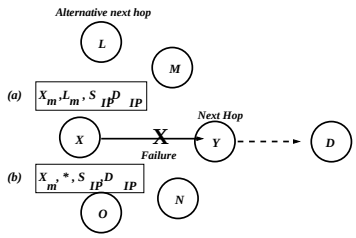

Fig. 2. Two cases when the communication link between X and Y fails.

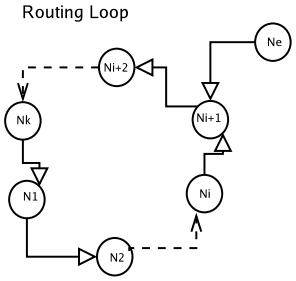

Fig. 3. Complete transient loops disappear in bounded time.

all of them. This technique detects duplicates and tolerates re-ordering of packets within a window of k packets. In evaluations, we observe that a window size as small as $k = 3$ is more than sufficient to detect and damp all duplicates even in the presence of packet reordering. The worst-case per-node state requirement in an N-node network would be $O(kN)$. Also note that, use of the Identification field in ECMF is quite different from the intricate mechanism of destination sequence numbers in AODV, which are used to altogether prevent routing loops as well as maintain route freshness. In contrast, the IP Identification field is used in ECMF only to detect and damp duplicate packets.

VI. Limiting the Impact of Transient Loops

We define a *complete* transient loop for a destination D as one in which each node in the loop has a valid routing entry towards the next hop node in the loop. A *partial* transient loop for a destination D is one in which at least one node in the loop does not have a valid routing entry for D. Consequently, such a node would broadcast the packets destined to D and the next node in the loop picks up the broadcast packet. No protocol can avoid *partial* transient loops whenever any intermediate node needs to broadcast a packet to an unknown destination. ECMF does not attempt to prevent transient loops at all costs. Rather, if and when transient loops are indeed formed, their impact on network performance is contained as follows. **(1)** *ECMF guarantees that complete transient loops last for less than a provably short bounded time interval* (We prove this claim below). **(2)** Since ECMF nodes perform duplicate packet detection (Section V), no packet will traverse a loop more than once. Rather a packet is discarded once it revisits any node. **(3)** Successive ECMF nodes decrement the TTL in each packet and discard them when TTL reaches zero.

Theorem 1: Lifetime of a complete transient loop for any destination D is less than or equal to MAX_ROUTE_LIFETIME.

Proof: Consider the complete loop in Figure 3 for destination D with k nodes N_1 to N_k, where the next hop for destination D at node N_i is N_{i+1} and at node N_k is N_1. By definition, node D itself cannot be part of the complete loop and hence is outside the loop. Furthermore, in order for node N_i to maintain N_{i+1} as its next hop entry for D, N_{i+1} must forward a packet with *source* IP address D_{IP} to N_i within MAX_ROUTE_LIFETIME interval of its learning the next hop. Two cases arise. In the first case, node D does not send a packet through any node in the complete loop for MAX_ROUTE_LIFETIME duration. Thus N_{i+1} cannot forward a packet with D_{IP} as source IP address to N_i. Consequently N_i's routing entry for D will expire and become invalid, thus breaking the complete loop within MAX_ROUTE_LIFETIME duration. In the second case, node D does send a packet through node N_{i+1} within MAX_ROUTE_LIFETIME interval. Since node D is outside the loop, the packet from D must enter the loop from some external node N_e. Without loss of generality, assume that N_e is a neighbor of node N_{i+1} that we are presently considering. Thus node D's packet arrives at N_{i+1} from N_e. However, N_e is different from N_{i+2}, which is the current next hop for D at node N_{i+1} (since N_e is outside the loop). Thus, as described in Section IV, N_{i+1} will switch to maintenance mode to re-learn its next hop for D by invalidating its routing entry for D via N_{i+2}. This too would break the complete loop within MAX_ROUTE_LIFETIME interval. ■

It is easy to see from the above argument that a complete loop soon decays into a partial loop in which one or more nodes do not have a valid routing entry towards D and broadcast the packets destined to D. However the partial loop also lasts only until the broadcasting node re-learns a new route to D, which would happen quickly if the broadcasting node lies along the path of data/dummy-IP packets arriving from D. Furthermore, D is *guaranteed* to send either a data or a dummy IP packet within ACTIVITY_INTERVAL.

