Title
Scalable Integrated Routing Using Prefix Labels and Distributed Hash Tables for MANETs

Permalink
https://escholarship.org/uc/item/6980x66f

Author
Garcia-Luna-Aceves, J.J.

Publication Date
2009-10-12

Peer reviewed
Scalable Integrated Routing Using Prefix Labels and Distributed Hash Tables for MANETs

J.J. Garcia-Luna-Aceves*†
†Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
Email: jj@soe.ucsc.edu

Dhananjay Sampath*
*Computer Engineering Department
University of California, Santa Cruz
Santa Cruz, CA 95064
Email: dsampath@soe.ucsc.edu

Abstract—We present AIR (Automatic Incremental Routing), a unified approach for scalable unicast and multicast routing in mobile ad hoc networks (MANET). In AIR, nodes run a distributed routing algorithm to assign prefix labels to themselves. The labels are assigned such that routing to unicast or multicast destinations is automatic, in that a route from any node to a destination is defined by the node’s prefix labels, and incremental, in that no relay node needs to know an entire path to any destination. We verify that AIR provides correct unicast and multicast routing, and present simulation results comparing AIR with AODV, OLSR, MAODV and ODMRP in MANETs. The results from these simulation experiments, as well as from tests carried out in a small testbed running AIR in wireless routers, illustrate that AIR offers substantial performance advantages over traditional unicast and multicast routing protocols, even in the case of small networks.

I. INTRODUCTION

End user applications for mobile ad hoc networks (MANETs) typically involve point-to-point or many-to-many communication among network nodes. These applications call for the support of both unicast and multicast routing in MANETs. Traditional approach to routing in MANETs assumes the use of node identifiers such as MAC or IP addresses to denote destinations in routing tables, and more importantly, the use of separate protocols for unicast (e.g., OLSR, AODV) and multicast (e.g., ODMRP) routing. Maintaining routing tables, built using node identifiers that are independent of the relative locations of nodes in the network, relies on some form of flooding or network-wide dissemination. This is because, in order for routing tables to establish entries to it, either a destination or a representative of a group must announce its presence using identifiers that have little to do with the topology of the network. Flooding occurs even when sources search for their intended destinations without knowing their location. Depending on the protocol, the resulting signaling may consist of the flooding of link states (e.g., OLSR), the dissemination of distances to destinations (e.g., DSDV), the flooding of route requests (e.g., AODV, DSR), or the flooding of group join requests (e.g., ODMRP). This poses a problem as the dissemination of control information on a network-wide basis does not scale with the number of nodes in a MANET.

The main contribution of this paper is the introduction of Automatic Incremental Routing (AIR), which is the first approach to unicast and multicast routing in MANETs in which routing is automatic, in that routes from any source to any destination is implicit in the labels assigned to nodes; and incremental, in that no relay node needs to know an entire path to a destination.

The short review of prior related work presented in Section II helps to highlight the novelty of our approach. We observe that no prior solution to routing in MANETs exists that attempts to reduce or avoid flooding of signaling messages, supports both unicast and multicast routing, and routes robustly in dynamic topologies. AIR is the first approach that eliminates the need for flooding any signaling packet, routes unicast and multicast traffic using an integrated routing structure, and allows the addition or deletion of nodes or links with limited impact to node labels and locally stored routing state.

Sections III and IV summarize the operation of AIR, which supports unicast and multicast routing in MANETs using routing tables based on prefix labels rather than node identifiers. AIR uses neighbor-to-neighbor signaling to label each node with a prefix label that denotes its position in the MANET relative to a root node. The root node is elected with the help of the same signaling messages used to assign labels. Given the prefix labels of a source and a destination, at least one route is defined implicitly at each hop by simply comparing the prefix label of the relaying node with the prefix label of the destination. As a result, the routing table of a node contains only prefix labels of its neighboring nodes, rather than identifiers of intended destinations.

AIR builds and maintains a Distributed Hash Table (DHT) to allow a source node to obtain the prefix label of a destination without flooding. Nodes use a consistent hash function to map their node identifiers, which are globally unique, to the prefix labels of anchor nodes that are in charge of storing the mappings between a node identifier and its corresponding prefix label. Each destination publishes its presence in the network by hashing its own identifier to obtain a prefix label, and then sends a unicast publish message with the mapping towards that prefix label. The node that best matches the target prefix label becomes the anchor node of the destination and stores the mapping. A source subscribes to a destination simply by first hashing the destination identifier with the
consistent hash function to obtain the prefix label of the anchor, and then sends a unicast subscribe message towards the anchor. For the case of multicasting, the publish-messages are group join requests from multicast receivers that form a shared multicast tree by forcing nodes between multicast receivers and the group anchor to join the multicast group. A multicast source simply sends its multicast data packets towards the anchor of the target multicast group. These packets are multicast over the shared multicast tree after they reach the first node that is already a member of the group.

