Hierarchical Routing in Ad hoc Mobile Networks

Elizabeth M. Belding-Royer
Department of Computer Science
University of California, Santa Barbara, CA 93106
ebelding@cs.ucsb.edu

Abstract

Clustering is a method by which nodes are hierarchically organized based on their relative proximity to one another. Routes can be recorded hierarchically, across clusters, to increase routing flexibility. Hierarchical routing greatly increases the scalability of routing in ad hoc networks by increasing the robustness of routes. This paper presents the Adaptive Routing using Clusters (ARC) protocol, a protocol that creates a cluster hierarchy composed of cluster leaders and gateway nodes to interconnect clusters. ARC introduces a new algorithm for cluster leader revocation that eliminates the ripple effect caused by leadership changes. Further, ARC utilizes a limited broadcast algorithm for reducing the impact of network floods. The performance of ARC is evaluated by comparing it both with other clustering schemes and with an on-demand ad hoc routing protocol. It is shown that the cluster topology created by ARC is more stable than that created by other clustering algorithms, and that the use of ARC can result in throughput increases of over 100%.

1 Introduction

The recent rise in popularity of mobile wireless computing devices has resulted in a number of protocol proposals for the establishment and maintenance of routes in these networks. Many of the routing protocols for ad hoc networks are classified as either *proactive* or *on-demand* routing protocols [18]. Proactive routing protocols are descendants of traditional distance vector and link state protocols, in that they maintain a routes between all pairs of nodes throughout the lifetime of the network. In contrast to proactive protocols, on-demand approaches create routes only as they are needed by a source node, and maintain these routes only as long as they are needed. These protocols typically have a route discovery cycle for route finding and a route maintenance algorithm for repairing link breaks in active routes.

The scalability of both proactive and on-demand protocols is limited due to the inherent characteristics of the protocols. The communication overhead of proactive protocols is $O(n^2)$, where n is the number of nodes in the network. On-demand protocols, on the other hand, are limited by their route discovery and maintenance techniques. Because of the flooding nature of the route discovery process, and because a route error message must typically travel back to the source node for each broken link, protocols designed in this manner have scalability difficulties when the networks have many nodes and/or span a large geographical area [11]. In a network with rapidly moving nodes, it may not be possible to maintain routes when broken links frequently occur. A new route may no sooner be discovered than another link break occurs in the route.

Some on-demand protocols offer variations of the basic route discovery and maintenance algorithms to improve their scalability. The Ad hoc On-Demand Distance Vector (AODV) protocol suggests using an expanding ring search to reduce the impact of the route discovery flood [13]. AODV

and the Relative Distance Micro-discovery Ad hoc Routing protocol (RDMAR) [1] both utilize local repair of link breaks to prevent the route error message from traveling back to the source node, if the broken link is closer to the destination than it is to the source node. Otherwise, an error message is sent to the source node. The Dynamic Source Routing (DSR) protocol maintains multiple routes per destination so that route discoveries do not need to be performed as often [9].

While these enhancements improve performance, they still do not allow the protocol to scale well to large networks [11]. One alternative routing scheme is clustering protocols. Clustering protocols place nodes into groups, called clusters, and perform hierarchical routing between these clusters. A hierarchical routing scheme can increase the robustness of routes by providing multiple possibilities for routing between clusters. There are a wide variety of clustering protocols. Many clustering protocols establish a cluster leader per cluster and utilize gateway nodes for cluster interconnection [3, 4, 6]; other algorithms take a distributed approach to cluster management [12, 19]. The chapter provided by Steenstrup in [20] provides an overview of many clustering protocols. Descriptions of example clustering protocols related to the work presented in this paper are given in section 2.

This paper presents the Adaptive Routing using Clusters (ARC) protocol. ARC is loosely based on the linked cluster architecture (LCA) presented in [3], in that similar cluster structures are created. Clusters are created with one cluster leader per cluster, and each node within the cluster must be within direct transmission range of the cluster leader. Inter-cluster communication is accomplished through the utilization of gateway nodes. While based on the LCA clustering scheme, there are some key differences between the two approaches. ARC allows for multiple communication paths between clusters, whereas the LCA protocol is limited to just one such path. Further, ARC defines a new method for determining when a cluster leader should revoke its leader status, which prevents the *rippling effect* of further leadership changes. The rippling effect results when one cluster leader change results in additional leadership changes in the network. While leadership changes are necessary so that the number of cluster leaders does not grow over time, these changes are expensive in terms of communication overhead and processing power. Hence they should be minimized. ARC also utilizes a new forwarding mechanism called *limited broadcast*, whereby the flooding of redundant control messages throughout the network can be greatly reduced. Because the transmission of control messages utilizes the resources of both the sender and the receivers, it is important to reduce the redundancy of control message transmissions so that battery power can be conserved.

The remainder of this paper is organized as follows. Section 2 presents related work, including overviews of the protocols used for comparison with ARC. Section 3 describes the operation of the ARC protocol. Section 4 discusses how ARC can be combined with an on-demand routing protocol so that routing between cluster leaders can be achieved. Through simulation, section 5 examines the characteristics of the cluster topology created by ARC. ARC is also integrated with an on-demand protocol to show the performance improvement that can be achieved by combining the protocols. Observations from the simulations are made in section 6. Finally, section 7 concludes the paper.

2 Related Work

The linked cluster architecture (LCA) [3], developed for packet radio networks, is one of the earliest clustering protocols for these networks. The LCA protocol organizes nodes into clusters based on node proximity. Each cluster has a clusterhead, and all nodes within a cluster are within direct transmission range of the clusterhead. Clusterhead election is based on node identifiers, where the node with the largest identifier in a given area becomes the clusterhead. Gateways in the overlapping region between clusters are used to connect the clusters. LCA specifies that there most be only one such gateway designated to interconnect the clusters at any given time. Pairs of nodes can also be

used to connect clusters if there are no nodes in the overlapping region.

Chiang, Wu, Liu, and Gerla have developed a clustering system known as Clusterhead Gateway Switch Routing (CGSR) [6]. In this protocol, packets are routed alternately between cluster leaders and gateways. The authors define various extensions that can be added to CGSR, such as priority token scheduling and gateway code scheduling, in order to control channel access. Additionally, they define a Least Cluster Change algorithm (LCC) that is designed to reduce the number of cluster leader changes, because such changes can result in a substantial amount of overhead. According to the LCC algorithm, a cluster leader change only occurs when two cluster leaders come within range of each other, or when a node is out of range of all other cluster leaders. When two cluster leaders come into contact, the leader that gives up its leader status is chosen by some means such as lowest ID or highest connectivity. Because some of the nodes in one cluster may not be members of the other cluster leader's cluster, the cluster leader change may cause some nodes to be left without a cluster leader. In this case, one or more of those nodes must become a cluster leader. Such changes can propagate across the network, causing a rippling effect of cluster leader changes, as described in Section 3.4. If nodes are moving rapidly, cluster leader changes may continually affect the topology of the network and generate a significant amount of control overhead.

Another approach to clustering is taken by Basagni in [4], which presents two clustering algorithms that take into account the movement speeds of the network nodes. The first of these is the Distributed Clustering Algorithm (DCA), which is intended for "quasi-static" networks in which nodes are slow moving, if moving at all. The other algorithm is designed for higher mobility and is called the Distributed and Mobility-Adaptive Clustering algorithm (DMAC). Both DCA and DMAC assign nodes different *weights* and assume that each node is aware of its respective weight. The weights are used in the determination of the cluster leaders. In the DMAC protocol, if two cluster leaders come into contact, the one with the smaller weight must revoke its leader status. Like the other clustering protocols, this requirement may cause additional leadership changes in the clusters, since each node that just lost its cluster leader must determine whether it is still within the communication range of any other leaders.

Lin and Gerla [12] take a different approach to clustering by requiring that clusters are *non-overlapping*, and by adopting a fully distributed approach to clustering which eliminates the need for a cluster leader. Their rationale for the elimination of the cluster leader is that such a leader must do extra work on behalf of the cluster, and hence may become a bottleneck for the cluster. An additional benefit to the elimination of cluster leaders is that it prevents the route centralization problem discussed in Section 6. However, one of the assumptions needed for their algorithm is that the network topology does not change during the algorithm execution. This assumption may be unrealistic, particularly in networks with rapidly moving nodes.

