Overlay, Borůvka-based, Ad-hoc Multicast Protocol: description and performance analysis

Andrea Detti, Nicola Blefari-Melazzi

University of Rome “Tor Vergata”, Electronic Engineering Dept., Italy

{andrea.detti, blefari}@uniroma2.it

Abstract— **This paper presents a novel MANET multicast protocol, named Overlay Borůvka-based Ad-hoc Multicast Protocol (OBAMP), and evaluates its performance. OBAMP is an overlay protocol: it runs only in the end-systems belonging to the multicast group. User data are distributed over a shared distribution tree formed by a set of non-cyclic UDP tunnels. OBAMP derives the distribution tree by approximating the Borůvka algorithm; the Borůvka algorithm is a classical tool (1926) to find the minimum spanning tree; thus, the distribution tree of OBAMP is an approximation of the minimum spanning tree of the connectivity topology at hand.**

OBAMP shows three distinctive advantages: i) its distribution tree closely resembles the minimum spanning tree; ii) it exploits broadcast communications (with favourable consequences on its efficiency); iii) its design takes into account not only overlay signalling but also network-layer signalling; thus, the protocol succeeds in limiting the overall signalling load, network+overlay. As a consequence, OBAMP has a low-latency and a high delivery ratio, even when the group size increases.

To prove this statement, we analyze the performance of OBAMP with ns-2 and compare it with two state-of-the-art protocols, namely ODMRP (a network-layer protocol) and ALMA (an overlay protocol). Both OBAMP and ALMA are assumed to use AODV as underlying routing protocol.

Also, we stress that we have implemented OBAMP, in Java, and we have tested it on the field, to prove its feasibility.

To allow fellow researchers to reproduce and test our work we published all simulation and implementation codes, in [11].

I. INTRODUCTION

An attractive use of Mobile Ad-hoc NETworks (MANETs) consists in performing cooperative work during occasional team tasks. In such scenarios, multicast traffic plays a more important role, with respect to the one that it would get in the public Internet. As a matter of fact, occasional team tasks are more in need of real time multipoint-to-multipoint services (e.g., push-to-talk, GPS positioning, etc.) than point-to-point ones.

MANET multicast protocols can operate as a network layer protocol or as an application layer protocol (known also as overlay). In the first case, all nodes run the protocol and all nodes can be involved in managing a multicast session, including nodes that are not members of the multicast session (e.g., ODMRP [2], MAODV [3], etc.).

On the contrary, an overlay multicast protocol involves only nodes that are members of the multicast session, i.e. the protocol is peer-to-peer (e.g., AMRoute [1], ALMA [7], PAST-DM [8]). User data and signalling information are transferred via transport layer tunnels (e.g. UDP sockets) among member nodes. Therefore, multicast functionality can be embedded within end-user applications, without any multicast support from the underlying network layer. This feature greatly simplifies the deployment of the multicast protocol and justifies the interest of the research community in this topic.

In this paper, we propose an overlay protocol, based on the so-called Borůvka algorithm [5]; we evaluate its performance by means of simulations; we compare it with two promising alternatives, ODMRP [2] (a network-layer protocol) and ALMA [7] (an overlay protocol); we implement it to assess its feasibility.

The main aim of OBAMP is to limit the network traffic, both user data and signalling information, so as to achieve a high delivery ratio and a low latency. Also, we would like the protocol to be scalable, i.e., to maintain these properties when the group size increases. This goal is achieved by: i) developing an efficient distribution tree that approximates as much as possible the minimum spanning tree (i.e., the overlay tree with the minimum number of network hops); ii) exploiting radio broadcasting; iii) taking into account and minimizing not only overlay signalling but also network-layer signalling.

An efficient distribution tree is a quite common objective for designers of overlay protocols. Radio broadcasting is exploited in only one (recent) proposal [8], while, to the best of our knowledge, only OBAMP tries to minimize not only overlay but also network-layer signalling[1].

The paper is organized as follows: in section II we discuss pros and cons of the overlay and network layer approaches; section III recalls some background on graph theory, useful to better understand the OBAMP theoretical foundations; section IV describes the protocol; section V focus on performance evaluation; section VI presents the conclusions. To avoid interrupting the flow of the paper, we report some additional results in appendices: Appendix I shows the sequence diagrams of the OBAMP procedures; Appendix II describes the format of the OBAMP messages; in Appendix III we extend the performance evaluation of section V, by analyzing a

[1] For instance, a frequent assumption when evaluating overlay multicast protocols is to assume a complete and always updated knowledge of the network topology, without considering the associated cost in terms of signaling exchanges and the routing inefficiencies due to the dynamics of the network links.

different source scenario; in Appendix IV we state three properties of the tree built by OBAMP.

Finally, we stress that we have implemented OBAMP, in Java, and we have tested it on the field, to prove its feasibility (see implementation codes in [11]). To the best of our knowledge this is the first time that an overlay multicast protocol is really deployed in practice. We believe that this is very important as it often happens that some protocol procedures or mechanisms may seem deceptively easy to implement. Practice brings about unforeseen problems and difficulties.

II. Overlay approach versus network layer approach

In this section, we briefly discuss pros and cons of the overlay and network layer approaches to implement a MANET multicast protocol. We compare the two approaches in terms of network performance (II.A) and in terms of easy of implementation (II.B).

We believe that, in line of principle, the network layer approach is better or at most equivalent than the overlay one, from a network performance point of view; nevertheless, particular instances of network layer protocols may turn out to perform worse than particular instances of overlay protocols.

On the other side, we argue that implementing an overlay MANET multicast protocol is less complex, quicker to deploy and easier to customize.

As a consequence, the research challenge that we face consists in devising an overlay multicast protocol that performs as better as (or even outperforms) the more promising network layer multicast protocols available nowadays.

A. Network performance aspects

Obviously, the overlay members are a subset of the network nodes. As a consequence, all the operations that are performed by an overlay protocol can be performed as well by a network layer protocol, exploiting the same set of nodes. However, a network layer protocol can exploit also the remaining nodes and thus it can potentially improve the overall performance. In the worst case, the network layer approach can avoid to exploit these added degrees of freedom and stay with the "basic" performance brought about by the set of member nodes.

In addition, an overlay protocol must coexist with an network layer unicast routing protocol. This implies potential inefficiencies: the two protocols need to interact and it may also happen that some functions are executed at both layers (e.g., neighbours discovery), thus increasing the overall overhead.

B. Implementation aspects

The main advantages of an overlay multicast protocol lie in:

- lower computational complexity (as pointed out in [7][9][8]): as a matter of fact, in the overlay case a generic node has to manage only the routing state(s) of the multicast session(s) to which it belongs to; instead, in the network layer case, a generic node has to manage the routing states of all ongoing multicast sessions.
- Fast deployment and time to market: an overlay multicast protocol can be integrated within an application software; this implies that the multicast protocol is distributed together with the application and can reach more easily all the interested users. This is especially important in the actual standardization scenario, in which unicast routing protocols are well established, whereas network layer multicast protocols are at a premature level [20]. The integration of the multicast protocol in the application package allows to avoid making assumption on the availability of underlying multicasting functionality, which is a distinctive advantage, as observed in [21].
- Easy of customization: this may be seen as another advantage of integrating the multicast protocol and the application software; the multicast protocol can be optimized or fine tuned as a function of the requirements of the application itself.

III. Theoretical Background

The aim of this section is twofold: a) to recall graph theory results showing what is the cheapest distribution tree in terms of number of hops on a given topology: it comes out that the best tree is the *minimum spanning tree* for the overlay approach, and the *Steiner tree* for the network layer approach; b) to describe the Borůvka algorithm, which is a classical tool to find the minimum spanning tree.

A. Graph theory background

Let us consider a generic multicast session at time t, over a given MANET, without detailing, for the moment, if it is implemented with an overlay or a network-layer approach. We can define:

- $\{N\}$ as the set of network nodes;
- $\{M\}$ as the set of member nodes of the multicast session, i.e., nodes belonging to the multicast group ($M \subseteq N$);
- $G=(V,E)$ as the connection graph, formed by the set $\{V_i\}$ of vertices and the set $\{E_{ij}\}$ of edges, where:
  - the vertex V_i is associated with the i-th network node able to perform multicast routing;
  - the edge E_{ij} is associated with the connection service provided by the underlying layer between vertex V_i and vertex V_j;
- $c(E_{ij})$ as the cost (or weight) of the edge E_{ij} measured in network hops;
- T as the minimum cost tree, i.e. a sub-graph of G that spans all members M and has the minimum cost, measured as the sum of the cost of the involved edges.

The graph G contains all the possible routes that can be used to setup the multicast session in the considered MANET. As a consequence, T is the *minimum* cost tree for the considered MANET.

Now, the graph G and the tree T may vary, depending on if we consider an overlay or a network-layer approach.

In the *overlay case* it turns out that: 1) only member nodes can perform multicast routing; 2) the connection service is supplied by the underlying TCP or UDP layers, which provide all Eij connections among members; these connections are named overlay links. In terms of graph theory these two features translate in: i) the edges of the graph are overlay links (and thus they may consist of more than one network link); ii) $V=M$; iii) G is the fully meshed connection graph connecting all the vertices; iv) the cost $c(E_{ij})$ is the number of network hops of the overlay link between member i and member j (assuming that the underlying unicast routing protocol minimizes the path length measured in number of hops, in that case the overlay link is also the shortest path between member i and member j); v) T is the sub-graph of G containing all vertices and having the minimum possible cost, namely the overlay *minimum spanning tree*.

In the *network-layer approach* it turns out that: 1) all nodes can perform multicast routing; 2) the connection service is supplied by the underlying Data Link / MAC layers, which provide only connections E_{ij} among adjacent nodes; these connections are named network links. In terms of graph theory these two features translate in: i) the edges of the graph are network links; ii) $V=N$; iii) G is the connection graph connecting only adjacent nodes; iv) $c(E_{ij})=1$; v) T is the sub-graph of G containing only the vertices of M and having the minimum possible cost, namely the *Steiner tree*.

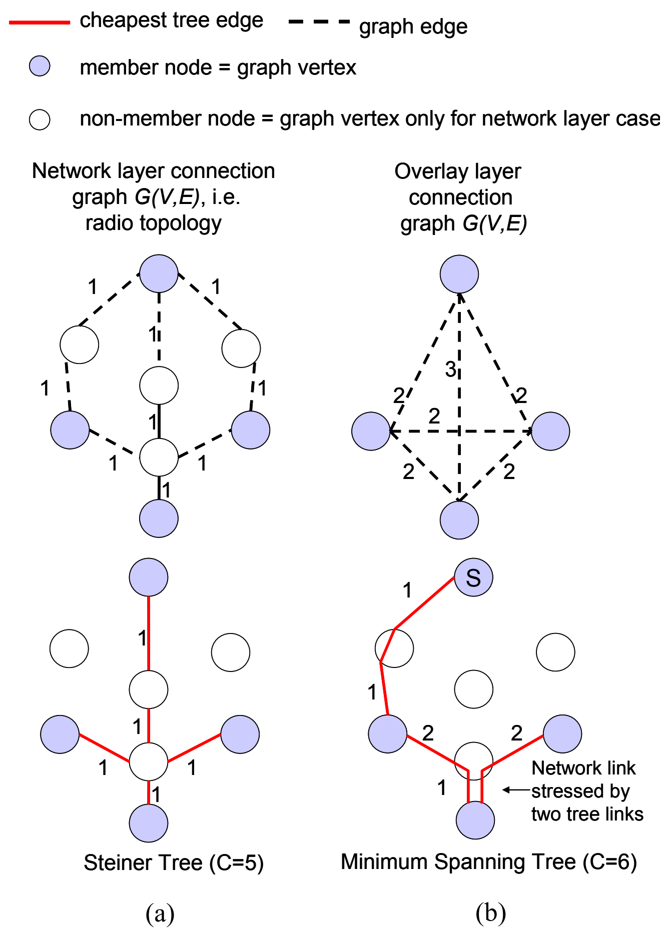

cheapest tree edge
graph edge
member node = graph vertex
non-member node = graph vertex only for network layer case
Network layer connection graph G(V,E), i.e. radio topology
Overlay layer connection graph G(V,E)
1
2
3
S
Network link stressed by two tree links
Steiner Tree (C=5)
Minimum Spanning Tree (C=6)
(a)
(b)

Fig. 1 - Example of connectivity graph and of resulting cheapest tree of a MANET in the two cases of network-layer (a) and overlay approach (b)

Fig. 1 shows an example of the connection graph G and of the corresponding cheapest tree in the network layer (a) and overlay (b) approach. In the network-layer case, the connection graph G equals the set of radio connections and the cheapest tree (a Steiner one) consists of 5 hops. In the overlay case, G is a full mesh among member nodes and the cheapest tree (the minimum spanning tree) consists of 6 hops.

