Pervasive and Mobile Computing 6 (2010) 144–157

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Fast track article

Hydra: Efficient multicast routing in MANETs using sender-initiated multicast meshes

Rolando Menchaca-Mendez [a,*], J.J. Garcia-Luna-Aceves [a,b]

[a] Computer Engineering Department, University of California, Santa Cruz, Santa Cruz, CA 95064, USA

[b] Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA

ARTICLE INFO

Article history:
Received 26 January 2009
Received in revised form 19 June 2009
Accepted 13 July 2009
Available online 17 July 2009

Keywords:
Ad hoc networks
Multicast routing
Mesh-based routing
Scalability
Multicast state aggregation

ABSTRACT

We present Hydra, the first multicast routing protocol for MANETs that establishes a multicast routing structure approximating the set of source-rooted shortest-path trees from multicast sources to receivers, without requiring the dissemination of control packets from each source of a multicast group. Hydra accomplishes this by dynamically electing a core for the mesh of a multicast group among the sources of the group, and aggregating multicast routing state in the nodes participating in multicast meshes, so that only control packets from the core are disseminated towards the receivers of a group. We prove that Hydra establishes correct routes from senders to receivers of a multicast group when multicast state information is aggregated. We also present simulation results illustrating that Hydra attains comparable or higher delivery ratios than the On-Demand Multicast Routing Protocol (ODMRP), but with considerably lower end-to-end delays and far less communication overhead. Results are shown for scenarios using 802.11 DCF and TDMA as the MAC layer protocols and using random waypoint and group mobility as mobility models.

1. Introduction

The objective of a multicast protocol for mobile ad hoc networks (MANET) is to enable communication between a sender and a group of receivers in a network where nodes are mobile and may not be within direct wireless transmission range of each other. These multicast routing protocols can be classified by the type of routing structure they construct and maintain; namely tree-based and mesh-based protocols.

A tree-based multicast routing protocol constructs and maintains either a shared multicast routing tree or multiple multicast trees (one for each sender) to deliver packets from sources to receivers. Several tree-based multicast routing protocols have been reported, and this approach has proven to deliver adequate performance in wired networks. However, in the context of MANETs, establishing and maintaining a tree or a set of trees in the presence of frequent topology changes incurs substantial exchange of control messages, which has a negative impact in the overall performance of the protocol [1].

On the other hand, a mesh-based multicast routing protocol maintains a mesh consisting of a connected sub-graph of the network that contains all receivers of a particular group and the relays needed to maintain connectivity. Maintaining a connected component is far simpler than maintaining a tree and hence mesh-based protocols tend to be simpler and more robust. Three representatives of this kind of protocols are the Core Assisted Mesh Protocol (CAMP) [1], the On-Demand Multicast Routing Protocol (ODMRP) [2], and the Protocol for Unified Multicast through Announcements (PUMA) [3].

[*] Corresponding author. Tel.: +1 831 427 2006.
E-mail addresses: menchaca@soe.ucsc.edu (R. Menchaca-Mendez), jj@soe.ucsc.edu (J.J. Garcia-Luna-Aceves).

1574-1192/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.pmcj.2009.07.010

A potential concern in mesh-based schemes is that, under high channel contention, these protocols may have poor performance if too many redundant relays are involved in the forwarding of multicast traffic.

Whether multicast routing protocols for MANETs build multicast trees or meshes, all of them are based on network-wide dissemination of control packets to inform the rest of the nodes about the existence of multicast groups. In receiver-initiated schemes, only one node, which in many schemes is called the core of the group, originates the dissemination of information about a multicast group reaching all other nodes, and receivers send explicit requests towards the core to join the group. This approach was originally introduced in the core-based tree (CBT) protocol [4]. In contrast, source-based or sender-initiated schemes have each multicast source originate the dissemination of state information that reaches all nodes in the network. This approach was originally introduced by Deering [5]. Given that the sender-initiated protocols proposed to date use per-source flooding, they do not scale well as the number of groups and sources increases. However, they can provide shortest paths from sources to destinations and avoid hot spots. On the other hand, core-based protocols incur far less overhead, but they do not establish shortest paths from sources to destinations, which leads to higher delays than the ideal shortest paths from sources to receivers. This approach also increases the potential of creating contention hot spots because they tend to concentrate the traffic around cores.

The work presented in this paper is motivated by the desirability of providing the best features from the two alternatives in the existing design space summarized above, and which we discuss in more detail in Section 2. The main contribution of this paper is to introduce and verify the *first* sender-initiated multicast routing protocol that incurs the same order overhead as receiver-initiated approaches, and yet establishes multicast meshes that approximate routing structures containing the shortest paths from multicast sources to their destinations.

Section 3 presents Hydra, a multicast routing protocol that creates a multicast mesh formed by a mixture of source-specific and shared sub-trees (or sub-meshes) using as few control packets as receiver-initiated schemes. The key ideas behind Hydra are: Restricting the dissemination of control packets to those regions of the network where other dynamically designated sender has previously discovered receivers, aggregating control messages from non-core senders, and electing a sender as the core in non-destructive manner. Section 4 shows that the algorithm used in Hydra to perform multicast state aggregation is correct (i.e., it establishes loop-free routes from sources to receivers). Section 5 describes the results of simulation experiments used to compare Hydra's performance with that of ODMRP by considering different numbers of sources, group sizes, node density, the use of 802.11 and TDMA as the underlying MAC protocol, as well as random waypoint and group mobility [6] as the mobility models. The results illustrate the performance benefits that should be expected from the approach implemented in Hydra, and that Hydra provides substantial performance improvements over ODMRP even in scenarios involving relatively small networks with few multicast sources. Hydra attains the same or better delivery ratios than ODMRP, and does so while transmitting from one third to one half the number of data packets sent in ODMRP and incurring end-to-end delays that can be close to an order of magnitude smaller than in ODMRP. The simulation experiments also compare different versions of Hydra based on the construction of trees or meshes. The results illustrate the tradeoffs of using different routing structures, and indicate the need for an adaptive strategy that builds trees or meshes based on the state of the network.

2. Related work

Many multicast routing proposals exist for MANETs, and due to space limitations we focus on just a few to highlight Hydra's novelty. The multicast ad hoc on-demand distance vector protocol (MAODV) [7] maintains a shared tree for each multicast group consisting of receivers and relays. Each multicast group has a group leader who is the first node joining the group. The group leader is responsible for maintaining the group's sequence number, which is used to ensure freshness of routing information. The group leader periodically transmits a group hello packet to become aware of reconnections. Receivers join the shared tree by means of a special route request (RREQ) packet. Any node belonging to the multicast tree can answer to the RREQ with a route reply (RREP). A sender joins the group through the node reporting the freshest route in a RREP with the minimum hop count to the tree. Data are delivered along the tree edges maintained by MAODV. If a node that does not belong to the multicast group wishes to multicast a packet, it has to send a non-join RREQ, which is treated like a RREQ to reach the group. As a result, the sender finds a route to a multicast group member. Once data are delivered to a group member, the remaining members receive the data along the multicast tree.

The adaptive demand-driven multicast routing protocol (ADMR) [8] maintains a source-based multicast tree for each sender of a multicast group. A new receiver performs a network-wide flood of a multicast solicitation packet when it needs to join the multicast group. Each source replies to the solicitation and the receiver sends a receiver join packet to each source that answered the solicitation. Each source-based tree is maintained by periodic keep-alive packets from the source, which allow intermediate nodes to detect link breaks in the tree by the absence of data or keep-alive packets. A new sender also sends a network-wide flood to allow existing group receivers to send receiver joins to the source. MZR [9] maintains source-based trees, like ADMR, but performs zonal routing, and hence its dissemination of control packets is less expensive.

In ODMRP [2], group membership and multicast routes are established and updated by the sources. Each multicast source broadcasts Join Query (JQ) packets periodically, and these are disseminated to the entire network to establish and refresh group membership information. When a JQ packet reaches a multicast receiver, it creates and broadcasts a Join Reply (JR) to its neighbors stating a list of one or more forwarding nodes. Nodes receiving JR listing them as part of forwarding groups forward the replies with its own list of forwarding nodes. A JR is propagated by each forwarding group member until it reaches a multicast source via the selected paths. This process establishes and updates the routes from sources to receivers and builds a mesh of nodes, the forwarding group. A source can multicast data packets to multicast receivers via selected routes and forwarding groups. Many ODMRP extensions have been proposed. One such extension is DCMP [10], which designates certain senders as cores and reduces the number of senders performing flooding. NSMP [11] is another extension to ODMRP aiming to restrict the flooding of control packets to a subset of the entire network. However, DCMP and NSMP fail to eliminate entirely ODMRP's use of multiple nodes flooding control packets for each group. MMARP [12] is another extension to ODMRP that builds its multicast mesh as the union of a set of trees that approximate Steiner trees rooted at each source. The authors report an improvement of 40%–50% in forwarding efficiency with respect to ODMRP; however, the control overhead is of the same order as ODMRP.