VII. Performance Evaluation

In this section we show that, even without explicit control messages, the performance of a backward learning algorithm, such as ECMF, is highly competitive, if not better, when compared to AODV and DSR protocols. Simulations were run using NS2 for different networks with 50 nodes. The number of connections were varied from 10 to 100 between randomly selected node pairs. We used the mobility scenarios from [1]. Mobility speed of the nodes varies between 0 to 20 meters/sec. Nodes move within a rectangular field of 1500mX300m according to the Random Waypoint Model. Each simulation is run for 900 simulated seconds. Pause times were varied from 0s to 900s. We used both TCP-based connections, (for bidirectional communication), and UDP-based Constant Bit Rate (CBR) connections (for unidirectional communication). The UDP CBR sources sent traffic at 4 packets/sec with 64 byte packets. Each data point is averaged over five simulation runs with a different random seed to vary the connection establishment pattern. We count the dummy IP packets as the control packets for ECMF. The MAX_ROUTE_LIFETIME is set to 5 seconds and the dummy IP packets are generated by the ECMF layer at a node after 4 seconds of silence on the part of the network application. For AODV and DSR, we count the Route Request (RREQ), Route Reply (RREP), Route Error (RERR), ARP requests/replies, and Hello messages as the control overhead. For DSR, we *do not* count the source routing information carried in packet headers towards the control overhead. The control packets are counted on a hop-to-hop basis rather than on end-to-end basis. For fair comparison, in AODV simulations we enabled the

Local Repair optimization (which attempts to repair a failed link locally before returning a RREQ message to source) and the Link Layer Detection optimization (which suppresses the transmission of "Hello" messages). Similarly, the simulation of DSR included the optimization of piggybacking RERR and RREP messages onto RREQ messages and the optimization of populating route cache by listening in promiscuous mode. We use three metrics : **(1)** *R/S* is the ratio of the number of packets received by the destination to the number of packets sent by the source. **(2)** *C/R* is the ratio of number of control packets to the number of data packets received by destination. **(3)** *Number of control packets* represents the total number of control messages transmitted.

A. Performance with TCP Traffic

R/S: Figures 4, 5, and 6 show that DSR performs best in terms of R/S values, followed by ECMF and AODV. In Figure 4, as the pause time increases, the R/S value approaches 1 (no packet loss). In Figure 5, as the number of connections increases, channel contention and packet losses increase resulting in a drop in R/S. As in the previous case, DSR performs best, followed by ECMF and finally AODV. Figure 6 shows a universal decrease in R/S with increase in mobility speed. The difference in R/S between DSR and ECMF is less than 1%. It can be seen from these figures that reducing the protocol complexity in ECMF does not automatically translate into better delivery ratio (R/S) when compared against DSR, because reduction in control overhead is offset to somewhat by increase in both data traffic. However, ECMF's delivery ratio remains competitive with both AODV and DSR.

C/R: Figures 7, 8, and 9 show that ECMF maintains a low value of *C/R* around 0.2 because it exploits the bidirectional nature of TCP traffic to maintain routing state without exchanging excessive control messages. DSR and AODV show higher value of C/R for smaller pause times than for larger pause times. ECMF experiences little variation in C/R whereas variation is C/R is large for both AODV and DSR. Figure 9 shows that increasing the mobility speed of nodes results in greater increase in C/R for DSR than for ECMF and AODV, because DSR is unable to benefit from the forward routing information carried in the reverse traffic.

Number of Control Packets: Figure 10 shows that number of control messages in ECMF is much lower than AODV and DSR. For 0s pause time, ECMF generates 4.5 times fewer control packets than AODV and 6 times fewer control packets than DSR. In Figure 11, we vary the number of TCP connections from 10 to 150, for a fixed 60s pause time and fixed mobility speed of 20m/s. One can see that number of control packets generated by ECMF remains much lower than AODV and DSR. As For 150 connections, AODV generates around 220K control packets, DSR generates around 125K control packets, while ECMF generates around 25K control packets. As with C/R, Figure 12 shows that increasing the mobility speed of nodes results in greater increase in number of control packets for DSR than for ECMF and AODV.

B. Performance with Constant Bit Rate (CBR) UDP Traffic

In Figure 13, ECMF achieves an R/S smaller than both AODV and DSR for smaller pause times. In Figure 14, ECMF and DSR achieve almost the same R/S which is smaller than AODV. When compared against results for TCP traffic, the above results show that, as expected, ECMF performs better with bidirectional traffic, than with unidirectional traffic. Figure 15 shows that AODV transmits almost 1.5 control packets for each received data packet. Although, DSR has approximately 0.8 C/R at 0s pause time, it reduces as the pause time increases. ECMF has a C/R less than 0.2 at any given point of time. Figure 16 shows that, as the number of connections increases, the C/R value drops for ECMF. With more number of connections, dummy IP packets are suppressed more often leading to lower C/R value. On the other hand, C/R value increases for both DSR and AODV. In Figure 17, AODV generates around 140K control packets for 0s pause time, DSR around 75K, and ECMF around 12K. In Figure 18, ECMF uses far fewer control packets than others for large number of connections. Due to space constraints, we omit the mobility speed variation results.

