Group Communication Applications for MANETs: Requirements and Real Implementations

Andrea Passarella and Franca Delmastro
CNR, IIT Institute
Via G. Moruzzi, 1 – 56124 Pisa, Italy
{andrea.passarella,franca.delmastro}@iit.cnr.it

Abstract

Desinging, developing and testing *real applications* for ad hoc network environments still deserves particular attention by the MANET research community. We believe that they represent a great incentive for mobile users to adopt the MANET technology in the daily life. From this standpoint, Group- Communication Applications are a very interesting class. In the chapter we motivate this claim by presenting examples of Group-Communication Applications that leverage *exclusive* features of ad hoc networks. Being run on completely on-demand networks they can be particularly valuable to end users. We then provide an overview of the main issues that should be addressed for such applications to be deployed in practice. As a case study, we present our prototype implementation of a Whiteboard Application. The prototype includes the networking support required by the Whiteboad, and thus can be used to test it in a real testbed. We present outcomes of a rich set of experiments in which we compare the performance achieved when alternative P2P systems are adopted in the network support. Results, highlight that designing network protocols (and P2P systems in particular) that exploit the *cross-layer* argument is of paramount importance to make Group-Communication Application feasible for real.

I. Introduction

Even though research on MANETs has been very active in the last decade, real applications addressed to people outside the research community still have to be developed. The typical simulation-based approach for the performance evaluation of MANETs is one of the main reasons of this. Often, simulation results turn out to be quite unreliable if compared to real-world measurements [2], [18]. Thus, real-world experiments are highly required for MANET applications to become reality, despite their high costs (in terms of time to set up) and intrinsic limitations (number of nodes). Furthermore, few efforts have been devoted by researchers to design applications that exploit the very features of ad hoc networks, and could thus be particularly valuable to MANET users. Most of the research has been carried out to analyse performance of networking protocols in isolation, whithout considering reasonable applications to be run on top of them.

By leveraging the self-organising nature of MANETs, Group-Communication Applications can be an outstanding opportunity from this standpoint. Group-Communication Applications can highly benefit from a networking environment like MANETs, in which networks can be set-up and torn donw completely on demand, without requiring any pre-existing infrastructure.

In this chapter we thus focus on this class of applications. In the first part, we motivate the idea of studying Group-Communication Application for MANETs and provide some example that have been proposed in literature (Section II). Supporting the Group Communication paradigm is not an easy task even in traditional wired networks. Addressing its requirement on MANETs is even more challenging. Thus, we present the main issues to be faced in Section III, and highlight the main research directions related to each issue.

The second part of the chapter is devoted to analysing a real case of GCA development. Specifically, we consider the *Whiteboard Application* (WB), which implements a distributed whiteboard among MANET users (Section IV). WB allows users to share drawings, messages and other dynamically generated content. To understand how WB works in reality we have implemented this application in a real MANET prototype. The WB application can be naturally supported by P2P systems. In our prototype, WB uses Scribe [7] to share content among users via application-level multicast trees. Scribe requires a P2P overlay network based on a Distributed Hash Table (DHT). Our prototype includes two alternative P2P solutions, i.e., Pastry [29] and CrossROAD [13]. Both of them implement the P2P commonAPI [10], and thus provide the same functionality to above layers. However, CrossROAD is explicitly designed for MANET environments. It reduces (with respect to Pastry) the network overhead related to the overlay management by exploiting cross-layer interactions with a proactive routing protocol. Specifically, the CrossROAD implementation is compliant with the cross-layer framework described in [11]. Finally, the prototype also includes both proactive and reactive routing protocols (i.e., OLSR [25] and AODV [3], respectively).

WB performance have been evaluated via an extensive measurement campaign (Section V, VI and VII). We evaluate our prototype at two different levels, i.e., we quantify *i)* the QoS perceived by WB users, and *ii)* the quality of the multicast tree generated by Scribe. First of all, we show how a proactive routing protocol performs better than a reactive one with regard to this kind of applications. Then, we highlight that a solution based on Pastry and Scribe is not very suitable for MANET environments. WB users perceive unacceptably high data loss and delay. Furthermore, both the Pastry overlay network and the Scribe multicast tree get frequently partitioned. This results in some WB users to be completely isolated from the rest of the network. Finally, we show that some of these problems can be avoided by using CrossROAD. Specifically, the structure of the Scribe tree is quite more stable when CrossROAD is adopted, and partitions problems experienced with Pastry completely disappear. Thus, CrossROAD turns out to be a very promising P2P system for MANET environments. We conclude the chapter by envisioning cross-layer optimisations that can significantly improve the P2P mutlicast system, as well (Section VIII).

II. Group Communication Applications and MANETs

A. Why Group Communication Applications on MANETs?

A high share of applications running on the current Internet is based on the client-server paradigm, the World Wide Web being the most popular example. Group-Communication Applications (GCA) such as Instant Messaging and Distributed Games are increasingly diffused, and take a completely different communication pattern. In GCA, a group of users want to communicate with each other, instead of issuing requests to a central server. Even though

some GCA implementations on the legacy Internet use client/server transactions (e.g., Instant Messaging typically uses central servers to detect user presence and availability), the eventual communication pattern among GCA users is more similar to P2P than to client/server systems. Furthermore, GCA must allow users to have a more dynamic behavior, and support coordination among them. Specifically, GCA users may appear and disappear dynamically, possibly resulting in high churn rates.

Group Communication Applications look particularly fit to be developed on MANETs. Recently, the research community is increasingly interested in issues related to Group Communication for ad hoc networks [1]. On the one hand, the networking systems designed for MANETs should already include several features that are needed by GCA. For example, MANET network protocols have to be resilient to very dynamic conditions in which nodes connect and disconnect dynamically. They should be designed having in mind a completely distributed environment, in which no central points of coordination can be assumed. They should be able to efficiently handle network partitions and joining of separated networks. On the other hand (and, most importantly) MANETs can be established without any pre-existing infrastructure. This makes plausible to run GCA completely on-demand, as soon as a group of users decide that they want to communicate. The only requirement is that users be sufficiently close to be connected to each other via possibly multi-hop paths. Temporary nodes' disconnections may also be tolerable, depending on the particular application.

Based on these remarks, we belive that Group Communication Applications is an exciting field to be explored to identify valuable applications for MANET users.

B. Examples of GCA on MANETs

Broadly speaking, any application in which a group of users want to communicate with each other can be seen as a Group Communication Application. So, it is fairly easy to envision real examples of GCA. For the sake of conciseness, in this section we mention three representative papers, in which authors describe a candidate application, and discuss issues related to implement it on MANETs.

In [16] authors discuss issues related to porting *Instant Messaging* to ad hoc networks. Clearly, the application is not new, but its increasing diffusion, and the potential benefit of running it on on-demand ad hoc networks, make IM a premier candidate to be deployed on MANETs. The work in [16] mainly deals with the problem of turning centralised operations of traditional IM systems into completely distributed ones. They mention peer discovery, presence and communication management as the main points to be addressed. They also identify components already available in literature that are able to cope with these issues.

In [14] authors focus on applications requiring coordinated actions among a set of *Mobile Robots*. While the paper mainly addresses multicast communication among robots, it also highlights a very promising scenario for Group-Communication Applications. Mobile robots can be used in several scenarios, ranging from hazard detection to search and rescue, to exploration in hostile or inacessible-by-human environments.

Finally, [27] considers *Augmented Reality Distributed Games*. Distributed games over the legacy Internet are already

largely diffused applications (e.g., Doom, Quake, etc.). Using MANETs as the network platform for such applications adds a further dimension. Since users are free to move and play with mobile devices, the gaming experience can be enhanced through augmented reality techniques. For example, in a Doom-like case, the game scene can be a real, physical place, while tools like armours, medi-kits, etc. can be virtual objects within the real scene. In [27] authors identify data management issues that have to be addressed to implement such applications. We survey these issues in more detail in Section III-B.

III. Requirements of Group Communication on MANETs

Supporting the Group Communication paradigm is clearly much more complex than supporting the traditional client/server paradigm. Groups are usually made up of more than two nodes, and group members can join and leave at arbitrary time and rate. The pattern of communication is usually point-to-multipoint (instead of point-to-point). Besides requiring efficient network support (e.g., multicasting) this also implies challenging data-management issues. The scarcity of networking resources available in MANETs makes these problem even more challenging.