CAMP [1] avoids the need for network-wide disseminations from each source to maintain multicast meshes by using one or more cores per multicast group. A receiver-initiated approach is used for receivers to join a multicast group by sending unicast join requests towards a core of the desired group. The drawbacks of CAMP are that it needs the pre-assignment of cores to groups and a unicast routing protocol to maintain routing information about the cores. PUMA [3] uses a receiver-initiated approach similar to that of CAMP in which receivers join a multicast group using the address of a special node (core), without the need for network-wide dissemination of control or data packets from all the sources of a group. PUMA implements a distributed algorithm to elect one of the receivers of a group as the core of the group, and to inform each router in the network of at least one next hop to the elected core of each group. Within a finite time, each router has one or multiple paths to the elected core. All nodes on shortest paths between any receiver and the core collectively form the mesh of the multicast group. A sender node can send packets to the multicast group by encapsulating them in unicast packets to the core along any of the paths to the core.

Regarding multicast state aggregation, the few existing proposals such as the one presented by Liu et al. [13] are targeted to the Internet and require nodes to know the entire multicast tree of a group which is not feasible in a MANET.

3. Hydra

Multicast sources in Hydra periodically broadcast Join Query (*JQ*) messages to establish a partial ordering of the nodes in the network. In the case of ODMRP and Hydra, the ordering is based on the nodes' distances in hops to the sources. This ordering is further used to route Join Reply (*JR*) messages from receivers to sources, forcing intermediate nodes to join either a mesh or a tree. However, Hydra uses three mechanisms to build a routing structure as close as possible to a set of source-rooted breadth-first trees (or meshes composed of the union of breadth-first trees) spanning all the receivers while incurring as few control packets as possible.

In contrast to ODMRP, Hydra uses an elected source as the core of the group, and this is the only source whose *JQ*s reach the entire network. Non-core sources take advantage of the routing state established by the core to identify connected sub-graphs containing themselves and the receivers of the group. This way, the scope of the dissemination of *JQ*s from non-core sources is restricted to these connected regions, and other parts of the network are not flooded with unnecessary control information.

Hydra also identifies regions of the network where two or more sources share common sub-graphs (meshes or trees) and performs routing-state aggregation, so that nodes located inside those common regions only keep routing state regarding one of the aggregated sources and receive *JQ*s and *JR*s only from and for that source. To detect the boundaries of a common sub-graph, Hydra compares the orderings established by previous sources with the ordering that is being established by the current *JQ* from a non-core source. If the ordering induced by the *JQ* is equivalent to the ordering established by a prior *JQ* from another source, then the current *JQ* is not forwarded any further and the two sources that have equivalent orderings are considered as aggregated. Two partial orderings over a graph are *equivalent* if the gradient vectors among neighbors obtained from the two orderings are the same. It is easy to see that if two sources order a region of a network in an equivalent way and if consistent criteria are used to break ties when selecting a parent when routing *JR* packets form receivers to senders, then both sub-trees (or sub-meshes) are identical in that region. Hence, both routing structures can share a single sub-graph without deviating from optimality too much in terms of the length of the paths from sources to the receivers located in the region under consideration. As the number of senders increases, the likelihood of finding equivalent regions also increases, because nothing prevents a source to share different sub-graphs with different sources, or a given sub-graph to be shared by more than two sources. This property helps the scalability of Hydra with respect to the number of sources. We also note that, while Hydra takes advantage of having a core, it is not necessary for its operation.

Hydra opportunistically groups control messages of different sources and groups into *control bundles*. However, in the rest of our description, we focus on the signaling intended for a single multicast group.

3.1. Join Query messages

The first active multicast source for a given group considers itself to be the core for that group and states so in the Join Query (*JQ*) message it broadcasts every *join query period*. As *JQ*s disseminate in the network, they inform nodes of the existence of the multicast group and its current core, and also create a partial ordering of the network based on the distance in hops from each node to the current core. If two or more sources become active concurrently in the same partition, a distributed election is held. Section 3.4 presents the details of the election algorithm.

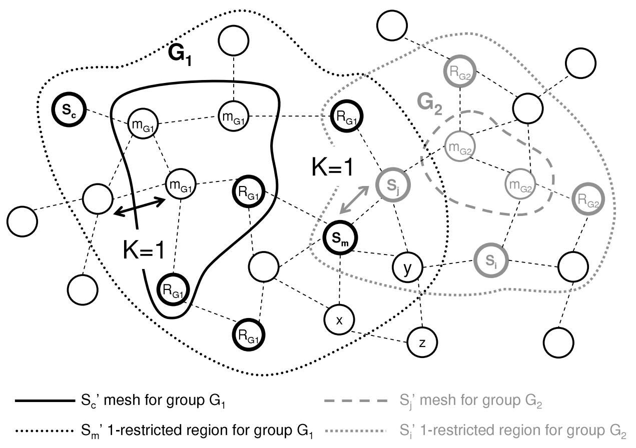

Fig. 1. *k*-restricted region. A *k*-restricted region of a non-core sender is a set of nodes that forward Non-Core Join Query messages generated by that source.

Sources other than the core in the same multicast group are considered regular sources or *non-core senders*. Non-core senders transmit Non-Core Join Query (*JQnC*) messages to build their own trees or meshes. A *JQ* is composed of:

- A packet type identifier.
- The address of the multicast group.
- A Time-To-Live field.
- The distance to the core.
- A sequence number.

In addition, a *JQnC* contains the address of the non-core sender, the distance to the non-core sender and the address of the selected parent towards the core. This last field is used to route *JQnC*s towards the mesh of the core.

Because non-core nodes benefit from the routing structure created by the core, the transmission of *JQnC*s is roughly synchronized with the reception of *JQ*s. Upon receiving a *JQ* with a larger sequence number, non-core senders wait for a random period of time that is much smaller than the join query period, but long enough to allow the establishment of the routing structure of the core before transmitting their next *JQnC*. That *JQnC* refreshes the routing information for that source. Non-core senders also send *JQnC* when they have data to send but no route is known to the receivers, or when the sender has not received a *JQ* from the core in the last two consecutive join query periods.

The objective of the combined use of *JQ*s and *JQnC* for a given multicast group is to order all nodes with respect to the core of the group, and to make the multicast routing structure (mesh or tree) as close as possible to the aggregation of the source trees of all the multicast sources in the group. *JQ*s must be sent to all nodes; however, the overhead due to the dissemination of *JQnC*s is reduced using the following two mechanisms.

The first way of reducing the overhead incurred with *JQnC*s consist of disseminating them only to a subset of the network composed of nodes that are part of the mesh or tree established by the core, nodes that lay in the path from the non-core sender to the core, and nodes located at most *k* hops away from them. The set of nodes that forward *JQnC*s for a given non-core sender is called the source's *k-restricted region of interest* or simply *k-restricted region*. This way, the dissemination of *JQnC*s is carried out only among nodes that are likely to be close to receivers, and other regions of the network do not receive irrelevant control information.

The optimal value of *k* for a *k*-restricted region depends on the topology of the network, as well as on the mobility of the nodes and on the length of the join query period. In our experiments, a sensitivity analysis showed that a reasonable value for *k* is one. In general, as the value of *k* grows, the *k*-restricted region grows larger, which helps to cope with mobility. In the worst case, the *k*-restricted regions cover the entire network, and the scheme degenerates to the case of flooding the network with control packets per sender per group as in ODMRP.

Fig. 1 illustrates the above concepts. The figure shows two multicast groups, G_1 and G_2, with their respective cores, S_c and S_j. Each group has a non-core source; S_m for G_1 and S_i for G_2. The mesh of core S_c of group G_1 is composed of nodes labeled m_{G1} and the mesh of core S_j of group G_2 is composed of nodes labeled m_{G2}. In the figure, the 1-*restricted region* of S_m is delimited by a dotted line. We observe that it contains the mesh constructed by the core of group S_c, which is delimited by a solid line, as well as the nodes located one hop way from the mesh or from the path from the non-core sender to the mesh. *JQnC*s generated by S_m are forwarded only by such nodes as node x or node y, which are located inside of the 1-restricted region, and nodes located outside of this region, such as node z, may receive the packet but do not forward it. The figure also presents a similar situation for group G_2.

The second way to reduce the number of *JQnC*s transmitted and the state kept at nodes consist of finding common sub-graphs and performing multicast state aggregation on these particular regions of the network. Nodes located in a common sub-graph only receive and forward *JQ*s (or *JQnC* s) of one of the sources that share that sub-graph and keep state about

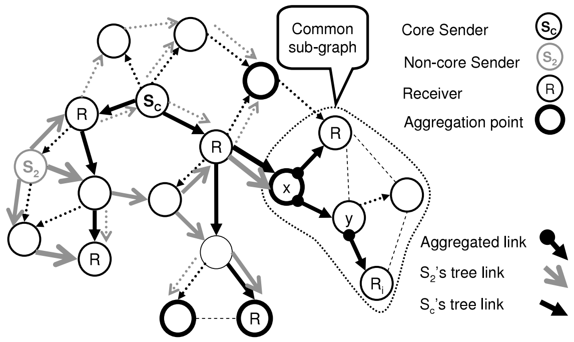

Fig. 2. Aggregated sub-graphs.

the source whose join queries are forwarded. Fig. 2 shows an example of a network in which two sources, S_c and S_2, share a common sub-graph. With this goal, we propose the *Dissemination of Multicast Aggregated-State (DIMAS)* algorithm. From the standpoint of message complexity, DIMAS behaves as simple flooding in the worst case. However, depending on the topology of the network, DIMAS stops disseminating control packets of non-core senders before covering the whole k-restricted region.