Section V presents the results of an extensive simulation study using the QualNet [23] simulator to illustrate the performance gains obtained with AIR compared to traditional routing in MANETs. The results show that AIR incurs much less overhead, and provides higher delivery rates and shorter end-to-end delays than traditional unicast and multicast routing protocols aimed at establishing shortest paths proactively or on demand. Lastly, Section VI summarizes the results obtained in a small ten-node testbed running AIR and OLSR to illustrate the fact that our new approach is more efficient than traditional routing even in small networks.

II. RELATED WORK

Several schemes have been proposed for routing in MANETs, and due to space limitations we only summarize a small representative sample to highlight the novelty of the approach adopted in AIR.

A. Unicast Routing protocols

Hierarchical routing schemes reduce signaling overhead by organizing nodes into clusters (e.g., [14], [24]). HSLS [22] and FSR [19] further reduce signaling of clustering schemes by limiting propagation of control messages based on their distance from an originating point. The key limitation of clustering schemes is that the affiliation of nodes to clusters is easily broken when nodes move. Re-establishing such affiliations involves flooding. An approach to reducing the amount of information communicated among nodes is to hash node identifiers of destinations into Bloom filters, which are then used in routing updates (e.g., [1], [3]). However, such schemes suffer from the existence of false positives, which forces nodes to use additional mechanisms to verify the existence of a route to a destination.

An alternative approach consists of establishing a distributed hash table (DHT) over a virtual topology defined on top of the physical network. Examples of this approach are Kademia [17], Tapestry [28], and VRR [4]. The advantage of this approach is that the DHT size grows only logarithmically with the number of intended destinations. The key limitation with approaches based on overlays is that a virtual link can correspond to a multi-hop path in the physical network topology. Accordingly, signaling overhead must be incurred to maintain such links, and this becomes excessive in large MANETs.

GPSR [13], XYLS [7], GLS [16]) are examples of using geographical coordinates for routing. These schemes are limited by the requirement of line-of-sight to satellites (for GPS based devices) and the overhead of discovering the geographical coordinates of intended destinations.

A number of schemes substitute the use of geographical coordinates with virtual coordinates consisting of the distances of nodes to a few reference nodes (beacons). Examples of this approach are Beacon Vector Routing (BVR) [10] and LCR [5]. The main limitation of these approaches is that multiple nodes may be assigned the same virtual coordinates, and there is no inherent uniqueness to a specific vector of distances to beacons. This results in either incorrect routing or the use of additional signaling (including flooding) aimed at resolving false positives.

Several compact routing schemes exist in which the routing tables of nodes grow sub-linearly with the number of nodes at the expense of using paths than may be longer than the shortest paths. Interestingly, all the compact routing schemes proposed to date are based on different forms of depth-first searches used to label nodes with identifiers that denote their relative location in a search tree. Tribe [26] is an example of this approach. Tribe partitions a finite address space into control regions corresponding to finite continuous intervals of addresses. The advantages of this approach are that routing tables are very small and the route between any two nodes is therefore implicit from their intervals, just as is the case in AIR. However, the limitation of these schemes is that they are not applicable to MANETs, because the addition or deletion of any node or link requires the relabeling of large portions of the network.

We are aware of only one prior approach based on prefix labels. DART [9] amounts to establishing clusters of nodes based on prefix address trees. The key limitation of this approach is that substantial relabeling of nodes occur when nodes move or links fail, which is the same node-to-cluster affiliation problem present in hierarchical routing.

B. Multicast Routing protocols

Multicast routing protocols for MANETs build either trees or meshes over which data packets are forwarded. These schemes are further classified as sender-initiated or receiver-initiated based on their signaling. In the receiver-initiated approach, introduced in CBT [2], also called shared-tree approach, only one node—called the core of the group—originates the dissemination of information about a multicast group reaching all nodes in the network. Receivers send explicit requests towards the core to join the group. Sources forward multicast data packets towards the core, and the packets are then multicast once they reach any node in the multicast tree or mesh. In contrast, source-based or sender-initiated schemes have each multicast source initiate the dissemination of state information that reaches all nodes in the network. Examples of sender-initiated approaches for MANETs are MAODV [20], ADMR [12], MZR [27], and ODMRP [15]. Examples of receiver-initiated approaches for MANETs are CAMP [11] and PUMA [25]. The limitation with both schemes is that they require the flooding of signaling packets from either the
AIR routes packets from sources to unicast or multicast destinations by the assignment of prefix labels to nodes, and the dynamic mapping of unique node identifiers (e.g., IP or MAC addresses) to prefix labels. The basis for the operation of AIR is the distributed establishment of the Labeled Directed Acyclic Graph (LDAG) rooted at an elected node. The root node is elected such that: (a) Each node is assigned a prefix label denoting the relative location of the node with respect to the root of the LDAG; (b) the prefix labels of a source and a destination define one or multiple valid routes between two nodes; and (c) node mobility, link or node failures and addition of new nodes have limited impact on the prefix labels already assigned to other nodes.