VIII. Conclusion

In this paper, we have proposed a set of techniques that can be applied to eliminate the use of explicit control messages in backward learning ad hoc routing protocols. The proposed techniques do not rely upon any extensive control infrastructure for loop prevention, such as spanning trees, destination sequence numbers, or source routing mechanisms. They are particularly suited for routing bidirectional communication sessions with minimal control overhead, and for unidirectional communication, use minimal implicit (data-like) control messages that require no special processing at intermediate nodes.

References

[1] The monarch project. httpe//www.monarch.cs.rice.edu/.

[2] I. Chakeres, E. Royer, and C. Perkins. Dynamic MANET on-demand routing protocol. IETF Internet Draft, October 2005.

[3] C.Perkins and P.Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94*, pages 234–244, 1994.

[4] R. Draves, J. Padhye, and B. Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *Proc. of MobiCom*, 2004.

[5] J. J. Garcia-Luna-Aceves and M. Spohn. Source-tree routing in wireless networks. In *ICNP*, 1999.

[6] Z. Haas, M. Pearlman, and P. Samar. The interzone routing protocol for ad hoc networks. IETF Internet draft, July 2002.

[7] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, volume 353. Kluwer, 1996.

[8] Y. Lee and G. Riley. Dynamic NIx-Vector routing for mobile ad hoc networks. In *Wireless Commn. and Networking Conf.*, 2005.

[9] C. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. *IEEE Workshop on Mobile Comp Sys and Applications*, 1999.

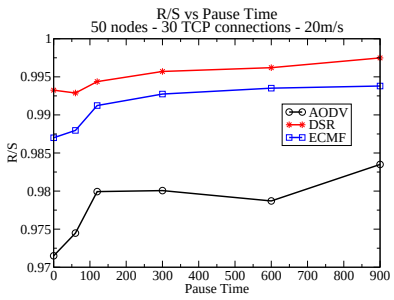

Fig. 4. R/S vs Pause Time for 50 nodes, 30 TCP connections, 20m/s.

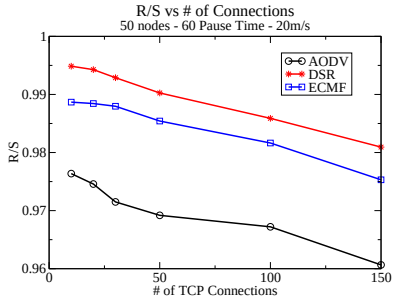

Fig. 5. R/S vs TCP connections for 50 nodes, 60s pause time, 20m/s.

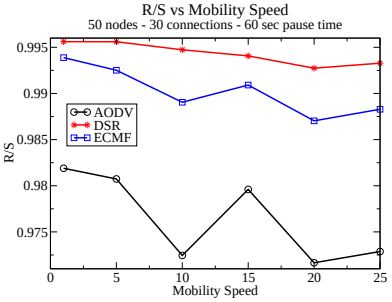

Fig. 6. R/S vs Mobility speed for 50 nodes, 30 TCP connections, 60s pause time

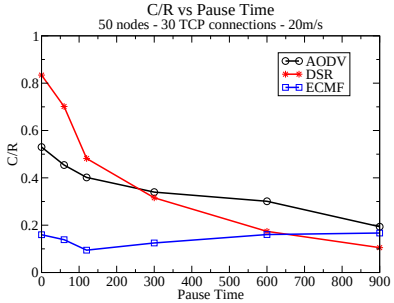

Fig. 7. C/R vs Pause Time for 50 nodes, 30 TCP connections, 20m/s.

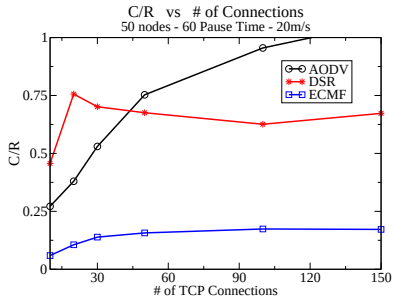

Fig. 8. C/R vs TCP Connections for 50 nodes, 60s pause time, 20m/s.

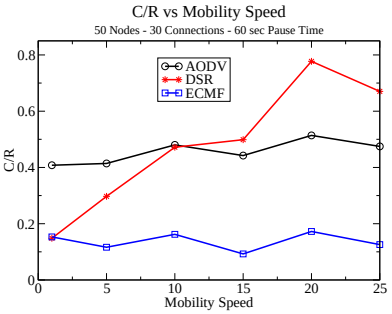

Fig. 9. C/R vs Mobility speed for 50 nodes, 30 TCP connections, 60s pause time

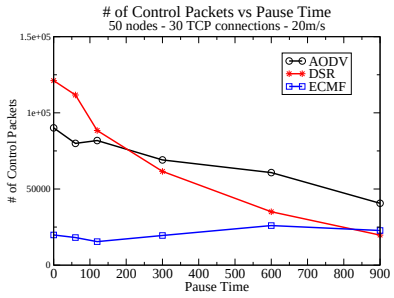

Fig. 10. Control packets vs Pause Time for 50 nodes, 30 TCP connections, 20m/s.