3.2. Join Reply messages

After receiving a *JQ* or a *JQnC* with a fresher sequence number, a receiver generates a Join Reply (*JR*) message. A *JR* contains:

- A packet type identifier.
- The address of the multicast group.
- The address of the sender (either core or non-core).
- The distance to the sender.
- The address of the selected parent towards the core.
- A sequence number.

A *JR* is routed back to the source following the reverse direction of the gradient of the distances established by the partial ordering obtained from the diffusion of *JQ*s. *JR*s travel on a hop-by-hop basis forcing intermediate nodes to join the multicast routing structure, until the *JR*s reach the first node that is already part of the multicast routing structure or a multicast source. If a *JR* is generated inside of a shared sub-graph, it travels along the shared region establishing routing state only about a single source. However, as soon as the *JR* reaches a node in which one or more *JQnC* were stopped by the DIMAS algorithm, a new *JR* is generated for each *JQnC* that was stopped in that particular node. Then, these *JR*s follow their own independent paths towards the different sources.

From the example shown in Fig. 2, we observe that a *JR* generated by receiver R_i travels to node y, and establishes routing state regarding only the core S_c. However, when the *JR* reaches the aggregation point x, two independent *JR*s are forwarded towards S_c and S_2. As the figure also show, these two *JR*s can follow independent paths towards their respective sources.

When nodes forward *JR*s, they can select one or more parents to reach a source. In the first case the resulting routing structure approximates a tree, while in the latter case is a mesh. In Hydra, we chose to keep state per source because we want to avoid forwarding data packets to places where they are not needed, e.g., towards other senders. This last design decision implies an increase in the state kept at the nodes. However, as our simulations show, bandwidth is a much more stringent bottleneck than memory, and spending extra memory in order to save bandwidth is a good tradeoff. Furthermore, keeping state per sender is necessary if the protocol has to support a source-specific multicast service model [14].

3.3. State aggregation: Dissemination of multicast aggregated-state (DIMAS)

The objective of the DIMAS algorithm is to reduce the number of control packets disseminated needed to establish a partial ordering that is as close as possible to the ordering that would be obtained by a per-source per-group dissemination. To do so, nodes determine if they are located in the boundary of a region that would likely be ordered by a *JQ* of a given source (say S_i) in an equivalent way as it was already ordered by a previous dissemination of *JQ* s generated by a different source (say S_j). If this is the case, then nodes stop the dissemination of control packets from S_i and mark that source as aggregated with S_j. Beyond this point, data packets generated by S_i are forwarded as if they were data packets from S_j. DIMAS$(i, snd : 1 \dots n)$ is executed at node i when it is about to relay a *JQnC* from sender *snd* to decide whether the message is to be sent or the sender *snd* is aggregated at node i. We assume that the execution of DIMAS is atomic. The following three rules are used to decide when to aggregate.

Rule 1: Upon reception of a *JQnC* with a larger sequence number, nodes wait for a period of time equal to *FWD_DLY* to collect packets forwarded by other neighbors. Based on the distances stated in these *JQnC*, nodes compute their own distance

```
Algorithm 1: DIMAS($i, snd : 1...n$)
1  $JQnC.snd \leftarrow snd$;
2  $JQnC.sn \leftarrow \max_{x \in Nbr | x.snd = snd}\{x.sn\}$;
3  $JQnC.dist \leftarrow \min_{x \in Nbr | x.snd = snd \wedge JQnC.sn = x.sn}\{x.dist\} + 1$;
4  $sGrdS \leftarrow tGrdS \leftarrow \emptyset$;
5  forall $x \in Nbr \mid x.snd = snd$ do
6      if $x.time + STL\_PRD > clock.getTime \wedge x.sn = JQnC.sn$ then
7          $\Delta \leftarrow JQnC.dist - x.dist$;
8          if $\Delta > 0$ then
9              $sGrdS \leftarrow sGrdS \cup \{(x.id, \Delta)\}$;
10 forall $y \neq (snd, nil, *, true) \in Snd$ do        /* Desc order respect
   snd's id */
11     $lsn \leftarrow \max_{x \in Nbr | x.snd = y.id}\{x.sn\}$;
12     $td \leftarrow \min_{x \in Nbr | x.snd = y.id \wedge x.sn = lsn}\{x.dist\} + 1$;
13     forall $x \in Nbr \mid x.snd = y.id$ do
14         if $x.time + STL\_PRD > clock.getTime \wedge x.sn = lsn$ then
15             $\Delta \leftarrow td - x.dist$;
16             if $\Delta > 0$ then
17                 $tGrdS \leftarrow tGrdS \cup \{(x.id, \Delta)\}$;
18     if $sGrdS = tGrdS$ then
19         $Snd \leftarrow (Snd - \{(snd, *, *, *)\}) \cup \{(snd, y.id, JQnC.sn, true)\}$;
20         return;
21 forall $y \neq (snd, nil, *, false) \in Snd \mid snd < y.id$ do   /* Desc order
   respect snd's id */
22     $tGrdS \leftarrow \emptyset$;
23     $lsn \leftarrow \max_{x \in Nbr | x.snd = y.id}\{x.sn\}$;
24     $td \leftarrow \min_{x \in Nbr | x.snd = y.id \wedge x.sn = lsn}\{x.dist\} + 1$;
25     forall $x \in Nbr \mid x.snd = y.id$ do
26         if $x.time + STL\_PRD > clock.getTime \wedge x.sn = lsn$ then
27             $\Delta \leftarrow td - x.dist$;
28             if $\Delta > 0$ then
29                 $tGrdS \leftarrow tGrdS \cup \{(x.id, \Delta)\}$;
30     if $sGrdS = tGrdS$ then
31         $Snd \leftarrow (Snd - \{(snd, *, *, *)\}) \cup \{(snd, y.id, JQnC.sn, true\}$;
32         return;
33 $Snd \leftarrow (Snd - \{(snd, *, *, *)\}) \cup \{(snd, nil, JQnC.sn, true)\}$;
34 $snd_{i,*}(JQnC)$;
```

Fig. 3. Total-DIMAS algorithm.

to the source and a set of pairs (*neighbor*, *gradient*), where the gradient is computed as the node's distance minus the distance reported by each neighbor. Then, nodes check if they have recently received *JQ*s or *JQnC*s from another source, such that the set $\{(neighbor, gradient) : gradient \geq 0\}$ matches with the one computed for the current source. If that is the case, nodes do not forward the *JQnC* and mark the senders as aggregated. If there is no match, nodes forward their own *JQnC* (with their computed distance).

Rule 2: If a node receives *JQnC*s generated by different sources at roughly the same time (within a *FWD_DLY* period) and if there is a match between the sets of gradient pairs, then the node forwards the control packet corresponding to the source with the *largest identifier* and stops the control packets corresponding to the other sources.

Rule 3: The core source is not aggregated to any non-core source. Aggregation is allowed only either among non-core sources, or with the core aggregating non-core sources. This rule has an exception that is described at Section 3.4.

There are two possible versions of the DIMAS algorithm. We call the one shown in the Fig. 3, Total-DIMAS, because it allows the aggregation of a non-core sender with any other sender. An alternative implementation that only allows aggregation of non-core senders with the core is called Core-DIMAS. The algorithm is composed of three main loops. The first loop (lines 5–9) computes the gradient set for the sender under consideration; it only takes into account information regarding the sender's current sequence number and those *JQnC* received during the last *STL_PRD* seconds. The second (lines 10–20) and third (lines 21–32) loops check for a match between the gradient sets of the sender under consideration and any other source of the same multicast group. The second loop only iterates over those senders for whom a *JQ* or *JQnC* has already been forwarded, and the third loop over senders form whom a *JQ* or *JQnC* is about to be sent in the current control bundle.

We use two separate loops to favor the aggregation with senders with more stable routing structures. In the same way, the algorithm can also be easily extended to favor the aggregation with the core. Both loops process the sender list in descending order to guarantee that per each set of senders with matching gradient sets, a single sender is selected to aggregate their multicast state.

3.3.1. *Data structures*

DIMAS uses a few data structures. The neighbor list *Nbr* is a set of tuples of the form $(id, snd, sn, dist, time)$. Each tuple contains data regarding the freshest (according to the largest sequence number) *JQ* or *JQnC* received from a given neighbor and originated by a particular source. The identifier of the neighboring node is *id*, *snd* is the identifier of the sender, *sn* is the freshest sequence number, *dist* is the distance to the sender and *time* is the node's local time at which the *JQ* or *JQnC* was received.