This said, we can conclude the section with an important consideration, already discussed in [6][7][9]. The overlay approach can not exploit non-member nodes for multicast routing; this implies that it is less efficient than the network-layer approach because the same network link may be stressed by more than one tree link (see Fig. 1), i.e. by more than one transmission of the same user data. In particular, it has been found that the ratio between the cost of the minimum spanning tree and that of the Steiner tree is limited to 0.9, in practical ad-hoc network scenarios [6]. This is an example of the performance issues discussed in II, about pros and cons of performing the multicast with an overlay or a network layer approach.

B. *Borůvka algorithm*

The Borůvka algorithm (1926) [5] finds the minimum spanning tree over a given graph. Alternative methods serving the same purpose are the Kruskal algorithm (1956) [17] and the PRIM algorithm (1957) [18]. We selected the Borůvka algorithm because it lends itself more easily to a distributed implementation. It works as follows:

```
1. make a list $L$ of {W} trees, where each
   tree is composed of a single vertex
2. while $L$ has more than one tree
      for each tree in $L$, find the smallest
      edge connecting the tree to another
      disjoined tree, thus forming a new tree
3. end
```

Let us define *n-level edges* the set of edges of the minimum spanning tree built by the Borůvka algorithm at the *n*-th iteration (in the while loop). It is easy to see that the first-level edges are the edges that connect nearest vertices. This definition will be useful in the sequel.

IV. OBAMP

A. *Main protocol features*

The goal of OBAMP is to limit the network traffic, both user data and signalling information, so as to achieve a high delivery ratio and a low latency, and to maintain these properties when the group size increases. The delivery ratio is the ratio between the number of non-duplicated delivered data bytes and the number of bytes supposed to be received by the multicast sinks. The latency is the time elapsing between the emission of a packet and its reception by the receiving multicast sink.

To this end, OBAMP: i) creates a cheap distribution tree; ii) exploits radio broadcasting; iii) limits the protocol overhead and the “induced” network layer-signalling.

OBAMP is a mesh-first overlay multicast protocol: first it builds an overlay network spanning all members (i.e., a mesh); then it builds the distribution tree by selecting a subset of non-cyclic overlay links belonging to the mesh. Fig. 2 reports an example of mesh and corresponding distribution tree in a 10-members case. Red lines are overlay links of the tree (and of the mesh as well, of course); dashed lines are overlay links of the mesh and not of the tree. Non-member nodes are not drawn, although they participate to the network layer routing.

Both the mesh and the distribution tree are periodically updated, to follow the dynamics of the network links. The procedures that build the mesh and the tree are named *mesh-create* and *tree-create*, respectively.

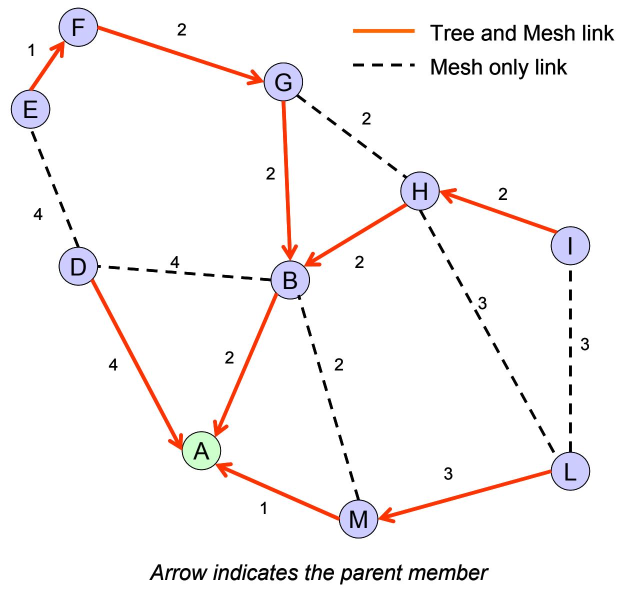


Fig. 2 - A mesh with a corresponding tree and related hop distance/cost (non-member nodes are not drawn)

1) Usefulness of the mesh-first approach

The mesh-first approach was originally proposed by AMRoute [1] and is alternative to the tree-first approach used, for instance, by ALMA [7], which builds directly the multicast tree, without previously forming a mesh.

The mesh-first approach constructs a structure more resilient to overlay link failures. As a matter of fact, in a tree-first approach the loss of an overlay link (e.g., due to a member hardware failure) would force a time-consuming process of neighbour discovery to select the “recovery” overlay link. During the discovery process, the tree is partitioned and loss phenomena can occur. On the contrary, in the mesh-first approach, it is often possible to quickly select a recovery overlay link among the mesh links, without the need of discovering available overlay links.

2) Performance issues driving the design process

This sub-section discusses some performance issues that explain the rationale that lies behind the definition of OBAMP.

Let us define the *efficiency* $\rho(t)$ of a generic distribution tree at time t as the ratio between the cost of the corresponding minimum spanning tree $C_{MST}(t)$ and the cost of the distribution tree $C_{tree}(t)$ itself, i.e. $\rho(t) = C_{MST}(t)/C_{tree}(t)$. As discussed in section III, the best choice for an overlay protocol is the minimum spanning tree and hence $\rho(t) \leq 1$. It is helpful to express $\rho(t)$ as follow:

$$\begin{aligned}
&\rho(t) = C_{MST}(t)/C_{tree}(t) = \rho_{mesh}(t)\cdot\rho_{tom}(t)\\
&\rho_{mesh}(t) = C_{MST}(t)/C_{meshMST}(t)\\
&\rho_{tom}(t) = C_{meshMST}(t)/C_{tree}(t)
\end{aligned} \tag{1}$$

where:

- $\rho_{mesh}(t)$ is the *mesh efficiency*, defined as the ratio between the cost $C_{MST}(t)$ of the minimum spanning tree and the cost $C_{meshMST}(t)$ of the minimum spanning tree of the given mesh at time t. This parameter measures the ability of the *mesh-create* procedure to include in the mesh overlay links belonging to the minimum spanning tree; as a matter of fact, if all minimum spanning tree links are contained in the mesh then $C_{meshMST}(t) = C_{MST}(t)$ and $\rho_{mesh}(t)=1$. On the other side, the more the mesh is “efficient”, in the sense just explained, the more are the opportunities for the *tree-create* procedure to build a cheap tree.
- $\rho_{tom}(t)$ is the *tree-over-the-mesh efficiency* defined as the ratio between $C_{meshMST}(t)$. and the cost $C_{tree}(t)$. This parameter is a measure of the effectiveness of the *tree-create* procedure; in the best case, when $\rho_{tom}=1$, the distribution tree will be the minimum spanning tree of the mesh.

The overall meaning of Eq. (1) is that the creation of a tree is a two-steps process and thus its efficiency is the product of the efficiency of the component steps.

Fig. 3 reports a simulation run of the cost of a tree obtained with OBAMP as a function of the time, $C_{tree}(t)$; also shown are $C_{meshMST}(t)$ and $C_{MST}(t)$.

$C_{MST}(t)$ is the minimum possible cost. If $C_{meshMST}(t)$ is greater than $C_{MST}(t)$, the obtained mesh does not contain the minimum spanning tree and $\rho_{mesh}(t) < 1$. If $C_{tree}(t)$ is greater than $C_{meshMST}(t)$, then the tree-create procedure is not finding the minimum spanning tree of the mesh and $\rho_{ton}(t)<1$.

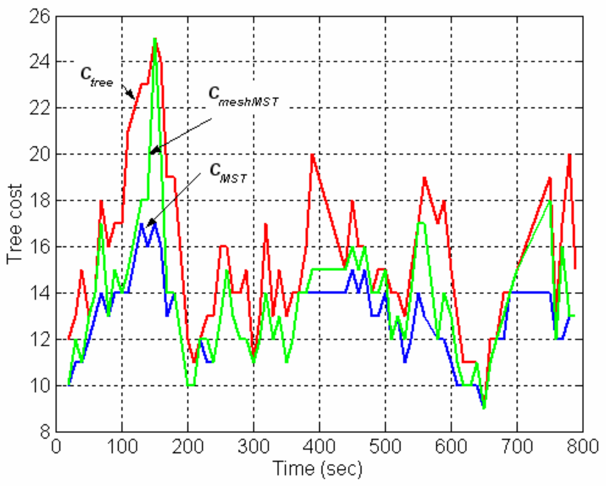


Fig. 3 - Simulation run of C_{tree}, $C_{meshMST}$ and C_{ton} in a multicast group of 10 members.

B. Protocol procedures

This section describes the protocol procedures. The sequence diagrams and messages format can be found in the Appendix I and II, respectively.

User data are distributed over a distribution tree that is created by means of suitable procedures. We start the description of the protocol by describing data distribution and then we introduce the procedures that we need to create the distribution tree.

1) Data-distribution

Let us define as neighbours two members connected by a mesh link. *Data-distribution* is executed in two different ways, depending on if the hop distance between the forwarding member and its neighbour is greater than one or equal to one:

– in the first case, each member forwards in unicast way the received (or generated) data on all overlay links of the distribution tree to which it is connected, with the exclusion of the receiving one.
– in the second case, data are distributed to members belonging to the mesh by means of only one radio broadcasting transmission, with a value of the IP Time To Live (TTL) equal to 1.

The rationale of the second case is that it may happen that more than one receiving member is within the coverage area of a sending member. In that case, it is more efficient to reach such members via radio broadcasting, instead that via unicast transmissions over the distribution tree.

Data duplication may occur because of loops in the mesh, loops in the tree, due to the dynamics of the MANET, and to other reasons. To limit such duplication, we resort to two mechanisms: i) each data message contains a field (named `JustForwardedMemberCode`) that codes the list of the members to which those data have already being sent. Members will not forward a message to other members contained in this list; ii) data duplication is further limited by means of temporal data caches, as done in [7].

Thanks to these mechanisms, only one transmission is sufficient to reach neighbours within the same radio coverage, independently from the tree topology. This feature maintains good performance even when the group size increases.

This said, we can now focus on how to create the mesh and the distribution tree.

2) Mesh-create

From Eq. (1), it is easy to see that the mesh-create procedure should return a mesh that contains the greatest possible number of overlay links belonging to the minimum spanning tree.

From this point of view, it is clear that the *full-mesh* will have $\rho_{mesh} = 1$ since it will surely contain all the overlay links of the minimum spanning tree. On the other side, the maintenance of a mesh link requires the exchange of overlay control messages and related, "induced", signalling at the network layer. The overall amount of signalling increases with the square of the multicast group size, *M*, since a full mesh has $M \cdot (M-1)$ overlay links. As a consequence, maintaining a full mesh is in contrast with the aim of achieving a good scalability performance, and a trade-off is in order.

For this reason, the mesh-create procedure builds a "trade-off mesh" that tries to limit the number of overlay links while at the same time tries to include as many as possible minimum spanning tree overlay links. We will see that the mesh created by our procedure will contain the *overlay links that connect nearest members*, plus other overlay links eventually needed to avoid group partition.

If we translate this in the terminology introduced in section III.B, we can say that the *mesh-create* procedure will surely include in the mesh at least the first-level edges of the minimum spanning tree. On the other side, the inclusion of the other overlay links of the minimum spanning tree is not guaranteed; this implies a small loss of mesh efficiency (e.g., in case of Fig. 3 the resulting mesh efficiency is ρ_{mesh}=0.94 instead of the ideal ρ_{mesh}=1). This inefficiency is the price to be paid to curb the signalling overhead, by limiting the number of mesh links.

The *mesh-create* procedure is made up of three elementary sub-procedures: *hello*, *fast-hello* and *link-pruning*. The *hello* and *fast-hello* sub-procedures periodically establish or refresh mesh links; in other words, their aim is to find the neighbours of each member, and to estimate their hop distance. We need to estimate the hop distance for a number of reasons; for instance, data are distributed differently, depending on the hop distance.

The *link-pruning* sub-procedure manages the removal of a mesh link.