Numerous other clustering schemes are possible. The approach taken in [10] defines a cluster as a set of nodes which are mutually reachable by a path of at most length k. In their paper, the authors focus on clusters where $k = 1$. The virtual subnet architecture described in [19] defines clusters as physical subnets and creates virtual subnets composed of members of each cluster. The virtual subnets provide the communication paths between the physical subnets. Finally, the Near-Term Digital Radio (NTDR) algorithms [21] create a clustering topology similar to that of the LCA approach. The key difference, however, is that the clusterheads must adjust their transmission range so that they can directly communicate with each other; gateways are not utilized.

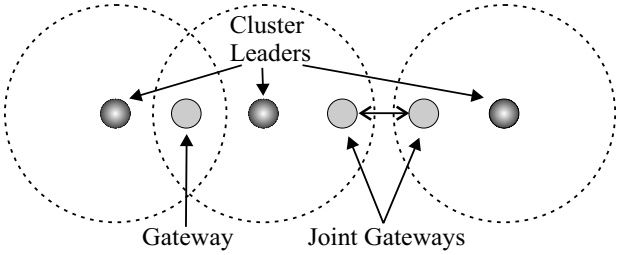


Figure 1: Example Cluster Topology.

3 Adaptive Routing using Clusters (ARC)

Clustering is a method by which nodes are placed into groups, called *clusters*, based on their proximity to one another. As discussed in the previous section, there are many variations on the creation of clusters. One method of clustering, originally proposed by LCA [3], requires a designated *cluster leader* per cluster. This node serves as the representative of the cluster and processes many of the control packets on behalf of the cluster members. Each member of a cluster must be within one hop of the cluster leader. Nodes that are members of more than one cluster are called *gateways*. Gateways are used by cluster leaders to route between clusters. It is often the case that a node is not a member of a certain cluster, but it is in direct contact with another node that is a member of that cluster. In this scenario, the pair of nodes serve as *joint gateways* for the two clusters, and can also be used to route between the clusters. Figure 1 illustrates the difference between gateways and joint gateways. When two cluster leaders are within transmission range of each other, they may communicate directly and eliminate the need for an intermediate gateway.

A clustering protocol can be run as an interface between the IP layer and the routing protocol to increase robustness of routes during node movement. In a network with flat addressing (i.e. AODV [15], DSR [9]), routes are recorded on a hop-by-hop basis. When a link between two nodes in an active route breaks, some action must be taken to repair the link. This action can be in the form of notifying the source node of the link break, repairing the link, etc. On the other hand, in a clustered network, addressing can be hierarchical and routes can be recorded on a cluster-by-cluster basis. Routing may be done either through cluster leaders [6], or a more distributed approach may be taken [12]. As an example, in figure 2 a flat routing scheme may record the route between the source (S) and destination (D) as

$$S \rightarrow CL_1 \rightarrow G_1 \rightarrow CL_2 \rightarrow D.$$

However, a hierarchical addressing scheme that defined routes between cluster leaders would record the route as

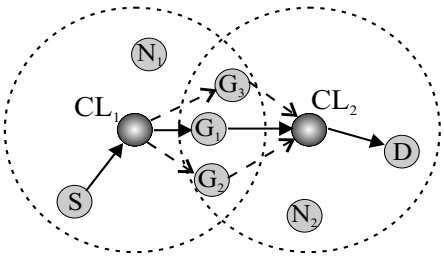


Figure 2: Hierarchical Addressing.

$S \rightarrow CL_1 \rightarrow CL_2 \rightarrow D$.

The primary difference lies in the link between cluster leaders CL_1 and CL_2. If the link between CL_1 and G_1 breaks, a flat addressing scheme requires the link to be repaired. However, with hierarchical addressing, *any* of the gateways between nodes CL_1 and CL_2 can be used to route between the two clusters. Hence, if node G_1 moves out of reach, G_2 or G_3 can be used instead. Furthermore, as gateways move out of range of CL_1 and CL_2, it is likely that other nodes, such as G_4 or G_5, will move such that they become gateways for the two clusters, thereby further maintaining the robustness of the route. In routes with longer path lengths, the robustness is increased substantially.

The Adaptive Routing using Clusters (ARC) protocol is based on LCA in that it creates a one-level cluster topology across a network of nodes. LCA was selected as the foundation for the ARC protocol because the clustering topology naturally lends itself to robust routing between clusters. The key variances between ARC and LCA are that ARC supports the use of multiple gateways between clusters, whereas LCA defines that only one cluster must be selected between a given set of clusters. Also, the basic formation of clusters in the two algorithms is different. In LCA, cluster leaders are selected by node identifiers, whereas in ARC they are selected by connectivity. Further, LCA has the requirement that nodes must remain synchronized and must maintain an accurate global time so that the control channel may be divided into epochs. ARC does not have this requirement. Finally, ARC introduces a new scheme for cluster leader management to reduce the rippling effect after leadership changes, and it utilizes limited broadcast for reducing the redundancy of network broadcasts.

Like LCA, nodes in the ARC network can have one of three status levels:

- `cldr` - a cluster leader
- `gateway` - a gateway for two or more clusters
- `node` - an ordinary node

A node's status level may change during the lifetime of the network based on the status level of its surrounding neighbors. The following sections describe the ARC protocol functions for creating and maintaining a clustered network topology.

3.1 ARC Tables

Each node running the ARC protocol maintains either one or two tables. All non-cluster leaders maintain a Cluster Leader Table. For every cluster of which the node is a member, there is an entry in this table containing the cluster leader IP address. Additionally, if the node can reach other cluster leaders through the use of a joint gateway, that cluster leader's IP address is recorded in the table as well. The IP address of each joint gateway that can be used to reach the cluster leader is likewise recorded in the table entry associated with that cluster leader.

If a node is a cluster leader, then it maintains two tables: a Node Table, and a Neighbor Table. Each entry of the node table contains an address of a node in that leader's cluster, as well as a list of the neighboring cluster leaders with which that node is able to communicate, either directly or through a joint gateway. Each entry of the neighbor table contains the address of a neighboring cluster leader, as well as a list of those gateways that can be used to reach the neighboring cluster leader.

3.2 Initialization

A node initializes by broadcasting a *hello* message. The hello message includes the node's IP address and current status level. During initialization, the node's status level is undefined. The hello message

also contains a list of a node's cluster leaders. This list is empty during initialization; however, after initialization it is used for learning local connectivity information. All hello packet transmissions are link local broadcasts; they have a time to live (TTL) value of one and are not rebroadcast by neighboring nodes.

After sending the initial hello, the node then waits a discovery period. During this time it records in its cluster leader table the IP address of any cluster leaders from which it receives a hello. At the end of the discovery period it determines whether it has heard from any cluster leaders. If it has not, this implies that it is not within any current cluster boundaries, and so it must become a cluster leader itself. The node therefore initializes its status to `cldr`. On the other hand, if it has heard from a cluster leader, it determines the number of cluster leaders from which it has heard. If it has received a hello from exactly one cluster leader, it initializes its status to `node`. Otherwise, if it has heard from multiple cluster leaders, its status is `gateway`, indicating that it can directly correspond with multiple cluster leaders and can thus be used to route between them.

Regardless of its achieved status level, the node broadcasts a hello indicating this new status. If the node is a `node` or a `gateway`, it also includes in the hello message a list of all cluster leaders from which it has received hellos. After initialization, the node continues to broadcast a hello message with its current status level and connectivity information every `hello_interval` seconds.

In the event that two nodes within transmission range of one another initialize at approximately the same time, it is possible that both will become cluster leaders. In this case, the method for merging clusters described in section 3.4 applies. If one cluster is a subset of the other cluster, the leader of the subset cluster revokes its leader status and joins the other cluster. Otherwise, if neither cluster is a subset of the other, both nodes remain leaders. See section 3.4 for further details.

3.3 Learning the Local Topology

When a node receives a hello from another node, it gleans whatever information about the local topology that it can from that hello message. There are four scenarios for reception of a hello message:

(i) Leader receives a hello from non-leader.
(ii) Leader receives a hello from another leader.
(iii) Non-leader receives a hello from leader.
(iv) Non-leader receives a hello from another non-leader.

The method by which the hello is processed depends upon which of these scenarios is satisfied.