The sender list *Snd* is a set of tuples $(snd, sWith, sn, fwd)$ that stores information regarding the state kept for each sender with identifier *snd*. *sn* is as defined before, *sWith* contains either the identifier of the source with which the sender *snd* is aggregated, or *nil* if the sender *snd* is not aggregated. The flag fwd indicates if the node has forwarded a *JQ* or *JQnC* for that sequence number. $sGrdS$, $tGrdS$ are list of pairs *(neighbor identifier, gradient value)* that state if a neighbor is parent, child or sibling of the current node for a given source. $sGrdS$, $tGrdS$ are temporary data structures that are used to compare the gradient sets of two sources. We also define two constants *FWD_DLY*, *STL_PRD*. The first one is the amount of time a node waits since it receives the first *JQ* or *JQnC* with a fresher sequence number until it sends its own *JQ* or *JQnC* message. The latter is the time within which a previously sent *JQ* or *JQnC* is considered to be recent, so that the source under consideration can be aggregated with that source. Typically, the value of *STL_PRD* is set to one third of the time between consecutive transmissions of *JQ* messages. We also assume that every node has its own independent clock (*clock*) that can be locally queried by calling a function *getTime*.

3.3.2. *Disseminating aggregation maps*

JQs and *JQnCs* can be augmented with an *aggregation map* containing pairs of nodes identifiers of the form (*aggregated, aggregatedWith*), where *aggregated* is the identifier of a source that is aggregated to other source with identifier *aggregatedWith*. The aggregation maps are stored at downstream nodes and can be used to decide which source's routing information has to be used when forwarding a data packet of an aggregated source. Aggregation maps permit the forwarding of multicast data packets from sources whose state has been aggregated. In effect, they are routing-table extensions, as we discuss in Section 3.5.

3.3.3. *Opportunistic grouping of control packets*

In order to take advantage of the broadcast nature of the the wireless medium and save extra bandwidth, nodes opportunistically stack as many control messages as possible in *control bundles*. Control bundles can contain a mixture of Join Query, Non-Core Join Query and Join Reply messages regarding different multicast groups and sources. Control messages are grouped opportunistically in the sense that nodes wait for a short period of time before transmitting a newly generated control message so that other control messages that may be generated within this period of time are sent in the same control bundle.

3.3.4. *Preventing aggregation chains*

DIMAS prevents the formation of *aggregation chains* inside of a node. An aggregation chain is of the form: S_i *is aggregated with* S_j, S_j *is aggregated with* S_k, S_k is *aggregated with* S_l and so on. The problem with aggregation chains is that they may violate the condition that a source should be aggregated with another source only if their respective *JQnCs* were received within an interval of *STL_PRD* seconds (lines 6, 14 and 26 of Algorithm 1). This "freshness" condition is important because it assures that the routing structure of a source is constructed using recent topological information. As an example, lets assume that at a given node, a *JQnC* generated by source S_l is received at time t_0 and that another *JQnC* of source S_k is received at time t_1 with $t_1 - t_0 < STL\_PRD$. Lets further assume that their gradient sets are the same. Then, from Algorithm 1, S_k is aggregated with S_l. Now, if a new *JQnC* generated by a third source S_j is received at time t_2 with $t_2 - t_1 < STL\_PRD$ and if the gradients sets of S_j and S_k are the same, then S_j may be aggregated with S_k, which leads to a potential violation of the freshness property if $t_2 - t_0 > STL\_PRD$. A simple way to avoid aggregation chains is to exclude from consideration those sources that were aggregated with any other source. This can be seen in Algorithm 1 at the *forall* cycles, defined at lines 10 and 21, which only iterate over sources with a *nil* value in their *aggregatedWith* field.

The state-aggregation algorithms just described may allow the creation of duplicate paths from senders to receivers. This can happen when *JQnC*s from a given sender s are stopped at a set of nodes S but removing the set of links with an endpoint in S do not create a partition between the sender s and a receiver r. Under this scenario, it is possible for a *JR* message generated by the receiver r for some other source S_x to establish an aggregated $s \rightsquigarrow r$ path in addition to the non-aggregated path established by the *JR* generated by r for s. However, establishing more than one path from senders to receivers is also common in mesh-based protocols. This is not a major drawback, because in some situations these extra paths may help to increase the reliability of delivery.

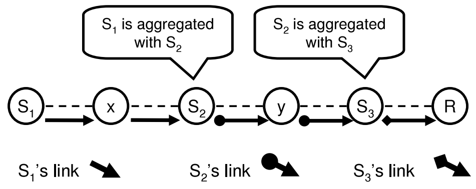

Fig. 4. Aggregated path. The $S_1 \rightsquigarrow R$ path is an aggregated path composed of the $S_1 \rightsquigarrow S_2$, $S_2 \rightsquigarrow S_3$ and $S_3 \rightsquigarrow R$ sub-paths.

3.4. Non-destructive core election

If a source has data to send to a multicast group, it first determines whether it has received a *JQ* from the core of that group. If that source node has, it adopts the core specified in the *JQ* it has received and transmits a *JQnC* advertising the same core for the group. Otherwise, it considers itself the core of the group and starts transmitting *JQ*s periodically to its neighbors, stating itself as the core of the group and a 0 distance to itself. Nodes propagate *JQ*s based on the best *JQ* they receive from their neighbors. A *JQ* with a higher core id is considered better than a *JQ* with a lower core id. Eventually, each connected component has only one core. If a sender becomes active for a group before other senders, then it becomes the core of the group. If several senders become active concurrently, then the one with the highest id is elected the core of the group.

A core election is also held if the network is partitioned. The election is held in the connected component of the partition that does not have the old core. A node detects a partition if it does not receive a fresh *JQ* from the core for three consecutive join query periods. Once a sender detects a partition and it has data to send, it promotes itself to the rank of core and participates in the core election. *JQ*s from nodes with lower ids are not discarded and the routing information regarding those senders is not destroyed. Instead, *JQ*s are just demoted to *JQnC* and they are forwarded using the *k*-restricted scheme and become susceptible of being stopped by the DIMAS algorithm. This scheme contrasts with the destructive schemes used in previous core-based multicast routing protocols (e.g., PUMA) in which the partial routing structure built by the dissemination of *JQ*s of senders that contended and lose an election is eliminated when *JQ* from senders with larger ids are received.

The way in which two partitions are merged depends on the type of join queries that traverse from one partition to the other. If a *JQ* reaches a new partition with a ''better'' core, then it is demoted to a *JQnC* and disseminated accordingly in that region of the network. The node or nodes that received the *JQ* from the core with a smaller id check if they have recently forwarded a *JQ* for the current core (for instance, within the last 100mS). If not, then they send *JQ*s that merge the partitions. While traversing the region of the network with the smaller core, *JQ*s force nodes to change to the new core but do not destroy their current routing information regarding the previously known sources. When the *JQ* is received by the senders located in the previously different partition with the smaller core, they generate new *JQnC*s with larger sequence numbers if at least one third of the join query period has elapsed since the last transmission of a join query.

For the case of Non-Core Join Queries we have the following options. If a *JQnC* reaches a previously different partition with a better core, then the behavior is analogous to the one just described with the only difference that there is no need of demoting the message because it is already a *JQnC*. On the other hand, when a *JQnC* arrives to a previously different partition with a smaller core, nodes that first receive the message aggregate the core stated at the arriving *JQnC* with the sender that originated it, and relay the *JQnC* but now stating as a core the core with the smaller id. Nodes located at the border of the region of the network with the core with smaller id are allowed to perform aggregation because (1) nodes in that region have not received a *JQ* from the core with larger id (for otherwise that sender would be the core), and (2) it is certain that their links are cut links between the core with the larger id and the receivers that may be located at the region with the core with smaller id. If the two regions remain connected long enough, then the next *JQ* generated by the core with larger id will force all nodes in the network to have a single core.

3.5. Forwarding multicast data packets

When a source has data to send, it first checks whether it has received at least one *JR* with the same sequence number as the last transmitted *JQ* or *JQnC*. If it is the case, the source considers the node from which it received the *JR* a child and transmits the data packet. If the source does not have any child, then it checks if has elapsed *ALLOW_NEXT_JQ* time since the last time it sent either a *JQ* or a *JQnC*. If so, it piggybacks the data packet in a *JQ* (or *JQnC*) with a newer sequence number and transmits it. Otherwise, the packet is silently dropped.

A multicast data packet received from a sender s_i is discarded by a node if a hit is found in the packet cache at the node based on the packet's sender and its sequence number. Otherwise, the receiving node inserts the *(sender's address, sequence number)* pair in its packet cache and determines the *address of the effective source* (*es*) of the packet, which is the one used to decide whether the packet has to be forwarded or not. The address of the effective source for a given packet is the original source s_i, or s_j if s_i was aggregated with s_j at the current node. Once the node determines the value of *es*, it forwards the data packet if it has at least one child for *es*. The effective source is obtained from the aggregation map maintained by each node.