The mesh is created (and maintained) in a distributed way: each member uses a number of parameters and state variables. An important data structure maintained by each member is the *neighbours list*, which contains the status information of the mesh links connected to the member; the members attached to the other end of such mesh links are the neighbours of that member. This status information is stored in a record containing 12 fields. The most important of these fields are: `NEIGHBOUR_IP_ADDRESS`; `NEIGHBOUR_CORE_ADDRESS`; `HOP_DISTANCE`; `EXPIRY_TIME`; `TREE_FLAG` (true if mesh link is a tree link).

a) Hello sub-procedure

The aim of the *hello* sub-procedure is to find the neighbours of each member and to evaluate the related hop distance. It is performed by all members when the nearest member is perceived to be no more than one network hop away or when the *neighbours list* is empty; otherwise the *fast-hello* sub-procedure is carried out.

The *hello* sub-procedure exploits a so-called expanding-ring search, and is executed every `HELLO_PERIOD` seconds. To describe it, let us consider a generic member, *S*. Member *S* sends out a sequence of broadcast HELLO messages, contained in IP datagrams with incremental values of the IP Time To Live (TTL) field[2]. The TTL field is incremented up to `MAX_HELLO_TTL` or until member *S* receives a HELLO REPLY message from another member. The IP TTL value set by

[2] We assume that the underlying network layer routing supports broadcast forwarding also for application layer data.

member S is copied in a specific field of the HELLO message.

When a member, R, receives an HELLO message, it creates (or refreshes) a mesh link toward member S. This means that a *neighbours list* entry is created (or updated) by setting `EXPIRY_TIME` equal to the current time plus 1.5 * `ALLOWED_HELLO_LOSS` * `HELLO_PERIOD` and by setting `HOP_DISTANCE` equal to the IP TTL value contained in the HELLO message. Then, member R sends to S a unicast HELLO REPLY message, piggybacking the originating IP TTL.

At the reception of the HELLOREPLY, member S stops the expanding-ring search and creates (or refreshes) a mesh link toward member R by creating (or updating) the relevant *neighbours list* entry, through the same procedure described above for R.

At the end of the *hello* sub-procedure, member S knows who its neighbours are and has an estimate of how far away each neighbours is. Thus, in principle, we do not necessarily need *fast-hello* sub-procedure. The raison d'être of this latter procedure is to improve performance.

The problem is that the estimate of the hop distance performed by the *hello* sub-procedure may be wrong, due to the fact that members are moving. This has serious implications in the data forwarding phase: if a member believes that its target is one hop away, it will use broadcast with a value of the IP TTL equal to 1 to reach it. If the target is NOT one hop away, the broadcast transmission will not be able to reach the target, causing data loss[3].

To alleviate this problem we have to improve the quality of the estimate of the hop distance. To do so, we could simply execute the *hello* sub-procedure more often, so that moving members are better tracked. However, this would increase the signalling overhead, especially when the hop distance is greater than one. Our solution is to devise a new sub-procedure, to be executed more frequently and only when the nearest member is perceived to be one network hop away.

b) *Fast-hello sub-procedure*

The *fast-hello* sub-procedure is executed only when the nearest member is perceived to be one network hop away. It is repeated with a period equal to `FAST_HELLO_PERIOD`, which is smaller than `HELLO_PERIOD`.

To describe it, let us consider a generic member, S. Member S sends out a FASTHELLO messages contained in a IP datagram with TTL=1 by means of a single broadcast transmission. When a member, R, receives a FASTHELLO message, it creates (or refreshes) a mesh link toward member S. This means that a *neighbours list* entry is created (or updated) by setting `EXPIRY_TIME` equal to the current time plus 1.5 * `ALLOWED_HELLO_LOSS` * `FAST_HELLO_PERIOD` and by setting `HOP_DISTANCE` equal to one.

An interesting comment is that the scope of the HELLO messages in both sub-procedures, *hello* and *fast-hello*, is limited to the hop distance between member S and its nearest member. Therefore, nearest members will be surely connected by mesh links, as previously stated.

Moreover, most of the HELLO messages are transferred with UDP/IP broadcast packets, avoiding the possible production of network layer signalling to perform route-discovery.

c) *Link-pruning sub-procedure*

The *link-pruning* sub-procedure removes outdated mesh links by using a soft-state approach. When the `EXPIRY_TIME` of a given mesh links is reached, then that mesh link is pruned by deleting its entry in the *neighbours list*, unless the mesh link is also a tree link. In the latter case: i) the mesh link is not pruned, to avoid group partition; ii) the related entry in the *neighbours list* is not deleted, but its hop distance is set to `MAX_HELLO_TTL + 1`, which has the meaning of *unknown* distance.

3) *Tree-create procedure*

Once that the mesh is built, OBAMP creates a shared distribution tree over the mesh by using the *tree-create* approach proposed by AMRoute [1]. We selected this approach for its capability of avoiding persistent tree loops (even if it does not avoid *temporary* tree loops).

In addition, to improve the *tree-over-the-mesh efficiency,* ρ_{tom} (see Eq. 1),we add to the AMRoute mechanism a novel feature, named *handling delay*.

The *tree-create* procedure is initiated by a special member, named *core,* which is chosen by means of a suitable procedure, named *core election*, described in the following sub-section 5.

Each member stores the identifier of its *core* (i.e., its IP address) and accept only control packet coming from its *core*.

The *core* sends out TREECREATE #x messages toward its neighbours[4] periodically, with a period equal to `TREE_CREATE_INTERVAL`. Each round of such messages refreshes the tree topology; the value of x identifies the x-th refresh round and is reported in a specific field of the TREECREATE message. Another field of the TREECREATE message indicates if the destination neighbour is or not the current nearest neighbour. TREECREATE messages are sent via broadcasting to neighbours at one hop distance, and via unicasting otherwise.

When a member, R, receives a TREECREATE message from another member, F, it delays the handling of this message for a time equal to `HANDLING_DELAY`. The member R determines the value of `HANDLING_DELAY` as follows:

```
if [(F is the nearest member of R) or (R is the
      nearest member of F)]
then HANDLING_DELAY=0
else HANDLING_DELAY =( dist -1)*$U_h$+r(0,1)* $U_h$ /10;
```

where U_h is the `HANDLING_DELAY` *time unit*; *dist* is the `HOP_DISTANCE` of the mesh link connecting F to R; $r(0,1)$ is

[3] On the contrary, unicast transmissions do not have this problem: unicast exploits network layer procedures that route the data to the destination and thus can find the target member even when the latter is moving.

[4] Two members are said to be neighbours of each other if they are connected by a mesh link.

a uniform random value in the (0,1) interval, used to differentiate the delays of paths with the same hop distance.

The first time that *R* handles a TREECREATE *#x* message, it marks the member from which it received the message as *current parent member*; the parent member of the previous refresh round is marked as *old parent member*. The *current parent member* is the closest upstream vertex of the tree toward the *core*.

If there is not a tree link between *R* and its *current parent member*, then *R* creates it by using a two ways handshake setup procedure (see Appendix I.B).

If *old parent member* is different from *current parent member*, then *R* tears down the tree link toward *old parent member* by means of two ways handshake tear-down procedure.

At this time, *R* can forward the TREECREATE *#x* message toward its neighbours, with the exclusion of the receiving one; TREECREATEs *#x* messages, which show up after that *R* has handled the first TREECREATE *#x* message, are discarded.

a) Three properties of the OBAMP tree

The OBAMP tree is characterized by the following three properties, demonstrated in Appendix IV.

Property A: *the tree created by OBAMP does not have persistent tree loops.*

Property B: *OBAMP builds a distribution tree that contains at least the fist-level edges of the minimum spanning tree.*

Property C: *the links of the OBAMP tree that are not first-level edges of the minimum spanning tree are the links of the macro-mesh that belong to the shortest-path rooted at the core.*

4) Outer-tree-create procedure

Up to now, we have implicitly assumed that the mesh spans all members. In reality, the *mesh-create* procedure discovers only nearest members whose distance is at maximum `MAX_HELLO_TTL` hops. Hence, the *mesh-create* procedure may create partitioned meshes with different *cores* and give rise to different distribution trees, one for each partitioned mesh. To cope with this issue the *outer-tree-create* procedure connects these different meshes and distribution trees by means of a tree link, creating a mesh and related distribution tree that span all members. The procedure works as follows.

Every `OUTER_TREE_CREATE_INTERVAL`, the *core* member of each partitioned mesh floods all the network with an OUTERTREECREATE message. To limit the signalling overhead, we choose for the `OUTER_TREE_CREATE_INTERVAL` a quite large value.

Only mesh cores handle OUTERTREECREATE messages. When *core A* of mesh *A* receives an OUTERTREECREATE from *core B* of mesh *B*, the procedure *core-election* is invoked and two things can happen:

1) *core B* is elected *core* of a new mesh resulting from the joining of mesh *A* and *B*; core *A* stops behaving as *core* and sets up a tree link with *B*, thus attaching "its" mesh to mesh *B*; this implies that also the related trees are now connected.
2) core *B* is not elected *core* of a new mesh resulting from the joining of mesh *A* and *B*; in that case core *A* floods all the network with another OUTERTREECREATE message, without waiting for the `OUTER_TREE_CREATE_INTERVAL`. In other words, since the joining operation failed, we start a new trial immediately to save time.

5) Core-election procedure

The procedure *core-election* is used to uniquely identify the *core* of the mesh and is used: i) when two meshes, each with its own core must be connected; ii) when a *core* leaves the network or when the network gets radio partitioned and a new *core* must be found.

In the first case, the procedure elects as new *core* the member with smallest IP address. This happens during the *outer-tree-create* procedure, as seen above.

In the second case, all members connected by means of a tree link with the *old core* become *cores*. The remaining members will: i) receive TREECREATE messages from these new *cores*; ii) accept only those coming along tree links; iii) elect as new *core* the generator of the accepted TREECREATE message. This is the only exception to the rule according to which members accept only control packet coming from their current *core*.

6) Member-join and member-leave procedures

When a member wants to join a group it simply elects itself *core* of a mesh formed by itself only; then the *outer-tree-create* procedure will take care of the joining of such atomic mesh with the rest of the group.

When a member wants to leave the group, it simply switches itself off. The neighbours connected to that member through tree links will perform the *tree-link-recovery* procedure (see below).

7) Tree-link-recovery procedure

The *tree-link-recovery* procedure has the aim of re-establishing the connectivity of the distribution tree in case of a failure of a member or when a member goes out of radio coverage.

Each member, *H*, monitors the activity of the tree links connected to itself. Each member sends out an ALIVEHELLO unicast message every `ALIVE_HELLO_PERIOD` on all tree links, unless *H* does not need to send other kind of unicast messages in this period.

If a member *H* does not receive any unicast message on a tree link in a period of time equal to `1.5*ALLOWED_ALIVE_HELLO_LOSS*ALIVE_HELLO_PERIOD`, then the tree link, *L*, is considered as faulty; hence, *L* is pruned and the neighbour connected to *L* is eliminated from the *neighbours list*[5]. Moreover, the following actions take place: i) if the eliminated neighbour is the *current parent member* of *H* but it is not the *core* of *H*, then *H* establishes a new tree link with the *core* in order to recover the connectivity of the tree; ii) if the eliminated neighbour is not the *current parent member* of *H,* then *H* does not do anything, because it will be the remote

[5] It is worth noting that similar events occur at the other end of the faulty link; thus, the tree-link-recovery procedure assures a symmetric pruning.

member of the faulty link that will establish a new tree link, in order to recover the connectivity of the tree; iii) finally, if the eliminated neighbour is the *current parent member* of *H* and it is also the *core* of *H*, then it means that the mesh of *H* has lost the *core* and *H* elects itself as *core*.

V. Performance evaluation

In this section, we analyze the performance of OBAMP with ns-2 and compare it with two state-of-the-art protocols, namely ODMRP [2] and ALMA. ODMRP is a network-layer protocol; as a consequence, the comparison between ODMRP and OBAMP must also take into account this aspect: with performance being equal, an overlay protocol should be preferred for its implementation advantages, mentioned in the Introduction.

ALMA [7] is one of the most promising overlay protocol and thus a good test for OBAMP. We compare OBAMP to ALMA and also to a modified version of ALMA, that we denote by ALMA-H.

Although we have implemented OBAMP, in Java, and we have tested it on the field, to prove its feasibility (see implementation codes in [11]), the limited number of available computers did not allow us to evaluate the OBAMP scalability when the group size increases. Thus, we resorted to carefully-designed simulations, taking into due account the recommendations given in [14] on how to produce meaningful simulation results. We start this section by describing the criteria that we followed to assure the so-called "simulation credibility".