Leader Receives a Hello from Non-Leader

Hello messages from non-cluster leaders list the cluster leaders with which the node is able to communicate. When a cluster leader receives a hello from a non-leader, it first ensures it has this node listed in its node table. Then, it compares the cluster leader list contained in the hello message with the list associated with the node's entry in the node table. It verifies that each cluster leader contained in the hello message is listed in the node table entry, and that there are not any stale entries for cluster leaders contained in the node table entry that are no longer in the hello message. It then goes through the reverse process by ensuring that there is a cluster leader entry in the neighbor table for each cluster leader listed in the hello message, and that the sending node is listed as a gateway for each such cluster leader.

Leader Receives a Hello from Another Leader

When a cluster leader receives a hello from another cluster leader, it verifies that there is an entry for this leader in the neighbor table. It notes that it can reach this leader directly, and then it determines whether it should give up its cluster leader status by the conditions given in section 3.4.

Non-Leader Receives a Hello from Leader

A node receiving a hello message from a cluster leader ensures that there is an entry for this cluster leader in its cluster leader table. It also notes that it can reach this cluster leader directly, i.e. it does not require a joint gateway. It then re-evaluates its status level. If it was previously a `node` and it can now communicate with more than one cluster leader, it updates its status to `gateway`. Otherwise, if this is the only cluster leader with which this node can communicate (either directly or through a joint gateway), it retains its `node` status.

Non-Leader Receives a Hello from Another Non-Leader

Finally, a node that receives a hello message from another non-cluster leader must determine for which cluster leaders this other node can serve as a joint gateway. The node receiving the hello iterates through the list of cluster leaders in the hello and notes which cluster leaders the neighboring node can reach directly. For each of these cluster leaders, the node ensures that there is a cluster leader entry in its cluster leader table. It also verifies that the source of the hello is listed as a joint gateway for each of those leaders. Finally, it checks that the neighboring node is not listed as a joint gateway for any cluster leaders no longer contained in the cluster leader list.

3.4 Maintenance

The ARC algorithm works simply through the periodic exchange of hello messages; no other messages are needed to maintain the cluster topology. Regardless of their status level, nodes maintain lifetimes for each of their neighbors. Each time a node receives a hello message from a neighbor, it updates the lifetime associated with that neighbor. If a node does not receive a hello message from a given neighbor within the remaining lifetime for that neighbor, all table entries for that neighbor expire and are deleted. The choice of lifetime and the frequency of the hellos effects the accuracy of neighborhood information [14]. If the lifetime is too short, neighbors may be prematurely deleted from the cluster tables. Alternatively, if the lifetime is too long, neighborhood information may become stale and inaccurate. Similarly, the greater the frequency at which hello messages are sent, the more up-to-date the routing information. Hello messages sent at longer intervals result in less timely information about the neighborhood. The simulation results presented in section 5 examine this effect.

The cluster hierarchy is relatively stable during node movement; nodes do not frequently change status levels. A node may update its status from `node` to `gateway` if it becomes able to communicate with more than one cluster leader, either directly or through a joint gateway. Similarly, if a gateway node moves out of contact of all but one of its cluster leaders, it downgrades its status to `node`. The most significant status change occurs either when a node becomes a cluster leader, or a cluster leader revokes it leader status and becomes an ordinary node.

Becoming A Cluster Leader

A node considers becoming a cluster leader when it loses contact with all of its cluster leaders (i.e. it is no longer a member of any cluster). It must be a member of at least one cluster to remain a

non-leader. This situation may occur if a node wanders towards the boundary of the network area. If the node is no longer a member of any cluster, it broadcasts a hello message, as in initialization (section 3.2). If there are one or more cluster leaders of which it is unaware in its vicinity, these leaders respond by sending the node a hello. This ensures that the node does not change its status prematurely. If the node does not receive any hello messages from cluster leaders in the next discovery period, it becomes a cluster leader. It then broadcasts a hello indicating its new status level.

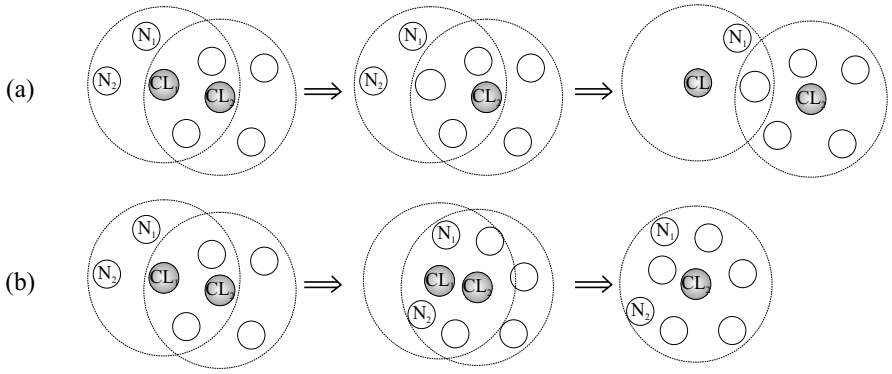


Figure 3: Cluster Leadership Change.

The failure of a cluster leader results in a situation similar to the one previously described. If a cluster leader fails, its member nodes notice the loss due to the lack of hello message reception from that leader. If a member node has other clusters of which it is a member, the node remains a member of those other clusters. On the other hand, if the failed cluster leader was the only cluster leader for a given node, that node acts according to the method described in section 3.2, whereby it broadcasts a hello message. If more than one node has lost its only cluster leader, each node broadcasts this hello when it notices the failure of the cluster leader. In this situation, more than one node may become a cluster leader if the time at which the nodes broadcast the hello is approximately the same. The new leaders then determine whether one of them should revoke its leader status according to the following algorithm.

Becoming A Node

A clustering protocol should have a method by which leaders give up their leader status in order to prevent a slow growth in the number of cluster leaders over the lifetime of the network. When a node wanders to the perimeter of the network, it often loses direct communication with its cluster leaders. Because it no longer has a cluster leader, it becomes a leader itself. Hence, if there is no method for a leader to become a non-leader, there will be a growth in the number of leaders over the course of the network lifetime. Taken to extremes, without a method to reduce the number of cluster leaders, a long-lived network could result in each node being a cluster leader.

A cluster leader knows it is within communication range of another cluster leader when it receives a hello from the other leader. In the algorithms described in [4, 6, 8], any time two cluster leaders move within transmission range of each other, one of the leaders must give up its leader status. The choice of which node remains a leader can be based on IP address, number of cluster members, etc. The problem with these methods, however, is that they can cause a *rippling effect* in cluster leader changes across the network. The rippling effect results in additional leadership changes within the network that are triggered by the initial change. Leadership changes are expensive due to the communication overhead and processing power utilization that results. In networks where routes

are recorded between cluster leaders, leadership changes result in routing changes, and hence may generate routing overhead as well. The rippling effect, therefore, can have a detrimental performance impact on a network and should be avoided.

Consider the example given in figure 3(a). Cluster leaders CL_1 and CL_2 come within range of each other. For this example, let the cluster leader with the higher IP address retain its leader status and the leader with the lower IP address becomes an ordinary node. Using this method, CL_1 becomes an ordinary node and joins CL_2's cluster, while CL_2 remains a leader. Then consider what happens to nodes N_1 and N_2. Because CL_1 is no longer a cluster leader, these two nodes are now not a member of any cluster. Hence, they each broadcast hellos, receive no response, and one becomes the new leader. In this way, the change of CL_1's status results in further status changes. Cluster leader changes are expensive as they result in routing changes and control overhead; hence they should be minimized.

Instead of requiring one leader to revoke its leader status whenever two cluster leaders come within transmission range of each other, the ARC algorithm requires one leader to become an ordinary node only when its cluster becomes a *subset* of the other leader's cluster. If one cluster is a subset of the other cluster, it is assured that there will not be any nodes left without a cluster leader, and the rippling effect is prevented. A cluster leader knows its cluster is a subset of another cluster because every member of its cluster is a direct gateway (i.e., not a joint gateway) for the other cluster leader.

Figure 3(b) illustrates the progression of one cluster as it becomes a subset of another. In the figure, the two cluster leaders have come into contact, but each cluster has members which are not also a part of the other cluster. As the nodes continue to move, each of the nodes in CL_2's cluster moves into the transmission range of CL_1, resulting in CL_2's cluster becoming a subset of CL_1's cluster. These two clusters then merge, and CL_1 becomes the overall cluster leader. Note that no further cluster reconfigurations are necessary because no node in CL_2's cluster is left without a leader after CL_2's status change.