Fig. 4 shows a simple example of a path composed of three concatenated paths, each of which corresponds to a path established for an aggregated source. In the figure, data packets generated by S_1 are routed by *x* using S_1 as the address of the effective source (*es*). When a S_1's packet reaches S_2, it determines from its aggregation map that the effective source for

S_1 is itself (S_2) and forwards the packet accordingly using its routing table. Nodes along the sub-path from S_2 to S_3 similarly determine that S_2 is the effective source for S_1. At node S_3, the effective source becomes S_3 itself, and the same is true for the relays in the sub-path from S_3 to R.

If aggregation maps are not communicated among neighbors as part of *JQnCs* and are maintained only locally, the original multicast data packet can be encapsulated in another multicast data packet with the aggregating source as the sender and the same group as the destination. At each hop, a relay decapsulates and encapsulates the packet before forwarding it.

4. Correctness of DIMAS

The following notation is used in this section. Let $G = (V, E)$ be a connected graph and $s \in V$ be any node of G. $T = (V, E)$ denotes a tree rooted in a designated node $s_c \in V$. $R \subset V$ is a set of nodes designated as receivers and $S \subset V$ is a set of nodes designated as senders. $d_s(u)$ is the distance in hops assigned to node u by an ordering initiated by node s.

Theorems 1 and 2 prove the correctness of the Core-DIMAS algorithm for a general topology. Theorem 3 shows the correctness of the Total-DIMAS algorithm only for acyclic graphs. In practice, a tree topology can be approximated by setting the value k of the k-restricted dissemination of *JQnC* to 0 and employing only one parent when establishing the routing structure of the core. However, our simulation results show that the Total-DIMAS and Core-DIMAS versions of our protocol achieve similar delivery ratios, even for values of $k > 0$ and when the core establishes a mesh instead of a tree.

Theorem 1. *Ordering the nodes in G with respect to its distance in hops to s establishes reverse loop-free paths from any node $u \in V$ to s by selecting as a next hop to s a node with a distance strictly less than the distance of the current node.*

Proof. We proceed by contradiction and assume that a loop is created in a path $u_0 \rightsquigarrow s = u_0u_1 \dots u_m s$ when a node u_j selects u_k, with $d_s(u_j) > d_s(u_k)$, as its next hop to s. Now, if there is a loop in the path, then there exist a down stream node u_i such that $u_i = u_k$. However we know that $d_s(u_o) > d_s(u_1) > \dots > d_s(u_i) > \dots > d_s(u_j) > d_s(u_k)$; hence $d_s(u_k) > d_s(u_k)$, which is a contradiction. ■

Theorem 2. *The Core-DIMAS algorithm can be used in conjunction with a join query and join reply signaling to establish loop-free routes from any sender $s \in S$ to any receiver $r \in R$.*

Proof. Let $s_i \in S$ and $r_j \in R$ be an arbitrary sender–receiver pair. From the specification of DIMAS, we know that a non-core sender s_i sends a $JQnC_{s_i}$ message to its neighbors when it needs to establish routes towards the senders. From the DIMAS specification we also know that every node in V relays $JQnC_{s_i}$, unless $JQnC_{s_i}$ is aggregated with the JQ_c sent by the core s_c. Now, if r_j receives a copy of $JQnC_{s_i}$, it follows from Theorem 1 that we can use the ordering established by the $JQnC_{s_i}$ to establish a loop-free path $s_i \rightsquigarrow r_j$. If this is not the case, then there is a set of nodes $U \subset V$ where the $JQnC_{s_i}$ messages were aggregated with JQ_c messages. Therefore, given that $JQnC_{s_i}$ could not reach r_j, we can identify a cut $C = (A, B)$ with $s_i, s_c \in A$ (otherwise s_i, s_c would not have the same gradient set at the nodes in U), $r_j \in B$ and cut edges of the form (u, v), where $u \in U \subset A$ and $v \in B$. Because C is a cut, any JR_{s_c} message generated by r_j has to reach a node in U (say u) on its way to s_c. Accordingly, we can establish a loop-free path from r_j to u, which can be concatenated with a $s_i \rightsquigarrow u$ path to get the desired $s_i \rightsquigarrow r_j$ path.

Now we just simply argue that the concatenation of aggregated paths is loop-free. We proceed by contradiction and assume that there is a loop in an aggregated path $s_i \rightsquigarrow r_j = s_iu_0u_1 \dots u_kr_j$ established using our algorithm. Hence, there must be a pair of nodes u_i, u_j in $s_i \rightsquigarrow r_j$ such that $u_i = u_j$. From Theorem 1, we know that u_i, u_j cannot be located in the same partition. However, because they are located in different partitions, they cannot be the same node. ■

Theorem 3. *In an acyclic graph $T = (V, E)$, the Total-DIMAS algorithm can be used in conjunction with a join query and join reply signaling to establish loop-free routes from any sender $s \in S$ to any receiver $r \in R$.*

Proof. Let $s_i \in S$ and $r_j \in R$ be an arbitrary sender–receiver pair. From the specification, we know that when a non-core sender s_i wants to establish routes towards the receivers it sends a $JQnC_{s_i}$ message to its neighbors. From the algorithm we also know that every node in V will relay its $JQnC_{s_i}$ message (with its computed distance to s_i) unless the $JQnC_{s_i}$ is aggregated with another JQ_{s_k} or $JQnC_{s_k}$ (we can use either JQ_{s_k} or $JQnC_{s_k}$ because we allow a non-core sender to be aggregated with other non-core sender) generated by a sender $s_k \in S$ with $s_k \neq s_i$. Now, if r_j receives a copy of the $JQnC_{s_i}$, then from Theorem 1, we can use the ordering established by $JQnC_{s_i}$ to establish a loop-free path $s_i \rightsquigarrow r_j$. If this is not the case, then the $JQnC_{s_i}$ was aggregated at an inner node $i \in V$ with other JQ_{s_k} that was received from the same node (say l) as $JQnC_{s_i}$, otherwise s_k and s_i would not have identical gradient sets. From the algorithm we also know that either $k > i$ or JQ_{s_k} had already been relayed when the $JQnC_{s_i}$ arrived at i.

Now, if we remove the edge (l, i) we have two sub-trees $T' = (V', E')$ and $T'' = (V'', E'')$ where $l, s_i, s_k \in V'$ and $i, r_j \in V''$. From lines 10 and 21 of the algorithm we know that if the $JQnC_{s_i}$ is aggregated with a JQ_{s_k} at i, then i has to relay (or it has already done that) JQ_{s_k} to its neighbors. Since T is a tree, and JQ_{s_k} was received from $l \in V'$ then, at the time the JQ_{s_k} was received at i, the remaining nodes in V'' have not received JQ_{s_k} yet, hence the JQ_{s_k} will be eventually relayed to all the nodes in T'' unless it is aggregated with other JQ_{s_m}. If JQ_{s_k} reaches r_j then we can use the ordering established by JQ_{s_k} to establish a loop-free path from r_j to i which can be concatenated with the path $s_i \rightsquigarrow i$ to get the desired $s_i \rightsquigarrow r_j$ path. If this is not the case, we can keep applying the same argument until r_j is reached. Now, we just have to argue that the concatenation of aggregated paths is loop-free. We proceed by contradiction and assume that there is a loop in an aggregated path $s_i \rightsquigarrow r_j = s_iu_0u_1 \dots u_kr_j$

Table 1
Simulation environment.

Total nodes	50	Node placement	Random	Simulation area	1400×1400 m^2
Data source	MCBR	Pkts. sent per src.	1000	Channel capacity	2000 000 bps
Tx. power	15 dbm				
MAC Protocol: WiFi					
Simulation time	150s				
Mobility model	Random waypoint	Pause time	10s	Min.–Max. Vel.	1–20 m/s
Mobility model	Group mobility	Grp. pause time	10s	Grp. Min.–Max. Vel.	1–10 m/s
		Node pause time	10s	Node Min.–Max. Vel.	1–20 m/s
MAC Protocol: TDMA (1 slot of 10ms per node in round-robin)					
Simulation time	30 min				
Mobility model	Random waypoint	Pause time	50s	Min.–Max. Vel.	1–2 m/s
Mobility model	Group mobility	Grp. pause time	50s	Grp. Min.–Max. Vel.	1–2 m/s
		Node pause time	50s	Node Min.–Max. Vel.	1–2 m/s

established using our algorithm. Hence in $s_i \rightsquigarrow r_j$ there must be a pair of nodes u_i, u_j such that $u_i = u_j$. From Theorem 1 we know that u_i, u_j cannot be located in the same partition, therefore, since they are located in different partitions they cannot be the same node. ■

5. Performance comparison

We present simulation results in which we compare six different variants of Hydra against ODMRP. The six variants consist of adding one or two parents per node when building the multicast routing structures, and using Total-DIMAS (Hydra-TA), Core-DIMAS (Hydra-CA) or no aggregation (Hydra-NA). We chose ODMRP for our comparison because it is a representative of the state of the art in multicast routing protocols for MANETs and has also become a *de facto* baseline for performance comparisons. Another reason for selecting ODMRP is that it constructs its forwarding mesh as the union of the source-specific trees of the senders of a particular group. Given that Hydra also builds a structure that is close to a set of source-specific trees or meshes, comparing Hydra against ODMRP allows us to highlight the benefits obtained by the signaling used in Hydra. Another popular protocol for performance comparisons is MAODV [7]. However, previous work [15] has shown that ODMRP clearly outperforms MAODV and hence we omit the latter in these experiments.