This type of election used for the root node in AIR is similar to the election of a root node in distributed spanning tree algorithms. Neighbor-to-neighbor Hello messages are used to create and maintain the LDAG. A node announces to its neighbors its own prefix label and the prefix labels it assigns to its children in the LDAG, and stores prefix labels it hears from all its neighbors. Each of these Hello messages carry a sequence number to establish its freshness. The soft-state nature of Hello messages is used to update the labels of the nodes as the topology changes. Hello messages are propagated from the root node in a breadth-first manner and percolates through the MANET. The Hello sent by a node also lists the identifiers and prefix labels of its one-hop neighbors, as well as the multicast groups to which the node belongs. Over time, each node knows the identifiers and prefix labels of nodes in its two-hop neighborhood. Figure 1(a) shows an example of an LDAG built with AIR in a MANET.

Clearly, a source must know the prefix label of its intended destination to route to it. To discover routes to destinations, nodes build and maintain a DHT and publish-subscribe to the mappings of node identifier and their prefix labels in this DHT. Each destination uses a consistent hashing function (e.g., as SHA-1) that takes as input the node identifier and returns a prefix label. If the node identifier corresponds to a multicast address, then the hash function returns a group-prefix-label. The node whose label is the closest match to the prefix label returned by the hash function becomes the anchor for that destination. A destination then publishes its presence in the network by sending its own mapping (i.e., its node identifier and its prefix label) to its anchor (see Figure 1(a)). To find a destination and subscribe to it, a source uses the same consistent hashing function with the destination node identifier as input. The source then sends a subscription request to the resulting anchor node. Anchor nodes forward requests and data traffic directly to the destinations by looking up the locally stored mappings.

In the case of multicasting, establishing receiver-initiated multicast trees is very similar to the manner in which unicast routes are established. The group-prefix-label derived from a multicast group identifier serves as the anchor of the multicast group, which serves the traditional role of the core [2], [11] of a multicast group. To join a multicast group, a multicast receiver first determines if any of its neighbors is on the group and if so it sends a join request to it; otherwise, the receiver simply hashes the multicast group identifier to obtain the group prefix label. A join-request is sent towards the node with the closest match to such a label, which serves as the anchor of the group. A join request is answered with a join reply by any node that is already part of the shared tree, and a reverse path is activated as the join reply is forwarded and relaying nodes become a part of the shared multicast tree for the group.
of the shared tree simply forwards multicast data packets to its neighbor to reach the core of the group. The TNT of each entry in the list indicating if node belongs to the list of multicast groups to which the neighbor belongs, with recent sequence number received from the neighbor; and a entry for each immediate neighbor of node neighborhood table node that is already a member of the group, in this case node.

A. Information Stored and Exchanged

Each node maintains a packet cache listing information about multicast data packets heard recently, and a signaling cache that stores and aggregates update information to be sent in the next Hello.

AIR uses a single type of control message called a Hello to carry out its signaling. A Hello sent by node contains the following information: (a) An LDAG identifier consisting of the node identifier of the root of the LDAG and the last sequence number created by the root, (b) the prefix label and node identifier of the node, (c) a sequence number created by the sending node, (d) the list of node identifiers and prefix labels for neighbors one and two hops away, and (e) a group information list (GIL).

The GIL in a Hello consists of a list of one or more entries describing the state of the sending node with respect to a multicast group. Each entry in the GIL specifies: (a) the identifier and group-prefix-label of a multicast group; (b) an action being taken for the group (join-request, join-reply, quit request, or member); (c) the node identifier and prefix label of the next hop selected to reach the core of the group; and (d) the node identifier of one or more nodes to which a join-reply is intended.

B. Structure of Prefix Labels

Let be the alphabet containing a finite number of symbols and be the set of all strings over such that . Every node labels its links to each of its neighbors with a letter from . If represents the letter assigned to the th link of any node then, where is the degree of the node. Hence, a unique letter is assigned to each link connecting any node to its neighbors. The labeling logic labels each node in the LDAG in a breadth-first fashion. Given that the LDAG can be organized as a -ary tree, with being the degree of the LDAG, each child (up to ) is assigned a prefix label as defined below:

Prefix Label: A prefix label for node is a word in such that , where is the prefix label obtained from the parent, \( \otimes \) is a concatenation operator, and \( \Lambda_{\text{parent}} \) is the prefix label obtained from the parent.