3.5 Limited Broadcast

In addition to increased robustness of routes, one of the primary benefits of clustering is that the flooding of control messages (such as AODV's Route Request (see section 4)) across the network can be reduced so that only the cluster leaders need to be flooded. This reduced broadcast mechanism is called *limited broadcast*. The premise of limited broadcast is that when a cluster leader broadcasts a packet with TTL greater than one (implying it should be rebroadcast), the gateways rebroadcast the packet only if they are able to reach a cluster leader that has not yet been sent the packet. The following example illustrates the limited broadcast algorithm.

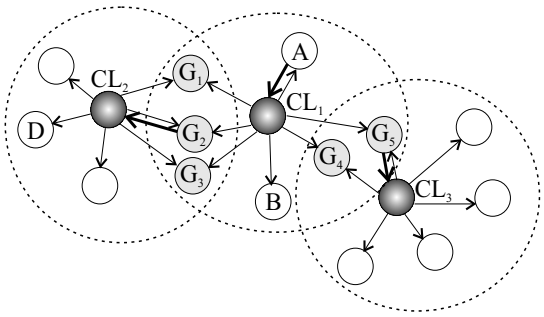


Figure 4: Limited Broadcast.

Suppose node A in figure 4 wishes to find a route to some destination D. Node A broadcasts its

route request with an ARC header is added to the packet. This header lists the cluster leaders that node A is able to reach directly, in this case CL_1. When cluster leader CL_1 receives the packet, it processes the packet and then rebroadcasts it. When a non-cluster leader receives the packet from either node A or cluster leader CL_1, it checks the list of cluster leaders that it can reach directly against the list contained in the packet header. If the node receiving the packet can reach at least one cluster leader that is not listed in the packet header, it buffers the packet, along with a list of cluster leaders it can reach, and sets a short random timer. During this time, the node listens for rebroadcasts of the packet by other nodes. If it receives a rebroadcast of the packet by another node, it checks the packet header's list of cluster leaders against its list that it has buffered with the packet. If any of the cluster leaders contained in its own list are included in the packet header, it marks those cluster leaders as having received the packet. When the node's timer expires, it examines its retransmission list to determine whether each cluster leader has already received the packet. If any of the cluster leaders have not had the packet retransmitted to them, the node retransmits the packet and includes in the packet header the identifier of *every* cluster leader that the node is capable of reaching, including those that have already received the packet. Otherwise, if every cluster leader has already been sent the packet, it drops the packet. Note that the gateway does not process the packet; it only determines whether it needs to retransmit it.

In the example, suppose gateway G_2's timer expires first. It has not heard the retransmission of the packet to CL_2, so it proceeds to retransmit the packet and lists CL_2 in the packet header. Gateway G_1 receives G_2's retransmission. It finds the packet in its buffer and checks the retransmission list associated with it. Because CL_2 is listed in its buffer, G_1 marks this leader as having received the packet. When its retransmission timer expires, G_1 notes that the only cluster leader listed in the retransmission list has already been transmitted the packet, and so it does not rebroadcast the packet. Gateway G_3 also proceeds in this manner. Similarly, suppose gateway G_5's timer expires before G_4's. G_5 rebroadcasts the packet and lists CL_3 in the packet header. When G_4 receives the packet, it marks CL_3 as having received the packet, and hence does not rebroadcast the packet when its timer expires. Note that a node such as B, which is not a gateway, does not buffer the packet for retransmission because there are no additional cluster leaders that it is able to reach. In this manner redundant broadcasts are suppressed, so that the impact of flooding is greatly reduced.

In the case that G_1's timer expires first, it rebroadcasts the packet to CL_2. It is possible that gateway G_3 is not within the transmission range of G_1. In this case, it does not overhear the transmission. When its own retransmission timer expires, gateway G_3 thinks that CL_2 has not yet received the packet, and so it retransmits the packet as well. The extra transmission decreases the reduction in redundancy provided by limited broadcast, but it does not affect the correct operation of the protocol.

In a network protocol without limited broadcast (such as in AODV without clustering), node A would broadcast the original route request. Every node within its transmission radius would rebroadcast it, and so forth, until nearly every node in the network had broadcast the packet. Because limited broadcast places additional restrictions on packet rebroadcasting, it is able to reduce the number of redundant broadcasts caused by a single original broadcast.

Note that because limited broadcast reduces the redundancy of broadcast transmissions, it performs optimally it a network which does not suffer from significant packet loss. In networks where packet loss is substantial, redundancy may be beneficial. The use of limited broadcast in these networks remains to be further investigated.

4 Combining ARC with AODV

The ARC clustering protocol can be run underneath virtually any on-demand routing protocol for ad hoc networks. In this scenario, ARC serves as an interface between the routing protocol and the IP layer. For the simulations, the Ad hoc On-Demand Distance Vector routing protocol (AODV) has been selected to provide routing capability. AODV is an on-demand routing protocol that utilizes a *route discovery* cycle to discover routes. A source node seeking a route broadcasts a *route request* packet. Any node with a current route to the destination unicasts a *route reply* to the source node. Additionally, when a link break in an active route occurs, a notification of that link break is sent to the source through a *route error* packet. Additional details of the protocol can be found in [13] and [14].

A cluster leader serves as a representative for the nodes in its cluster and processes routing packets on the node's behalf. To accomplish this, the ARC protocol intercepts all AODV control packets before they reach the AODV module. If the node is a cluster leader, the packet is passed to AODV for processing. If the node is not a cluster leader, the packet is either forwarded or buffered, depending on the type of packet. If it is a unicast packet, such as a route reply, the packet is forwarded by the gateway to the next cluster leader on the route to the destination of the RREP. Otherwise, if the packet is a broadcast packet such as a route request, then the gateway either rebroadcasts the packet or buffers it, depending on whether or not limited broadcast is being utilized.

Because non-cluster leaders do not process AODV control packets, they do not learn of routes through route requests and replies. It is the responsibility of the cluster leader to inform gateway nodes of routes of which they should be aware. This is accomplished through a new packet type called the *Route Activation* packet (RTAct). When a cluster leader receives a route reply, the next hop indicated is the next cluster leader towards the destination. Unless the cluster leader is within transmission range of this next hop cluster leader, it must select a gateway to use to reach that neighbor. It chooses one of the gateways listed in the neighbor table entry for the neighboring cluster leader, and then unicasts this node a RTAct message. The RTAct contains the destination node's address, the address of the next hop cluster leader to reach the destination, and the number of hops to the destination.

When the gateway receives this packet, it creates a route table entry for the destination. If this node cannot reach the next hop cluster leader directly, but instead requires a joint gateway, then it in turn sends the joint gateway the RTAct message. In this way, when the gateway nodes receive data packets addressed to the destination, they know the next hop to which to forward the packet.

In the event that a gateway selected as the next hop along an active route moves out of reach of either of its cluster leaders along that route, a new gateway must be chosen to connect the two cluster leaders. The cluster leader upstream of the break selects a new gateway from its neighbor table entry for the neighboring leader. It unicasts to this gateway a RTAct packet that contains all of the necessary information to continue routing to the destination. If there are no other gateways between the two cluster leaders, the upstream cluster leader times out the route, notifies AODV of the broken link, and then AODV sends a route error message to the source of the route. Because of the likelihood of multiple gateways between cluster leaders, many link breaks can be patched through the selection of an alternate gateway.

As an optimization, non-leader nodes could snoop routing control traffic in order to learn routing information. This would eliminate the need for the RTAct message after the RREP is received. However, once an initial gateway along the route moves out of range, a new gateway must be selected. Because this gateway did not receive the original RREP packet, the RTAct message is still needed to notify it of the routing information. For consistency, and to remove the workload of processing routing control messages from non-leader nodes, the RTAct message is used in all instances to notify

gateways of routing information.

5 Simulations

Two sets of simulations have been performed - ARC alone, and ARC combined with AODV. The first set of simulation results are used to evaluate the cluster topology ARC creates. The characteristics of the topology is examined, and then the stability of the cluster hierarchy is investigated by comparing the subset algorithm with two other popular methods for cluster leader revocation. In the second set of simulations, the performance of the ARC/AODV combination is compared against that of AODV by itself to determine the benefit that clustering provides.