We use packet delivery ratio, end-to-end delay and number of packets relayed per packet received at receivers as our performance metrics. The latter metric can be seen as the cost in terms of bandwidth that the protocol pays to achieve a given delivery ratio.

The multicast protocols are tested with 802.11 DCF (WiFi) and TDMA as the underlying MAC protocols. The former is the most commonly used MAC in the MANET literature and the latter allows us to isolate multicast signaling and construction of routing structures from the effects of collisions at the MAC layer. In addition, we used random waypoint and group mobility [6] as our mobility models. The first model allows us to test the protocols on general situations in which each node moves independently, and the latter models situations in which the members of a team tend to move in groups.

We used the discrete event simulator Qualnet [16] version 3.9 which has the advantage of providing a realistic simulation of the physical layer. The software distribution of Qualnet itself has a very stable version of ODMRP, which was used for the ODMRP simulations. For PUMA simulations we used the original code used in [3]. Each simulation was run for ten different seed values. In our graphs we report the average and standard deviation computed over these ten independent runs. To have meaningful comparisons, all the protocols use the same period (join query period for ODMRP and Hydra, and announcement period for PUMA) to refresh their routing structures (3s for 802.11 and 30s for TDMA). For ODMRP, the forwarding group timeout was set to three times the value of the join query period, as advised by its designers. Table 1 lists the details of the simulation environment.

5.1. Results using 802.11 DCF

We first focus on an experiment in which the number of concurrent active senders changes. For this experiment we set the value of k to 1. Hence, only nodes that are at most one hop away from the routing structure of the core disseminate $JQnC$s. Each sender transmits 20 packets of 256 bytes per second and the group is composed of 20 nodes. We observe from Fig. 5(a) that all the versions of Hydra (TA, CA and NA for mesh and tree topologies) perform similar or slightly better than ODMRP for more than one source. Among the Hydra variants, we can observe that the mesh versions tend to outperform the tree versions only for a small number of senders, whereas the tree versions clearly outperform the mesh versions for six to twelve senders. This is an indication of how soon the extra redundancy used by the mesh versions becomes counterproductive as the load injected into the network increases. We can also observe that the versions that use aggregation are capable of attaining delivery ratios equivalent to the ones attained by the versions that do not use aggregation, but incur much less control overhead. In these experiments, the tree and mesh versions of Hydra-TA transmitted an average of 77.89% and 78.73% of the Join Queries ($JQ + JQnC$) transmitted by the tree and mesh versions of Hydra-NA, respectively.

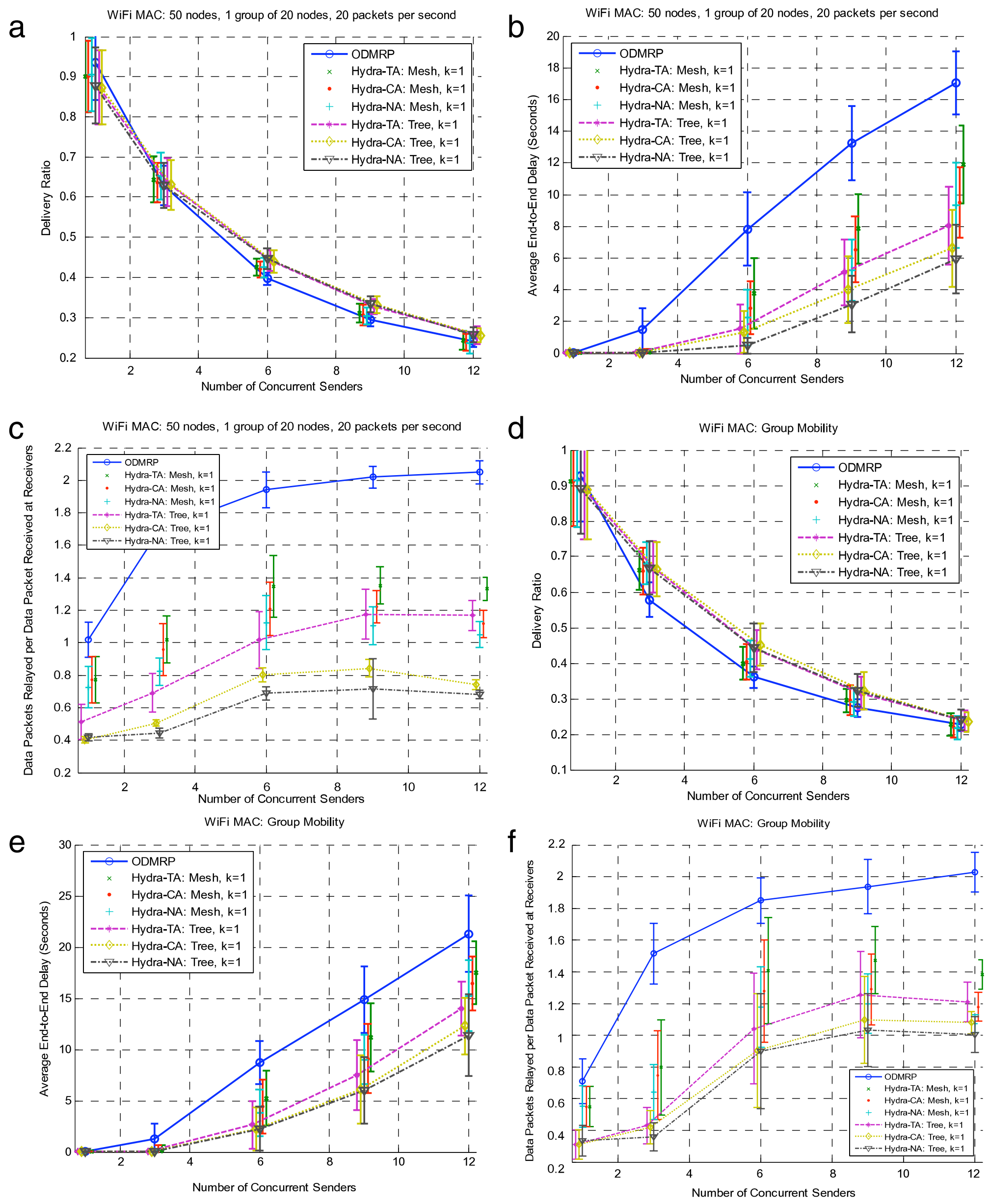

Fig. 5. 802.11 DCF MAC: Performance with increasing number of concurrent active sources. (a–c) Random waypoint mobility. (d–f) Group mobility. (a and d) Delivery ratio. (b and e) Average end-to-end delay. (c and f) Average number of data packets relayed per data packet received at receivers.

Fig. 5(c) shows the average number of data packets relayed per packet received by receivers. We can see that ODMRP incurs considerably more redundancy than the Hydra variants. As we would expect, the variant of Hydra that uses less redundancy is the tree version with no aggregation, while the one that uses more redundancy is the mesh version with total aggregation. In general, the versions that use total aggregation generate more redundancy than the core aggregation versions, which in turn generate more redundancy than the versions that do not use aggregation. This is due to the fact that the no-aggregation versions try to establish source-specific shortest-path trees or meshes, while the structures created by the aggregated versions are not necessarily composed of shortest paths. If we analyze these two metrics, we notice that the versions of Hydra are able to attain higher delivery ratios at a much smaller cost. This is a strong indication that the routing structures built by Hydra are more efficient than ODMRP's mesh.