A. Simulation credibility criteria

Credibility criteria adopted: i) we publish all the simulator source code and support files (e.g., movement scenarios, TCL files, post-processing routines, etc.) in [11], to assure the reproducibility of our study; ii) we produce the movement traces to be used as inputs of the simulations by using the "random trip model" [10], to assure that the stationary regime is reached; iii) we repeat each simulation ten times with different movement traces and random seeds and we plot the 95% confidence intervals in all figures; iv) we check, by means of post-processing procedures, that all simulation results do not significantly change in the last 100 seconds of simulation, to assure that we have reached the stationary regime within each simulation run.

B. Simulation tool

The simulation tool is based on ns2.29 [13] running on a cygwin32 platform. Unfortunately, none of the benchmarked protocols is available in the all-in-one ns2.29 package. Thus, we had to add the code simulating the three protocols as follows.

As regards OBAMP, we modelled it from scratch. The ODMRP simulation code has been taken from [12]. As regards ALMA, we developed the code from scratch too, implementing two versions of this protocol, both proposed in [7]; in the first one, the metric used for parent selection is the round trip time (and we call it simply ALMA); in the second one (that we call ALMA-H) the metric is the number of hops. According to the ALMA Authors, the latter choice improves the performance of ALMA, in terms of tree efficiency. Finally, we assume that the ALMA signalling packets are 8 bytes long (this value is not given in [7]).

C. Simulation details

Each simulation run lasts 800 s and all members join the group within the first 10 seconds. The simulated network is made up of 50 mobile nodes that move in a region of 1000m x 1000m. Nodes move according to a random way point model with constant pause times of 30 s and with a constant speed. The radio channel is modelled as free-space. The transmission power is regulated so that the radio coverage of each node is 250m. The MAC layer is IEEE 802.11 operating in DCF mode with a constant 2 Mbps bit rate. We use AODV as underlying routing protocol [4], without local repair and with link-layer detection. In Tab. 1 we report the other AODV configuration parameters.

Tab. 1 – AODV main configuration parameters

Parameter	*Value*
ACTIVE_ROUTE_TIMEOUT	10 sec
MY_ROUTE_TIMEOUT	20 sec
NETWORK_DIAMETER	30
RREQ_RETRIES	2
REV_ROUTE_LIFE	1.8 sec
NODE_TRAVERSAL_TIME	0.04 sec
MAX_RREQ_TIMEOUT	3.0 sec
DELAY	0.05 sec
BCAST_ID_SAVE	30 sec
TTL_START	1
TTL_THRESHOLD	7
TTL_INCREMENT	1

Tab. 2 – OBAMP main configuration parameters

Parameter	*Value*
HELLO_PERIOD	5 sec
FAST_HELLO_PERIOD	1 sec
ALIVE_HELLO_PERIOD	1 sec
TREE_CREATE_INTERVAL	5 sec
OUTER_TREE_CREATE_PERIOD	15 sec
MAX_HELLO_TTL	4
U_h (handling delay unit)	250 ms

Tab. 3 – ALMA (ALMA-H) main configuration parameters

Parameter	*Value*
UPDATE_PERIOD	5 sec
THRESHOLD_LEVEL1	30 ms(1 hops)
THRESHOLD_LEVEL2	45 ms(2 hops)
THRESHOLD_LEVEL3	65 ms(2 hops)
THRESHOLD_LEVEL4	80 ms(3 hops)

Tab. 4 – ODMRP main configuration parameters

Parameter	*Value*
JOIN QUERY refresh interval	3 sec
Acknowledge timeout for JOIN Table	25 msec
Maximum JOIN table retransmissions	3

The main configuration parameters of the benchmarked multicast protocols are reported in Tab. 2, Tab. 3 and Tab. 4 for OBAMP, ALMA and ODMRP, respectively.

As regards the traffic model, we consider a *multi-source* scenario where each member sends out data packets of 256 bytes, with an inter-packet interval such that the overall data traffic is equal to 16 kbps. This allow us to evaluate the impact of the increase in the number of sources while the offered traffic remains constant. Our multi-source scenario is more realistic for ad hoc networks, since it models many-to-many communications (e.g., push-to-talk [19]), which are widely believed to be among the most likely sources of load in a MANET.

D. Simulation results

In the following we compare the four protocols at hand in two set of figures. In the first set, we assume a constant speed of 10 m/s and we vary the group size; in the second set we assume a group size equal to 20 members and we vary the node speed. The 95% confidence intervals are always plotted, when they are not visible it means that they are smaller than the curve markers.

The first figure, Fig. 4, reports a parameter commonly named "byte sent per byte delivered" (*BSBD*) as a function of the group size. The *BSBD* is the ratio between the overall number of bytes sent by the network nodes (on the IP-MAC interface) and the number of data bytes delivered to the multicast sinks at the application layer (excluding duplicate packets).

In Fig. 5 we plot the delivery ratio, i.e. the ratio between the number of non-duplicated delivered data bytes and the number of bytes supposed to be received by the multicast sinks, as a function of the group size.

In Fig. 6 we plot the average data latency, i.e., the time elapsing between the emission of a packet and its reception by the receiving multicast sink, as a function of the group size.

In Fig. 7 we plot the number of control bytes sent by all network nodes, as a function of the group size. In the case of overlay protocols (OBAMP, ALMA and ALMA-H) we also distinguish between network layer and overlay signalling.

In Fig. 8 we plot the average tree efficiency vs. group size. We recall that the tree efficiency $\rho(t)$ at time t is the ratio between the cost of the corresponding minimum spanning tree $C_{MST}(t)$ and the cost of the distribution tree $C_{tree}(t)$ itself. The average tree efficiency is the average of $\rho(t)$ over time.

Fig. 9, Fig. 10, Fig. 11 report the following performance indicators versus the node speed: byte sent per byte delivered, delivery ratio and average latency. The performance of ALMA is not reported; in fact, for the considered value of group size (equal to 20 members), the performance of ALMA is considerably worse than the other protocols and the related curves would be so distant from the ones of the other protocols to make the figures not easily readable.

In the following we comment all these figures by comparing our solution first to ODMRP and then to ALMA.