To evaluate the stability of the subset algorithm, the algorithm is compared against two alternatives for leader revocation. The first is the Least Cluster Change (LCC) algorithm implemented by the CGSR protocol [6]. In this approach, when two leaders come within transmission range of each other, one of the leaders must give up its leader status. The leader with the least ID becomes an ordinary node, while the leader with the greater ID remains a leader. The other leader revocation algorithm uses a weight-based metric. When two leaders come within direct transmission range of each other, the leader with the greatest weight remains a leader, while the leader with the lower weight becomes an ordinary node. This is the algorithm utilized by the DMAC protocol [4]; however in this work, the metric for the weight calculation is not specified. A different work [5], however, investigates a variety of weight metrics and their impact on cluster stability. For the simulations presented here, the weight for each leader is assigned to be the number of members in its cluster. The leader with the most cluster members remains a leader. In the case of a tie, the LCC algorithm is used to select the remaining leader. By examining the number of status level changes during the lifetime of the network, the overall stability of the cluster topology can be evaluated.

5.1 Environment

The simulations have been performed using the GloMoSim [2] simulation package. This simulator models the OSI seven layer network architecture and includes models of IP routing and UDP. The simulator also allows for network node mobility, thereby providing for simulation of mobile ad hoc networks.

The mobility model used in each of the simulations is the random direction model [17]. In each simulation, nodes are initially placed randomly within a predefined $L \times L$ area. Each node then chooses a random direction between 0 and 360 degrees, a speed from some predefined range of speeds, and then proceeds in that direction at that speed. Once the node reaches the boundary of the area, it chooses a period of time to remain stationary. At the end of this "pause time," the node chooses a new direction, this time between 0 and 180 degrees, adjusted relative to the boundary along which the node is located. The node then resumes movement at a newly selected speed. This process repeats throughout the simulation, causing continuous changes in the topology of the underlying network. The pause time used during the simulations is 30 seconds. The node movement is varied between either zero and 5m/s, or zero and 10m/s, as indicated by the following graphs.

The MAC layer protocol used in the simulations is the IEEE standard 802.11 Distributed Coordination Function (DCF) [7]. In the simulations, the data rate is 2 Mb/sec, and the data packet size is 64 bytes. There are 20 randomly selected source/destination pairs, and each source has a sending rate of four pkts/sec. The propagation model used is the free space model [16] with threshold cutoff included in the GloMoSim simulation package. The free space model has a power signal attenuation of $1/d^2$, where d is the distance between nodes. The radio model used also has capture capability,

Number of Nodes	Room Size
50	1000m×1000m
100	1500m×1500m
250	2400m×2400m
500	3450m×3450m
750	4300m×4300m

Table 1: Room Sizes for Network Simulations

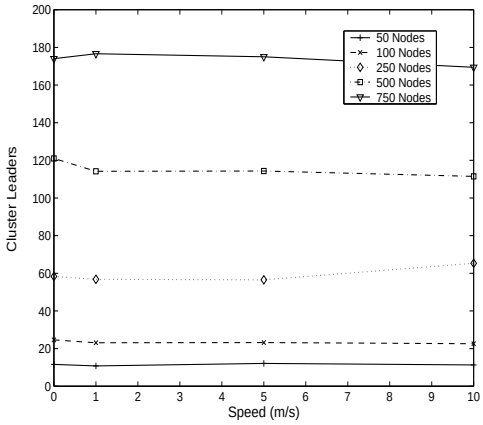


Figure 5: Total Cluster Leaders.

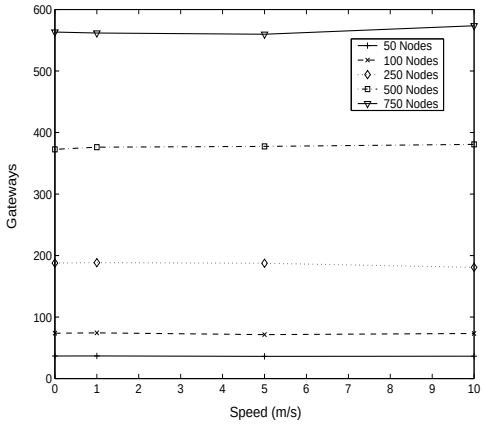


Figure 6: Total Gateways.

where it can lock on to a strong signal during interference from other signals, and still receive the packet. Other interfering packets with weaker signal strength are dropped.

Simulations of the basic ARC protocol, and ARC combined with AODV, both with and without limited broadcast, have been performed. Networks with between 50 and 750 nodes are evaluated. The room sizes for the different numbers of nodes are shown in table 1. These areas were chosen so that the node density is approximately constant across the different simulations, and hence the impact of network size can be investigated. On average, there are seven neighbors per node. The protocols have also been evaluated in networks of increasing density. To evaluate the characteristics of the ARC protocol, a network of 100 nodes with varying transmission ranges was simulated. The impact of network density on the ARC/AODV combination was also evaluated in a network of 100 nodes. The nodes in this network also had increasing transmission ranges in a constant sized network. This resulted in increasing density.

Ten simulation runs were completed for each protocol/network size/node speed combination, where each run had a different initial network configuration. The results of these simulations were averaged together to produce the resulting graphs. Each simulation models 300 seconds of real time.

5.2 Performance Results

The following three sections present the results achieved by the simulations.

ARC Characteristics

The topological formation of the different cluster hierarchies created by ARC is examined in figures 5 and 6. Figure 5 illustrates the number of cluster leaders at simulation end in each of the different

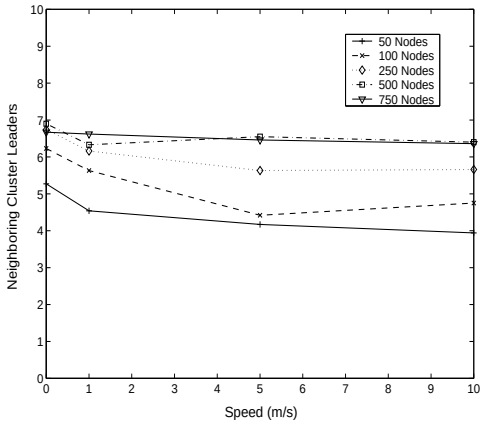


Figure 7: Neighbors per Cluster Leader.

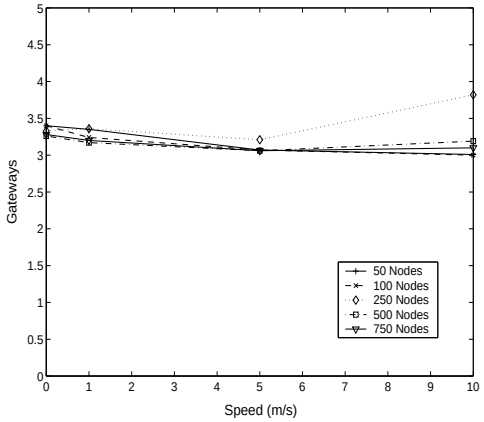


Figure 8: Gateways per Neighboring Cluster Leader.

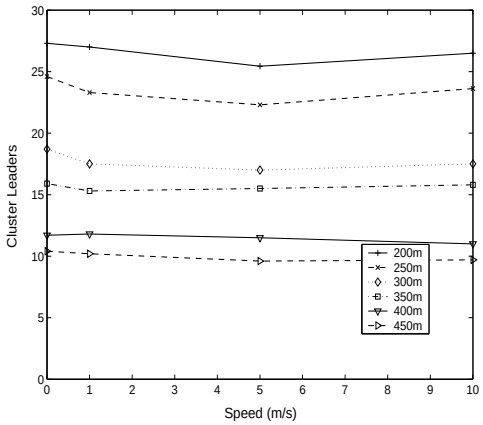


Figure 9: Number of Cluster Leaders in Varying Density Networks.