It follows from the above definition that the prefix label of a node uniquely identifies the node in a given LDAG.
The prefix labels of nodes define a *predecessor* relation in the LDAG, denoted by $\rightarrow$, such that for any two nodes $s$ and $d$ in the LDAG:

1) $s \rightarrow d : s$ precedes $d$. In this case, $d$ can be reached by traversing a prefix path of descendents of $s$ in the portion of the LDAG rooted at $s$.

2) $d \rightarrow s : d$ precedes $s$. In this case, $d$ can be reached upstream from $s$ by traversing a prefix path of ancestors of $s$.

3) $d \approx s : d$ and $s$ are not directly related. In this case, $d$ can be reached by traversing the LDAG from $s$ up to the first common ancestor of $d$ and $s$, say $A$, such that $A \rightarrow s$ and $A \rightarrow d$ hold, and then down from $A$ to $d$.

The topology of a MANET changes constantly and nodes join or leave the network arbitrarily, which results in inconsistent labels while nodes are updating them. AIR enforces a *strict labeling* by ordering prefix labels using sequence numbers as well. Let $L$ represent a tuple $(S_n^a, \Lambda_n^a)$, where $S_n^a$ denotes the sequence number that originates at $a$ and is forwarded by node $n$, and $\Lambda_n^a$ denotes the prefix label of the current node with respect to $a$. $S$ is a monotonically increasing integer, while $\Lambda$ is a word in $\Sigma^*$. We define an operator $\prec$ over the set of ordered-pair of identifiers $L$. If $L_n^a, L_y^a$ are two such tuples then,

$$\{L_n^a \prec L_y^a \mid L_{x,y} \in \Sigma\}$$

$$if \{ (S_n^a < S_y^a) \lor [(S_n^a = S_y^a) \land (A_n^a \prec A_y^a)] \}$$

It can be shown that the operator $\prec$ is anti-reflexive and transitive over the set $\Sigma^*$, and that these prefix labels can be used to establish an ordering among the nodes.

**C. Electing a Root and Assigning Prefix Labels**

A root node is elected in a distributed fashion using Hello messages. Ties between two nodes are broken by their node identifiers, for example. If a node is not labeled, it looks up its neighbor table to determine if any node is labeled. If no node is labeled and its local timer for labeling expires, the node elects itself as the root node. The self-elected root node then sends Hello messages with its own label and assigns labels to the neighboring nodes.

The node with the lowest node identifier enforces the ordering among the nodes in the LDAG as described in Algorithm 1. In this paper, for the sake of simplicity, we only consider prefix labels assigned by a single root node.
D. Routing on AIR

To route to a destination \( d \), node \( s \) chooses the link to any of its two-hop neighbors that offers the maximum length of prefix label that matches with the prefix label of the destination. This is simply maximum matching prefix logic, shown in Algorithm 2, which selects the next hop using a greedy strategy that considers the two-hop neighborhood of a node, and can find shorter paths than the traditional prefix-tree routing by leveraging the richer path diversity of an LDAG compared to a prefix tree. For instance, if there exists a label in the two-hop neighborhood that is lexicographically closer to the destination, the next hop is chosen such that the packet is forwarded to that node instead of routing via the prefix-tree parent. We ensure that this greedy strategy does not encounter a local minima by randomly selecting a next hop when all nodes offer the same matching prefix.

From the predecessor relation induced by the prefix labels in the LDAG of a MANET, a given node selects a next hop to a prefix label according to the possible cases allowed by the predecessor relation.

E. Building the DHT

As we stated before, to avoid bottlenecks and single-points of failure, the mappings of node identifiers to prefix labels is distributed within the network. Node publish and subscribe to both unicast and multicast destinations using the same LDAG. Algorithm 3 briefly describes these mechanisms.

1) Publishing Destinations: Each node publishes the mapping between its node identifier and its prefix label to an anchor node according to Algorithm 3. The node with a prefix label that is the closest match in its two-hop neighborhood to the prefix label stated in a publish request becomes the designated anchor for the mapping and stores it.

The frequency of the update messages is controlled by the topology changes in the neighborhood of the node. Each node also maintains a list of prefix labels assigned to it and its corresponding parent. If the node detects a change in its prefix label, it then sends an update to its anchor.