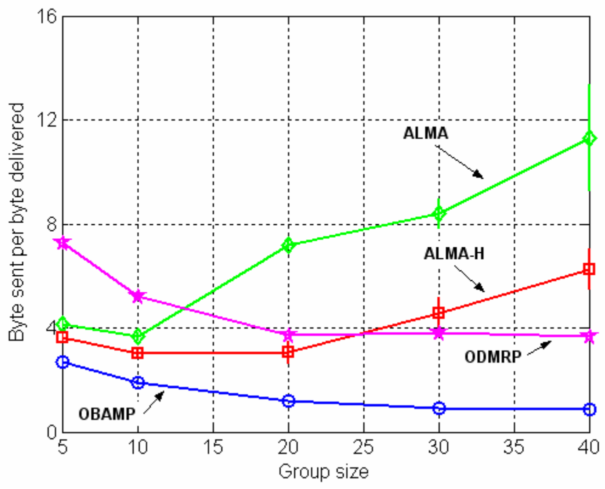


Fig. 4 - Bytes sent per bytes delivered vs. group size

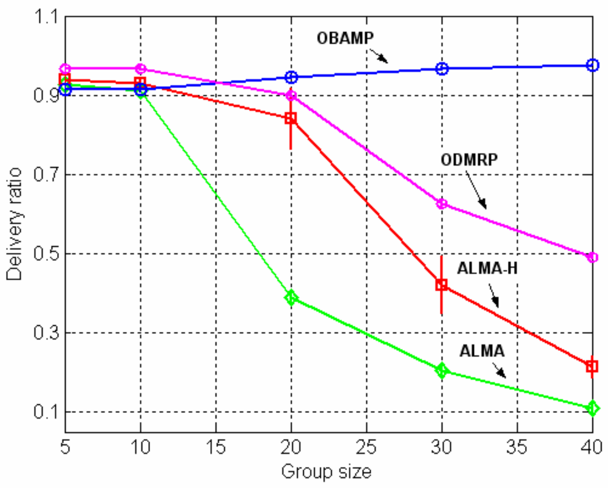


Fig. 5 - Delivery ratio vs. group size

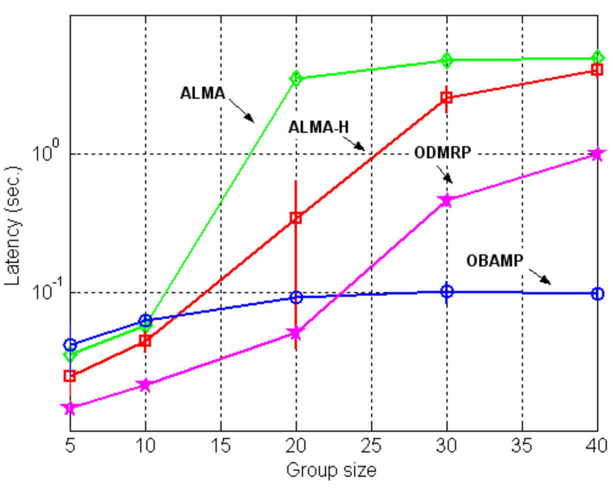


Fig. 6 - Average data latency vs. group size

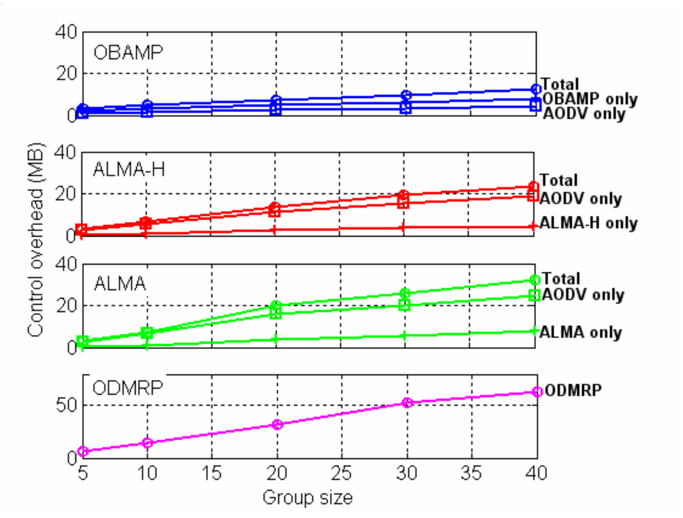


Fig. 7 - Number and type of control mega bytes transmitted vs. group size

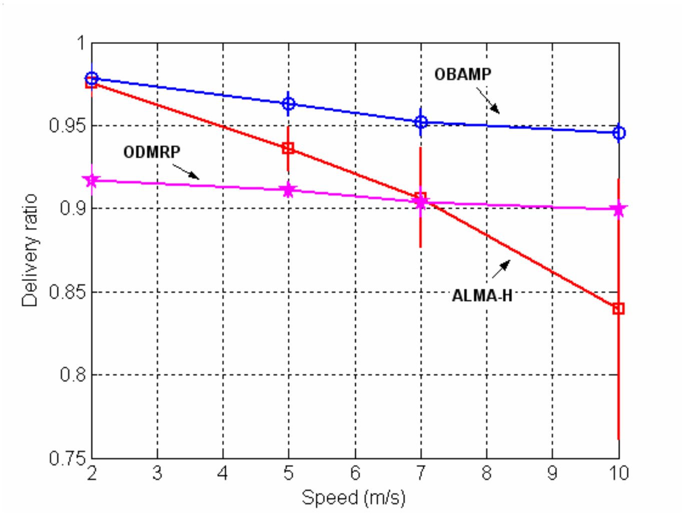


Fig. 10 - Delivery ratio vs. node speed

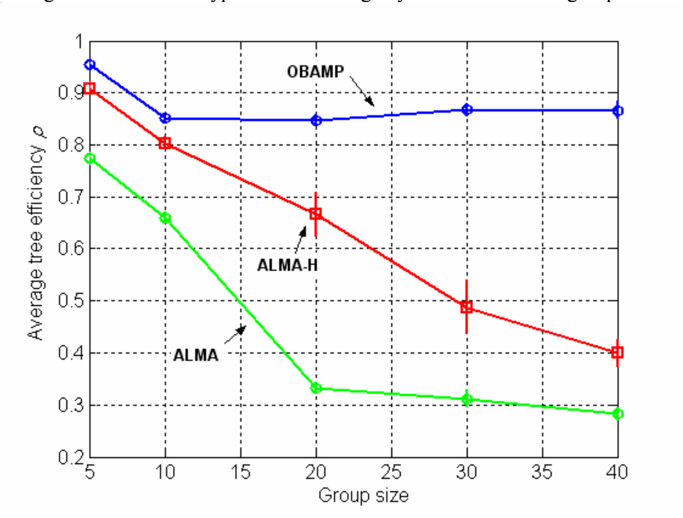


Fig. 8 - Average tree efficiency vs. group size

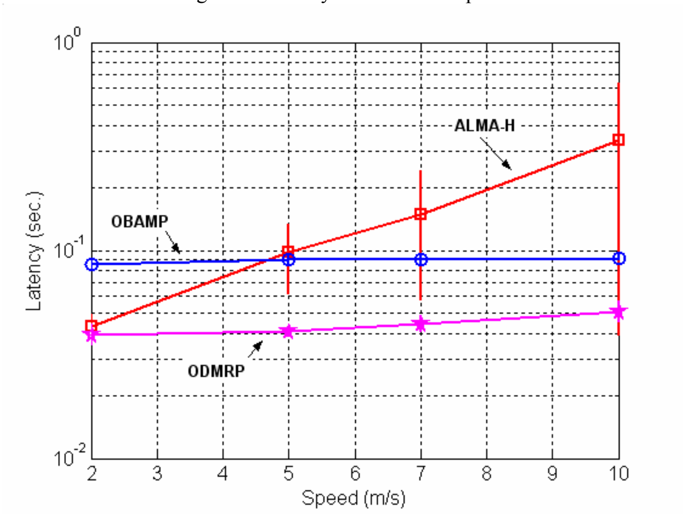


Fig. 11 - Average data latency vs. node speed

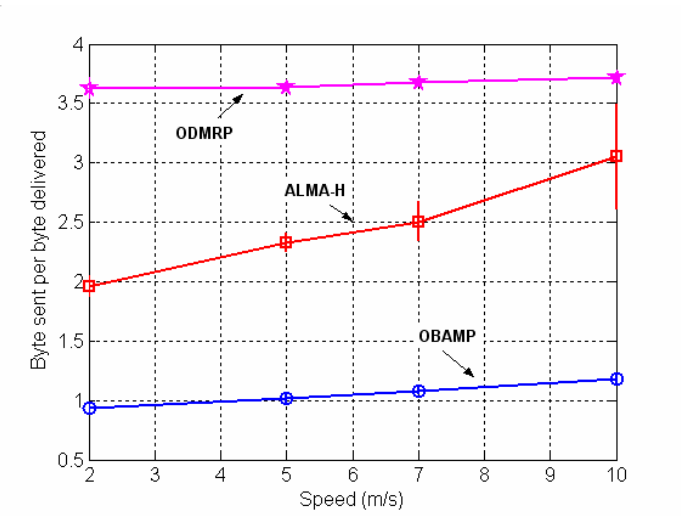


Fig. 9 - Bytes sent per bytes delivered vs. node speed

1) OBAMP vs. ODMRP

We start our comparison by looking at the performance as a function of the group size.

As we can see from Fig. 4, the *BSBD* decreases as the group size increases for both OBAMP and ODMRP. This is due to the ability of these protocols to exploit radio broadcasting for data distribution: when the group size increases, more members can be reached by the same broadcast transmission and the *BSBD* decreases. As the group size keeps increasing, the effect of control traffic becomes more noticeable and the curves flatten. The important result of Fig. 4 is that OBAMP limits the network traffic much better than ODMRP. This advantage is confirmed by Fig. 7 which shows that the number of control bytes of OBAMP is significantly smaller than that of ODMRP.

As regards OBAMP's user-perceived performance, when the group size increases, Fig. 5 and Fig. 6 show that the delivery ratio and the latency increase only moderately, demonstrating the scalability performance of the proposed protocol. The first (positive) effect is due to the natural data redundancy provided by broadcasting, which is more and more apparent as the group size increases. The second (negative) effect is due to the increase of the control traffic which implies a higher latency.

Fig. 5 shows that the OBAMP's delivery ratio remains higher than 90% for all values of the considered group sizes while ODMRP's delivery ratio suffers a heavy decrease for group sizes greater than 20 nodes, due to its significant network load. Correspondingly, Fig. 6 shows that a similar phenomenon occurs for the data latency.

Let us now analyze what happens when we vary the node speed. The Fig. 9 and Fig. 10 show that OBAMP performs better than ODMRP as regards the byte-sent-per-byte parameter and the delivery ratio. On the contrary, Fig. 11 shows that OBAMP performs worse than ODMRP in terms of latency, even if the difference is at most 50 ms.

Overlooking the specific numerical values, we observe that both OBAMP and ODMRP have a weak dependence from the node speed; this behaviour is an evidence of their scaling properties as a function of the node speed.

In all fairness, we must observe that the multi-source traffic scenario has been selected because it is of interest for a MANET but on the other side turns out to be the worst possible traffic load for ODMRP. As a matter of fact, in the Appendix III, we show that in a single-source traffic scenario ODMRP and OBAMP have similar (and good) performance. This happens because in the single-source case ODMRP has a smaller amount of overhead, with respect to the multi-source case.

2) *OBAMP vs. ALMA and ALMA-H*

Also in this case, we begin comparing the protocols by varying the group size.

ALMA does not implement radio broadcasting; as a consequence, when the group size increases, the number of transmissions must increase and so does the network load, the latency and the loss phenomena. In addition, our results show that ALMA-H provide better performance than ALMA as anticipated by the Authors of [7]. As a consequence, we focus the following comparison on ALMA-H vs. OBAMP.

In the case of ALMA-H we see that the *BSBD* rapidly increases (Fig. 4), the delivery ratio suffers a heavy decrease (Fig. 5) and the latency increases to undesirable values (Fig. 6).

In Fig. 7 we can see how OBAMP and ALMA-H behave versus the amount of control traffic. We notice that the amount of overlay control signalling of the two protocols (curves "OBAMP only" and "ALMA only") is comparable. Regarding the amount of network signalling induced at the network layer (curve "AODV only"), OBAMP succeeds in limiting this kind of overhead whereas ALMA-H does not. Moreover, the induced overhead is a major fraction of the overall control traffic of ALMA-H. This fact demonstrates the need of taking into due account this source of overhead, when devising an overlay protocol. If we do not take care of this issue, we would risk that an overlay protocol perfectly working in ideal routing conditions, drastically collapses in real settings.

In Fig. 8 we compare the average tree efficiency of OBAMP and ALMA and we find that OBAMP possesses good scalability properties, as far as the tree create procedures are concerned, and good overall efficiency values.

We now compare the protocols as a function of the node speed. The Fig. 9, Fig. 10 and Fig. 11 show that OBAMP performs better than ALMA-H; with the exception of the average latency values in the speed range 2 m/s ÷ 5 m/s. Furthermore, we note that, when the node speed increases, the performance gap between OBAMP and ALMA-H increases as well. This is an evidence that OBAMP scales better than ALMA-H versus the node speed.

As regards the difference between a single-source scenario and a multi-source one, in the Appendix III we show that no significant difference can be observed between the two traffic models. The reason is that the considered overlay protocols form a shared overlay tree by means of mechanisms that do not depend on where the data source are. As a consequence, keeping constant the offered traffic, the performance are quite independent on the sources location.

VI. Conclusion

Overlay multicasting is a valuable approach to support many-to-many communications in MANETs. A critical figure of merit for a MANET scenario is the ability of the multicast protocol to scale with the group size and to work satisfactorily for a wide range of values of the node speed. To obtain these results it is important, in our opinion, to design the protocol with the follow three guidelines in mind: i) build a distribution tree that approximates the minimum spanning tree as much as possible; ii) design the protocol so as to limit not only the overlay signalling but also the induced network layer signalling; iii) exploit radio broadcasting.

We did follow these guidelines and we ended up with a protocol that exhibits better scalability performance in a many-to-many communications scenario than two state-of-the-art protocols such as ALMA and ODMRP.

A last consideration is that the OBAMP protocol has the ability to account for asymmetric links. This requires only to add a suitable field in the Hello and FastHello messages. This feature is not presented in this paper for space limitations, but is implemented in the test-bed described in [11].

Acknowledgment

This work has been developed by the University of Rome "Tor Vergata" and RadioLabs within the project Virtual Immersive COMmunications (VICOM) founded by the Italian Ministry of Instruction University and Research (MIUR).

The authors would like to thank the Ph.D. Claudio Loreti of Univeristy of Rome "Tor Vergata" for ns-2 simulation code and protocol design; in addition, Remo Pomposini of RadioLabs for the Java implementation and test-bed.

References

[1] J. Xie, et al., “AMRoute: Ad Hoc Multicast Routing Protocol,”*ACM/Baltzer Mobile Networks and Applications*, vol. 7 , no. 6, 2002, pp. 429 – 439.

[2] Sung Ju Lee, William Su, and Mario Gerla, "On-demand multicast routing protocol in multihop wireless mobile networks," *ACM/Baltzer Mobile Networks and Applications*, vol. 7, no. 6, 2002, pp. 441-453.

[3] E. M. Royer and C. E. Perkins, "Multicast Ad hoc On-Demand Distance Vector (MAODV) Routing, " In Proc. of the Second *IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[4] C. Perkins, E. Royer and S. Das. “Ad Hoc On Demand Distance Vector (AODV) Routing,” *IETF* RFC 3561.

[5] J. Nesetril, E. Milkov and H. Nesetrilov, “Otakar Boruvka on minimum spanning tree problem: translation of both the 1926 papers, comments, history”, *Discrete Math.*, vol. 233, 2001, pp. 3-36.

[6] A. Detti, C. Loreti, P. Loreti, “Effectiveness of Overlay Multicasting in Mobile Ad-Hoc Networks, ” *IEEE International Conference on Communications*, vol.7, 20-24 June 2004, pp. 3891 - 3895

[7] Min Ge, Srikanth V. Krishnamurthy and Michalis Faloutsos, “Application versus network layer multicasting in ad hoc networks: the ALMA routing protocol,” *Elsevier Ad Hoc Networks Journal*, vol. 4, no. 2, pp. 283-300, 2006.

[8] Ki-Il Kim; Sang-Ha Kim, "A novel overlay multicast protocol in mobile ad hoc networks: design and evaluation," *Vehicular Technology, IEEE Transactions on* , vol.54, no.6, pp. 2094- 2101, Nov. 2005

[9] C. Gui and P. Mohapatra, “Efficient Overlay Multicast for Mobile Ad Hoc Networks,” *Proc. 2003 IEEE Wireless Comm. and Networking Conf.*, vol.2, pp. 1118-1123.

[10] Le Boudec, J.-Y.; Vojnovic, M., "Perfect simulation and stationarity of a class of mobility models," *Proc. of IEEE INFOCOM 2005*, vol.4, 13-17 March 2005, pp. 2743- 2754.

[11] www.radiolabs.it/obamp

[12] “Wireless Multicast Extensions for ns-2.1b8” at http://www.monarch.cs.cmu.edu/multicast_extensions.html .

[13] “The Network Simulator - ns-2” at http://www.isi.edu/nsnam/ns/ .

[14] S. Kurkowski, T. Camp , M. Colagrosso, “MANET simulation studies: the incredibles,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 4, pp. 50-61, Oct. 2005,

[15] S. Lee, W. Su, J. Hsu, M. Gerla, R. Bagrodia, “A Performance Comparison Study of Ad Hoc Wireless Multicast Protocols,”, *Proc. IEEE INFCOM 2000*, vol. 2, March 2000, pp. 565 – 574

[16] A. Detti, C. Loreti, R. Pomposini, “Overlay Borůvka based Ad Hoc Multicast Protocol – Demonstration”, *Proc. IFIP Med-Hoc-Net 2006 – demo session*, 14-17 June 2006, Lipari (Italy)

[17] J. B. Kruskal, “On the shortest spanning subtree and the traveling salesman problem”, *Proc. of the American Mathematical Society*, no. 7, pp. 48–50, 1956

[18] R. C. Prim, “Shortest connection networks and some generalisations”, *Bell System Technical Journal*, no. 36, pp. 1389–1401, 1957

[19] L.A. DaSilva, G.E. Morgan, C.W. Bostian, D.G. Sweeney, S.F. Midkiff, J.H. Reed, C. Thompson, “The resurgence of push-to-talk technologies”, *IEEE Communication Magazine*, vol. 54, no. 6, pp. 48-55, January 2006.

[20] Internet-Drafts Database Interface, MANET Working Group at https://datatracker.ietf.org/public/idindex.cgi?command=show_wg_id&id =1132

[21] H. Eriksson, “MBone: The Multicast Backbone,” *Communications of the ACM*, vol. 37, no.8, 1994, pp. 54–60.

APPENDIX I
ILLUSTRATIVE SEQUENCE DIAGRAMS OF OBAMP PROCEDURES

A. *Data-distribution procedure*

Fig. 12 shows the *data-distribution* procedure in an example network. In this example the procedure is carried out by member *C* and the data source is member *F*.

At the reception of a data message from *F*, member *C* performs two types of forwarding: i) toward the neighbours *E* and *G*, by means of unicast UDP/IP packets: in fact these members have a distance from *C* greater than one network hop and they are connected by a tree link with *C*; ii) toward the neighbours *A, B* and *D*, by means of a single broadcast UDP/IP packet with IP TTL=1, as these members are at one hop distance from *C* and they are connected by a mesh link with *C*.

We notice that the `JustForwardedMemberCode` (`JFMC`) field of the data packet header transmitted by *F* toward *C* includes only the members *F* and *C*. When *C* subsequently forwards this data, it increases the `JFMC` scope by including also members *A,B,D,E,G*. From now on, no more transmission of this data will be allowed, since all members are included in the `JFMC` field.

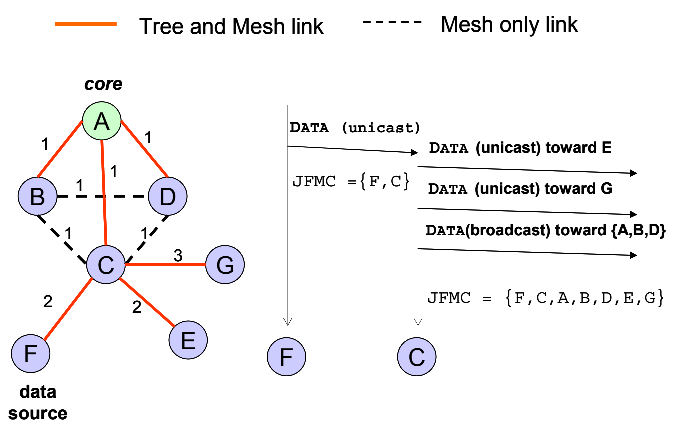


Fig. 12 - The *data-distribution* procedure performed by *C* for a data message coming from *F*

B. *Hello sub-procedure*

Fig. 13 shows the *hello* sub-procedure in an example network. The sub-procedure is performed by member *A*. The TTL values reported in the figure are the content of the `TTL` field of the HELLO messages, i.e., the initial TTL value of the IP packet that contains the overlay message.

At the beginning of the *hello* sub-procedure, member *A* transmits an HELLO message within an UDP/IP broadcast packet with IP TTL = 1. This HELLO message is received only by an intermediate node (denoted by a white circle) that discards the packet, as the node does not belong to the multicast group.

After a short timeout, member *A* re-transmits the HELLO message and increases the IP TTL to 2 hops. This time, the intermediate node re-broadcasts the HELLO message, which is received by member *B*. Member *B*, in turn, creates (or refreshes) the mesh link *B-A*, by inserting *A* in its neighbour list; moreover, member *B* sends back the HELLOREPLY message in a unicast way.

At the reception of HELLOREPLY, member *A* creates (or refreshes) the mesh link *A-B* and the current round of the *hello* sub-procedure ends.

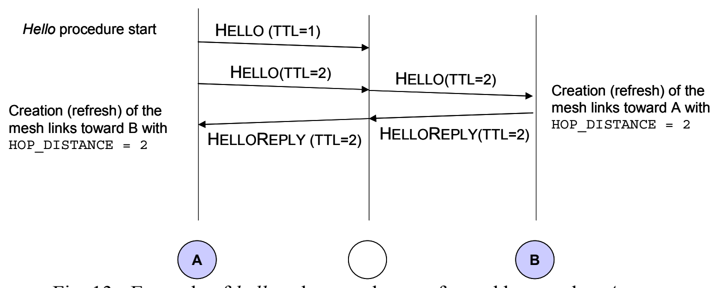

Hello procedure start
HELLO (TTL=1)
HELLO(TTL=2)
HELLO(TTL=2)
Creation (refresh) of the mesh links toward B with HOP_DISTANCE = 2
HELLOREPLY (TTL=2)
HELLOREPLY(TTL=2)
Creation (refresh) of the mesh links toward A with HOP_DISTANCE = 2
A
B

Fig. 13 - Example of *hello* sub-procedure performed by member *A* .

C. *Fast-hello sub-procedure*

Fig. 14 reports the exchange of FASTHELLO messages between member *B* and *C*, which are one hop away from each other. When *C* receives a FASTHELLO message from *B*, it creates or refreshes the mesh links toward *B*, and vice-versa.

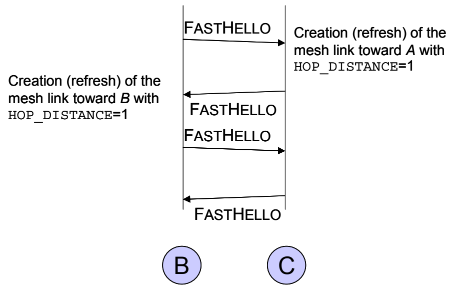

FASTHELLO
Creation (refresh) of the mesh link toward *A* with HOP_DISTANCE=1
Creation (refresh) of the mesh link toward *B* with HOP_DISTANCE=1
FASTHELLO
FASTHELLO
FASTHELLO
B
C

Fig. 14 - FASTHELLO messages exchanged between members *B* and *C*.

D. *Tree-create procedure*

Fig. 15 shows the *tree-create* procedure performed in an example network, with a distribution tree having a cost of 4 hops and formed by two tree links: *A-B* and *A-C* (Fig. 15a). Member *B* is the *A*'s nearest, while the *B*'s nearest is *C* and vice-versa.

At the procedure starts, the *core* *A* sends out two TREECREATE #x messages toward its mesh neighbours: *B* and *C*.

At the reception of the TREECREATE #x message (Fig. 15b): i) *B* applies an *handling delay* equal to zero seconds, since *B* is the nearest of *A*; ii) *C* applies an *handling delay* equals to 0.25*1 + 0.95*0.25/10 seconds, since the hop distance *A-C* is equals to 2 hops, U_h is equals to 0.25 and assuming that `r(0,1)` draws the random number 0.95.

Consequently, *B* immediately handles the TREECREATE #x message coming from *A* (Fig. 15c) and confirms the overlay link toward *A* as a tree link, since this TREECREATE message is the first handled one during the refresh round #x. In other words, *A* becomes *B*'s parent member and *B* is a descendant of *A*. Because the overlay link *A-B* is already set as tree link, no tree link switching occurs and *B* forwards the TREECREATE #x message to *C*.

When *C* receives this TREECREATE #x message from *B* (Fig. 15d), it applies an handling delay equals to zero seconds, as *B* is its nearest member. Consequently, *C* immediately handles such control message. This TREECREATE message is the first one handled during the round #x, therefore *C* setups a tree link toward *B* and tears down the tree link toward *A* (Fig. 15e).

The *tree-link-setup* procedure is the following (Fig. 15f): *C* sends a TREECREATEACK message toward *B* and sets the *C-B* overlay link as tree link. At the reception of the TREECREATEACK message, *B* sends back a TREECREATECONF message and sets the *B-C* overlay link as tree link. When *C* receives the TREECREATECONF message, the *tree-link-setup* procedure ends and the tree link tear down procedure starts.

The *tree-link-tear-down* procedure is the following (Fig. 15f): *C* sends a TREECREATENACK message toward *A*. At the reception of the TREECREATENACK message, *A* sends back a TREECREATENACKCONF message and sets the *A-C* overlay link as mesh link. When *C* receives the TREECREATENACKCONF message, it sets the *C-A* overlay link as mesh link and the *tree-link-tear-down* procedure ends[6].

To cope with control packets loss, the setup and tear down procedures adopt a retransmission policy based on timeouts.

As a final comment on Fig. 15, we observe that the handling delay mechanism has changed the distribution tree from a sub-optimal tree, with a cost of 4 hops (Fig. 15a), to a more efficient tree, formed by 3 hops (Fig. 15e). Without the handling delay mechanism, the tree would have remained that of Fig. 15a.

E. Outer-tree-create procedure

Fig. 16a shows two disjoined meshes and inner trees, whose *cores* are members *A* and *D*, respectively.

At a given time, the *core A* performs the *outer-tree-create* procedure by flooding the network with the OUTERTREECREATE message.

At the reception of this message, the *core D* executes the *core-election* procedure, which returns as *winner core* core *A*. Therefore, *D* stops to behave as *core*, sets *A* as its *core* and establishes a tree link with *A* by means of the tree link setup procedure. The two disjoined meshes and inner trees are now connected.

Let us notice that, at the end of the tree link setup procedure, the descendants of *D*, i.e., members *E* and *F*, still have as their *core* member *D*. This *core* ambiguity will be solved with the next *tree-create* procedure; the TREECREATE message sent by *core A* will pass through the tree links *D-E* and *D-F* and the receiving members *E* and *F* will switch the *core* from *D* to *A* (see IV.B.5).

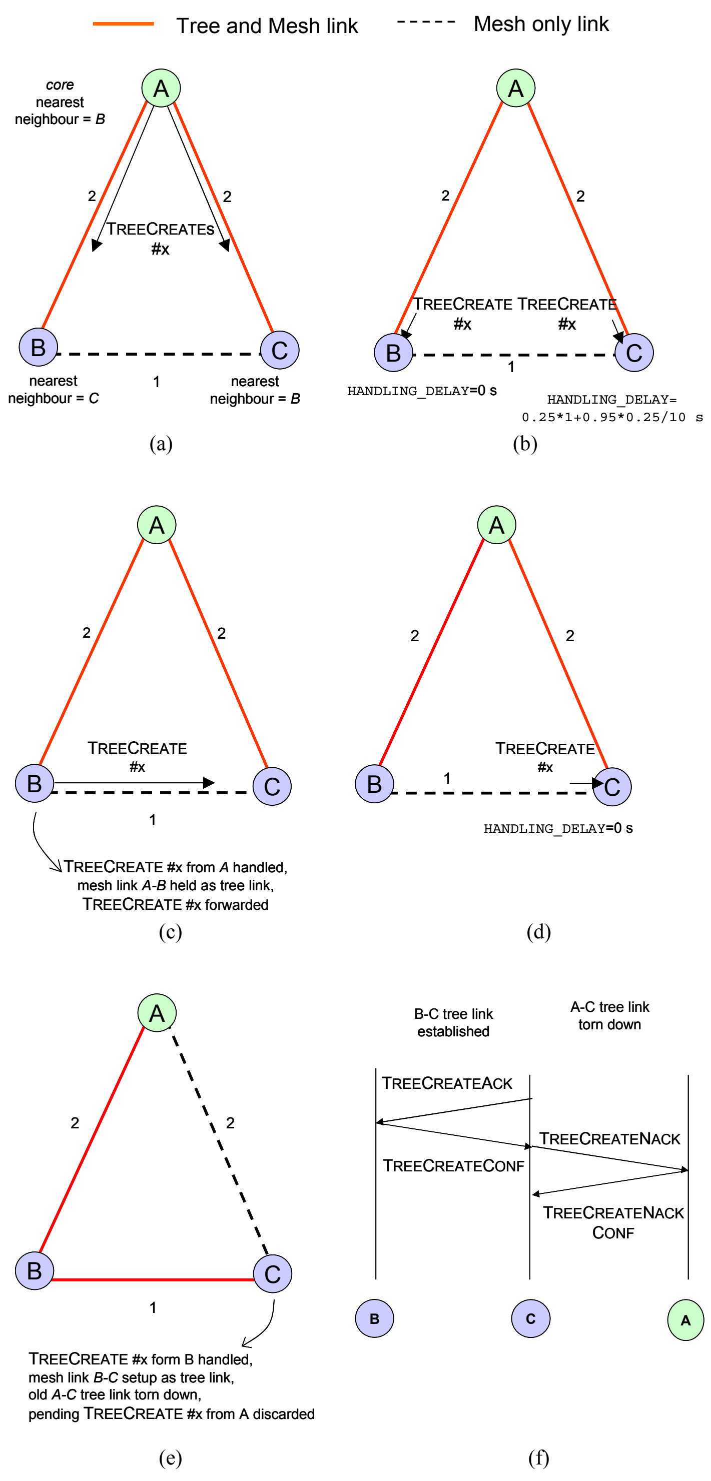


Fig. 15 - Steps of the *tree-create* procedure in an example network (a,b,c,d,e) and exchange of control packets for the *tree-link-setup* and *tree-link-tear-down* procedures (f); non-member nodes are not drawn.

[6] We point out that a temporary loop in the tree occurs since the start of the tree create setup procedure and till the end of tree link tear down procedure.

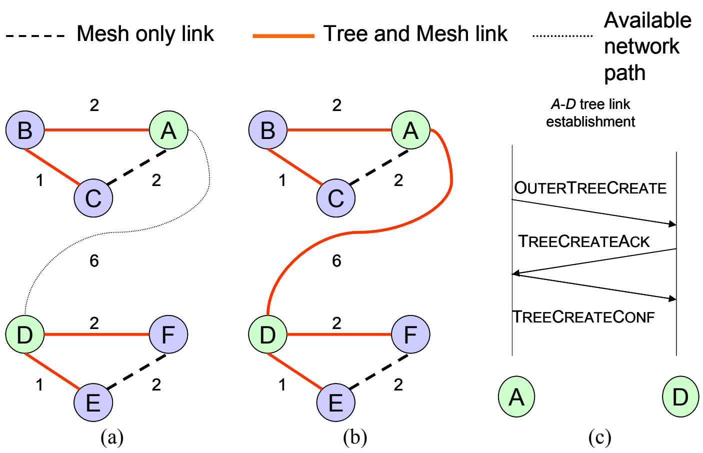


Fig. 16 - The *outer-tree-create* procedure in an example network (a,b) and the related exchange of control packets (c).

F. *Tree-link-recovery procedure*

We report the behaviour of the tree-link-recovery procedure in two example cases. In the first one, the procedure faces a radio partition; in the second one, a hardware failure.

Fig. 17a shows a working network configuration, which in Fig. 17b suffers of a radio partition that impedes the connection between member *D* and core *A*. In this situation, no unicast message will be exchanged on this tree link and the procedure detects the overlay link failure. Then, *core A* purges *D* from its *neighbour list* (being *D* a descendant of *A*), and implicitly tears down the *A*-*D* tree link. At the other end of the link, *D* elects itself as *core* (being *A* both the *core* and the *parent member* of *D*), so forming a disjoined mesh and inner tree. When, eventually, the radio propagation will allow to reconnect the two meshes, then the *outer-tree-create* procedure will perform this task, as shown in Fig. 17c (see section IV.B.4).

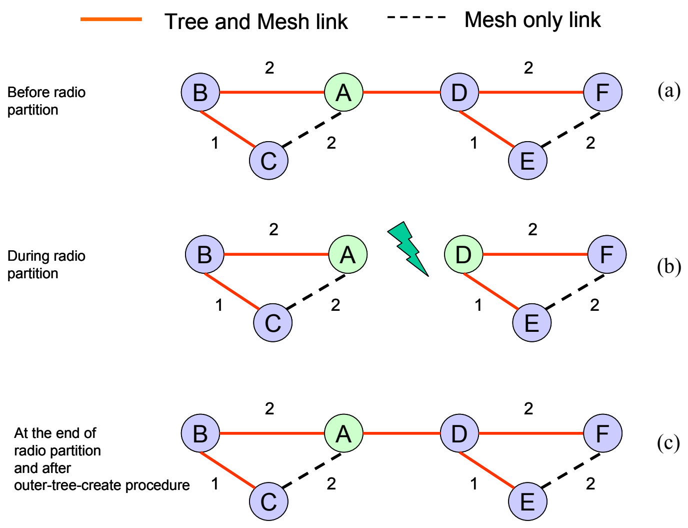
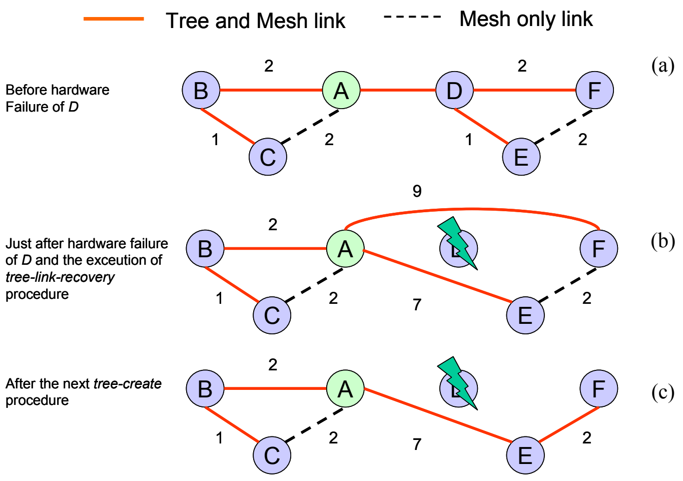


Fig. 17 - *Tree-link-recovery* procedure facing a radio partition (non-member nodes are not drawn)

Fig. 18a shows a working network configuration, which in Fig. 18b suffers of the hardware failure of member *D* (if *D* leaves the group, the same chain of events here described will occur). The descendants *E* and *F* detect the failure by not receiving any more unicast messages from *D*. Consequently, *E* and *F* select as *parent member* the *core A*, re-establishing a suboptimal tree connectivity. The tree inefficiency will be recovered at the next *tree-create* procedure. When the faulty member will repair its hardware (or join again the group), it will be the *core* of a mesh formed by only itself and it will connect itself to the other members, by means of the *outer-tree-create* procedure.



Tree and Mesh link
Mesh only link
Before hardware Failure of D
Just after hardware failure of D and the exceution of tree-link-recovery procedure
After the next tree-create procedure
(a)
(b)
(c)

Fig. 18 - *Tree-link-recovery* procedure facing a hardware failure

APPENDIX II
FORMAT OF THE OBAMP MESSAGES

In the following we will describe each field only once, even if it is contained in more than one message.

A. *Data message*

Fig. 19 reports the DATA message structure.

The `MessageID` field is contained in all the OBAMP messages and identifies the message type (DATA, HELLO, FASTHELLO, etc.).

The `SequenceNumber` field counts the number of messages generated for each message type.

The `SourceIP` field contains the IP address of the member that generates the message.

The `JustForwardedMemberCode` (JMFC) field is the bit-map that codes the list of the members for which further forwarding is forbidden[7].

1 byte	1 byte
MessageID	SourceIP
SequenceNumber	JustForwardedMemberCode
(8 bytes)	

Fig. 19 –DATA message

B. *HELLO messages*

Fig. 20 reports the structure of the HELLO message.

The `CoreAddress` field is the IP address of the *core* of the member originating the HELLO message.

1 byte	1 byte
MessageID	SourceIP
SequenceNumber	TTL
CoreAddress	

Fig. 20 – HELLO message

Fig. 21 reports the structure of the HELLOCONF message.

The `HelloSequenceNumber` field is used to piggy back the sequence number value of the originating HELLO message. This allows the source of the HELLO message to avoid handling outdated responses.

1 byte	1 byte
MessageID	SourceIP
HelloSequenceNumber	TTL
CoreAddress	

Fig. 21 – HELLOCONF message

Fig. 22 and Fig. 23 report the structures of the FASTHELLO ad ALIVEHELLO messages, respectively.

1 byte	1 byte
MessageID	SourceIP
SequenceNumber	CoreAddress

Fig. 22 – FASTHELLO message

1 byte	1 byte
MessageID	SourceIP

Fig. 23 – ALIVEHELLO message

C. *TreeCreate messages*

Fig. 24 reports the structure of the TREECREATE message and of the OUTERTREECREATE message, being equal to each other. The `NearestFlag` field is used by the forwarding member to inform the receiver that it is the nearest neighbour.

The `JustForwardedMemberCode` field codes the list of members for which further forwarding is forbidden, since these members have just handled this TREECREATE message. Each member puts itself in this list before forwarding the TREECREATE message.

1 byte	1 byte
MessageID	SourceIP
SequenceNumber	CoreAddress
Nearest flag	JustForwardedMemberCode
(8 bytes)	

Fig. 24 –TREECREATE and OUTERTREECREATE messages

Fig. 25, Fig. 26, Fig. 27 and Fig. 28 report the control messages used during the *tree-link-setup* (TREECEATEACK and TREECREATECONF) and *tree-link-tear-down* (TREECREATENACK and TREECREATENACKCONF) procedures.

The fields not described up to now are `ACKSequenceNumber` and `NACKSequencenumber`. Such fields are used to piggy back the sequence number value of the originating TREECREATEACK and TREECREATENACK messages, respectively. This allows the source of the TREECREATEACK messages (and of the TREECREATENACK messages) to avoid handling outdated responses.

1 byte	1 byte
MessageID	SourceIP
SequenceNumber	CoreAddress

Fig. 25 – TREECEATEACK message

1 byte	1 byte
MessageID	SourceIP
ACKSequenceNumber	CoreAddress

Fig. 26 – TREECREATECONF message

1 byte	1 byte
MessageID	SourceIP
SequenceNumber	CoreAddress

Fig. 27 –TREECREATENACK message

1 byte	1 byte
MessageID	SourceIP
NACKSequenceNumber	CoreAddress

Fig. 28 –TREECREATENACKCONF message

[7] This field coding aims at reducing the field length. The coding assumes that the network contains at most 64 hosts and is assigned a class C addressing space, and works as follows: the *k*-th bit is set to 1 when we want to forbid forwarding to the member whose least significant byte of its IP address is equal to *k*. Under different assumptions, the coding must be suitably modified, eventually resorting to a brute-force approach of creating a list of IP addresses.

APPENDIX III
PERFORMANCE EVALUATION OF A SINGLE-SOURCE SCENARIO

The aim of this section is to replicate the analysis of section V with a single-source traffic model, which is believed to be a more favourable scenario for ODMRP. We analyze the performance only as a function of the group size.

We assume a single-source traffic model in which a single member node sends out CBR traffic at 16 kbps, with a payload of the data packets equal to 256 bytes.

Fig. 29, Fig. 30, Fig. 31, Fig. 32 reports the following performance indicators as a function of the group size: byte sent per byte delivered, delivery ratio, average latency and tree efficiency. We first compare OBAMP and ODMRP and then OBAMP and ALMA.

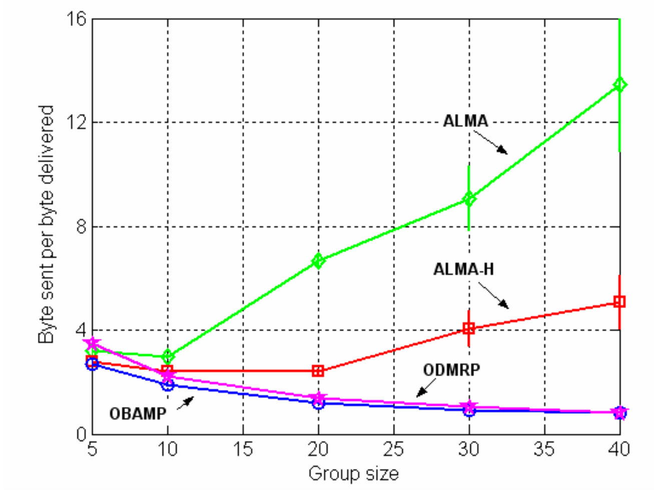


Fig. 29 - Bytes sent per bytes delivered vs. group size

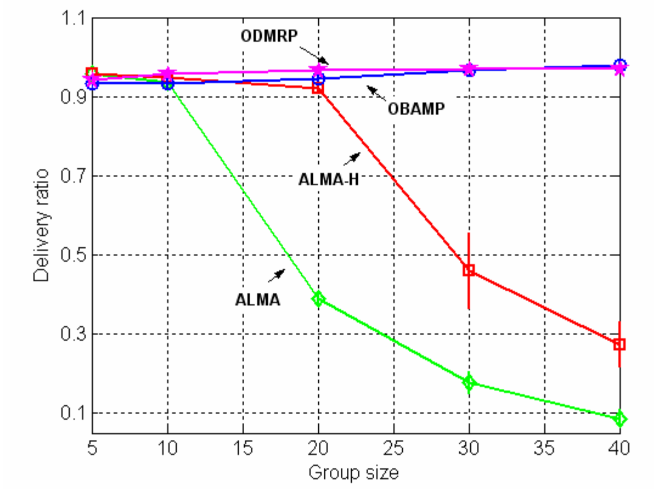


Fig. 30 - Delivery ratio vs. group size

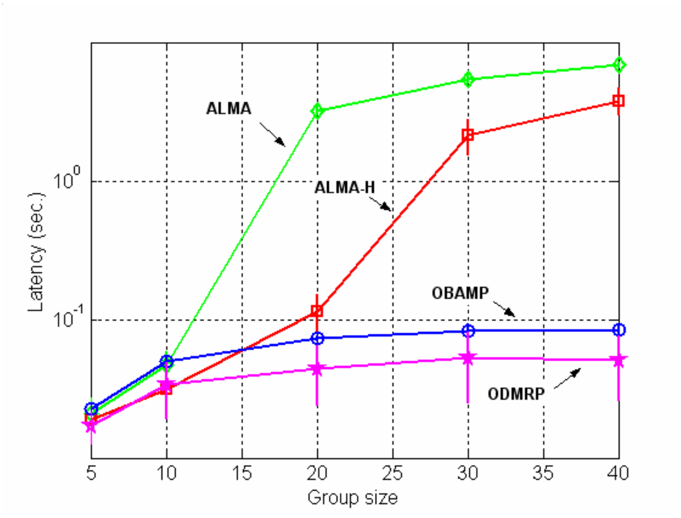


Fig. 31 - Average data latency vs. group size

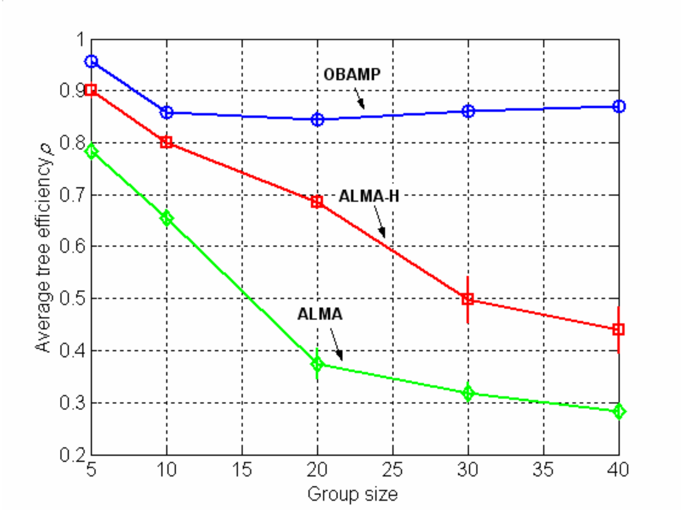


Fig. 32 - Tree efficiency vs. group size

1) *OBAMP vs. ODMRP*

If we compare the ODMRP performance in single and multi source cases (V.D), we find out that the single-source traffic model is a more favourable scenario for ODMRP.

The Fig. 29, Fig. 30 and Fig. 31 show that OBAMP and ODMRP have similar performance in the single-source case, with the exception of a small penalty of OBAMP as regards the average latency (with a gap of about 30 ms).

2) *OBAMP vs. ALMA*

The performance of OBAMP and of ALMA in the single-source case (Fig. 29, Fig. 30, Fig. 31, Fig. 32) are very similar to the corresponding ones in the multi-source case case (Fig. 4, Fig. 5, Fig. 6). Thus, we can state that the conclusions reached for the multi-source case hold for the single-source case as well.

APPENDIX IV
THREE PROPERTIES OF THE OBAMP TREE

In this section we proof three properties of the OBAMP tree.

<u>Property A</u>: *the tree created by OBAMP does not have persistent tree loops.*

Proof of property A: let us consider the tree as if it were a directed tree rooted at the *core*. Under this working assumption, we can say that a tree loop is generated when during the same refresh round a member H chooses as *current parent member* a member K that is connected lower down in the tree (i.e., a descendant) with respect to itself. Then, in a generic refresh round #*x,* a generic member H can not select as *current parent member* any member K that is placed on a tree vertex that descends from H. In fact, if K forwarded the TREECREATE #*x* message to H, then H would discard it, since the same message has already passed through H (given that K is a descendant of H). As a consequence, the *tree-create* procedure avoids tree loops and, in turn, group partition.

<u>Property B</u>: *OBAMP builds a distribution tree that contains at least the fist-level edges of the minimum spanning tree.*

<u>Assumption:</u> *for the sake of simplicity we assume that the handling-delay is the only source of delay in the network. The consequence of this assumption and the way to make it uninfluential in practical cases are discussed at the end of the section.*

Proof of property B: let us define first-level tree a tree formed at the first iteration of the `while` loop of the Borůvka algorithm (see III.B). A first-level tree is a set of connected first-level edges; this set may contain, at a minimum, a single edge. As previously discussed, the OBAMP mesh surely contains all the first-level edges and, hence, it contains all the first-level trees. To complete the proof we must now show that, during a refresh round of the *tree-create* procedure, the first-level edges will be surely marked as tree link. Let us focus our attention on a given first-level tree, T. Let us define as t_k the time instant in which the first TREECREATE #*x* message is handled by any member of T. Given that we are assuming that the TREECREATE #*x* message does not experience any other delay than the *handling delay*, the TREECREATE #*x* message will be immediately forwarded on all mesh links that form T and immediately handled by the members of T. In fact, the *handling delay* is set to zero, for these overlay links. Now, since this is the first TREECREATE #*x* message of the refresh round #*x*, all these mesh links will be surely selected as tree links; q.e.d.

To give an example of property B, let us consider Fig. 33, where the first-level trees are: {*A-B-C*}, {*D-E*}, {*F-G*}, {*H-I-L*} and {*M-O*}. Let us focus our attention on a generic first-level tree, i.e., {*H-I-L*}.

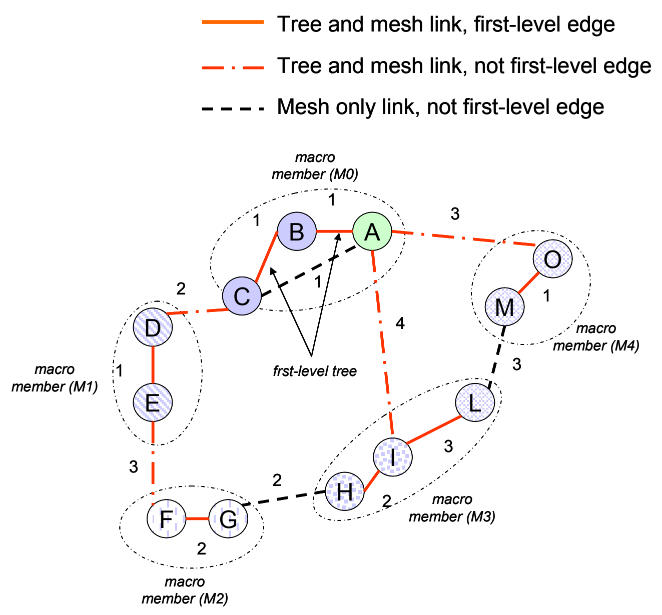


Fig. 33 - An example mesh and tree configuration; the *core* is member *A*

When at time t_k member I handles the first TREECREATE #*x* message coming from the *core* A on the overlay link A-I, member I forwards this message toward members H and L. The receiving members H and L immediately handle this message because the related *handling delay* is zero; as a matter of fact H is the nearest member of member I and I is the nearest member of member L[8]. Consequently, the I-H and I-L overlay links are set as tree links and these overlay links are precisely first-level edges. The same thing occurs within the other first-level tree, as anticipated by property B.

The third property is concerned with the other links of the OBAMP tree; i.e., those links that are not the fist-level edges of the minimum spanning tree.

In order to focus our analysis only on those links that are not fist-level edges, in the following we introduce a new mesh, named macro-mesh, in which the first-level edges are hidden, as follows. Let us define as macro-member the set of members forming a first-level tree and the mesh links connecting such members. The OBAMP mesh can be seen also as a mesh connecting such macro-members, which we call macro-mesh. In the same way, the OBAMP tree can be seen also as a tree connecting such macro-members, which we call macro-tree.

In Fig. 34 we give an example of such definitions by reporting the macro-tree and the macro-mesh of the configuration shown in Fig. 33.

[8] We are assuming that in case of mesh links with equal hop distance, the shortest mesh link is considered the one that at the other end has a member with the smallest IP address.

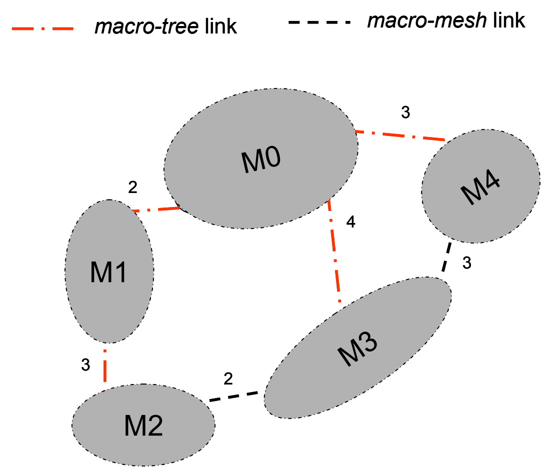


Fig. 34 - *Macro-mesh* and *macro-tree* related to the configuration of Fig. 33

This said, we can state the:

Property C: *the links of the OBAMP tree that are not first-level edges of the minimum spanning tree are the links of the macro-mesh that belong to the shortest-path rooted at the core.*

Assumption: *as for property B, we assume that the handling-delay is the only source of delay in the network.*

Proof of property C: during a generic *tree-create* refresh round *#x*, the TREECREATE *#x* message passes through a *macro-member* instantaneously, since within the *macro-member* the path forming the first-level tree has an *handling delay* equal to zero. In addition, the *handling delay* of the mesh links connecting *macro-members* is an increasing function of the hop distance. As a consequence, the first TREECREATE *#x* message that is handled by a member of *macro-member M* and that turns the transporting mesh link into a tree link, is received on the shortest-path of the *macro-mesh* from the *core* to *M*.

To present an example of this property we consider again Fig. 33 and focus our attention on the overlay links that connect *macro-members*. These overlay links are the links of the macro-mesh reported in Fig. 34.

Let us consider a generic macro-member, e.g., *M2*, and assume that the *core* sends out the TREECREATE #x at time t_0. The macro-member *M2* can receive the TREECREATE *#x* message from the following paths: *M0-M1-M2, M0-M3-M2, M0-M4-M3-M2*, whose hop distances on the macro-mesh are equal to 5,6 and 8 hops, respectively.

The TREECREATE *#x* message passing through the path *M0-M1-M2* will be handled by *M2* at time t_0+2*$\mathrm{U_h}$+3* $\mathrm{U_h}$[9]. Where 2*$\mathrm{U_h}$ is the *handling delay* of the overlay link *M0-M1* and 3*$\mathrm{U_h}$ is the *handling delay* of the overlay link *M1-M2*. Hence, the overall *handling delay* is a linear function of the hop distance of the path on the macro-mesh, that is 2+3=5 hops.

The same reasoning can be repeated for the other paths *M0-M3-M2, M0-M4-M3-M2*.

Consequently, the first TREECREATE *#x* message handled by *M2* passes through the overlay link *M1-M2* because this overlay link belongs to the path of the macro-mesh that has the smaller hop distance from the *core* to *M2* (property C), that is the path *M0-M1-M2*. The same reasoning can be repeated for the other tree links that connect other macro-members.

We notice that the shortest path approach followed by OBAMP for the tree links that not are first-level edges leads to a distribution tree that is suboptimal with respect to the minimum spanning tree. For instance, in Fig. 33, the cheapest overlay link to connect macro-member *M3* to the rest of the tree should be *G-H*; instead, the shortest path approach of OBAMP set as tree link the more expensive overlay link *A-I* .

Future work could try to identify better performing procedures to limit this inefficiency, possibly without increasing too much the signalling load.

We conclude this section by commenting the assumption requiring that the *handling delay* is the only source of delay in the network. Actually, it can be shown that the two properties hold also if the *handling delay time unit,* $\mathrm{U_h}$, is greater than the maximum network delay. In other words, this parameter must be chosen great enough so that the other sources of delay do not alter the order of precedence in choosing tree links established by the *handling delay* parameter by itself.

Thus, a proper choice of the parameter $\mathrm{U_h}$ makes not needed the assumption made at the beginning of this section.

[9] We are neglecting the small r(0,1) random value of the *handling delay* computational algorithm (IV.B.3)