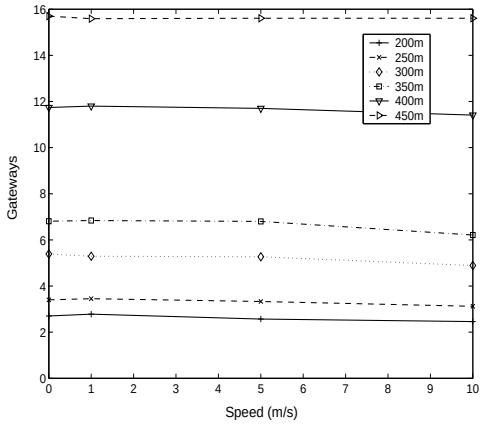


Figure 10: Gateways per Neighboring Cluster Leader in Varying Density Networks.

simulations, while figure 6 shows the number of gateways at simulation end. The graphs show that the number of nodes at each status level is fairly stable for the different mobility speeds, and that there is an increase in the number of cluster leaders proportionate to the increase in the network size. The number of cluster leaders in each simulation is approximately 25% of the number of nodes in the network. The node density is consistent in the different size networks; consequently, the percentage of cluster leaders and gateways should likewise be constant. This is verified by the figures.

The graphs also indicate that the ordinary nodes (i.e., nodes that are neither leaders nor gateways) typically comprised less than 2% of the network nodes. The vast majority of the non-leaders are able to communicate with two or more cluster leaders. This leads to high inter-cluster connectivity and increased route robustness, since there are multiple gateways between each cluster. The number of nodes at each status level, however, is dependent on the density of the network, as shown in the varying density experiments.

Figures 7 and 8 illustrate the high inter-cluster connectivity levels achieved by the protocol. Figure 7 indicates the average number of cluster leaders that are reachable by a given cluster leader through its gateways. The number of neighboring cluster leaders remains relatively constant over the

Transmission Range	Neighbors
200m	4.8
250m	6.2
300m	9.7
350m	12.1
400m	18.4
450m	22.7

Table 2: Average Number of Neighbors.

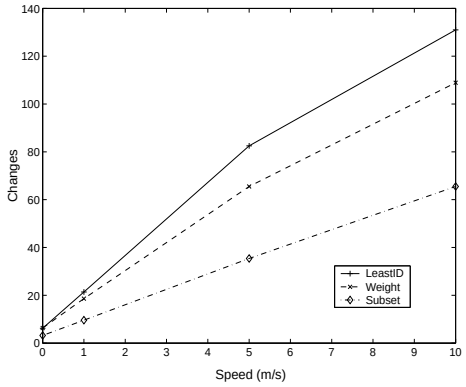


Figure 11: Total Cluster Leader to Node Changes.

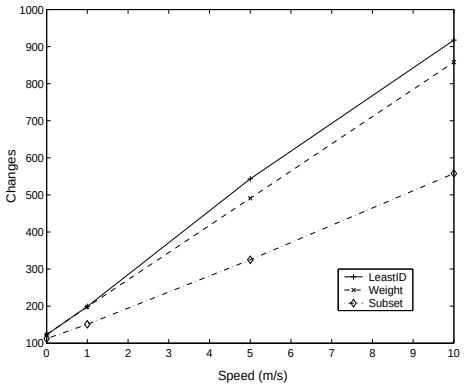


Figure 12: Total Status Level Changes.

varying mobilities, and increases slightly for the greater network sizes due to the larger number of cluster leaders in these networks. The figure indicates a high connectivity level for all networks, with the average cluster leader being able to reach between four and seven neighboring cluster leaders through its gateways.

The number of gateways between clusters is indicated by figure 8. With an average of three to four gateways between clusters, routes are more robust to node movement; if one gateway moves out of range, there are likely to be many others capable of maintaining the connection.

Figures 9 and 10 illustrate the relationship between the network density and the cluster topology. These figures show a 100 node network contained in a 1500m×1500m room. The transmission radius is varied between 200m and 450m for each of the simulations. The average number of neighbors in each network is given in table 2. Figure 9 illustrates that networks with higher density have fewer cluster leaders. Because the nodes are more tightly packed together, more nodes are members of a given cluster. Conversely, figure 10 shows that as the network density decreases, the nodes become more sparse, and so there are fewer gateways available to connect cluster leaders.

In the remaining simulations, the network sizes are as indicated in table 1.

Leader Revocation

Figures 11 through 13 evaluate the effectiveness of the subset algorithm for leader revocation. For this set of experiments, 50 nodes are modeled in a room of size 1000m×1000m. The transmission range of each node is 250m. All the other parameters are the same as the previous set of simulations. In the following graphs, the LCC algorithm is indicated by the "LeastID" label.

Figure 11 shows the total number of cluster leader to node changes throughout the simulation. As the average node speed increases, the number of such changes rises. At higher speeds, nodes change neighbors more rapidly, and so the probability that two cluster leaders come within transmission range of each other increases. The LeastID algorithm results in the greatest number of leadership changes during the simulation, often more than twice as many cluster leader changes as the Subset algorithm. The Subset algorithm yields the fewest number of cluster leader changes. This is primarily due to the more strict requirement for a node to give up its leader status.

Figure 12 represents the total number of status changes in the simulations. This includes gateway-to-node and node-to-gateway status changes. This figure is consistent with the previous result, in that the LeastID and Weight algorithms produce the greatest number of status level changes, while the Subset algorithm produces the fewest.

The previous two graphs alone do not give an indication of the relative stability of the three algorithms. The stability of the cluster topologies is more closely reflected by figure 13. This figure represents the number of cluster leaders at each second during the first 140 seconds of the simulations. The figure indicates that the Subset algorithm results in a fairly stable number of cluster leaders that fluctuates between 10 and 12 cluster leaders, after initially fluctuating to 13 cluster leaders. The LeastID and Weight algorithms, however, show much wider variance. The LeastID approach fluctuates rapidly between 9 and 15 cluster leaders, while the Weight algorithm oscillates between 8 and 12 cluster leaders. In particular, the LeastID approach varies between 9 and 15 cluster leaders on a period of only one second. A high number of cluster leader changes results in significant communication overhead for routing updates and cluster reconfigurations, and leads to an unstable hierarchical topology.

ARC and AODV

The sending frequency of hello messages represents a tradeoff between bandwidth utilization and power consumption, versus the most up-to-date routing information. Figure 14 represents the number of packets able to be delivered by ARC running over AODV without the limited broadcast mechanism in a network of 100 nodes. The hello frequency of cluster leaders is held constant at one hello per second, while that of the non-cluster leaders is varied. The lines in the graph indicate the different hello sending frequencies of the non-cluster leaders. For example, the line labeled "500 msec" indicates a hello message sending frequency of one hello every 500 msec for non-leader nodes. As was shown in figure 5, cluster leaders comprise approximately 25% of the network nodes. Figure 14 indicates that the difference in the number of delivered data packets increases as the rate of mobility

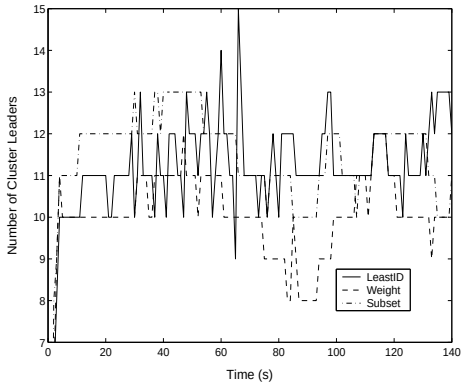


Figure 13: Number of Cluster Leaders during Simulation.

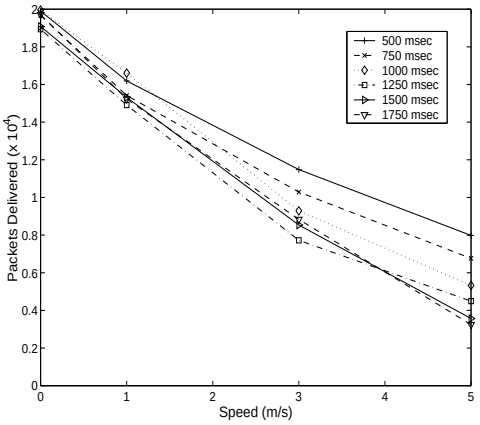


Figure 14: Number of Packets Delivered for Varying Hello Frequency (Without Limited Broadcast).