In the following of this section we survey the main problems one has to face with to build Group-Communication Applications. We classify them in three categories, i.e., *Group Management*, *Data Management*, and *Network Support*.

A. Group Management

Essentially, Group Management is about defining who is part of which group, and providing a consistent view of the group to the members. An example of Group Management system is described in [6], where authors present a middleware support for Group Communication named AGAPE. The AGAPE Group Management module defines two possible roles for group members. Managed Entities (ME) just exploit AGAPE to communicate with other group members. Locality Manager Entities (LME) implement the Group Management functionality. The different roles are used to define cluster-like structures that help reducing the network overhead related to Group Management. The AGAPE Group Management module defines the IDs of groups, allows nodes to advertise their presence in the group and to join/leave the group dynamically. Furthermore, AGAPE supports admission control for new nodes willing to join the group, and distributes group views, i.e., advertises group members to each other. Therefore, AGAPE includes a representative set of features that any Group-Communication management system should provide. Other examples of Group Management systems are presented in [20], [28].

B. Data Management

In any distributed system data management is a focal point. In Group Communication this is actually a very challenging issue, due to the presence of multiple data producers and consumers. Natural questions to be answered are where in the network it is better to place the data (close to the consumer? close to the producer?), if and how to replicate data, how to maintain consistency, etc. Ad hoc networks add further dimensions to the problem because: *i)* no central point of aggregation should be assumed, and *ii)* bandwidth resources are very precious.

In [27] (distributed gaming) the data authors focus on is the game *state*. It is very important that the game state be managed carefully, since players' actions are based on it. Since no central server is available in MANETs, the game state is replicated on each node. Then, mechanisms are included to keep the state consistent among the nodes. The authors explicitely deal with state management upon network partitions and rejoins, which are common events in MANETs. Since players in different partitions are not able to communicate, they cannot influence each other behavior. Thus, the game state evolves separately in the different partitions, and a reconciliation policy merges the states of joining partitions in a global, consistent one (see also [4], [19] for Group Communication in partitioned networks). Finally, another fundamental requirement discussed in [27] is the causal and temporal ordering of message delivery. Intuitively, the Group Communication support must ensure that correlated events are received in the correct order by each user. For example, in a Doom-like game it is not acceptable that an enemy falls dead before the shooter fires. In [27] authors present an analytical framework to derive maximum delay jitters the network may allow in order to keep causal and temporal ordering.

Data management for Group Communication does not only mean keeping correct state among the group users, but also making users able to locate and get the required data efficiently. To this end, data-centric routing strategies have been proposed. The main idea is that data should not be routed based on topological information (like in IP networks), but based on the data semantic. Two main families can be identified within data-centric routing, i.e., topic-based routing, and content-based routing. In topic-based routing data are categorised in topics. Each topic is then assigned to a particular node in the network. Distributed mechanisms allow users to map a topic to the corresponding node, thus allowing them to store and retrieve data related to the topic. Content-based system are more flexible, since do not require data classified by topic. Instead, users declare their interests, and the system must be able to locate data that match (also partially) those interest, and route them to the respective nodes. Even though such systems were originally designed for the legacy Internet [12], [29], they are particularly suited to support Group Communication Applications on MANETs. These systems require minimal configuration at the application level, are usually self managing and self healing, and can be designed without any point of centralisation. For example, CrossROAD [13] is a system for subject-based routing explicitely designed for MANET environments. Also, hybrid systems exploiting both topic-based and content-based features have been proposed for ad hoc networks [30].

C. Network Support

Group Management and Data Management are two core blocks of any Group Communication system. However, the ground on which both are founded is an efficient networking environment. In this section we identify the main communication patterns used in such GC networking supports. For more details on this particular topic, the interested reader is referred to [24].

In general, the traditional unicast pattern widely adopted in client/server applications is not very suitable for Group Communication Applications. Rather, multicasting, broadcasting, and gossiping are more fit. All of them allow point-to-multipoint communication, which is the prevalent pattern in Group Communications.

Broadcasting is a straightforward way of disseminating information to all nodes in the sender transmission zone at once, since wireless transmissions are broadcast in nature (the wireless advantage). However, uncontrolled broadcast leads to well-known problems like flooding and broadcast storms. Also, reliable broadcast is usually a problem, since no straightforward way is available to understand if all intended receivers actually got the message. Geocast is broadcast scoped within a particular geographic region. It is used by applications that have to send messages to all nodes in a particular region. It shares the advantages and drawbacks of broadcast, and, in addition, it also needs that nodes can locate themselves in the physical space.

Multicasting is often the most efficient way of achieving point to multipoint communication. Several approaches have been proposed in literature to implement multicast over MANETs. A classic one (borrowed from the wired Internet) is building a *tree* that connects all nodes in the multicast group [5]. A different tree can be built for each sender in the group, or a single shared tree can be used to carry messages possibly generated by any node in the group. Tree-based multicasting is very popular in the legacy Internet, but it is questionable whether it is the best approach also for MANETs. Specifically, the intrinsic instability of wireless links make the tree quite unstable as well, possibly resulting in low delivery rates and high management traffic. To overcome these drawbacks, mesh-based multicasting has been proposed [21]. Group members are not joined by a tree, but by a mesh. This structure is more resilient to link failures and node movements thanks to its redundancy, and thus mesh-based multicasting usually achieves higher delivery rates.

Both these approaches require nodes to keep some state about the multicast structure. A first argument against this requirement is that both meshes and trees may encompass also node that are not interested in the group, but that have to be member of the structure (and hence keep the state) nevertheless. To overcome this drawback, middleware-level solutions have been proposed, in which the mesh or the tree is built on top of an overlay network that just encompasses the intended multicast group members [15], [17]. Even though non-group members have possibly to forward traffic for the group, they are not forced to keep the state of the structure anymore. Another advantage of middleware-level solutions is the option of implementing multicast protocols on top of a P2P Distributed Hash Table (DHT). This is actually a very popular approach in the most recent proposals for middleware-level multicast for the Internet ([7] [26] [31] [9]). This approach is very interesting for a number of reasons. Firstly, the task of defining a network structure that just encompasses the edge nodes is assigned to the DHT, and has not to be implemented by the multicast protocol itself. Secondly, the multicast protocol leverages the self-organising and self-recovery features of the DHT. Finally, the same DHT can be shared by several higher-level services running besides the multicast protocol. To the best of our knowledge, the feasibility of such systems on multi-hop ad hoc networks has not been investigated yet. In this work we take this approach, and we choose Scribe ([7]), because it is one of the most recent and popular p2p multicast protocols, and has shown to outperform other similar approaches sharing the same concepts [8].

A second argument against mesh- and tree-based multicasting is that they are *structured* solutions. Maintaining these structures may be very costly in MANET environments, due to the high dynamism of the network. Therefore, *structure-less* solutions have been proposed. RDG [23] uses *gossiping* to avoid structure maintenance. Specifically,

in RDG each member of a group knows a few nodes that are members of the same group. Upon generating (or receiving) a message, a node forwards it to the group members it knows. RDG is able to guarantee good reliabilty, but it introduces possible replicated receptions. On the other hand, DDM [22] exploits unicast routing information to implement multicast delivery. A sender inspects the unicast routing table to get the next hop towards each other member in the group. Then, it classifies group members based on the respective next hop. If members R1 and R2 happen to share the same next hop N, then a single message is sent to node N, making it responsible for forwarding the message to R1 and R2. Structure-less solutions like RDG and DDM look suitable for MANETs, since they do not require management traffic that can be very costly. However, a drawback of both of them is the fact that they need to diffuse information about which nodes belong to the group. In [23], [22] this is achieved via periodic flooding and polling, which are costly operations as well.

The work in [23] can be also seen as an example of gossiping protocol. The main idea of this approach is that senders of a multicast group select a random subset of nodes in the group to send data to. The same process is repeated at receiving nodes for a given number of turns, this number and the size of the random subsets being protocol parameters. It has been shown that such protocols are actually able to deliver messages with high probability to all intended receivers [23]. A side effect of gossiping is that the message replication rate is quite difficult to control. Evaluating such an approach in comparison with p2p multicasting is an interesting topic which is however out of the scope of this work.