2) Subscribing to Destinations: Active unicast sources subscribe to unicast destinations by first hashing the node identifier to get the label of the anchor. The unicast subscription request (typically the first data packet) travels all the way to the corresponding anchor node, which can then forward to the unicast destination.

Multicast receivers subscribe to the multicast groups in much the same way as above. However, multicast subscriptions by receivers are resolved by the first member of the multicast group reached by the join request. The anchor of a multicast group is that node whose prefix label is the closest match to the group prefix label stated in a subscription request for the group.

Sources route towards the anchor of a unicast or multicast destination, choosing the next hop that best matches the anchor’s prefix label.

Algorithm 3: Publish Algorithm

<table>
<thead>
<tr>
<th>Data: nodeId, nodeLabel, prevLabel, timeout, pkt</th>
</tr>
</thead>
<tbody>
<tr>
<td>anchorLabel = Fn(getHash(nodeId))</td>
</tr>
<tr>
<td>nextHop = Fn(getNextHop(nbrTable, anchorLabel))</td>
</tr>
<tr>
<td>if nodeLabel ≠ prevLabel or timeout == True then</td>
</tr>
<tr>
<td>/* set packet information */</td>
</tr>
<tr>
<td>pkt.type = ‘update’;</td>
</tr>
<tr>
<td>pkt.mapping = (nodeId, nodeLabel);</td>
</tr>
<tr>
<td>pkt.anchorLabel = anchorLabel;</td>
</tr>
<tr>
<td>pkt.prevHopLabel = nodeLabel;</td>
</tr>
<tr>
<td>pkt.srcLabel = nodeLabel;</td>
</tr>
<tr>
<td>/* send the message to the nextHop */</td>
</tr>
<tr>
<td>Fn(sendPacket(pkt, nextHop, anchorLabel));</td>
</tr>
<tr>
<td>Fn(resetTimer(timeout))</td>
</tr>
<tr>
<td>prevLabel = nodeLabel;</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>/* wait for next time-out or event */</td>
</tr>
<tr>
<td>Fn(Wait())</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

Algorithm 4: Handling Publish packets

<table>
<thead>
<tr>
<th>Data: pkt, nodeId, nbrTable, anchorLabel</th>
</tr>
</thead>
<tbody>
<tr>
<td>if pkt.type == ‘update’ then</td>
</tr>
<tr>
<td>/* Node that matches the anchorLabel */</td>
</tr>
<tr>
<td>closest is assigned as anchor */</td>
</tr>
<tr>
<td>nextHop = Fn(getNextHop(nbrTable, anchorLabel))</td>
</tr>
<tr>
<td>if nextHop == ‘no valid next hop’ then</td>
</tr>
<tr>
<td>/* no node matches the anchor */</td>
</tr>
<tr>
<td>label closer than current node */</td>
</tr>
<tr>
<td>Fn(storeMapping(self, pkt.mapping))</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>/* found better mapping */</td>
</tr>
<tr>
<td>Fn(sendPacket(pkt, nextHop, anchorLabel))</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

F. Adaptive Timers

Hellos are transmitted periodically by a node to all its immediate neighbors. The periodicity of Hellos is determined by the nature of the updates that need to be conveyed. Two separate timeout values, a long-timeout (LT) and a short-timeout (ST) are maintained by each node. Updates to the neighborhood information or GIL are aggregated until either timeout expires. Any event that changes the next hop towards the core of a group for which the node is active fires the short-timeout. When the ST expires, the node sends its Hello with the aggregated updates stored in its signaling cache, and the node takes the steps needed to maintain multicast trees. The long-timeout is fired when the topology is relatively static and fewer messages are needed to maintain the multicast trees of
Algorithm 5: Subscribe Algorithm

Data: destAddress, nbrTable, nodeLabel

anchorLabel = Fn (getHash(destAddress));
nextHop = Fn (getNextHop(nbrTable, anchorLabel));

if nextHop ≠ 'no valid next hop' then
  /* set packet information */
  pkt.type = 'subscription';
  pkt.sessionType = 'unicast' or 'multicast';
  pkt.destination = destAddress;
  pkt.anchorLabel = anchorLabel;
  pkt.prevHopLabel = nodeLabel;
  pkt.srcLabel = nodeLabel;

  /* send packet to nexthop */
  Fn (sendPacket(pkt, nextHop, anchorLabel));
  Fn (setTimer(timeOut));
else
 /* sending packet to self since no
nextHop available */
  Fn (sendPacket(pkt, self, nodeLabel));
end