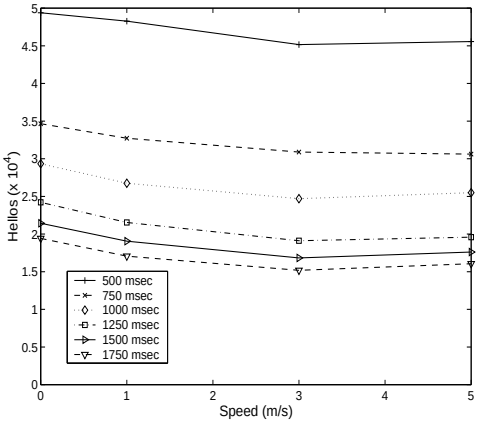
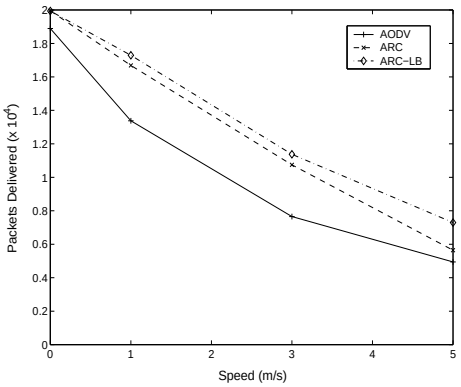
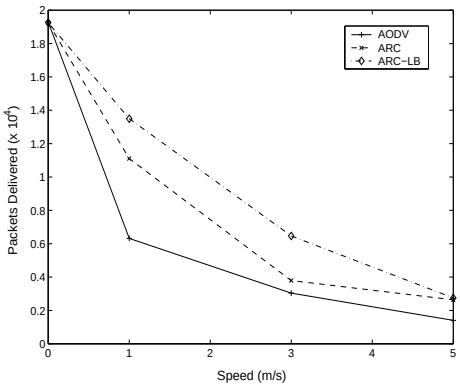


Figure 15: Number of Hellos Transmitted for Varying Hello Frequency.



(a) 100 Nodes



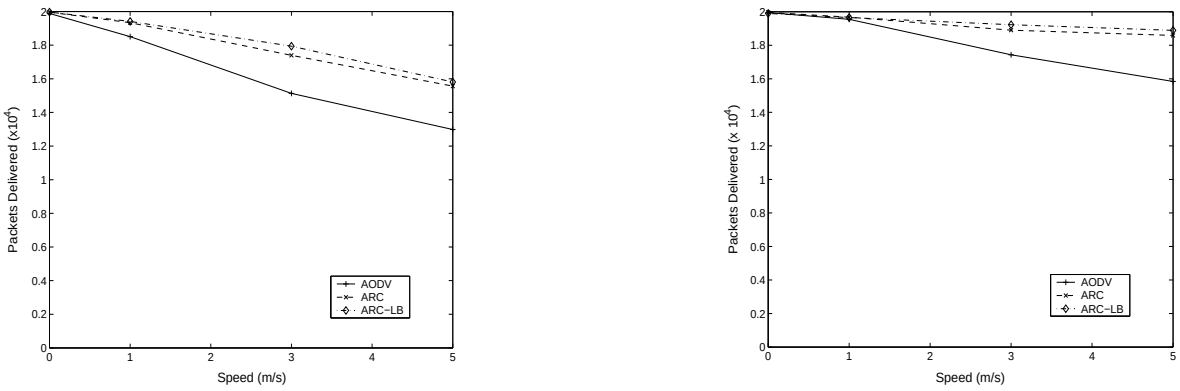
(b) 750 Nodes

Figure 16: Number of Packets Delivered.

increases. For slow mobilities, a sending interval of 1000 msec produces the greatest number of delivered packets, while for faster mobility the highest frequency produces the best results. The drawback to the increased sending rate, however, is shown in figure 15 as the total number of hello messages transmitted by all network nodes during the simulation. To balance the number of delivered data packets with the number of hello messages produced, a sending interval of 1000 msec is used for the remainder of the simulations.

Figures 16 and 17 show the number of data packets delivered to destinations in each of the different sized networks. In each scenario, ARC with Limited Broadcast (ARC-LB) outperforms ARC without Limited Broadcast. Figure 16 illustrates the results in networks of increasing size, while figure 17 illustrates the results in the 100 node network with increasing density.

For moderately moving networks, figure 16 indicates that both ARC versions show superior performance to AODV. The difference is largest with slower moving nodes, and diminishes somewhat as the mobility increases. This is due in part to the selection of 1000 msec as the hello sending rate for non-cluster leader nodes. If a more frequent sending rate had been chosen, the performance



(a) 350m Range

(b) 450m Range

Figure 17: Number of Packets Delivered.

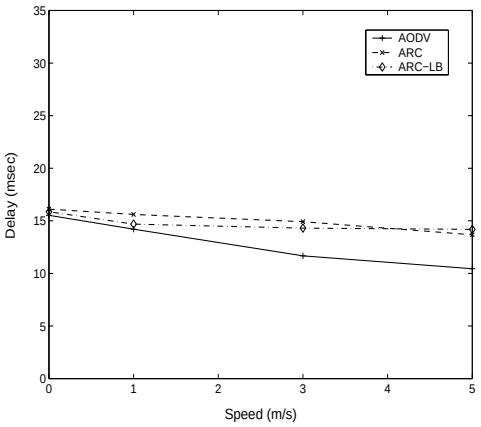
would have improved for the higher mobilites, at the expense of more hello messages. At the moderate node mobilities, however, a performance improvement of over 100% is shown in the 750 node network.

Interestingly, figure 17 shows an opposite trend from figure 16. In figure 17, the performance improvement increases as the node mobility increases. As the network density increases, there are increasingly more nodes per cluster, as indicated by figure 9. With more nodes per cluster, there are more gateways to neighboring clusters, and hence there are more routing options. As the network mobility increases, AODV has a more difficult time maintaining routes due to the greater frequency of link breaks. However, due to the greater number of gateways, the ARC protocols can more frequently patch broken links between clusters. This results in fewer AODV-initiated route repairs, and hence higher overall throughput.

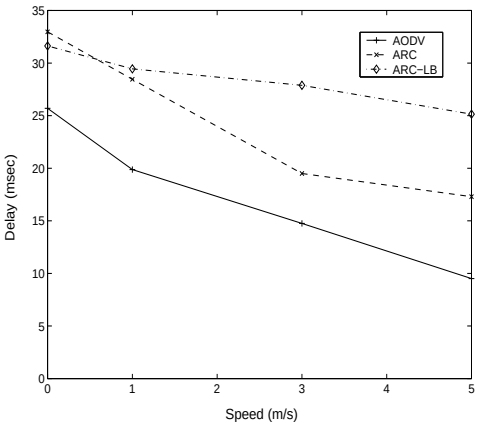
The average delay witnessed by data packets is shown in figure 18. The delay is the total average time for data packets to reach their destination once they are generated by the source node. The overall trend of the graphs shows that the delay increases as the size of the network grows. This is due to the increase in path length as the number of hops needed to connect a source and destination grows with the size of the network. The clustering protocols consistently have greater delay than AODV. The path lengths determined by the clustering protocols are generally a hop or two greater than those found by AODV. This path length disparity is examined in figure 19. Also, because there is more traffic in the clustered networks due to the periodic messaging requirement, nodes experience longer channel acquisition delays than they do in the AODV network. This delay contributes to the overall delay for the data packets to reach their destinations.

The average path length for each network is shown in figure 19. The cluster leader-gateway-cluster leader routing requirement can tend to produce longer path lengths. As the figure shows, however, the average path length in ARC is only a couple of hops longer than that in AODV.

Finally, the mean number of AODV-initiated route repairs in the 100 node network with 450m transmission range is shown in figure 20. An AODV-initiated route repair is a repair that includes a route error transmission to the source node and, consequently, a new route discovery process. The clustering protocols achieve a drastic reduction in the number of route repairs during the lifetime of the network. The hierarchical routes provide greater routing flexibility due to the multiple routing choices between cluster leaders. The link breaks can be repaired through the selection of a new



(a) 100 Nodes



(b) 750 Nodes

Figure 18: Delay.

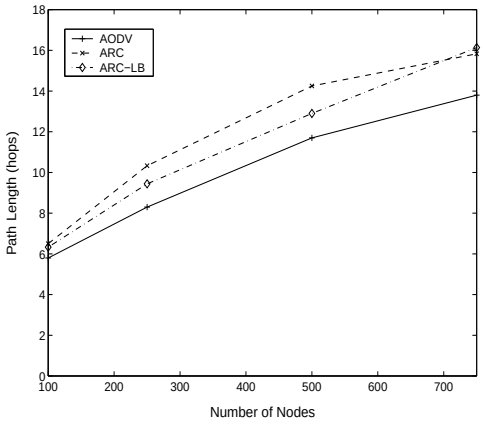


Figure 19: Path Length.