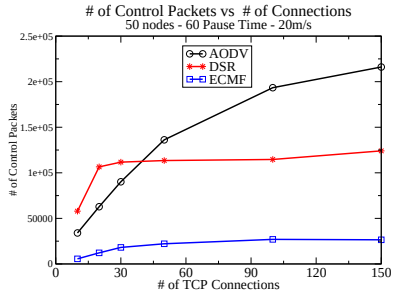

Fig. 11. Control packets vs TCP Connections for 50 nodes, 60s pause time, 20m/s.

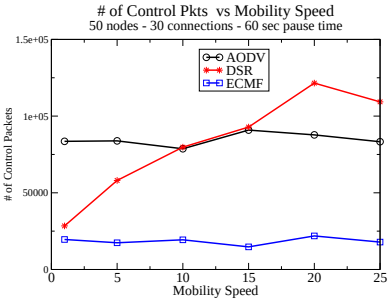

Fig. 12. Control pkts vs Mobility speed. 50 nodes, 30 TCP connections, 60s pause time.

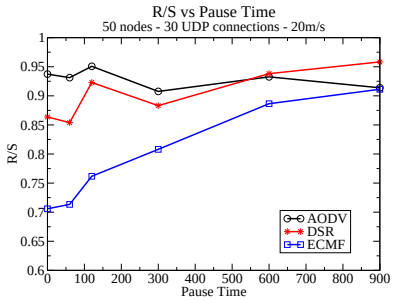

Fig. 13. R/S vs Pause Time for 50 nodes, 30 UDP connections, 20m/s.

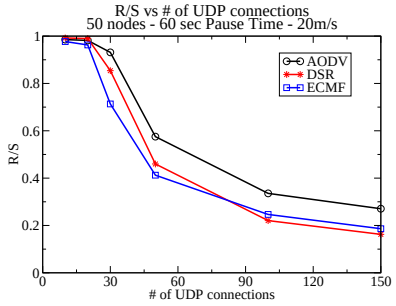

Fig. 14. R/S vs Number of UDP connections for 50 nodes, 60s pause time, 20m/s.

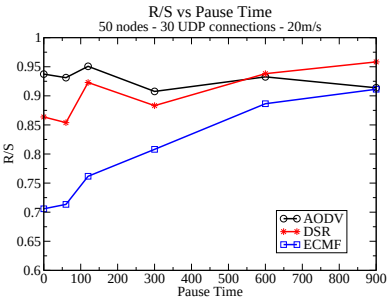

Fig. 15. C/R vs Pause Time for 50 nodes, 30 UDP connections, 20m/s.

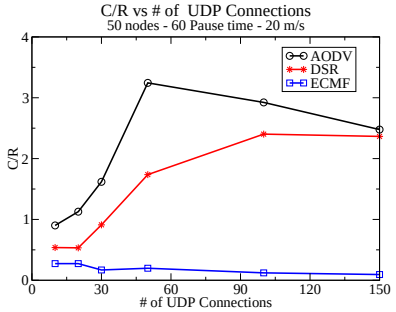

Fig. 16. C/R vs UDP Connections for 50 nodes, 60s pause time, 20m/s.

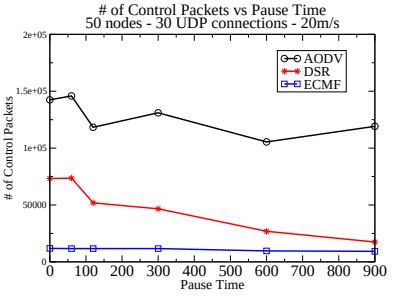

Fig. 17. Control pkts vs Pause Time for 50 nodes, 30 UDP connections, 20m/s.

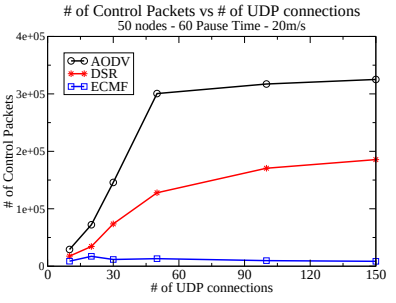

Fig. 18. Control pkts vs UDP Connections for 50 nodes, 60s pause time, 20m/s.