Algorithm 6: Handling Subscribe packets

Data: pkt, nodeLabel, nbrTable, anchorLabel

if pkt.type == 'subscription' then
  if pkt.sessionType == 'unicast' then
    nextHop = Fn (getNextHop(nbrTable, anchorLabel));
    if nextHop == 'no valid next hop' then
      /* This node matches the anchor
      label closest. Look up the
      mapping for the destination */
      destLabel = Fn (lookupMapping(pkt.destAddress));
      nextHop = Fn (getNextHop(nbrTable, destLabel));
      Fn (sendPacket(pkt.data, nextHop, destLabel));
    else
      Fn (sendPacket(pkt, nextHop, anchorLabel));
    end
  else
    /* Packet is a multicast packet */
    if node.multicastGroup == pkt.multicastGroup then
      /* Node is a part of multicast
      group send data in pkt to all
      members of the group */
      for each nbr in nbrTable do
        if nbr in multicast group then
          Fn (sendPacket(pkt.data, nbr, anchorLabel));
        else
          continue;
        end
      end
      /* add the nbr that sent
      subscription to the multicast
      group */
      Fn (addNbrToGroup(node.multicastGroup.members, pkt.prevHop));
    else
      /* Packet is forwarded towards the
      core of the group, current node
      subscribes to the group as well */
      Fn (subscribe(anchorLabel, nbrTable, nodeLabel));
    end
  end
end

G. Node Dynamics

Nodes acquire new labels when they reposition themselves in the network. While changes to nodes joining as a fringe node is trivial, changes in the prefix label of internal nodes are more involved. As Algorithm 7 illustrates, a node tries to find paths to its peers in the event a parent node moves/fails. If it finds a valid path to all its peers, then it emulates the parent node until the expiration of a relabel – timer. This prevents dropping of existing packet flows. Moreover, each node maintains its old label for one complete time-period after relabeling to avoid loops or packet drops.

Changes to labels of anchors or cores does not affect the topology much as the node that is next closest to the prefix label or the group-prefix-label automatically assigns itself as the anchor or core.

various groups. Figure 3 shows an example of the adaptive update timers. These timers ensure that the frequency with which a node transmits its Hellos is a function of the topology changes around the node.

![Fig. 3. Message aggregation using adaptive timers](image)

H. Group Dynamics

Group membership is maintained using simple soft-state logic through the exchange of Hellos periodically. Although each Hello specifies control information in the GIL for one or multiple multicast groups, the steps taken for a given multicast group are independent of other groups.

A receiver interested in joining a particular multicast group hashes the name of the group to obtain the corresponding group-prefix-label. It then sends a join-request for the group in the GIL of its Hello stating the identifier and prefix label of the neighbor node it chooses along one of its implicit routes to the group-prefix-label.

A core node of the multicast group $G$ receives a join-request for that group in the Hello from a neighbor $x$. It then, sends a join-reply entry for group $G$ in the GIL of its own Hello and states the identifier of node $x$ as the recipient of the reply. The same action is taken by a node that is a member of multicast
Algorithm 7: Handling node dynamics

Data: nbrTable, oldNbrTbl, topoTestTime, topoTimeOut, repair = False;
while topoTestTime < topoTimeOut do
    if nbrTable ≠ oldNbrTbl then
        /* one hop neighborhood has changed */
        if (getNextHop(nbrTable, '0') == 'no valid next hop') then
            /* next hop to root does not exist */
            for each nbr in oldNbrTbl do
                if Fn(checkPathExists(nbr)) then
                    repair = True;
                else
                    repair = False;
                    /* subtree is disconnected from rest of the DAG, initiating relabeling */
                    Fn(initLabeling( ));
                end
            end
        end
    else
        /* Valid next hop to root node exists. Do nothing */
        Fn(incrementTopoTestTime());
        continue;
    end
    else
        /* no topology change. increment timer */
        Fn(incrementTopoTestTime());
        continue;
    end
end

Group \( G \) when it receives a join-request for that group in the \textit{Hello} from a neighbor stating that the node is the next hop to the core of the group. The core of a group \( G \) becomes a member of the group and starts sending a `member` entry for group \( G \) in the GIL of its \textit{Hellos} after receiving the first join-request for the group.

If node \( x \) is not a member of a multicast group \( G \) and receives a join-request from a neighbor stating the node as the next hop to the core of group \( G \), then node \( x \) sends a join-request of its own in the GIL of its next \textit{Hello}. Its join-request states its choice of next hop to the core of \( G \) along one of its implicit routes to the core.

A `member` entry for group \( G \) in the GIL of the \textit{Hello} from a node serves as an update of the multicast state for all its neighbors. It also helps nodes decide when to forward multicast data packets. A multicast receiver that does not serve as a relay and chooses to leave a group simply stops including an entry for group \( G \) in its \textit{Hellos}.