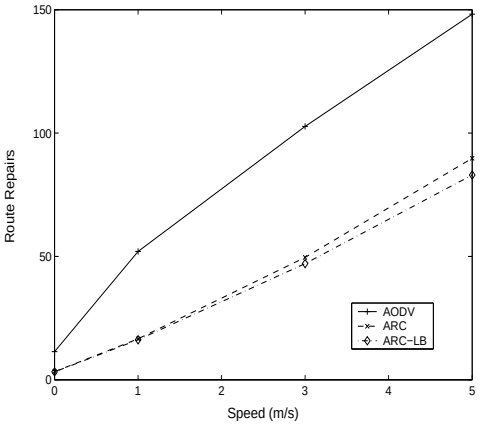


Figure 20: Route Repairs.

gateway, and hence do not require a new route discovery. This results in routes that are more robust to link failures, and hence fewer route reconstructions are needed. The difference increases as the node movement rises, due to the greater frequency of route breaks.

6 Observations

The utilization of the ARC protocol for the establishment of a hierarchical network topology results in a greater number of packets delivered to destinations due to the achievement of longer-lived routes. When routes are recorded hierarchically (i.e., cluster leader to cluster leader), *any* gateway that lies between the two cluster leaders can be used to route between them. When one gateway moves out of range of either cluster leader, it is likely that there are one or more gateways that are able to take its place. This routing flexibility results in increased robustness of routes, thereby enabling more data packets to be delivered to the destinations.

Further benefits of ARC include a reduction in the number of routing protocol control packets

that network nodes must process. Because control packet processing is centralized at the cluster leaders, non-cluster leaders are relieved of the responsibility of processing these packets. This does not result in an increase in routing packet processing by the cluster leaders because they would still be required to process the packets in a network with flat addressing. Additionally, the limited broadcast mechanism provides a further reduction in the flooding overhead generated by suppressing redundant control messages.

ARC is not without its drawbacks, however. Because of the need to continuously maintain the cluster topology, periodic hello messages must be transmitted by each node to signify its presence (or lack thereof). While the sending frequency of the hello messages can be adjusted, the rate is directly related to the freshness of topology information. The higher the sending frequency, the more accurate the cluster information and, consequently, the more accurate the routing information. It would be beneficial to develop a scheme whereby the hello frequency could be varied based on the average mobility of the network nodes. For slower, or even stagnant, networks, hello messages could be sent relatively infrequently. On the other hand, for more rapidly moving networks, the sending rate could be increased. This would result in the optimum connectivity information, but would require some method for nodes to determine their mobility rate.

Another disadvantage of some clustering protocols, including ARC, is the centralization of routes through the cluster leaders. This centralization results in the extra taxation of cluster leaders' power supply. Because routes are centralized, when all possible routes between two cluster leaders are finally exhausted, a large number of broken routes result. It would be beneficial to investigate the migration of established routes from the cluster leader to gateway nodes, where the cluster leader could "oversee" the routing, while not directly participating in it. Such a scheme, however, would require coordination on the part of the cluster leaders, as they would need to have information about which gateways are within communication radius of which other gateways. In her chapter on cluster-based networks, Steenstrup discusses some methods for distributed routing in these networks [20].

7 Conclusions

This paper has presented the Adaptive Routing using Clusters (ARC) algorithm for increasing the scalability of ad hoc networks. The ARC algorithm establishes a hierarchical topology in a network by grouping nodes into clusters and designating one node per cluster to serve as the cluster leader. ARC records routes on a cluster leader to cluster leader basis, as opposed to utilizing flat addressing. This results in an increase in the robustness of routes. Robust routes are longer-lived and are hence able to deliver more data packets to their destinations. Additionally, the ARC algorithm defines a new metric for merging clusters whenever one cluster becomes a subset of another. This scheme prevents the *rippling effect* and results in a cluster topology which is more stable than that created by other clustering protocols. Further, through the *limited broadcast* mechanism, ARC is able to achieve a reduction in the number of redundant routing control messages broadcast throughout the network. When combined with AODV, ARC achieves throughput improvements of over 100%.

8 Acknowledgments

Although this work has a single author, the work was partially completed under the direction of the author's doctoral advisor, P. Michael Melliar-Smith. This work was supported in part by a UC Core grant with Cubic Defense Systems.

References

[1] G. Aggelou and R. Tafazolli. RDMAR: A Bandwidth-efficient Routing Protocol for Mobile Ad Hoc Networks. In *Proceedings of the Second Annual ACM International Workshop on Wireless Mobile Multimedia (WoWMoM)*, pages 26–33, Seattle, WA, August 1999.

[2] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla. GlomoSim: A Scalable Network Simulation Environment. Technical Report CSD Technical Report, #990027, UCLA, 1997.

[3] Dennis J. Baker and Anthony Ephremides. The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm. *IEEE Transactions on Communications*, 29(11):1694–1701, November 1981.

[4] Stefano Basagni. Distributed Clustering for Ad Hoc Networks. In *Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms, and Networks*, pages 310–315, Australia, June 1999.

[5] Christian Bettstetter and Roland Krausser. Scenario-Based Stability Analysis of the Distributed Mobility-Adaptive Clustering (DMAC) Algorithm. In *Proceedings of the* 2_{nd} *Annual Symposium on Mobile Ad hoc Networking and Computing*, Long Beach, California, October 2001.

[6] Ching-Chuan Chiang, Hsiao-Kuang Wu, Winston Liu, and Mario Gerla. Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel. In *Proceedings of IEEE Singapore International Conference on Networks (SICON)*, pages 197–211, April 1997.

[7] IEEE Standards Department. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Standard 802.11–1997*, 1994.

[8] Mingliang Jiang, Jinyang Li, and Yong Chiang Tay. Cluster Based Routing Protocol (CBRP) Functional Specification. *IETF Internet Draft, draft-ietf-manet-cbrp-spec-00.txt*, August 1998. (Work in Progress).

[9] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In Tomasz Imielinski and Hank Korth, editors, *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[10] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan. A Cluster-based Approach for Routing in Dynamic Networks. *ACM SIGCOMM Computer Communication Review*, pages 49–65, April 1997.

[11] Sung-Ju Lee, Elizabeth M. Royer, and Charles E. Perkins. Ad hoc Routing Protocol Scalability. To appear in the International Journal on Network Management, 2002.

[12] Chunhung Richard Lin and Mario Gerla. Adaptive Clustering for Mobile Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, September 1997.

[13] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir R. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. *IETF Internet Draft, draft-ietf-manet-aodv-10.txt*, March 2002. (Work in Progress).

[14] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc On-Demand Distance Vector Routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, LA, February 1999.

[15] Charles E. Perkins and Elizabeth M. Royer. The Ad hoc On-Demand Distance Vector Protocol. In Charles E. Perkins, editor, *Ad hoc Networking*, pages 173–219. Addison-Wesley, 2000.

[16] Theodore S. Rappaport. *Wireless Communications, Principles & Practices*, chapter 3, pages 70–74. Prentice Hall, 1996.

[17] Elizabeth M. Royer, P. Michael Melliar-Smith, and Louise E. Moser. An Analysis of the Optimum Node Density for Ad hoc Mobile Networks. In *Proceedings of the IEEE International Conference on Communications*, Helsinki, Finland, June 2001.

[18] Elizabeth M. Royer and C.-K. Toh. A Review of Current Routing Protocols for Ad-Hoc Mobile Networks. *IEEE Personal Communications*, 6(2):46–55, April 1999.

[19] J. Sharnoy. An Architecture for Mobile Radio Networks with Dynamically Changing Topology Using Virtual Subnets. *ACM Mobile Networks and Applications (MONET)*, 1(1):75–86, 1996.

[20] Martha Steenstrup. Cluster-Based Networks. In Charles E. Perkins, editor, *Ad Hoc Networking*, chapter 4. Addison-Wesley Publishers, 2000.

[21] J. Zavgren. NTDR Mobility Management Protocols and Procedures. In *Proceedings of the IEEE Military Communications Conference (MILCOM)*, November 1997.