Regarding the average end-to-end delay attained by the protocols, From Fig. 5(b) we can observe that the versions of Hydra clearly outperform ODMRP. There are two important factors behind this behavior. The first one is the queueing delay, which is strongly correlated to the amount of redundancy used by the protocol. Hence, protocols that incur less redundancy when forwarding data packets and that induce less control traffic tend to attain lower end-to-end delays than the ones with increased level of redundancy. The other factor is the length of the paths followed by the data packets. Therefore, the

Mobility (pause time)	0S		5S		10S		15S		20S	
	Avg	SD	Avg	SD	Avg	SD	Avg	SD	Avg	SD
	Delivery Ratio									
ODMRP	0.632	0.037	0.639	0.055	0.630	0.049	0.629	0.045	0.630	0.050
Hydra-TA:Mesh	**0.656**	**0.057**	0.646	0.048	0.644	0.057	**0.646**	**0.056**	0.635	0.054
Hydra-CA:Mesh	0.649	0.053	**0.651**	**0.060**	0.637	0.049	0.641	0.057	**0.643**	**0.059**
Hydra-NA:Mesh	**0.657**	**0.046**	**0.649**	**0.057**	**0.653**	**0.057**	**0.645**	**0.060**	**0.643**	**0.059**
	End-to-End Delay (S)									
ODMRP	1.691	1.374	1.570	1.308	1.513	1.349	1.594	1.490	1.291	1.232
Hydra-TA:Mesh	0.166	0.360	0.180	0.288	0.101	0.164	0.172	0.427	0.134	0.247
Hydra-CA:Mesh	0.179	0.311	**0.049**	**0.048**	0.107	0.174	**0.078**	**0.146**	0.068	0.115
Hydra-NA:Mesh	**0.092**	**0.143**	0.076	0.107	**0.026**	**0.004**	**0.082**	**0.114**	**0.036**	**0.022**
	Number of Data Packets Relayed per Data Packet Received at Receivers									
ODMRP	1.714	0.125	1.699	0.090	1.696	0.101	1.727	0.113	1.695	0.120
PUMA	1.717	0.344	1.840	0.428	1.863	0.421	1.747	0.332	1.768	0.349
Hydra-TA:Mesh	0.986	0.140	1.064	0.171	1.020	0.146	1.001	0.184	1.034	0.162
Hydra-CA:Mesh	1.005	0.182	0.948	0.113	0.958	0.158	0.940	0.176	0.923	0.167
Hydra-NA:Mesh	**0.864**	**0.155**	**0.866**	**0.122**	**0.822**	**0.084**	**0.889**	**0.146**	**0.848**	**0.085**
Terrain Dimensions	1200x1200m		1300x1300m		1400x1400m		1500x1500m		1600x1600m	
	Delivery Ratio									
ODMRP	0.659	0.029	0.646	0.033	0.630	0.049	**0.619**	**0.049**	**0.597**	**0.070**
Hydra-TA:Mesh	0.685	0.048	0.670	0.052	0.644	0.057	**0.618**	**0.057**	0.586	0.062
Hydra-CA:Mesh	**0.703**	0.052	0.672	0.049	0.637	0.049	**0.618**	**0.062**	0.587	0.059
Hydra-NA:Mesh	**0.709**	**0.053**	**0.680**	**0.052**	**0.653**	**0.057**	**0.617**	**0.064**	0.589	0.062
	End-to-End Delay (S)									
ODMRP	2.875	1.531	2.098	1.575	1.513	1.349	0.854	0.972	0.464	0.602
Hydra-TA:Mesh	0.667	0.765	0.267	0.382	0.101	0.164	0.058	0.071	0.038	0.016
Hydra-CA:Mesh	**0.384**	**0.598**	0.313	0.542	0.107	0.174	0.033	0.009	0.031	0.007
Hydra-NA:Mesh	**0.345**	**0.645**	**0.175**	**0.304**	**0.026**	**0.004**	**0.027**	**0.005**	**0.027**	**0.004**
	Number of Data Packets Relayed per Data Packet Received at Receivers									
ODMRP	1.660	0.119	1.699	0.128	1.696	0.101	1.663	0.137	1.690	0.093
Hydra-TA:Mesh	1.050	0.258	1.017	0.194	1.020	0.146	0.994	0.132	1.092	0.082
Hydra-CA:Mesh	0.888	0.225	0.967	0.246	0.958	0.158	0.982	0.096	1.010	0.112
Hydra-NA:Mesh	**0.802**	**0.231**	**0.846**	**0.187**	**0.822**	**0.084**	**0.851**	**0.087**	**0.917**	**0.094**
Group size (receivers)	10		15		20		25		30	
	Delivery Ratio									
ODMRP	**0.656**	**0.062**	0.636	0.052	0.630	0.049	0.625	0.034	0.620	0.045
Hydra-TA	0.650	0.063	0.651	0.055	0.644	0.057	0.639	0.058	0.634	0.050
Hydra-CA	**0.656**	**0.055**	**0.654**	**0.048**	0.637	0.049	**0.649**	**0.048**	0.645	0.054
Hydra-NA	0.647	0.059	**0.654**	**0.055**	**0.653**	**0.057**	**0.649**	**0.058**	**0.653**	**0.061**
	End-to-End Delay (S)									
ODMRP	0.7229	0.7501	1.2985	1.2738	1.5128	1.3494	1.5305	1.381	1.7905	1.4114
Hydra-TA:Mesh	0.0327	0.0271	0.0861	0.1705	0.1008	0.1641	0.271	0.4881	0.2859	0.4022
Hydra-CA:Mesh	**0.022**	**0.0034**	**0.0345**	**0.0209**	0.1074	0.1739	0.1853	0.252	0.2194	0.3614
Hydra-NA:Mesh	**0.0209**	**0.0019**	**0.0242**	**0.0043**	**0.0259**	**0.0043**	**0.0966**	**0.152**	**0.1395**	**0.2859**
	Number of Data Packets Relayed per Data Packet Received at Receivers									
ODMRP	2.857	0.248	2.145	0.158	1.696	0.101	1.368	0.095	1.200	0.067
Hydra-TA:Mesh	1.464	0.215	1.158	0.206	1.020	0.146	0.855	0.182	0.745	0.094
Hydra-CA:Mesh	1.304	0.207	1.080	0.124	0.958	0.158	0.829	0.115	0.741	0.094
Hydra-NA:Mesh	**1.250**	**0.140**	**0.972**	**0.122**	**0.822**	**0.084**	**0.734**	**0.116**	**0.647**	**0.126**

Fig. 6. Performance when varying mobility, node density, and group size with 802.11 DCF MAC.

versions of Hydra that build routing structures that are closer to source-specific trees, namely the ones that use less or no aggregation, tend to attain lower end-to-end delays than the versions that use more aggregation.

Fig. 5(d–f) present results for the group mobility model in which the 20 nodes that belong to the multicast group move around inside of a square region of 900 $\times$ 900 m^2. In the group mobility model, each group decides its group mobility direction and speed randomly. Each node then decides its internal mobility randomly and computes its actual mobility by summing the two mobility vectors [6]. The remaining 30 nodes, including sources, move following the random waypoint mobility model. From Fig. 5(d) we can notice that the delivery ratio attained by the Hydra family of protocols is very close to the one attained for the random waypoint mobility model. However, that is not the case for ODMRP whose delivery ratio is diminished. The reason is as follows. Given that receivers are concentrated in a particular region of the network and that sources are spread all over the simulation area, ODMRP meshes tend to cover more regions where no receivers and only sources are located and hence, to route data packets towards places where no receiver is located. This is also the reason for its reduced forwarding efficiency as shown in Fig. 5(f). On the other hand, the end-to-end delays (Fig. 5(e)) attained by all the protocols is increased with respect to the one attained for the random waypoint mobility model. The reason for this is that, on average, the paths from sources to receivers are longer when nodes move according to the group mobility model.

Fig. 6 summarizes a number of simulation results for three concurrent sources and different scenarios where we vary the mobility, node density, and multicast group size. Because of space limitations, we only show results for the mesh versions of