V. Performance Comparison

We modeled AIR using a discrete event simulator called QualNet [23] (version 4.0) and compared AIR separately with unicast and multicast protocols. We chose AODV and OLSR for unicast and MAODV and ODMRP as multicast protocol benchmarks. We also ran tests to compare AIR with nodes running both unicast and multicast protocols by combining AODV and OLSR with ODMRP. In each of our experiments, we use three metrics to compare the performance of the protocol: The total number of control packets transmitted per node; the delay measured per flow on an end-to-end basis; and the packet delivery ratio, defined as the number of data packets successfully delivered.

Simulations were instrumented in networks of nodes placed in a terrain of dimensions 1500m X 1500m and nodes were placed randomly within unit blocks (each of which was 5% of the dimension of the region). Radios in the nodes were 802.11 with CSMA channel access and transmission power of 10dbm. The simulation time was set to 450 seconds and data sources were generators that produced a constant bit rate (CBR) at a rate of 10 packet per second. Each multicast source was allowed to transmit up to 1000 packets.

The random waypoint mobility model was chosen to simulate the movement of nodes. While it is common observation that this mobility model is more aggressive than real scenarios, it helps us understand the performance of each protocol under the worst possible case. Each simulation was run 10 times with different seeds to avoid any artifact of pseudo random number generators.

Key parameters of all routing protocols were selected to ensure sufficient likeness for comparison. Periodic refresh timeouts for ODMRP and MAODV were set to 3 sec. Maximum group timeouts were set to 3 times the refresh timeouts. AODV’s horizon threshold was set to the same value as ODMRP’s TTL-Max. Protocols running in combination had separate trace files and statistics gathered were combined.

A. Unicast Traffic with increasing Pause Times

Our first scenario demonstrates the robustness of AIR in the face of mobility by comparing the different protocols under pause times increasing from 0 to 300 seconds in a MANET of 400 nodes. Figures 4(a), 4(b), and 4(c) show the results for this scenario. We observe that when nodes move constantly, all protocols have low delivery ratios. OLSR suffers because of too many LSUs being propagated to the entire network after each topology change. AODV manages to catch up faster as the rate of mobility goes down; it leverages the unexpired entries in its routing table cache to respond to nodes that are not moving as fast.

AIR utilizes anchors very effectively. It achieves almost 15% higher delivery ratio than either OLSR or AODV and yet has 20% less overhead than the two. The higher delivery ratio is explained by fewer control messages transmitted, resulting in lesser congestion across the network. The end-to-end latency of AIR is also low under high mobility because fewer packets are retransmitted. While it is hard to outperform
flooding in terms of latencies, flooding under overloaded conditions results in severe performance degradation and this is where AIR shows better end-to-end latencies. The performance of AODV and OLSR come closer to that of AIR only in relatively static scenarios.

B. Multicast Traffic with increasing Pause Times

In this scenario, we simulated only multicast traffic to observe how AIR compares with multicast routing protocols. Each multicast group was set to contain 20 members and sources of these groups sent 10 packets per second. The multicast groups were assigned unique global addresses. Figures 5(a) to 5(c) show the results for this scenario. We can see that AIR delivers 20% more than MAODV and almost 10% more than ODMRP while at the same time incurring almost 40% less overhead. Note that, for short pause-times of up to 50 seconds, AIR manages to deliver packets two to three seconds faster as well. Under longer pause times, it is hard to outperform network-wide floods in terms of latency. AIR outperforms both protocols in the combined performance of delivery, overhead and latency.

AIR performs better than MAODV and ODMRP, because of the reduced amount of control overhead generated by AIR, and in particular the absence of periodic network-wide flooding of control packets. In MAODV, whenever a receiver disrupts the shared-tree by leaving and joining at a different location, it floods the network with a RREQ. On the other hand, in ODMRP, a multicast source floods the entire network
periodically with a join-query message. In contrast, disruptions due to mobility trigger fewer updates in AIR, and control messages are propagated throughout the network. This reduces any congestion that may occur owing to network wide floods and improves delivery significantly.

C. Combined Unicast, Multicast Traffic with Increasing Pause Times

We ran simulations of ODMRP with two different flavors of unicast routing protocols. AODV and OLSR characterize two different class of unicast routing protocols, one is a reactive routing protocol and the other a proactive one. As we can see from Figures 6(a), 6(b) and 6(c), AIR does better in terms of all three metrics. ODMRP with reactive routing does better for smaller pause times. The congestion caused by repeated broadcasts of control messages causes the delivery ratio to drop almost by 35%. AIR manages to deliver packets at the same rate as before and sends fewer messages without much bias for either types of traffic.