IV. A real case study: the Whiteboard Application

After having detailed the main requirements of Group Communication Applications in the previous section, we now focus on a real case study, i.e. a distributed Whiteboard Application (WB). More exciting examples of GCA can be surely taken into consideration. However, WB is very simple, and this allows us to concentrate on general issues arising when really implementing GCA support on MANETs.

The Whiteboard can be used to share dynamically generated content (e.g., drawings, messages, ..., see Figure 1). Each user runs a WB instance on her mobile device, and selects a *topic* she wants to associate to (i.e. “treasure hunting”). Each topic is linked with a canvas on which she can draw strokes or type text. On the same canvas, the user directly sees strokes and text generated by others.

The Whiteboard has been actually implemented in a real MANET prototype on which we ran an extensive set of experiments. We also included in the prototype a full network stack, implementing WB data management and network support. To not add too many dimensions to the problem space, we decided not to include any group management support. In our tests the set of group members is known in advance, and it does not change over time.

WB uses a topic-based multicast protocol to join group members, and disseminate WB data to the group members. Therefore, we implement the data management system and part of the network support via a topic-based P2P middleware suite. Specifically, Scribe [7] is used as the topic-based multicast protocol, since it has shown to outperform other similar solutions [8]. As better described in Section IV-B, Scribe allows a node to send messages addressed to a

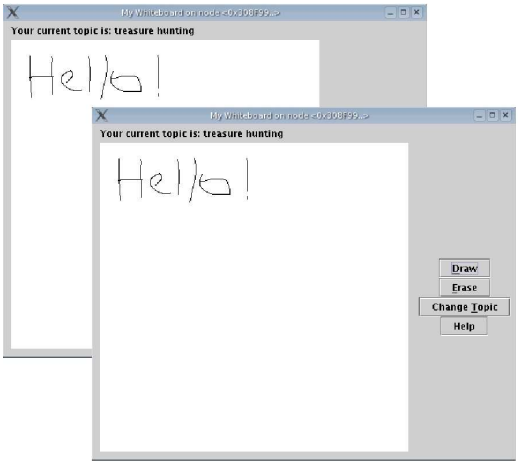

Fig. 1. The Whiteboard Application

topic by simply knowing the topic name, instead of the whole set of nodes belonging to the group. Scribe is designed to work on top of a P2P structured overlay network implementing a DHT. As a first step, we used Pastry [29] as the overlay network, since Scribe was originally designed on it. Even though Pastry was designed for the legacy wired Internet, evaluating it through an experimental testbed allows us to understand if such an off- the-shelf solution is suitable. Then, we replaced Pastry with CrossROAD [13], which, being designed for MANETs, provides the same functionality optimised for this environment. The resulting network architectures are shown in Figure 2. As is discussed in detail in this book, the main difference between Pastry and CrossROAD is that CrossROAD exploits cross-layer interactions with a proactive routing protocol (OLSR) to maintain a correspondence between the physical and the logical address spaces (identified by the DHT). When used on MANETs, this approach proves to be far more efficient than the original Pastry. More details about Pastry and CrossROAD can be found in the dedicated chapter of this book. For the reader convenience in the next section we briefly recap their main features.

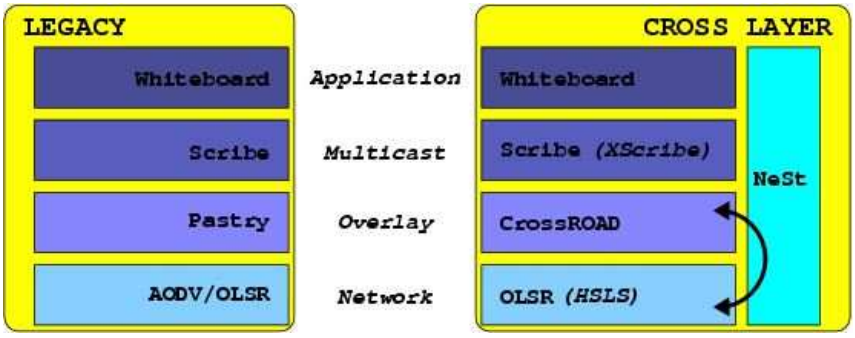

Fig. 2. Networking stacks: legacy (left) and cross layer (right).

Note that the system we have described so far is not yet fully optimised with respect to what discussed in Section III-C. Specifically, the multicast support is implemented at the overlay level, but no multicast system is then included in the network layer. Instead, the real communication at the network level occurs via traditional unicasting. A big improvement we are working on is to further optimise the network support by designing a multicast system at the

network level able to allow Group-Communication Applications to route data in a topic-based fashion.

A. Pastry and CrossROAD

Pastry is a P2P system based on a DHT to build a structured overlay network (*ring*) at the middleware level. A logical identifier (node id) is assigned to each node hashing one of its physical identifiers (e.g., IP address, hostname). Messages are sent on the overlay by specifying a destination key k belonging to the logical identifiers' space. Pastry routes these messages to the node whose id is numerically closest to k value. To route messages, Pastry nodes maintain a limited subset of other nodes' logical ids in their internal data structures (middleware routing tables). Periodic data exchange between nodes of the overlay are needed to update the state of the overlay. Finally, in order to initially join the overlay network, each Pastry node executes a bootstrap procedure, during which it initialises its middleware routing table by collecting portions of other nodes' routing tables. Specifically, each nodes has to connect to an already existing Pastry node (i.e., it needs to know its IP address) in order to correctly start the bootstrap procedure.

The bootstrap phase and the periodic data exchange between nodes constitute the main network overhead of Pastry. CrossROAD, that is a Pastry-like P2P system explicitly designed for MANETs, drastically reduces the Pastry overhead by exploiting cross-layer interactions with a proactive routing protocol. Specifically, CrossROAD defines a cross-layer Service Discovery protocol in order to broadcast information about upper-layer services (e.g. Scribe) through the proactive flooding of routing packets, and to maintain an association between nodes' IP addresses and provided services. Hence, each CrossROAD node can autonomously build the overlay, by simply hashing the IP address of nodes providing the same service. In this way the overlay network related to a particular service is maintained with almost *negligible network overhead* in comparison with Pastry. Furthermore, CrossROAD *i)* is completely self-organising, since it does not require any bootstrap procedure, and *ii)* correctly manages cases of network partitioning and topology changes with the same delays of the routing protocols.

B. Scribe Operations

Scribe exploits topic-based routing to build multicast groups (Figure 3(a)). From the standpoint of the application running on Scribe, the group is identified by a *topic*. Scribe uses the hash function provided by Pastry (or CrossROAD) to generate the topic id (t_{id}) in the logical space of node ids. In order to join the Scribe tree, nodes send a `join` message on the overlay with key equal to t_{id}. This message reaches the next hop (say, N) towards the destination on the overlay network. The node originating the `join` message is enrolled as a child of N. If not already in the tree, N itself joins the tree by generating a `join` message anew. Eventually, such a message reaches the node whose id is the closest one to t_{id} and it is not further propagated. This node is defined as the *root* of the Scribe tree.

Application messages are sent on the overlay with key equal to t_{id} (see Figure 3(b)). Hence, they reach the Scribe root, which is in charge of delivering them over the tree. To this end, it forwards the messages to its children, which further forward them to their children, and so on.

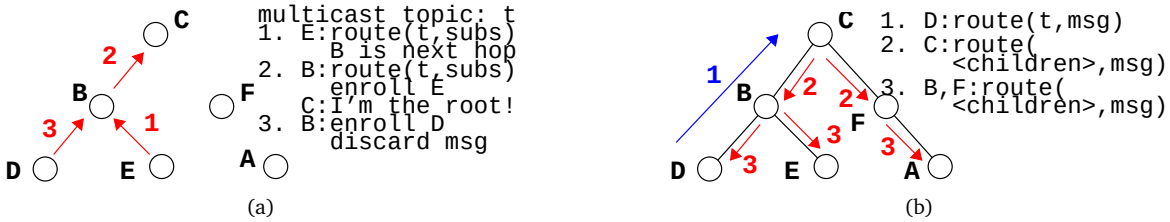

Fig. 3. Scribe building the tree (a), and disseminating messages (b).

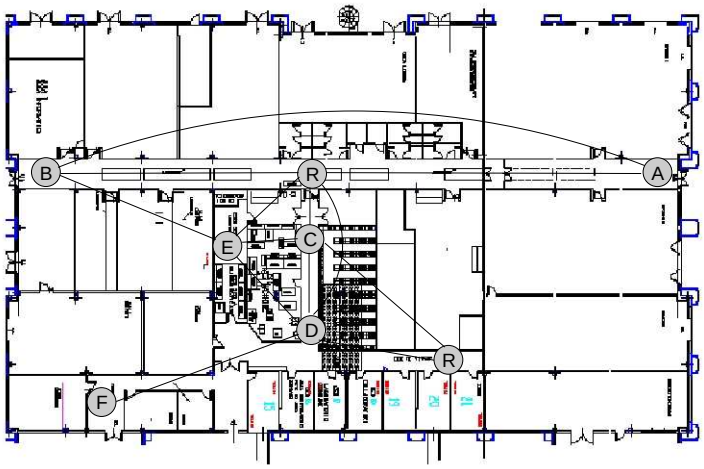

Fig. 4. Map of the experiment setup

Finally, the Scribe maintenance procedure is as follows. Each parent periodically sends a `HeartBeat` message to each child[1]. If a child does not receive any message from the parent for a given time interval (20 s in the default case), it assumes that the parent has given up, and re-executes the join procedure. This simple procedure allows node to discover parent failures, and re-join the tree, if the case.

V. Experimental Setup

The experiments reported in this chapter are based on a *static* MANET. This allows us to highlight limitations that originate from Pastry and Scribe design, rather than to mobility. Extending the results in the case of mobility is subject of future work.

The experiment testbed is as depicted in Figure 4. We set up an indoor MANET consisting of 8 nodes. To have an homogeneous testbed, all nodes are IBM ThinkPad R50 laptops. We use the built-in Intel PRO-Wireless 2200 802.11 card, with `ipw2200` driver (on Linux 2.6 kernel). The data rate is set to 11 Mbps. In addition the transmission power of each card has been adjusted to reproduce the topology shown in the figure and obtain a multi-hop ad hoc network. During the experiments, nodes marked A through F participate in the overlay network, and run the WB application (they will be throughout referred to as "WB nodes"). Nodes marked with "R" just acted as routers. It is worth pointing out that this setup lies within the "802.11 ad hoc horizon" envisioned in [18], i.e. 10-20 nodes, and 2-3 hops. Therefore, it is a valid example of possible real-world MANETs.

[1]Application-level messages are used as implicit `HeartBeats`.

In order to have a controllable and reproducible setup, a human user at a WB node is represented by a software agent running on the node. During an experiment, each software agent interleaves active and idle phases. During an active phase, it draws a burst of strokes on the canvas, which are sent to all the other WB nodes through Scribe[2]. During an idle phase, it just receives possible strokes from other WB nodes. After completing a given number of such *cycles* (a cycle is defined as a burst of strokes followed by an idle time), each agent sends a `Close` message on the Scribe, waits for getting `Close` messages of all the other nodes, and shuts down. Burst sizes and idle phase lengths are sampled from exponentially distributed random variables. The average length of idle phases is 10 s, and is fixed through all the experiments. On the other hand, the average burst size is defined on a per-experiment basis. As a reference point, we define a traffic load of 100% as the traffic generated by a user drawing, on average, one stroke per second. Finally, the number of cycles defining the experiment duration is fixed through all the experiments. Even at the lowest traffic load taken into consideration, each agent draws – on average – at least 50 strokes during an experiment. For the performance figures defined in this chapter (Section V-A) this represents a good trade-off between the experiment duration and the result accuracy.

Some final remarks should be pointed out about the experiment start-up phase. Nodes are synchronised at the beginning of each experiment. Then, in the Pastry case, the bootstrap sequence occurs as follows[3]: node C starts first, and generates the ring. Nodes E and D start 5 seconds after C, and bootstrap from C. Node B starts 5 seconds after E and bootraps from E. Node A starts 5 seconds after B and bootraps from B. Finally, node F starts 5 seconds after D and bootraps from D. After this point in time, the Scribe tree is created and, finally, WB instances start sending application messages (herafter, WB messages). In this way, the Scribe tree is built when the overlay network is already stable, and WB starts sending messages when the Scribe tree is completely built.

A. Performance Figures

Since Pastry and Scribe have been conceived for fixed networks, we investigate if they are able to provide an adequate Quality of Service to users in a MANET environment. To quantify the ”WB user satisfaction” we use two performance indices:

- *Packet Loss*: at each node i, we measure the number of WB messages received and sent (R_i and S_i, respectively) during an experiment; the packet loss experienced by node i is defined as $pl_i = 1 - \frac{R_i}{\sum_i S_i}$.
- *Delay*: the time instant when each packet is sent and received is stored at the sending and receiving node, respectively. In this way, we are able to evaluate the delay experienced by each node in receiving each packet. If d_{ij} is the delay experienced by node i in receiving packet j, and N_i the total number of packets received by i during an experiment, the average delay experienced by node i is defined as $D_i = \frac{\sum_j d_{ij}}{N_i}$.

Furthermore, we define two more indices, to quantify the quality of the multicast tree created by Scribe.

[2]Please note that in our software configuration each stroke generates a new message to be distributed on the Scribe tree.
[3]The same schedule is also used to start CrossROAD, even though a CrossROAD node does *not* need to bootstrap from another node.

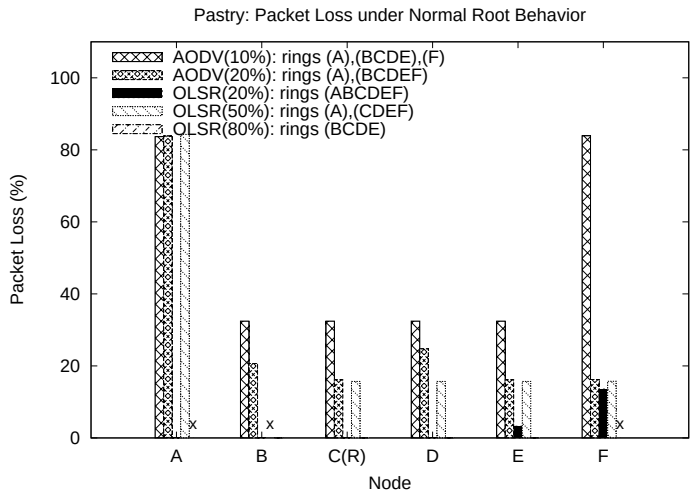

Fig. 5. Packet Loss w/o MSRN crash

- *Node Stress*: for each node, it is defined as the average number of children of that node. If t_{ij} is the time interval (within an experiment) during which node i has n_j children, the average node stress of node i is $NS_i = \frac{\sum_j n_j t_{ij}}{\sum_j t_{ij}}$.
- *Re-subscriptions*: for each node, we count the number of times (during an experiment) this node sends new subscriptions requests, because it cannot communicate with the previous parent anymore.

VI. Performance with Pastry

The results we report in this section are obtained by using Pastry as DHT, and either OLSR or AODV as routing protocols. Experiments are run by increasing the traffic load starting from 20% up to 80%.

Before analysing the experimental results in detail, let us define what herafter will be referred to as "crash of the Scribe Root Node". In our configuration Pastry assigns node ids by hashing the IP address and the port used by Scribe on the node. Hence, each node always gets the same node id. Furthermore, the topic used by the WB users is always the same. Under the hypothesis that Pastry generates a single ring encompassing all WB nodes, the Root of the Scribe tree (i.e., the node whose id is closest to the WB topic id) is the same through all the experiments, and is node C in Figure 4. This node will be throughout referred to as the Main Scribe Root Node (MSRN). Due to the Scribe algorithm, each WB message to be distributed on the tree is firstly sent to MSRN, that then forwards it to its children. Often, this is an excessive load for MSRN, which, after some point in time, becomes unable to deliver all the received messages. Messages are thus dropped at the MSRN sending queue. We refer to this event as a crash of MSRN. Of course, since the application-level traffic is randomly generated, the MSRN crash is not a deterministic event.

A. User QoS

Figures 5 and 6 show the packet loss and the delay indices experienced by the WB nodes considering experiments where the MSRN *does not* crash. Specifically, we consider AODV experiments with 10% and 20% traffic load, and OLSR experiments with 20%, 50% and 80% traffic load, respectively. There is no point in running AODV experiments with higher traffic load, since the system performance with AODV is very low, even with such a light traffic load. In

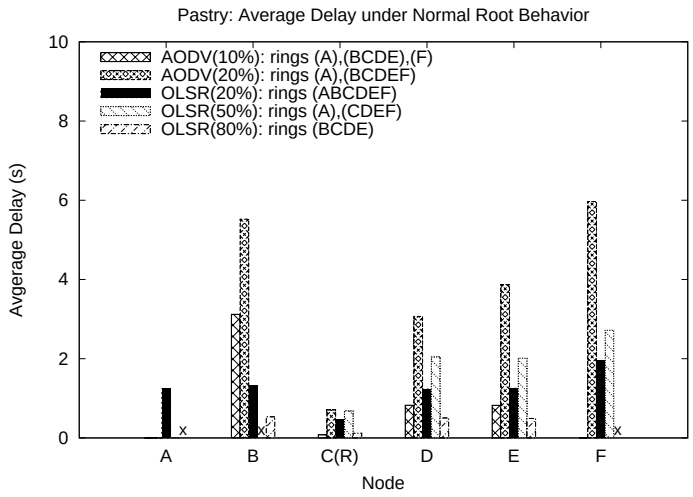

Fig. 6. Delay w/o MSRN crash

the figure legend we also report the rings that Pastry builds during the bootstrap phase (please note that, theoretically, just one ring should be built, encompassing all WB nodes). Finally, an "x" label for a particular node and a particular experiment denotes that for that experiment we are not able to derive the index related to the node (for example, because some components of the stack crashed during the experiment).

Figure 5 allows us to highlight an important Pastry weakness. If a WB node is unable to successfully bootstrap, it starts a new ring, and remains isolated for the rest of the experiment. In MANET environments, links are typically unstable, and the event of a WB node failing to contact the bootstrap node is quite likely. Clearly, once a node is isolated, it is unable to receive (send) WB messages from (to) other nodes for the rest of the experiment, and this results in packet losses at all nodes. In the "AODV 10%" experiment, nodes A and F are isolated, and create their own rings. This results in packet loss of about 80% at those nodes (i.e., they just get their own WB messages, which is about one sixth of the overall WB traffic), and about 33% at nodes B, C, D and E. Similar remarks apply to the "OLSR 50%" experiment. It is more interesting to focus on the "AODV 20%" experiment. In this case, node A is isolated, while nodes B, C, D, E and F belong to the same ring. As before, A's packet loss is about 80%. The packet loss at the other nodes due to the isolation of node A is about 18% (one sixth of the overall traffic). It is interesting to notice that nodes B and D experience a *higher* packet loss, meaning that they are unable to get WB messages generated within the "main" Pastry ring (i.e., nodes B, C, D, E, F). Finally, in the case "OLSR 20%", Pastry is able to correctly generate a single ring, and the packet loss is quite low. In the case "OLSR 80%" nodes A and F crash. However, the packet loss experienced by the other nodes is negligible.

Similar observations can be drawn by focusing on the delay index (Figure 6). First of all, it should be pointed out that the delay related to nodes that are the sole member of their own ring (e.g., node A in the "AODV 10%" case) is obviously negligible. Even though – in general – the delay in this set of experiments is low, it can be noted that better performance are achieved by using OLSR instead of AODV. Finally, it should be noted that MSRN (node C) always experiences a lower delay with respect to the other nodes in the same ring.

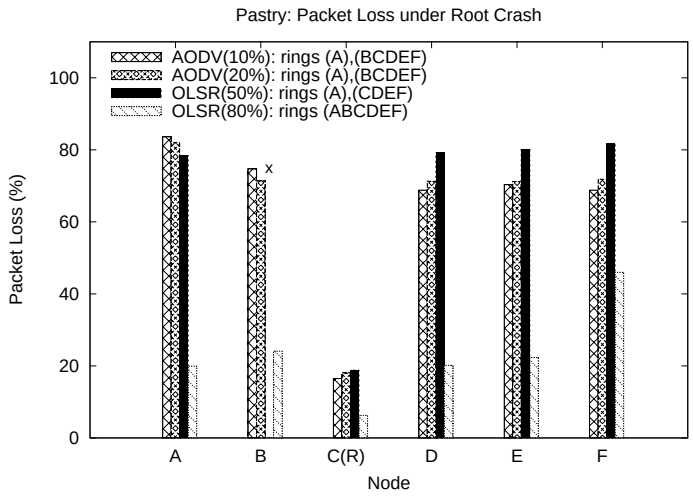

Fig. 7. Packet Loss w/ MSRN crash

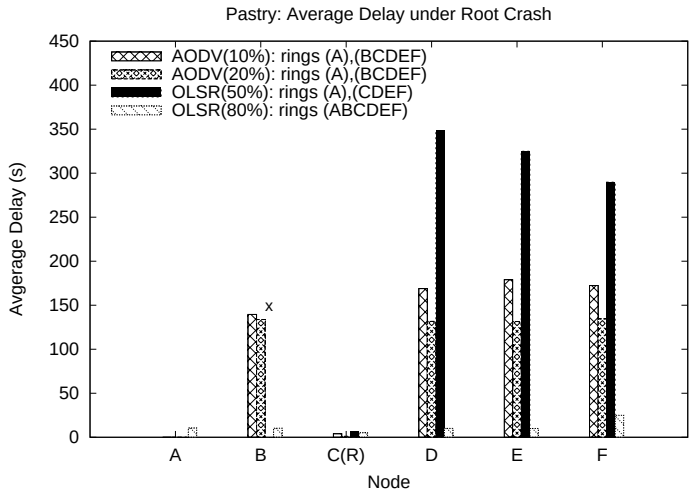

Fig. 8. Delay w/ MSRN crash

Figures 7 and 8 show the packet loss and the delay indices in cases of MSRN crash. The packet loss experienced by nodes in the *same* ring becomes higher than in cases where MSRN does not crash. In the first three experiments, node A isolation causes a packet loss of about 18% on the other nodes. Hence, the remaining 60% packet loss is ascribed to the MSRN crash. Quite surprisingly, OLSR with 80% traffic load shows better performance than OLSR with 50% traffic load. It is also interesting to note that the packet loss at MSRN is always lower than at other nodes in the same ring. This highlights that MSRN is able to get, but unable to *deliver* over the Scribe tree WB messages generated by other nodes. Similar observations can be drawn by looking at Figure 8, as well. The delay experienced by nodes B, D, E and F can be as high as a few *minutes*, either by using AODV or OLSR. Finally, the delay experienced by MSRN is very low in comparison to the delay experienced by the other nodes.

To summarise, the above analysis allows us to draw the following observations. The Pastry bootstrap algorithm is too weak to work well in MANETs, and produces unrecoverable partitions of the overlay network. This behavior is generally exacerbated by AODV (in comparison to OLSR). Furthermore, MSRN is clearly a bottleneck for Scribe.

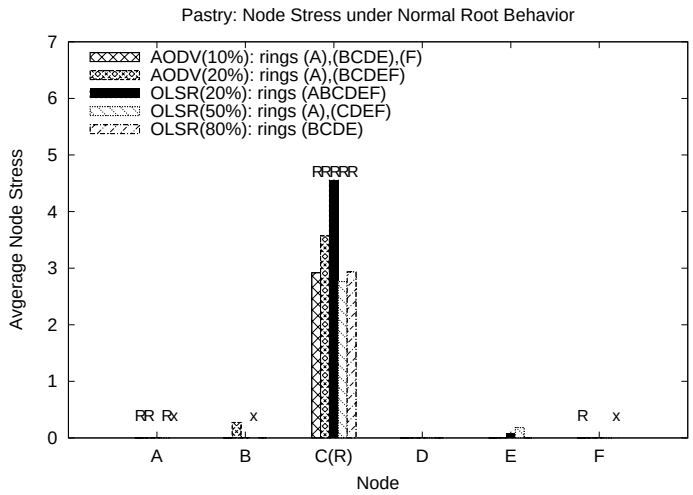

Fig. 9. Node stress w/o MSRN crash

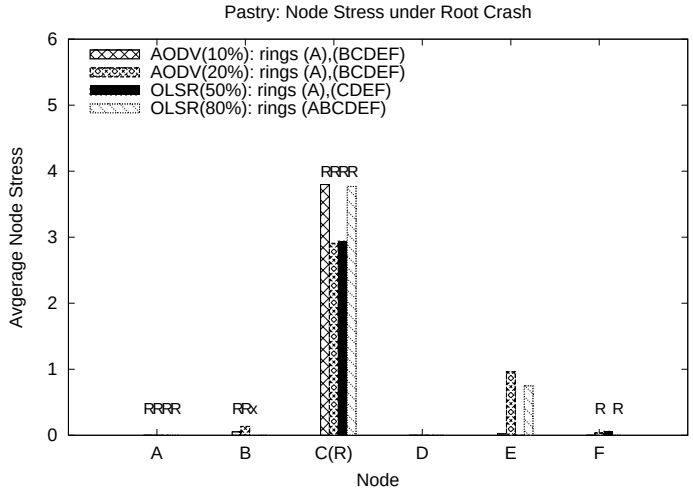

Fig. 10. Node stress w/ MSRN crash

MSRN may be unable to deliver WB messages also with moderate traffic loads, resulting in extremely high packet loss and delay. Moreover, the performance of the system in terms of packet loss and delay is unpredictable. With the same protocols and traffic load (e.g., OLSR and 50% traffic load), MSRN may crash or may not, resulting in completely different performance figures. In cases where MSRN crashes, packet loss and delay are clearly too high for WB to be actually used by real users. However, even when MSRN does not crash, the high probability of WB users to be isolated from the overlay network makes Pastry-based solutions too unreliable. These results suggest that Pastry and Scribe need to be highly improved to actually support group communication applications such as WB in MANET environments.

B. Multicast Tree Quality

In this section we analyse the node stress and re-subscription indices, with respect to the same experiments used in the previous section.

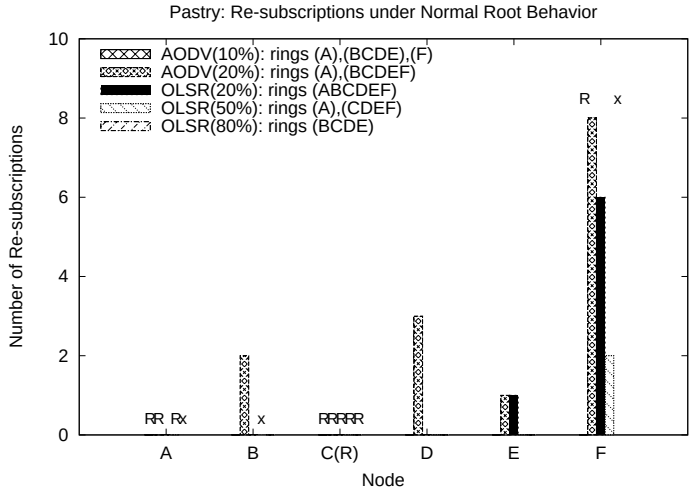

Fig. 11. Re-subscriptions w/o MSRN crash

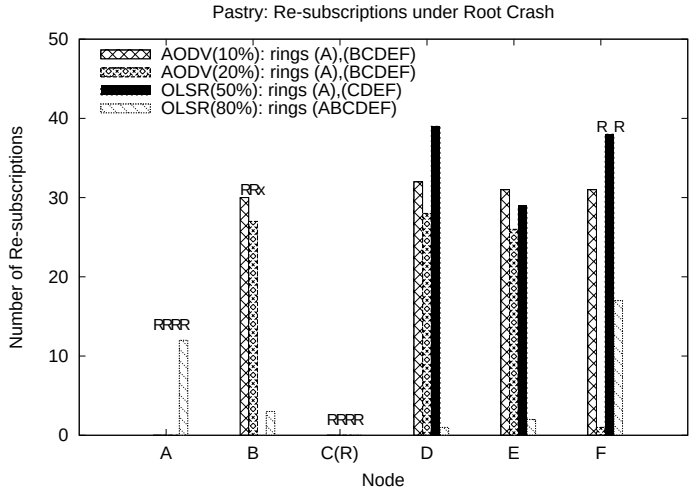

Fig. 12. Re-subscriptions w/ MSRN crash

Figures 9 and 10 plot the average node stress with and without MSRN crashes, respectively. In both cases, the node stress is significantly higher at MSRN than at any other node. This means that the Scribe tree is a one-level tree, and MSRN is the parent of *all* the other nodes. This behavior is expected, and can be explained by recalling the way Scribe works. In our moderate-scale MANET, all nodes are in the Pastry routing table of each other. Hence, Scribe `join` messages reach MSRN as the first hop, and MSRN becomes the parent of all other nodes (in the same ring). Together with the way application-level messages are delivered, this phenomenon explains why MSRN is a bottleneck, since it has to send a distinct message to *each* child when delivering WB messages over the tree. This is a major limitation of the Scribe algorithm, and optimisations of the P2P system are clearly not sufficient to cope with it.

In Figures 9 and 10 we have added “R” labels to indicate nodes that occur to become Scribe Root during the corresponding experiment. When MSRN does not crash (Figure 9) other nodes become Scribe root only as a side effect of a failed Pastry bootstrap. On an isolated WB node, Scribe builds a tree which consists only of the node itself, that is thus the root. However, Scribe partitions may also occur due to congestion at the Pastry level in cases

where MSRN crashes. By looking at Figure 10, it can be noticed that nodes other than MSRN may become root also if they belonged (after the Pastry bootstrap phase) to the same overlay network of MSRN. This phenomenon occurs, for example, at node A in the OLSR 80% case, and at node B and F (whenever they become root). It should be noted that a node with id n_1 (other than MSRN) becomes root when *i)* it looses its previous parent, and *ii)* the Pastry routing table does not contain another node id n_2 closer to the WB topic id than n_1. Figure 10 shows that the congestion at the Pastry level is so high that the Pastry routing table of some nodes becomes incomplete (i.e., MSRN disappears from other nodes' routing table). Thus, the Scribe tree gets partitioned in several isolated sub-trees. Clearly, this contributes to the high packet loss measured in these experiments. Another effect of Pastry congestion during MSRN crashes is a possible reshaping of the Scribe tree. Figure 10 shows that the average Node Stress of E is close to 1 in the "AODV 20%" and "OLSR 80%" cases. This means that MSRN disappears from the Pastry routing table of some node, which – instead of becoming a new root – finds node E to be the closest one to the WB topic id. This phenomenon could be considered a benefit, since it reduces the MSRN node stress. However, it derives from an incorrect view of the network at the Pastry level, originated from congestion.

Figures 11 and 12 show the re-subscription index for the same set of experiments. Figure 11 shows that, when MSRN does not crash, the Scribe tree is quite stable. Most of the re-subscriptions occur at node F, which is the "less connected" node in the network (see Figure 4). In these experiments, the performance in the AODV cases is worse than in OLSR cases. Furthermore, upon MSRN crashes (Figure 12), the number of re-subscriptions increases drastically, even in case of "well-connected nodes" (i.e., node B, D and E). MSRN crashes make other nodes unable to get messages from their parent (i.e., MSRN itself), increasing the number of re-subscriptions. It is interesting to point out that this is a typical positive-feedback control loop: the more MSRN is congested, the more re-subscriptions are sent, the more congestion is generated.

To summarise, the multicast tree generated by Scribe on top of Pastry is quite unstable, especially in cases of MSRN crashes. The tree may get partitioned in disjoint sub-trees, and many re-subscriptions are generated by nodes. Furthermore, Scribe is not able to generate a well-balanced multicast tree, since MSRN is the parent of all the other nodes. Directions to optimise Scribe are discussed in Section VIII.

VII. Performance with CrossROAD

In this section we show that using a P2P system optimised for MANETs is highly beneficial to the stability of the Scribe tree. In this set of experiments, we use CrossROAD instead of Pastry, and set the traffic load to 20%, 50% and 100%, respectively. We concentrate on the performance figures related to the quality of the multicast tree, i.e., the average node stress (Figure 13) and the number of re-subscriptions (Figure 14). A complete evaluation of the User Satisfaction parameters, as well as further optimisations of the Scribe algorithm, are subjects of future work.

The first main improvement achieved by using CrossROAD is that neither the overlay network nor the Scribe tree get partitioned. CrossROAD is able to build a *single* overlay network in all the experiments. Furthermore, even at very high traffic loads (e.g., 100%), MSRN is the *only* root of the Scribe tree. Therefore, CrossROAD is able to overcome

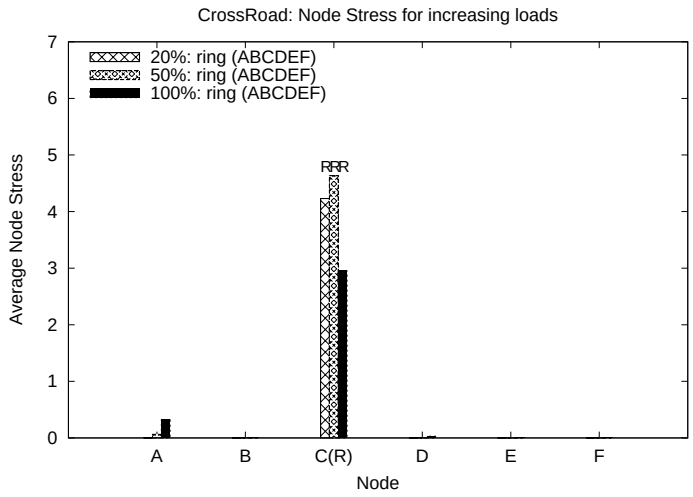

Fig. 13. Node Stress with CrossROAD

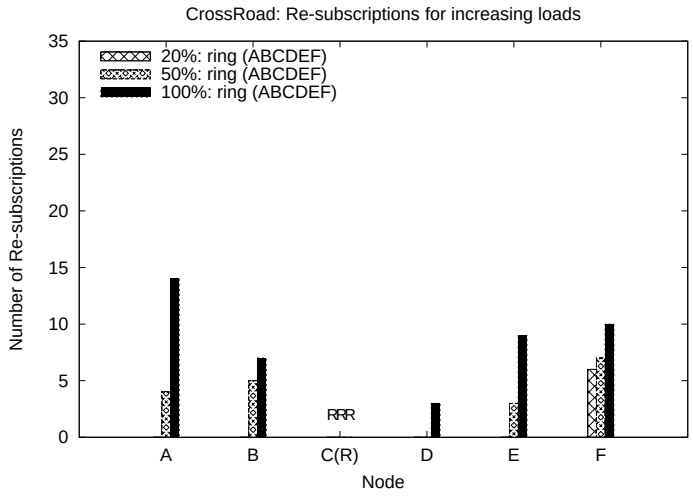

Fig. 14. Re-subscriptions with CrossROAD

all the partition problems experienced when Pastry is used.

Figure 13 clearly shows that the node stress still remains quite unbalanced among the nodes. MSRN is typically the parent of all the other nodes, and this contributes to make it a bottleneck of the system, as highlighted above. This behavior is expected, since it derives from the Scribe algorithm, and cannot be modified by changing P2P system.

Finally, Figure 14 shows that the Scribe tree is more stable (i.e., requires less re-subscriptions) using CrossROAD instead of Pastry. To be fair, we have to compare Figure 14 with both Figures 11 and 12. It is clear that CrossROAD outperforms Pastry when used on top of AODV. The “20%” case of CrossROAD should be compared with the “OLSR 20%” case of Figure 11, since in both experiments the overlay network is made up of all nodes. The number of re-subscriptions measured at node F is the same in both cases, while it is higher at node E when Pastry is used. The CrossROAD “50%” case shows a higher number of re-subscriptions with respect to the “OLSR 50%” case in Figure 11. However, it should be noted that in the latter case the overlay network encompasses less nodes, and hence the congestion is lower. It should also be noted that, with the same nodes in the overlay network, the same protocol stack

and traffic load, Pastry experiments may suffer MSRN crashes (Figure 12). In this case, the number of re-subscriptions is much higher than in the CrossROAD case. Finally, results in the CrossROAD "100%" case should be compared with the "OLSR 80%" case of Figure 12, since the overlay network is the same in both experiments. CrossROAD achieves comparable performance, and at some nodes it outperforms Pastry, even if the application traffic is significantly higher.

A. Overlay-Network Management Cost

In the previous section we have shown that adopting CrossROAD significantly improves the performance of Scribe. In this section we highlight that one of the main reasons for this improvement is the big reduction of the network overhead. This is a key advantage in MANET environments.

Figure 15 shows the network load experienced by nodes A, C and the two nodes which just act as routers, during the Pastry "OLSR 80%" experiment in which MSRN crashes[4]. Each point in the plot is computed as the aggregate throughput (in the sending and receiving directions) over the previous 5-seconds time frame. We take into consideration the traffic related to the whole network stack, from the routing up to the application layer. Specifically, nodes A and C are representative for WB nodes, pointing out the difference with nodes that just work as routers. The discrepancy between the curves related to node A and C confirms that the MSRN node has to handle a far greater amount of traffic with respect to the other WB nodes, due to the Scribe mechanisms. Furthermore, it should be noted that the curves related to the two routers can hardly been distinguished in Figure 15, since they are about $400Bps$. This means that the lion's share of the load on WB nodes is related to Pastry, Scribe and the WB application.

Figure 16 plots the same curves, but related to the "100%" CrossROAD experiment. Also in this case, MSRN (node C) is more loaded than the other WB nodes. However, by comparing Figures 16 and 15 we can highlight that the Pastry network load is far higher than the CrossROAD network load. By considering the average value over all nodes in the MANET, the Pastry load is about 3 times greater than the CrossROAD load. More specifically, the average load of C and A is 48.5 KB/s and 16.5 KB/s in the Pastry case, while drops to 21.1 KB/s and 2.96 KB/s in the CrossROAD case. The reduction of the network load achieved by CrossROAD is thus 56% at node C and 82% at node A. Since the other stack components are exactly the same, CrossROAD is responsible for this reduction[5]. Furthermore, it should be noted that, during several time intervals, the load of node A is just slightly higher than that of "routing" nodes. This suggests that the additional load of CrossROAD management with respect to the routing protocol is very limited.

VIII. Conclusions and Open Directions

Results presented in this chapter allows us to draw the following conclusions. Pastry and Scribe seem not to be good candidates to support Group-Communication Applications in MANET environments. Pastry is particularly weak during the bootstrap phase, causing the overlay network to be partitioned into several subnetworks, and some nodes to be unable to join application services. Further partitions may occur in the Scribe tree due to congestion at the Pastry level. Finally, the delivery algorithm implemented by Scribe generates a severe bottleneck in the tree, which is highly prone

[4]We do not take into account AODV experiments, since OLSR has clearly shown to outperform AODV.
[5]The actual reduction is even higher, since the application-level traffic is 100% in the CrossROAD case.

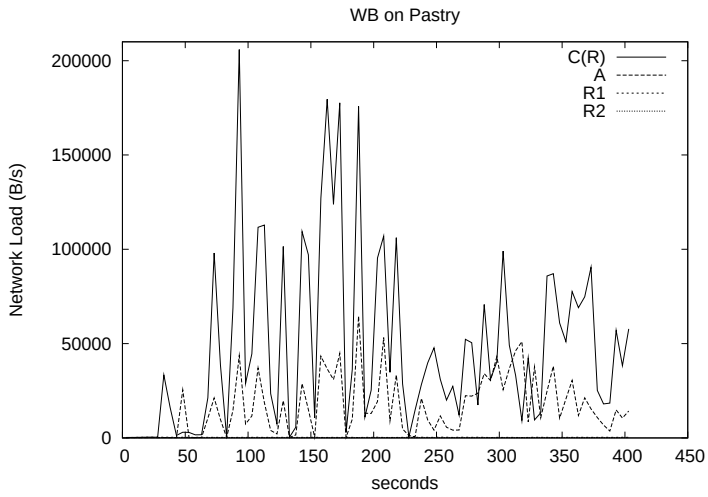

Fig. 15. Network Load with Pastry

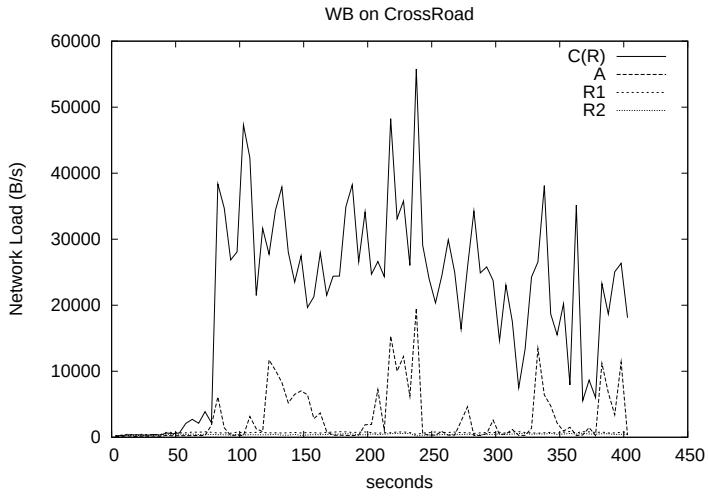

Fig. 16. Network Load with CrossROAD

to get overladed. All these limitations result in unacceptable levels of packet loss and delay for applications. Many of these problems can be avoided by adopting a cross-layer optimised P2P system such as CrossROAD. Thanks to the interactions with a proactive routing protocol, CrossROAD is able to avoid all the partitioning problems experienced by Pastry, and to drastically reduce the network overhead.

Clearly, CrossROAD cannot solve the problem of bottlenecks in the Scribe trees. Therefore, optimised versions of Scribe are required for group communication applications such as WB to be really developed in MANETs. What we believe is really needed is a multicast system that still allows topic-based routing (like Scribe), but also implements multicast at the network level efficiently. The lessons learned while designing and developing CrossROAD are of paramount importance in respect of this. CrossROAD actually achieves a similar goal, since it allows nodes to route data in a topic-based fashion, guaranteeing efficient data delivery at the network level. Actually, extending such a cross-layer approach also to P2P multicast systems seems to be a very promising direction.

REFERENCES

[1] D.A. Agarwal, D.A. Tran, and M.D. Yarvis Editors, International Journal of Wireless and Mobile Computing (IJWMC), Special Issue on Group Communications in Ad Hoc Networks, Inderscience Publishers, vol. 3, 2005.

[2] G. Anastasi, E. Borgia, M. Conti, E. Gregori and A. Passarella, "Understanding the Real Behavior of Mote and 802.11 Ad hoc Networks: an Experimental Approach", Pervasive and Mobile Computing, Vol. 1, Issue 2, pp. 237-256, July 2005.

[3] AODV, Dept. of Information technology at Uppsala University (Sweden), http://user.it.uu.se/ henrikl/aodv/.

[4] O. Babaoglu, R. Davoli, and A. Montresor, "Group communication in partitionable systems: Specification and algorithms", IEEE Transactions on Software Engineering, 27(4), pp. 308-336, Apr. 2001.

[5] E.M. Royer and C.E. Perkins, "Multicast Operation of the Ad-hoc On-Demand Distance Vector Routing Protocol", ACM MobiCom, Seattle, Washington, 1999.

[6] D. Bottazzi, A. Corradi, R. Montanari, "Context-Awareness for Impromptu Collaboration in MANETs", Second IEEE Annual Conference on Wireless On-demand Network Systems and Services (WONS05), St. Moritz, Switzerland, Jan. 19-21, 2005.

[7] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "SCRIBE: A large-scale and decentralised application-level multicast infrastructure", IEEE Journal on Selected Areas in Communication (JSAC), Vol. 20, No, 8, October 2002.

[8] M. Castro, M. B. Jones, A-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang and A. Wolman, "An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer overlays", Infocom 2003, San Francisco, CA, April, 2003.

[9] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High bandwidth multicast in cooperative environments", Proc. of ACM SOSP, October 2003.

[10] F. Dabek and B. Zhao and P. Druschel and J. Kubiatowicz and I. Stoica, "Towards a common API for Structured Peer-to-Peer Overlays", Proc. of the the 2nd International Workshop on Peer-to-peer Systems (IPTPS'03), Berkeley, CA, Feb. 2003.

[11] M. Conti, G. Maselli, G. Turi, and S. Giordano, "Cross layering in mobile ad hoc network design", IEEE Computer, Feb. 2004.

[12] Carzaniga, A. et al., "Achieving scalability and expressiveness in an Internet-scale event notification service". In Proc. 19th ACM Symp. on Principles of Distributed Computing, 2000.

[13] F. Delmastro, "From Pastry to CrossROAD: Cross-layer Ring Overlay for Ad hoc networks", in Proc. of Workshop of Mobile Peer-to-Peer 2005, in conjuction with the PerCom 2005 conference, Kauai Island, Hawaii, Mar. 2005.

[14] S.M. Das, Y.C. Hu, C.S.G. Lee, and Y.-H. Lu, "An Efficient Group Communication Protocol for Mobile Robots", 2005 IEEE International Conference on Robotics and Automation (ICRA 2005), Barcelona, Spain, Apr. 18-22, 2005.

[15] M. Ge, S.V. Krishnamurthy, and M. Faloutsos, "Overlay Multicasting for Ad Hoc Networks", Proc. of the Third Annual Mediterranean Ad Hoc Networking Workshop (MedHocNet 2004), June 2004.

[16] D. Greene and D. O'Mahony, "Instant Messaging & Presence Management in Mobile Ad-Hoc Networks", First IEEE PerCom Workshop on Mobile Peer-to-Peer Computing (MP2P 2004), Orlando, FL, Mar. 2004.

[17] C. Gui and P. Mohapatra, "Overlay Multicast for MANETs Using Dynamic Virtual Mesh", ACM/Springer WINET, to appear.

[18] P. Gunningberg,H. Lundgren, E. Nordström and C. Tschudin, "Lessons from Experimental MANET Research", Ad Hoc Networks Journal, (Special Issue on "Ad Hoc Networking for Pervasive Systems"), Vol. 3, Number 2, March 2005.

[19] I. Keidar and D. Dolev, "Totally Ordered Broadcast in the Face of Network Partitions. Exploiting Group Communication for Replication in Partitionable Networks", Chapter 3 of Dependable Network Computing, pp. 51-75, D. Avresky Editor, Kluwer Academic Publications, Jan. 2000.

[20] M.-O. Killijian, R. Cunningham, R. Meier, L. Mazare, and V. Cahill, "Towards group communication for mobile participants", First ACM Workshop on Principles of Mobile Computing (POMC) August 29-30, 2001, Newport, Rhode Island, USA.

[21] S.-J. Lee, W. Su, and M. Gerla, "On-Demand Multicast Routing Protocol in Multihop Wireless Mobile Networks", ACM/Baltzer Mobile Networks and Applications, special issue on Multipoint Communication in Wireless Mobile Networks, 2000.

[22] L. Ji and M.S. Corson, "Explicit Multicasting for Mobile Ad Hoc Networks", Mobile Networks and Applications, Vol. 8, pp. 535-549, 2003.

[23] J. Luo, P.T. Eugster, J.-P. Hubaux, "Probabilistic reliable multicast in ad hoc networks", Ad Hoc Networks, Vol. 2, pp. 369-386, 2004.

[24] P. Mohapatra, C. Gui, J. Li, "Group Communications in Mobile Ad Hoc Networks", IEEE Computer, vol.37,no.2,pp. 52-59, Feb. 2004.

[25] OLSR, Andreas Tonnesen, Institute for informatics at the University of Oslo (Norway), http://www.olsr.org.

[26] S.Ratnasamy, M.Handley, R.Karp, and S.Shenker, ”Application-level multicast using content-addressable networks”, Proc. of the 3rd International Workshop on Networked Group Communication, November 2001.

[27] N. Reijers, R. Cunningham, R. Meier, B. Hughes, G. Gaertner, V. Cahill, “Using group communication to support mobile augmented reality applications”, the 5th IEEE International Symposium on Object-oriented Real-time distributed Computing(ISORC 2002), Crystal City, VA, Apr.29 - May 1, 2002.

[28] G.-C. Roman, Q. Huang, and A. Hazemi, “Consistent group membership in ad hoc networks”, 23rd ACM International Conference on Software Engineering, pages 381-388, May 2001.

[29] A. Rowstron and P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems”, Middleware 2001, Germany, November 2001.

[30] E. Yoneki and J. Bacon, “Dynamic Group Communication in Mobile Peer-to-Peer Environments”, The 20th Annual ACM Symposium on Applied Computing Santa Fe, New Mexico, March 13 -17, 2005.

[31] S. Q. Zhuang, B. Y. Zhang, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, ”Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination”, Proc. of the Eleventh International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), 2001.