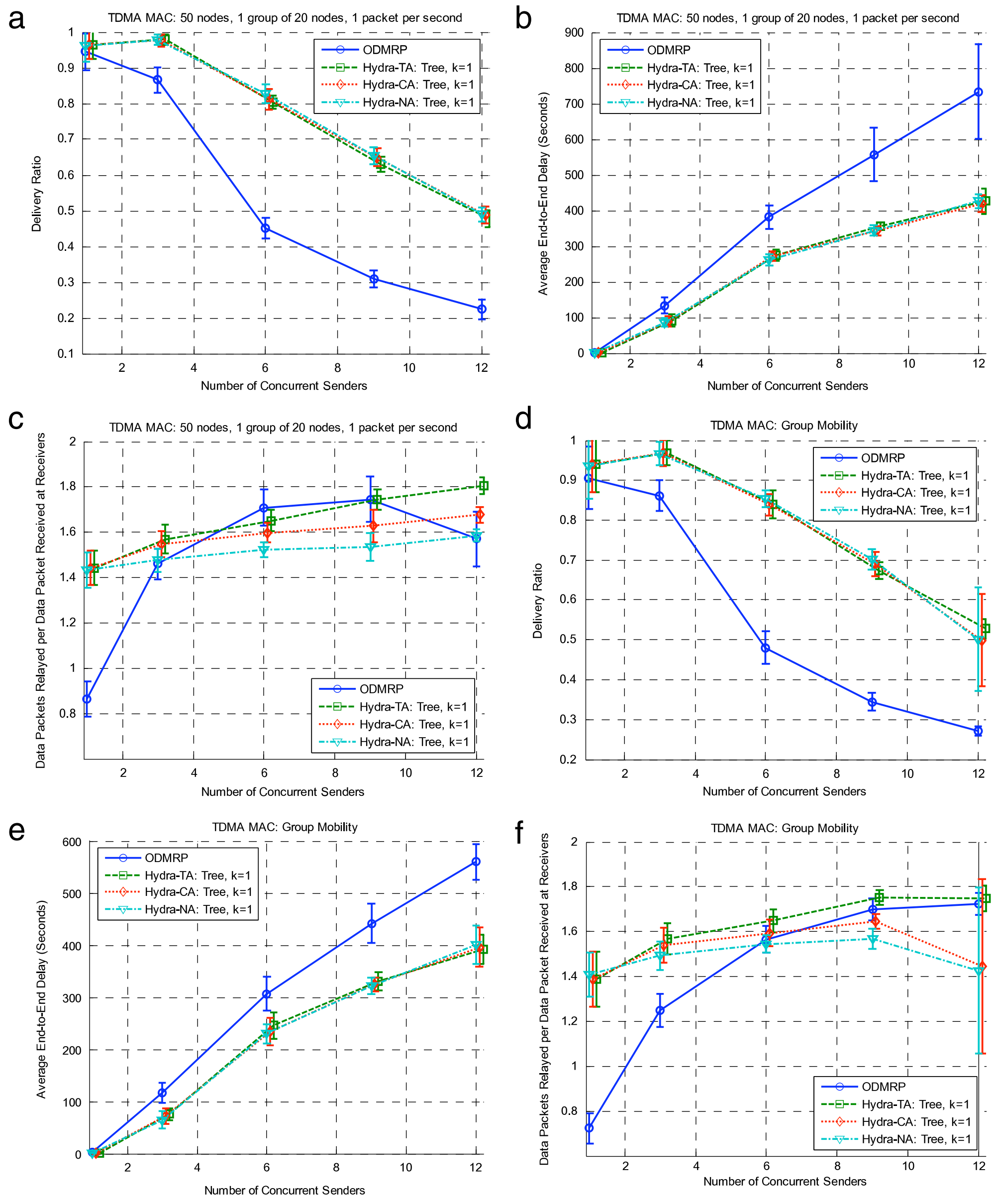

Fig. 7. TDMA MAC: Performance with increasing number of concurrent active sources (a–c) Random waypoint mobility. (d–f) Group mobility. (a and d) Delivery ratio. (b and e) Average end-to-end delay. (c and f) Average number of data packets relayed per data packet received at receivers.

Hydra. From the figure, we can observe that the variants of Hydra attain similar or better delivery ratios than ODMRP across the three scenarios. We can also observe that the Hydra variants incur far less forwarding overhead (around 30%–50%) than ODMRP, and render close to an order of magnitude improvement in delay with respect to ODMRP.

5.2. Results using TDMA

The results shown in Fig. 7(a–c) present a sample of the behavior of the protocols when the transmission of control information (by assigning control packets to a higher priority queue) is isolated from the effects of the data traffic. For these experiments, each sender transmits 1 packet of 256 bytes per second and the group is composed of 20 nodes. TDMA allocates 1 slot of 10 ms per node in round-robin fashion by address. In this scenario the join query period, the multicast announcement period, the sources' data rate and the mobility parameters are scaled so that all the protocols have good performance for one source given that nodes are able to access the medium only two times per second. For one sender, all the protocols have equivalent performance in terms of delivery ratio and end-to-end delay, as should be expected. However, as the number of sources increases, the performance of ODMRP drops sharply, which is due to the per-source per-group flooding strategy it uses. For six or more sources, the Hydra family provides an improvement of around 30% in delivery ratio and up to 40% in end-to-end delay. The decreased amount of data overhead shown by ODMRP for 12 sources is due to the fact that many more data packets are being dropped early.

Given that collisions are prevented by the scheduled nature of TDMA, the main reason for packets being lost consist of packets dropped due to queues being overflowed. This situation is particularly adverse to ODMRP and any routing protocol that transmits a considerable amount of high-priority control packets, which tend to stall the lower priority data packets.

Fig. 7(d–f) present the results attained by the protocols when the group mobility model is used. In this scenario we can observe an improvement in the three performance metrics of all the protocols over the performance attained with random waypoint mobility. This is due to the fact that receivers tend to be concentrated in the region of the network, and hence the trees established by the protocols are less shallow which is reflected in the improved forwarding efficiency shown in Fig. 7(f). As a result of a better forwarding efficiency we observe a reduction in the number of data packets sent to network which benefits the delivery ratio (Fig. 7(d)) and end-to-end delay (Fig. 7(e)) because it also reduce the number of packets dropped due to overflowed queues as well as the queueing delay.

6. Conclusions

We introduced and verified the Hydra multicast routing protocol for MANETs. Hydra is the first multicast routing protocol that establishes routing structures that approximate those built with sender-initiated approaches, but incurring the communication overhead of a receiver-initiated approach. This is accomplished by limiting the dissemination of control information from non-core sources to small regions of the network, and aggregating routing information by establishing common sub-graphs that are shared by two or more senders. Hydra can work in either mesh or tree mode by restricting the number of parents that are forced to join the routing structure. Cores are elected using a non-destructive core election protocol that does not destroy recently established trees (or meshes). To save bandwidth and take advantage of the broadcast nature of the wireless medium, Hydra opportunistically combines control messages from different groups and sources into a single control packet. We showed that the aggregation algorithms establish loop-free routes from senders to receivers. We also presented the results of a series of simulation experiments illustrating that Hydra attains higher delivery ratios and considerably lower end-to-end delays than ODMRP, while inducing far less retransmission overhead, even in relatively small networks with few multicast sources.

Acknowledgments

Work was partially sponsored by the U.S. Army Research Office (ARO) under grant W911NF-05-1-0246, by the National Science Foundation under grant CNS-0435522, by the Baskin Chair of Computer Engineering, by the UC MEXUS-CONACyT program and by the Mexican National Polytechnic Institute.

References

[1] J.J. Garcia-Luna-Aceves, E.L. Madruga, The core-assisted mesh protocol, IEEE Journal on Selected Areas in Communications 17 (8) (1999) 1380–1394.
[2] S.-J. Lee, M. Gerla, C.-C. Chiang, On-demand multicast routing protocol, in: Proc. of the IEEE Wireless Comm. and Net. Conf., 1999. WCNC, vol. 3, 1999, pp. 1298–1302.
[3] R. Vaishampayan, J.J. Garcia-Luna-Aceves, Efficient and robust multicast routing in mobile ad hoc networks, in: Proc. of the IEEE Conf. on Mob. Ad-hoc and Sensor Syst., 2004, Oct. 2004, pp. 304–313.
[4] T. Ballardie, P. Francis, J. Crowcroft, Core based trees (cbt), SIGCOMM Computer Communication Review 23 (4) (1993) 85–95.
[5] S.E. Deering, Multicast routing in internetworks and extended lans, SIGCOMM Computer Communication Review 18 (4) (1988) 55–64.
[6] X. Hong, M. Gerla, G. Pei, C.-C. Chiang, A group mobility model for ad hoc wireless networks, in: Proc. of ACM/IEEE MSWiM'99, 1999, pp. 53–60.
[7] E.M. Royer, C.E. Perkins, Multicast operation of the ad-hoc on-demand distance vector routing protocol, in: MobiCom'99: Proc. of the 5th Annual ACM/IEEE Intl. Conf. on Mob. Comp. and Net., ACM, 1999, pp. 207–218.
[8] J.G. Jetcheva, D.B. Johnson, Adaptive demand-driven multicast routing in multi-hop wireless ad hoc networks, in: MobiHoc'01: Proc. of the 2nd ACM Intl. Symp. on Mob. Ad Hoc Net. & Comp., ACM, 2001, pp. 33–44.
[9] V. Devarapalli, D. Sidhu, Mzr: A multicast protocol for mobile ad hoc networks, in: Proc. of the IEEE Intl. Conf. on Comm., 2001. ICC 2001, vol. 3, 2001, pp. 886–891.
[10] S.K. Das, B.S. Manoj, C.S.R. Murthy, A dynamic core based multicast routing protocol for ad hoc wireless networks, in: MobiHoc'02: Proc. of the 3rd ACM Intl. Symp. on Mob. Ad Hoc Net. & Comp., ACM, 2002, pp. 24–35.
[11] S. Lee, C. Kim, Neighbor supporting ad hoc multicast routing protocol, in: MobiHoc'00: Proc. of the 1st ACM Intl. Symp. on Mob. Ad Hoc Net. & Comp., ACM, 2000, pp. 37–44.
[12] P.M. Ruiz, A.F. Gomez-Skarmeta, Reducing data-overhead of mesh-based ad hoc multicast routing protocols by steiner tree meshes, in: In Proc. of IEEE SECON 2004, 4–7 Oct. 2004, pp. 54–62.
[13] J.-H. Cui, J. Kim, D. Maggiorini, K. Boussetta, M. Gerla, Aggregated multicast a comparative study, Cluster Computing 8 (1) (2005) 15–26.
[14] H. Holbrook, B. Cain, Source-specific multicast for ip. internet-draft, November 2000.
[15] K. Viswanath, K. Obraczka, G. Tsudik, Exploring mesh and tree-based multicast routing protocols for manets, IEEE Transactions on Mobile Computing 5 (1) (2006) 28–42.
[16] Qualnet 3.9, scalable network technologies: http://www.scalablenetworks.com.