D. Combined Traffic with Varying Group Size and Shape

In this scenario, the topology of the network is static and the number of multicast groups is increased from 1 to 6. Each of these groups was divided such that the total members in the groups increased by a factor, corresponding to the number of groups. However, the size of each group was determined uniformly at random.

Therefore, in the first experiment with 1-group, the group size was set to 10 members. In the second experiment with 2-groups, the group size was set to 20 members, and the constitution of each of the group was determined randomly such that it totaled 20 members. In this case, group-1 was setup with 8 nodes and group-2 with 12. Similarly the number of groups were varied from 1 to 6. The average distances between each group member was set to vary from 3 hops to 7 hops. This helped us to observe the effect of having wide-spread group members vs. spatially co-located ones.

We observe from Figures 7(a) to 7(c) that AODV+ODMRP performs better for smaller groups that are spatially co-located. However, as the number of groups increases, its performance degrades independently of how the groups are located in the network. AIR outperforms both AODV+ODMRP and OLSR+ODMRP as the number of groups increases, and the performance of AIR is very similar for any number of multicast groups. These results should be expected, given that the cores of groups do not generate any signaling traffic in AIR.

VI. TESTBED EVALUATION

We designed a prototype of AIR to demonstrate some of its characteristics using a testbed. We wrote AIR in Python, because it enabled rapid prototyping. The testbed consisted of 10 nodes and each node in the network was equipped with a 802.11b/g radio on a Mini-ITX board running Debian. Each node establishes a UDP connection with the neighboring nodes to exchange messages. We built our own version of flow control without explicit acks to leverage the use of the hello messages. The testbed nodes were deployed within a building as shown in Figure 8 and were placed such that any source-destination pair was connected over multiple hops. We evaluated the behavior of AIR under different scenarios. As a benchmark, we used the OLSR protocol from [6] for our comparisons.

Due to space limitations, we briefly discuss a single scenario to show a proof of concept and also demonstrate some crucial properties of AIR. To characterize the effects of mobility, we studied the scenario where a node is moved to a different part of the network after 10 minutes of static placement, and determined the overhead incurred by AIR and the OLSR protocol. Table I shows the performance results for the two protocols. Note that while there is a slight improvement in the delivery ratio in AIR, the number of control messages transmitted when a single node moved is almost four times less in AIR than in OLSR.

<table>
<thead>
<tr>
<th></th>
<th>Delivery Ratio</th>
<th>Overhead single link</th>
<th>Overhead Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLSR</td>
<td>95.224%</td>
<td>63 pkts</td>
<td>5400 pkts</td>
</tr>
<tr>
<td>AIR</td>
<td>97.117%</td>
<td>16 pkts</td>
<td>3100 pkts</td>
</tr>
</tbody>
</table>

TABLE I
IMPLEMENTATION RESULTS
necessarily reflect the views of the funding agencies.

The Baskin Chair of Computer Engineering. Any opinions, search Office (ARO) under grant W911NF-05-1-0246 and by routing schemes even in very small wireless networks. can offer significant performance advantages over traditional using a Python implementation of AIR demonstrate that AIR to compare its performance with the performance of tradi-
tional unicast and multicast routing protocols, OLSR, AODV , and

destination are implicit in the prefix label assigned to nodes; and
incremental, in that no relay node needs to know an entire path to any destination. We described salient aspects of our implementation of AIR, and used simulation experiments to compare its performance with the performance of traditional unicast and multicast routing protocols, OLSR, AODV, ODMRP and MAODV in particular. Testbed experiments using a Python implementation of AIR demonstrate that AIR can offer significant performance advantages over traditional routing schemes even in very small wireless networks.

VII. CONCLUSION

We presented the first unified approach to unicast and multicast routing in MANETs that eliminates completely the need for flooding of signaling packets in the network, or the network-wide dissemination of signaling packets from each source or each destination. Our approach is called Automatic Incremental Routing (AIR) because routing to any unicast or multicast destination is automatic, in that the routes to the destination are implicit in the prefix label assigned to nodes; and incremental, in that no relay node needs to know an entire path to any destination. We described salient aspects of our implementation of AIR, and used simulation experiments to compare its performance with the performance of traditional unicast and multicast routing protocols, OLSR, AODV, ODMRP and MAODV in particular. Testbed experiments using a Python implementation of AIR demonstrate that AIR can offer significant performance advantages over traditional routing schemes even in very small wireless networks.

VIII. ACKNOWLEDGMENTS

This work was supported in part by the U.S. Army Research Office (ARO) under grant W911NF-05-1-0246 and by the Baskin Chair of Computer Engineering. Any opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES