P2P Multicast for Pervasive Ad hoc Networks

Franca Delmastro, Andrea Passarella, and Marco Conti
CNR, IIT Institute
Via G. Moruzzi, 1 – 56124 Pisa, Italy
{firstname.lastname}@iit.cnr.it

Abstract

Integrating p2p services in multi-hop ad hoc networks is today a hot topic. General multi-hop networks, and pervasive systems in particular, can greatly benefit from high-level middleware abstractions able to provide a friendly and powerful substrate for applications development. The p2p paradigm is particularly suitable in this case, because ad hoc networks are decentralised, self organising and self healing. Despite the vast literature on p2p systems in legacy wired networks, providing efficient p2p services for wireless ad hoc networks is still an open issue. Furthermore, evaluating legacy p2p services in pervasive environments can give good hints for innovative service architectures and the interoperability of p2p systems in heterogeneous networks. Motivated by these remarks, in this work we focus on p2p *multicast* services. Specifically, starting from a reference solution in legacy wired networks (Scribe), we design a cross-layer optimised protocol (XScribe) that addresses most of the Scribe problems on ad hoc networks. XScribe exploits cross-layer interactions with a proactive routing protocol to manage group membership. Furthermore, it uses a lightweight structure-less approach to deliver data to group members. By jointly using experimental results and analytical models, we show that, with respect to Scribe, XScribe significantly reduces the packet loss and the delay experienced by multicast receivers, and increases the maximum throughput that can be delivered to the multicast groups. We also exploit analytical models to highlight limitations of XScribe, showing that they are actually general problems of p2p multicast, due to delivering messages via pure p2p policies. We finally suggest that the direction to address this issue is to extend the cross-layer interactions among the routing and the p2p level to the data-delivery phase, as well.

Index Terms

Multi-hop ad hoc networks, p2p systems, multicast, cross-layer

I. Introduction

A. Background and Motivations

During the last few years there has been increasing interest to integrate p2p systems and multi-hop ad hoc networks (see, for example, [1], [2]), because they share a number of common features. Both are completely self organising, decentralised and self healing. Both allow users to join and leave dynamically, with possibly high churn rates. Both are good platforms to host “p2p” applications in which

nodes directly communicate with each other without any centralised service. Furthermore, p2p systems could help developing sensible applications for multi-hop ad hoc networks, which still represents a great barrier for this technology to become truly pervasive. Specifically, applications for pervasive networks definitely require middleware-level services that provide abstractions from large sets of physical devices. The decentralised and dynamic nature of pervasive systems makes the p2p paradigm suitable for providing such middleware-level abstraction.

One of the main obstacles to legacy p2p systems integration in multi-hop ad hoc networks is the fact that p2p systems are typically based on opposite assumptions with respect to what really holds in ad hoc networks. Legacy p2p systems are designed to scale up to thousands of nodes. Furthermore, the networking environment for which they are thought is typically resource rich, especially in terms of bandwidth. Thus, legacy p2p systems usually trade increased bandwidth consumption for higher scalability. Scalability to thousands of nodes is not a major issue for *flat* multi-hop ad hoc networks. Actually, theoretical and experimental results show that large-scale flat ad hoc networks are not very likely, because of intrinsic wireless capacity constraints [3], [4]. Based on these observations [4] defines an ”ad hoc horizon” for realistic flat ad hoc networks, consisting of 10 to 20 nodes, with peer-to-peer communications spanning 2 to 3 hops. Furthermore, ad hoc networks are definitely not a resource rich environment, particularly in terms of bandwidth. Thus, trading increasing bandwidth consumption for greater scalability is not the best design choice. Experimental results show that, because of this mismatch, legacy p2p systems usually fail when used ”as-they-are” in multi-hop ad hoc environments [5], [6]. One of the main issues is therefore how to *efficiently* provide the same kind of p2p services implemented in legacy wired networks also in multi-hop ad hoc networks. Fortunately, previous studies show that p2p systems implementing DHTs can be designed to achieve high performance in ad hoc networks too (e.g. CrossROAD [7] and Virtual Ring Routing (VRR) [8]). However, a lot of work has still to be done to extend these results to more complex p2p platforms.

Besides efficiency, another main concern to be addressed is the portability of legacy p2p applications on ad hoc networks, and the interoperability between p2p systems running on heterogeneous (wired/wireless) networks. P2p applications can represent a great inheritance for wireless networks and pervasive environments improving the popularity of this technology among users communities. The CommonAPI defined in [9] represents a first step in this direction even though it has been originally defined for wired p2p systems. In fact, it defines interfaces between components of a p2p system and towards the applications, thus granting easy portability of p2p applications across different p2p systems' implementations. Then, to port distributed applications on wireless networks, a cross-layer extension of the original common API (named *X*L-CommonAPI) has been defined in [10], with special emphasis on cross-layer support to the upper p2p layers. The XL-CommonAPI exports fundamental routing- and DHT-level information to upper-layer services, thus allowing them to exploit complete knowledge about both the network topology and the overlay status. Exploiting cross-layer information has proved to be very important for both CrossROAD and VRR. Besides this, designing p2p components on top of the XL-CommonAPI grants easy portability of applications across wired and wireless networks, and their

interoperability in heterogeneous environments.

B. Contributions

Starting from these considerations, in this paper we focus on p2p *multicast* services. Specifically, we consider p2p multicast protocols running on top of a p2p substrate providing a DHT (in Section III we discuss the reasons of this architectural choice). We choose a legacy solution made up of Pastry (at the DHT level), and Scribe (at the multicast level), since it is one of the most efficient solutions designed for wired networks [11].

After recalling the main features of Pastry and Scribe (Sections III and IV), we highlight the inefficiencies of such a p2p system when used as-it-is on ad hoc networks (Section IV-B). Based on these remarks, we propose a new solution that leverages cross-layer interactions between the p2p and the routing levels to optimise the multicast system implementation (see Figure 1). Specifically, we replace Pastry with CrossROAD (whose main features are recalled in Section III), and Scribe with XScribe (which are extensively described in Section IV). Since CrossROAD implements the XL-CommonAPI, and XScribe exploits cross-layer interactions via this interface, this is also an example of how to exploit the XL-CommonAPI to optimise p2p services for ad hoc networks.

While CrossROAD has already proved to outperform Pastry on ad hoc networks (e.g., [12]), the main focus of this paper is the design and evaluation of XScribe. The main advantages of XScribe over Scribe are: *i)* managing group membership with minimal overhead by exploiting the periodic traffic of a proactive routing protocol and *ii)* delivering multicast data to intended receivers without requiring any networking structure (such as trees or meshes) built and maintained exclusively for multicasting purposes. In this sense, XScribe is a *stateless, cross-layer, p2p* multicast protocol.

In Section V we evaluate the XScribe performance in comparison with Scribe, in terms of packet loss and delay. We report results from experiments run on a real multi-hop ad hoc network, implementing both p2p solutions. Results show that, in the majority of the cases, XScribe is able to halve the packet loss and the delay experienced by Scribe *at the same time*.

To better understand the p2p systems' behavior, in Section VI we develop an analytical model, which provide expressions for the *saturation throughput* of both Scribe and XScribe. The saturation throughput is defined as the maximum throughput that the multicast system is able to fairly deliver from groups' senders to groups' receivers. After validating the model by experimental results, we analyse the scalability bounds of Scribe and XScribe varying several parameters, i.e., the number of sender and receivers, the network size, and the number of multicast groups active at the same time. The model demonstrates that both systems can be used in small- and medium-size networks, i.e., within the ad hoc horizon. Specifically, in these scenarios they are able to support reference applications such as multi-player games [13]. Furthermore, we also highlight that XScribe is able to increase the scalability bounds of Scribe with respect to all the considered parameters. Specifically, the proportional increase of the saturation throughput achieved by XScribe can be as high as 50%. Despite its good performance, results also suggest XScribe limitations and ways to further improve it. Specifically, XScribe implements

a pure p2p delivery policy, in which no information about physical paths is exploited. We highlight that this is the main direction along which XScribe can be further improved, by blending together p2p multicast features and layer-3 multicast mechanisms.

II. Related work

Multicast support implemented at the p2p layer is just one of the available options proposed in the literature. Usually, multicast protocols are classified as operating at the network layer (L3), or at the application layer, where application denotes all possible layers above the transport. Application-level multicast runs only at nodes involved in the related application and it just requires standard unicast support from the routing level. The most recent proposals for wired environments ([14] [15] [16] [17]) run the multicast protocol on top of an overlay network based on a DHT. This approach is very interesting for several reasons, as discussed in Section III. To the best of our knowledge, the feasibility of this approach on multi-hop ad hoc networks has not been investigated yet. Application-level multicast has been proposed also for ad hoc networks (e.g., ALMA [18] and PAST-DM [19]), even though it is not implemented on top of a DHT. We believe that exploiting a DHT to implement multicast services is an interesting direction to investigate also in ad hoc networks.

On the other hand, it should be pointed out that application-level multicast potentially generates path stretch because just a subset of nodes can be used to deliver the data. Moreover, nodes that do not participate in multicast groups have to forward data nevertheless. Therefore, it is not yet clear whether multicast for ad hoc networks should be implemented at the routing or at the application level. Examples of multicast protocols for ad hoc networks implemented at the routing level are MAODV [20] and ODMRP [21] (other proposals, either implemented at the application or at the routing level, are described in [22]). From this standpoint, there is actually an increasing trend towards blending together, in ad hoc networks' stacks, features usually implemented at the routing and at the p2p levels (e.g., [8], [23], [7]). Therefore, an interesting solution could be a multicast system implemented within an integrated routing layer that also includes p2p features. Results presented in this paper can be seen as an helpful intermediate step towards this eventual goal.

The vast majority of multicast protocols for ad hoc networks proposed in the literature adopts structured approaches, i.e., the multicast protocol builds and maintains networking structures exclusively related to data delivery. For example, MAODV and ALMA use shared trees, while ODRMP and PAST-DM use meshes. A few others implement a structure-less approach, in which the multicast protocol does not rely on any structure exclusively related to multicast. Structure-less multicast requires that each sender is aware of at least a fraction of the receivers of the multicast group (we will refer to this property as *receiver awareness* in the following). Receiver awareness allows senders to directly deliver messages to receivers along the paths already built by the underlying routing protocol. Basically, receiver awareness replaces multicast structures. Examples of structure-less multicast protocols for ad hoc networks are DDM [24] and RDG [25]. A structure-less approach avoids the cost of maintaining another networking structure separated from the one already provided by the routing protocol. The main drawback of such

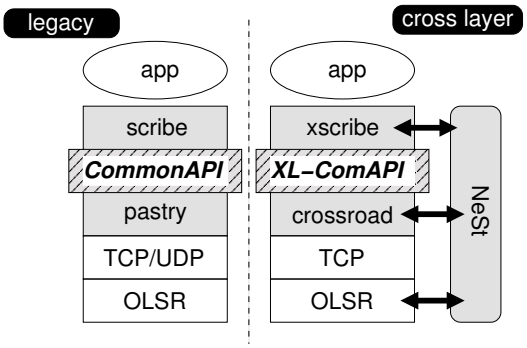


Fig. 1. Network architectures.

an approach is clearly the cost of implementing receiver awareness (usually, receivers have to register with the senders, which then periodically poll them with alive messages).

XScribe actually follows a structure-less approach, since it exploits the network structure already defined by the underlying layers. With respect to DDM and RDG, XScribe works on top of a DHT and can straightforwardly support p2p applications. Furthermore, it implements receiver awareness in a more efficient way than periodic polling, and thus it better exploits the advantages of a structure-less design.

This paper complements our previous work on p2p multicast for multi-hop ad hoc networks. Specifically, in [6], [26] we extensively evaluated the performance of Pastry and Scribe in ad hoc networks. In this work we focus on the comparison between the legacy solution (Pastry & Scribe), and the cross-layer one (CrossROAD & XScribe). The main design features of XScribe have been presented in [27], together with a preliminary evaluation. In this paper we provide a more complete evaluation, both from an experimental and from an analytical standpoint.

III. DHTs for P2P multicast

Before describing the features of the p2p platforms shown in Figure 1, it is worth briefly discussing why using a DHT to support p2p multicast is an interesting idea, and why using *structured* overlay networks is a reasonable choice. Using a DHT below the p2p multicast level is 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.

Being mainly designed for handling exact-match queries, a structured overlay (i.e., a DHT) is the most natural support for a p2p multicast protocol, in which single nodes have to send data to a well-defined set of receivers. In comparison with unstructured and hybrid overlays, structured overlays are usually considered *i)* less efficient in terms of management, *ii)* not able to exploit nodes' heterogeneity, and *iii)* not able to support content-based queries. However, the discussion on these points is still open. For example, [28] shows that it is possible to address points *i)-iii)* by still using Pastry, and by

achieving performance at least comparable to that provided by unstructured overlay networks. Thus, even though conceptually interesting, porting our analysis of p2p multicast protocols to unstructured and hybrid overlays is not the most compelling issue.

A. Pastry vs. CrossROAD

Pastry defines a DHT using a logical circular address space. The logical address of a node is the hashed value of its IP address (a proper hash function maps IP addresses and strings to logical addresses). A key is associated to each message sent on the overlay. Pastry delivers the message to the node whose logical id is the closest one to the hashed key. For the sake of scalability, each node keeps a partial view of the overlay network, i.e., it just knows about a limited set of nodes. The nodes that are kept in the set guarantee the forwarding correctness, i.e., that messages sent from anywhere eventually reach the correct destination.

The main costs of Pastry in terms of networking overhead are due to *i)* the overlay creation and management that require periodic communications between nodes, and *ii)* the multi-hop middleware routing caused by the incomplete knowledge of the overlay at each node, which possibly results in significant path stretches. These costs are well justified in large-scale wired networks, where the ability to scale to large number of nodes (possibly thousands) is correctly traded for additional bandwidth consumption (brought by points *i)* and *ii)* above). However, the cost of this trade-off is turned upside-down in multi-hop ad hoc networks, where the number of nodes is limited, bandwidth is scarce, and paths should be kept as short as possible since nodes communications can be severely affected by unstable links.

CrossROAD has been designed to overcome Pastry's inefficiences on multi-hop ad hoc networks. It provides the same DHT features, but implements them via a cross-layer optimised approach. Specifically, CrossROAD interacts with a *proactive* routing protocol via the cross-layer architecture proposed in [29]. As shown in Figure 1, these interactions are mediated by the NeSt module, which provides standard interfaces for cross layering, thus granting both performance optmisations and stacks manageability.

A node wishing to join a CrossROAD overlay embeds few bytes into periodic routing advertisements, announcing its participation to a specific service. This information eventually reaches all the other nodes in the network through the proactive flooding of the routing protocol. Therefore, every node in the overlay knows the IP addresses of all the other nodes currently running the same service, and it is able to autonomously build the overlay by simply hashing their IP addresses. In this way, CrossROAD drastically reduces the bandwidth overhead with respect to Pastry ([30], [5], [31]), because building the overlay does not requires connections between nodes to exchange overlay information. Furthermore, p2p messages always travel just one-hop on the overlay (assuming that the overlay view is consistent at all nodes), because every node has a *complete* view of the overlay. Even though using a proactive routing protocol over ad hoc networks might sound costly, experimental results [30] showed that the overhead of the proactive routing protocol (OLSR in the particular case) is completely affordable. More details about CrossROAD operations can be found in [7] and [12].

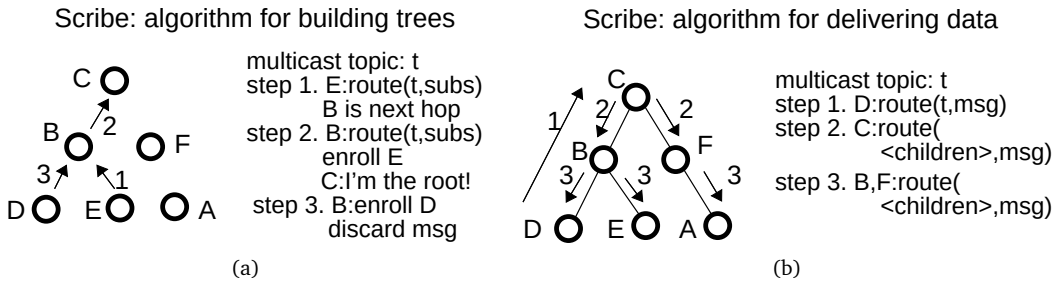


Fig. 2. Scribe tree construction (a), and data delivery (b)

IV. Multicast systems

A. Scribe

Scribe is a p2p shared-tree multicast protocol. It identifies each tree with a topic, and defines a root node for each topic as the node in the overlay whose address is the closest one to the hashed topic. For example, in Figure 2 the root is node C. Multicast trees are built through the well-known reverse path algorithm. Each node willing to join the tree sends a subscribe message specifying the topic as the key. An intermediate node (in the overlay path between the subscribing node and the root) that receives such a message, either subscribes itself to the same topic if it is *not* a member of the tree (e.g., node B after step 1 in Figure 2(a)), or discards the message otherwise (e.g., node B in step 3 in Figure 2(a)). In both cases, it enrolls the node from which it has received the message as a child. Messages to be delivered over the tree are first sent towards the root of the topic (step 1 in Figure 2(b)), and subsequently delivered by each parent to its children (steps 2 and 3 in Figure 2(b)). Parent-child relationships are periodically refreshed through HeartBeat Messages sent by each parent to each child (application messages are also used as implicit HeartBeats). Upon missing a specified number of HeartBeats, a child assumes that the parent is no longer in the overlay, and sends a new subscribe message. This also allows to identify the new root upon a failure or disconnection of the previous one. Children of the old root node detect that the root node is no longer there, and send subscribe messages that eventually arrive to the node currently closest to the topic, i.e., the new root. Nodes at lower levels in the tree are not affected by root failures. Finally, the current root node periodically checks whether it is still the node with the closest id to the topic. If it is not (e.g., because a new node joined the overlay), a new node has to become root. The old root sends a subscribe message to the new one, attaching the whole tree to the new root.

B. Scribe strengths and limitations

The design of Scribe has a number of interesting features. First of all, it allows the application to define multicast groups in a straightforward way by identifying groups with topics translated into logical ids in the DHT address space. Furthermore, when the network is stable enough, it minimizes the tree management traffic thanks to the implicit HeartBeats mechanism. Finally, it leverages the self-organising and self-healing features of the underlying DHT very effectively. The mechanisms described

in the previous section to detect parent failures, recover the tree structure, and managing root changes guarantee loop freedom, and keep recovery traffic local to the nodes actually involved in the failure detection and repair.

Despite these nice properties, Scribe also have features that are definitely not suitable for multi-hop ad hoc networks. Firstly, the data delivery mechanism is highly centralised. Messages have firstly to reach the root, and are then forwarded on the tree. Clearly this is not efficient since it tends to saturate network resources around the root node, as shown by experiments in [32].

Secondly, Scribe uses a structured approach to multicasting, which in general results in higher path stretches with respect to the minimum stretch granted by the DHT. Let us focus again on Figure 2(b), and specifically on the path followed by messages sent by D to reach node E. Even though the mechanism could be optimised by avoiding messages to go through the root node, messages from D to E have to pass through node B anyway, because they have to be delivered over the structure defined by the multicast protocol. In general, it is easy to show that the best path on the overlay between D and E does not necessarily includes B. Thus, such a delivery policy generally extends the physical path between senders and receivers more than what is strictly required by the DHT.

A third problem is also related to the structure defined by Scribe. Under dynamic or unstable conditions, the management traffic might grow significantly. Link instability, which is typical of wireless environments, may actually result in temporaneous nodes disappeareance from the overlay. This may thus cause management traffic at the Scribe level to repair the multicast structure. Overall, if the link instability is caused by network congestion, such recovery mechanisms actually exacerbate the problem instead of contributing to fix it, and may result in tree partitions and nodes' isolation [32].

Finally, the fourth problem we highlight is the fact that Scribe uses a *pure p2p* data delivery mechanism, in the sense that no information about the real paths taken by messages in the network is used. For example, when node B in Figure 2 delivers a message to D and E, it generates two distinct copies of the message. If the paths between B and D, and between B and E, partially overlap, such a pure p2p delivery results in unnecessary replication of message transmissions over the physical network.

C. XScribe

To cope with most of the problems highlighted in Section IV-B, we propose XScribe, which is a replacement for Scribe optimised through cross-layer interactions. Purposely, XScribe still uses a pure p2p data delivery strategy, not addressing the fourth problem highlighted in the previous section. On the other hand, it is able to deal with the other Scribe limitations. XScribe is heavily inspired by *stateless, explicit* multicast approaches such as DDM [24] and RDG [25]. The main differences between these protocols and XScribe are that *i)* XScribe is implemented at the P2P level, and therefore provides a quite more friendly support for applications with respect to standard layer-3 multicast, and *ii)* its group membership policy is implemented via a very efficient cross-layer approach, that drastically reduces the management traffic. For ease of explanation, we divide the XScribe operations into *data dissemination* and *membership management*, and discuss each aspect separately.

1) Data dissemination: As in DDM, in XScribe each sender of a multicast group is aware of all the other group members (we will refer hereafter to this property as *receiver awareness*). Specifically, in XScribe each sender keeps a list with the logical addresses (in the overlay network) of the group members. Locally generated messages are disseminated in the group by sending to each member a distinct message over the overlay network. Thanks to receiver awareness, no structure related to multicast is required, and XScribe is thus a structure-less protocol. Clearly, the main issue of this approach is how to implement receiver awareness efficiently. The membership-management section addresses this aspect.

Even though this data-dissemination mechanism can be further optimised, it addresses the first, second, and third problems we have discussed in Section IV-B. The first advantage of the structure-less approach is to avoid centralisation, since no root node is defined anymore. In addition, XScribe also avoids management traffic and path stretches brought in Scribe by the multicast structure. If we consider a particular sender-receiver pair, in XScribe the sender directly sends messages to the receiver, thus achieving the minimum path length according to the underlying DHT. It is worth recalling that in CrossROAD all the nodes in the overlay are just one hop away from each other. Therefore, messages sent from a multicast sender to a receiver actually travel along the shortest path according to the underlying L3 routing protocol.

Despite these advantages, it is easy to show that the XScribe dissemination policy could be further optimised. The repeated unicast used by XScribe is definitely a feature to be improved. However, note that in small-scale ad hoc networks Scribe tends to build two-level trees in which all the leaf nodes are direct children of the root node[1]. Thus, data dissemination from the root node occurs via repeated unicast in Scribe too. Despite this sub-optimal behavior, the results presented in the following sections show that XScribe is already able to significantly improve Scribe. Furthermore, the current version of XScribe allows us to stress the limits of a pure p2p delivery mechanism applied to multicast, and to understand how to further improve it (see Section VIII for a discussion on this point).

2) Membership management: XScribe uses a cross-layer policy to manage group-membership, inspired by the main principle of CrossROAD. In fact, even in this case, the main idea is to exploit the proactive routing traffic to spread information around.

We assume that multicast groups can be mapped to positions in a *group bitmask*. One could envision a number of ways to define mappings such as, for example, exploiting the fingerprint of the group id in a Bloom filter. XScribe maintains a group bitmask local to each node, which stores the multicast groups the node is subscribed to. As soon as the node subscribes to some groups (i.e., the bitmask is not completely cleared), the local bitmask is embedded into periodic messages generated by the routing protocol, and disseminated in the network. Further subscriptions/unsubscriptions are disseminated by setting/clearing the corresponding bit(s) in the bitmap embedded into routing packets. By inspecting received routing packets each node in the network can be aware of all the members of the available

[1]This is essentially because in small-scale networks nodes tend to be aware of *all* the other nodes at the Pastry level.

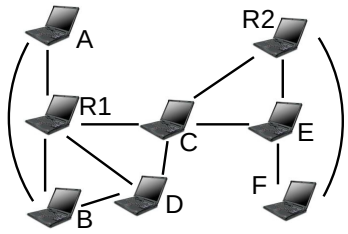


Fig. 3. Main Topology of the experimental testbed

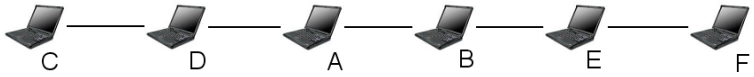


Fig. 4. Chain topology

multicast groups, which is exactly the information required by the data dissemination algorithm. A node is assumed to have ceased to be part of the network when no bitmasks of that node are received for a specified amount of time. Clearly, nodes that do not run the XScribe layer are not be able to join multicast groups, while they are still be able to run other applications based on CrossROAD.

Essentially, this is the same mechanism CrossROAD uses to advertise nodes' presence in the overlay, and it is highly efficient from a bandwidth overhead standpoint. For example, if the bitmask size is 128 bits, and the period used by the routing protocol to send updates is 2 seconds (the default value for Hello packets in OLSR), the overhead injected by a XScribe node is just 8 Bps. Furthermore, XScribe information does not need to be sent in separate frames at the MAC level, not resulting in additional accesses to the shared medium. This is very important whenever the network starts to be even slightly congested. Actually, experimental results related to CrossROAD (see [33], [12]) show that the networking overhead to disseminate p2p-level information through the routing protocol is minimal. Since the additional information required by XScribe is very small, we do not quantitatively investigate this protocol's aspect, while we focus the performance evaluation on data delivery figures.

Finally, note that the membership management policy represents one of the main differences between XScribe and the other stateless multicast protocols such as DDM or RDG. Both DDM and RDG require each receiver to send a join message to each sender of the group. Furthermore, they require periodic polling to refresh membership. These aspects of the protocols are usually neglected in evaluations, but they may represent a significant overhead.

V. Experimental Evaluation

A. Methodology

To compare the legacy and the cross-layer multicast p2p systems we ran experiments in a real ad hoc network setup. Specifically, we used the Pastry and Scribe implementations provided by the Rice University (FreePastry [34]) for the legacy system, while we implemented XScribe on top of CrossROAD, directly exploiting its cross-layer interactions to spread group information on the network. To have a

realistic application environment, we also implemented a simple Whiteboard Application (WB) on top of both systems, allowing users to share drawings, writings, etc. Specifically, users run a Whiteboard instance on nodes of the ad hoc network, select a topic to join to, and then they are able to draw on a canvas, and receive drawings from the other users that selected the same topic. All the nodes whose users subscribe to the same topic form a multicast group. The software implementing the protocol stacks can be downloaded from `http://bruno1.iit.cnr.it/xscribe_exp/`.

The testbed we have used represents a small-scale multi-hop ad hoc network with two different topologies, as shown in Figures 3 and 4. All the nodes were IBM ThinkPad R50 laptops with integrated 802.11b wireless card (Intel PRO-Wireless 2200). The OS was `linux-2.6.12.3`, loading the `ipw2200` driver for the network card. The network topology shown in Figure 3 represents the main scenario of our experimental evaluation. In this case nodes A to E ran the whole protocol stacks, including the WB application, while nodes R1 and R2 just worked as routers. On the other hand, the chain topology shown in Figure 4 has been used to highlight some anomalies of the routing protocol, as deeply explained in the following.

In all the experiments we assumed that all the nodes running the WB application were interested in the same topic. Therefore, the multicast protocol worked with a single multicast group encompassing all nodes. In both topologies, node C was the root of the Scribe tree. In the main topology, nodes C and D generated traffic at the application layer, thus acting as senders of the multicast group, while the other nodes only received application-level traffic. In the chain topology the only sender was node C, and all the other nodes were receivers. Actually, in both scenarios we used `iptables` to emulate multi hopping, and we cross-checked that paths were actually multi-hop by inspecting packet traces collected during the experiments. Although this methodology is not able to completely capture all the effects of wireless links' intricacies, it allows us to closely approximate the system behavior in a realistic multi-hop setting. We did not run experiments in mobile conditions, in order to separate the effects of the different multicast architectures from those of nodes mobility. Analysing the multicast systems in mobile scenarios is one of the main subjects of future work.

To have a controllable and reproducible environment, in our tests WB was not run by humans, but by simulated users. Each user alternated between ON and OFF phases. During ON phases it drew strokes on the canvas, while during OFF phases it did nothing but receiving other users' strokes. Both ON and OFF phases lengths were exponentially distributed. Each trial was composed by 100 active/idle cycles, and in any configuration each node running WB generated at least 500 messages[2]. To make trials start at the same time at different nodes, we synchronised the nodes before each trial, and scheduled the trial to start at the same time on each node. In the following, each trial configuration is identified by the application-load index, measured as the number of Packets Per Second (pps) generated by each user. This index is defined as the ratio between the average number of strokes generated in a cycle, and the average duration of an active/idle cycle. We found that this simple index is sufficient to correctly

[2]A distinct message was sent for each stroke. The size of each message was 1448 bytes.

identify usage cases for our environments.

We characterise the architectures' performance at each node in terms of packet loss and delay statistics. Specifically, the packet loss at node i is measured as $1 - \frac{R_i}{\sum_{j=1}^{N} S_j}$, where R_i is the number of messages received by node i, N the number of senders in the group, and S_j is the number of messages generated by the j-th sender. Packet delays were measured by timestamping the transmission time at the sender, and the reception time at the receiver (recall that nodes were synchronised). For nodes acting as senders, we did not take into account locally generated packets to compute the performance figures. Each configuration was replicated for 5 trials to have i.i.d. samples of the packet loss and delay statistics. For each configuration, we hereafter provide average values and confidence intervals of the performance figures (with 90% confidence level, unless otherwise stated).

B. The chain topology

Since experimental results obtained in the main scenario presented some anomalies strictly dependent on the routing protocol, we decided to run some trials on the chain topology, to have a simpler setup and achieve an easier interpretation of the routing behavior. In this setup Scribe and XScribe behavior is remarkably similar. In the case of Scribe, since the root node is the only sender, it does not represent a centralisation point, since it does not receive traffic to be forwarded. Furthermore, the traffic pattern generated by Scribe and XScribe to deliver application-level data to receivers is the same (i.e., data generated by C are delivered to each receiver on a dedicated TCP connection in both cases). Actually, since XScribe adds group-membership information to routing packets, the routing protocol behaves slightly different in the two cases, but the effects of routing protocol anomalies are very similar. For these reasons, in this section we discuss results from XScribe experiments, that better highlight these anomalies. Specifically, we discuss a single trial in which node C generates 20pps (similar remarks apply also to the other configuration that we have used).

In this specific experiment nodes A, B, and D measure no packet loss, while nodes E and F measure 37% and 40% packet loss, respectively. The log files show that node F loses *all* the routes for a few seconds during the trial, while in the same period node E has only a valid route to F (while the routes to all the other nodes disappeare). At the same time, all the other nodes (i.e., from C to B) lose the routes to both E and F. This fact is not so surprising in this kind of scenario, since it is well-known that in a multi-hop chain topology the loss of some Hello packets can cause a network partition.

Unfortunately, these route failures cause spikes of application messages delays. In correspondence of these events, nodes E and F experience delays in the order of 20 seconds, while the other nodes experience delays no greater than 5 seconds throughout the whole experiment. It is quite surprising that such route losses, and the associated delay spikes, also occur at very light traffic loads (5 pps), i.e., when the network is not congested. This suggests that they could be caused by some misbehavior of the routing protocol. To investigate this hypothesis, we deeply analysed the routing behaviour directly examining the packets sent on the network through the `tcpdump` sniffer.

According to the standard OLSR specification [35], Hello messages should be sent every 2 seconds, and TC messages (aggregating the link state of several nodes) every 5 seconds, or upon a topology change. Links advertised by Hello and TC messages are valid for the next 6 and 15 seconds, respectively. If they are not refreshed within these validity timeouts, they are removed from the routing table. Referring to the OLSR implementation we used we noticed that, even though Hello and TC messages are generated on time by the OLSR process (i.e., precisely every 2 and 5 seconds), they can experience significant delays either in being actually sent, or in being processed at the receiving side. Such delays occur because *i)* this specific implementation aggregates routing messages in single packets to reduce the network load, and *ii)* routing packets are not prioritised at the MAC level, and are thus queued together with application-level packets (especially at high taffic loads).

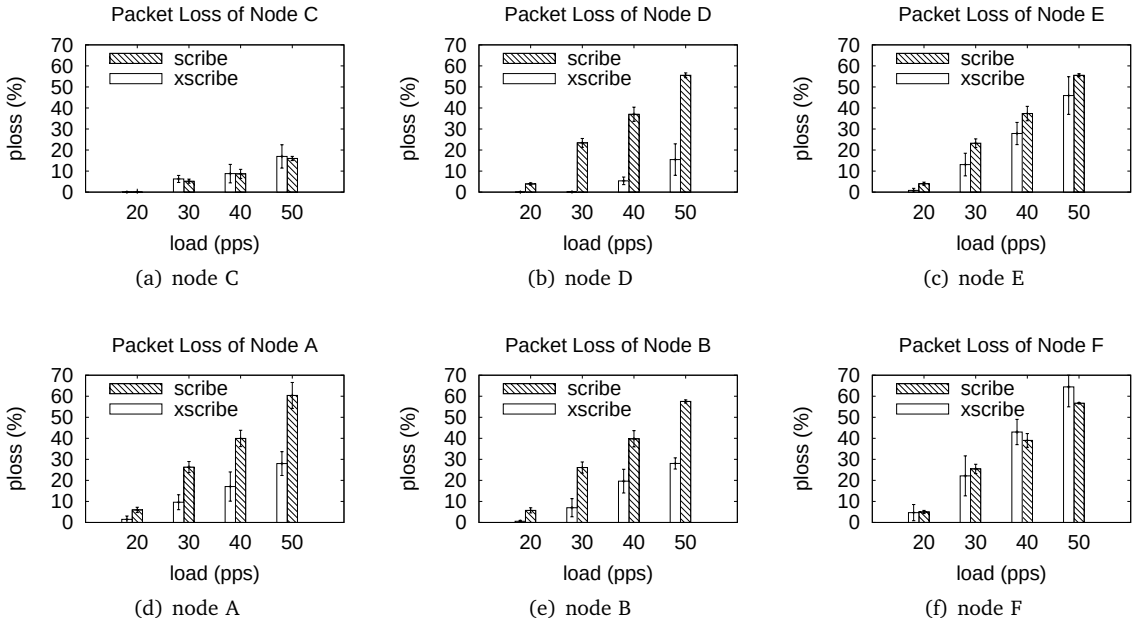
The delayed sending of routing packets makes thus the entire system more susceptible to route failures. Maintaining the original values of routes validity timeouts, it can be sufficient to lose one routing packet, or receive it with a delay higher than the timeout, to cause the route failure. In our specific trial, node F loses an Hello packet sent by node E, and receives the next packet 10 seconds after the last one correctly received. This causes the expiration of the validity timeout in F's routing table for the link with E, and the consequent route failures. Similar events have been noticed at node E, i.e. it loses an Hello from B, and all the related routes. This example highlights how the additional delays of routing messages weaken the protocol reaction to packet loss.

Such misbehaviors partly depend on the particular routing protocol implementation, but are highly determined by the fact that the routing traffic is not prioritised at the MAC level. Anyway, we decided to eliminate from our logs spikes in application message delays that occur because of these routing anomalies. This allows us to evaluate the p2p multicast solution in an optimistic case, in which the routing protocol behaves as expected, and we can fairly compare the architectural differences between Scribe and XScribe.

C. Main evaluation scenario

The main scenario is represented by the network topology shown in Figure 3. In this case the topology is characterised by multiple paths connecting pairs of nodes, and the root node (C) is located at the center of the network, thus being well-connected with all the other nodes. In the following subsections we deeply analyse the two performance indices, i.e., the packet loss (Section V-C.1), and the delay (Section V-C.2).

1) Packet loss: Figure 5 shows the packet loss experienced by each node under the two alternative architectures as a function of the application load (loads below 20pps resulted in no packet loss, and are thus omitted). Plots are presented starting from the center of the topology (node C), towards the edges. Since both architectures use TCP at the transport layer, one could expect not to see any packet loss. Actually, both Pastry and CrossROAD use internal queues (of the same size) to store messages going to be sent. Packet loss actually occurs when these queues fill up, and is thus a side effect of



(a) node C (b) node D (c) node E

(d) node A (e) node B (f) node F

Fig. 5. Packet loss at each node

network congestion[3].

Several interesting observations can be drawn from these plots. In general, it is evident that XScribe drastically reduces the packet loss with respect to Scribe. In more details, at nodes D, A, and B the packet loss is always more than halved with respect to Scribe. At node E the packet loss experienced by Xscribe is lower than in case of Scribe but the reduction is less marked. Instead, different observations should be drawn with respect to node C (the Scribe root), and node F. Referring to node C, the packet loss experienced by the two systems is almost the same at the different traffic loads, and is reasonably low. Actually, this had to be expected. In fact, both in Scribe and XScribe, node C receives only packets from node D. Since the packet loss is mainly due to overflows at the sender's queue, the two systems experience the same performance. On the other hand, analysing the packet loss measured by node F, XScribe experiences slightly higher values only for high traffic loads (i.e. 40 and 50 pps). Note that in case of Scribe node F receives all the messages from a single TCP connection originated by node C, which is 2-hop away. In the case of XScribe, node F receives the message over two concurrent TCP connections (from C and D), one of which spans 3 hops. Thus, at high traffic loads the longer connection suffers more than the other one, and the sender's queue at node D fills up more quickly causing a higher packet loss.

As a final remark, it should be noted that the packet loss we measured is influenced by the routing anomalies discussed in Section V-B. In correspondence of route failure events, TCP retransmits failed packets until a new route is established. Thus, the senders' queues at the p2p level tend to fill up and

[3]Properly dimensioning the queues to find the right balance between delay and packet loss depends on the particular application demands (actually, in other sets of experiments we have completely removed packet losses by allowing queues to grow unlimited).

TABLE I
DELAY STATISTICS AT EACH NODE (SECONDS)

node	load (pps)	average		99th percentile	
		Scribe	XScribe	Scribe	XScribe
C	5	0.173±0.048	0.108±0.013	0.590±0.178	0.432± 0.067
	20	1.03±0.170	0.5±0.054	3.05±0.382	1.53±0.246
	50	2.10±0.084	1.64±0.249	3.89±0.122	4.28±0.593
D	5	0.167±0.09	0.095±0.011	0.314±0.055	0.432± 0.067
	20	1.38±0.158	0.497±0.055	4.07±0.665	2.09± 0.389
	50	3.38±0.653	1.81±0.373	4.81±1.38	4.38±0.307
E	5	0.21±0.058	0.113±0.007	0.827±0.204	0.432± 0.089
	20	1.89±0.294	0.588±0.042	6.00±1.33	2.56± 0.669
	50	4.57±0.087	1.88±0.353	7.52±0.379	7.72±1.27
A	5	0.162±0.079	0.111±0.006	0.852±0.197	0.385± 0.058
	20	2.06±0.347	0.592±0.055	6.32±1.39	2.00±0.206
	50	4.91±0.348	2.01±0.369	7.78±0.357	5.42±1.11
B	5	0.167±0.10	0.079±0.032	0.835±0.206	0.322± 0.051
	20	2.03±0.295	0.445±0.110	6.28±1.20	2.00±0.333
	50	4.75±0.057	1.92±0.342	7.70±0.405	5.75±0.621
F	5	0.217±0.063	0.176±0.067	0.859±0.252	0.514± 0.07
	20	2.01±0.222	0.750±0.088	6.25±1.15	2.99±0.551
	50	4.55±0.30	4.08±1.04	7.68±0.369	9.34±0.532

packet loss becomes more and more likely. Unfortunately, it is not possible to establish the exact share of packet loss related to these events. Since routing misbehavior is almost equally probable for Scribe and XScribe, our results might overestimate the packet loss, but correctly rank the performance of the two protocols.

Overall, the experimental results show that in the majority of the cases XScribe performs better than Scribe even though it generates more TCP connections, and, fixed a usability threshold, XScribe allows the application to operate at higher loads than Scribe does.

2) Delay: Table I shows the average values and the 99th percentiles of delays experienced by each node. We also associated with every single value its confidence interval. Three traffic loads have been selected, representative for light, medium, and high loads, and experimental results are almost self-explanatory. When XScribe is used, the average delay is generally more than halved, while the 99th percentile is reduced by at least 1.4x, except for nodes E and F that experience higher percentiles in case of XScribe, but only for high traffic loads (50 pps). Coupled with the results related to packet loss, this means that, in the majority of the cases, for the same application load XScribe is able to drastically reduce the packet loss and, *at the same time*, to reduce the average delay.

As a final remark, it is worth pointing out that in this sets of experiments the root node of Scribe is located at the center of the topology, which minimises the distance with all the other nodes. However, it has been shown in [6] and [26] that, whether the root node is placed at one edge of the network, the performance can be far worse than that presented here. Thus, performance in case of Scribe highly depends on the particular position of the root node in the network topology, while XScribe is immune to this problem.

VI. Model of Scribe and XScribe bounds

To complement the results presented in Section V we now investigate the scalability bounds of both Scribe and XScribe with respect to the multicast group size, the number of senders in the group, and the number of multicast groups. Specifically, we present an analytical model providing the Scribe and XScribe *saturation throughput*, defined as the maximum application load generated by *each* sender that the multicast protocol is actually able to deliver to *all* the receivers. Loads higher than the saturation throughput can be delivered in general just to a *subset* of the receivers. The model is validated by experimental results, and then used to identify the parameter ranges for which the saturation throughput is greater than a specific application load.

To have sensible values for the application load, we refer to the characterisation provided in [13]. In that work, authors focus on massive multiplayer games implemented via p2p multicast system (they actually consider Scribe), and show that each multicast receiver can receive as much as 100 packets per seconds. If aggregation techniques are applicable, the majority of nodes receive between 0 and 10 pps. Therefore, we consider application loads between 1 and 100 pps as representative loads.

A. Modelling Assumptions

The saturation throughput of Scribe and XScribe are hereafter referred to as γ_S and γ_{XS}, respectively. For the sake of simplicity, we now consider a single multicast group. We extend the analysis to the case of multicast groups in Section VII-C. Moreover, we assume that all the senders of the multicast group generate the same application-level load.

In the model we also assume that all the nodes are inside the same Carrier-Sensing (CS) range. The CS range in 802.11 networks has proved to be quite larger than the transmission range, especially at high transmission rates. For example, the CS range measured in [36] is 7 times larger than the transmission range at 11 Mbps, 3 times larger at 5.5 Mbps, 2 times larger at 2 Mbps, and 1.6 times larger at 1 Mbps. This means that nodes 7 hops away from each other (or even more) may be in the same CS range. As we mostly focus on multi-hop ad hoc networks within the ad hoc horizon (2-3 hops, 10 to 20 nodes) assuming that all the nodes are in the same CS range is reasonable.

The last modelling assumption is that the overlay networks adopt UDP as the transport protocol. This is a reasonable choice for a number of applications that could be run on top of p2p multicast systems, which do not necessarily require 100% reliability. For example, using UDP would provide to the Whiteboard Application reduced delays for an increased packet loss, which could be a reasonable trade off. Furthermore, using UDP simplifies the analysis, and, in our networking scenario, provides an upper bound for the saturation throughput achievable over TCP.

B. Model in the case of a single group

Let us firstly focus on XScribe, and specifically on a single sender-receiver pair. Let us finally consider the case in which the network is not overloaded, i.e., all the traffic generated by the senders can be

delivered to the receivers. If n denotes the size of network in number of nodes, then the average length of the multi-hop path between the sender and the receiver is $O(\sqrt{n})$ [37]. According to [37], we assume that the average of this length can be expressed as $k\sqrt{n}$, being k a constant value. The average number of (successful) MAC-level accesses to deliver a single packet from the sender to the receiver is thus $k\sqrt{n}$. Therefore, from the MAC-level standpoint, a UDP flow spanning $k\sqrt{n}$ hops and generating γ bits per second at the sender is equivalent to $k\sqrt{n}$ flows spanning just one hop, generating γ bps each. The average load imposed on the network by one such flow is therefore $\gamma \cdot k\sqrt{n}$. In the case of f multiple concurrent flows generating γ bps each, by assuming a fair scheduling among the flows at each node, the total average load on the network is $f \cdot \gamma \cdot k\sqrt{n}$.

Based on the above remarks it is straightforward to derive the saturation throughput for XScribe. In the saturation condition each sender generates a UDP flow of γ_{XS} bps towards each receiver. Clearly, each sender "sees" the same number of receivers, throughout referred to as n_R. Note that the number of nodes in the group is $n_R + 1$. Therefore, n_R is also the number of UDP flows generated by each sender. The total load imposed on the network by each sender is thus, on average, $\gamma_{XS} \cdot k\sqrt{n} \cdot n_R$. If n_S is the number of senders in the group, the total load imposed on the network is $\gamma_{XS} \cdot k\sqrt{n} \cdot n_R \cdot n_S$. By definition, the total load in the saturation condition has to meet the maximum load sustainable by the network, denoted as Γ. Γ is actually the saturation throughput of an 802.11 ad hoc network in which all the nodes are in the same CS range. Closed form expressions for this figure are available in the literature (the interested reader can refer, for example, to [38]). The saturation throughput of XScribe can be finally derived as follows:

$$\Gamma = \gamma_{XS} \cdot k\sqrt{n} \cdot n_R \cdot n_S \Rightarrow \gamma_{XS} = \frac{\Gamma}{k\sqrt{n} \cdot n_R \cdot n_S} \ . \tag{1}$$

In the case of Scribe the analysis is a bit more involved, even though the guidelines are similar. Let us firstly focus on the traffic outgoing from the root node, and addressed to the other nodes in the multicast group. Recall that the root node receives all the traffic from all the senders, and delivers it to all the nodes in the multicast group (each sender receives back from root its own traffic too). Assuming that the root node is also a member of the group, it has to deliver data to n_R other nodes (because the size of the group is $n_R + 1$). Below the saturation point, the root node of the group delivers all messages to all the group's receivers. If n_S senders generate γ bps each, then n_R flows originate from the root node, each carrying the whole traffic generated by the senders, i.e., $\gamma \cdot n_S$ bps. Finally, each flow covers a path long, on average, $k\sqrt{n}$ hops. Therefore, the total load generated by the root node in the saturation condition is $\gamma_S \cdot k\sqrt{n} \cdot n_R \cdot n_S$.

In addition to the traffic generated by root, in the case of Scribe we have also to take into account the traffic generated by each sender towards root. Specifically, in the saturation condition each sender generates a flow of γ_S bps towards root, which results in an average load on the network of $\gamma_S \cdot k\sqrt{n}$ bps. To count the number of such flows we have to distinguish between two cases, i.e., the root node is or is not a sender of the multicast group. Clearly, the former case provides a best-case condition with

respect to the saturation throughput, because the network has to carry one flow less than in the latter case. The number of flows towards the root node is indeed $n_S - 1$ in the former case, and n_S in the latter.

By denoting with γ_S^o the saturation throughput in the best case (when root is a sender), and with γ_S^w the saturation throughput in the worst case, the following equations hold:

$$\Gamma = \gamma_S^o \cdot k\sqrt{n} \cdot n_R \cdot n_S + (n_S - 1) \cdot \gamma_S^o \cdot k\sqrt{n} \Rightarrow \gamma_S^o = \frac{\Gamma}{k\sqrt{n} \cdot n_S \left(1 + n_R - \frac{1}{n_S}\right)} , \tag{2}$$

$$\Gamma = \gamma_S^w \cdot k\sqrt{n} \cdot n_R \cdot n_S + n_S \cdot \gamma_S^w \cdot k\sqrt{n} \Rightarrow \gamma_S^w = \frac{\Gamma}{k\sqrt{n} \cdot n_S \left(1 + n_R\right)} . \tag{3}$$

Finally, it is easy to show that the probability of the root node being a sender of the multicast group is equal to $\frac{n_S}{n_R+1}$. Therefore, we can evaluate the saturation throughput in the Scribe case as follows:

$$\begin{aligned} \gamma_S &= p\,(\text{root being a sender}) \cdot \gamma_S^o + p\,(\text{root not being a sender}) \cdot \gamma_S^w = \\ &= \frac{n_s}{n_R + 1} \cdot \gamma_S^o + \left(1 - \frac{n_s}{n_R + 1}\right) \cdot \gamma_S^w \end{aligned} \tag{4}$$

C. Extension to the case of multiple groups

The number of active groups in the network is hereafter referred to as n_G. For the sake of simplicity, we assume that all groups have the same number of senders (n_S), are all of the same size ($n_R + 1$), and all their senders generate the same amount of traffic. The extension of Equation 1 to the case of multiple groups is straightforward. For fairness reasons, before the saturation point all the multicast groups should deliver to their receivers the same throughput, say γ. Based on the same outline followed to derive Equation 1, the load generated in the network by n_G groups, each delivering γ bps to its receivers, is $n_G \cdot \gamma \cdot k\sqrt{n} \cdot n_R \cdot n_S$. Therefore, the saturation throughput of XScribe when n_G groups are active can be derived as follows:

$$\Gamma = n_G \cdot \gamma_{XS}^{(n_G)} \cdot k\sqrt{n} \cdot n_R \cdot n_S \Rightarrow \gamma_{XS}^{(n_G)} = \frac{\Gamma}{n_G \cdot k\sqrt{n} \cdot n_R \cdot n_S} . \tag{5}$$

Again, the evaluation of the saturation throughput of Scribe is a bit more involved. Let us assume that for h groups out of the n_G the root node is also a sender of the group. Before the saturation point, each group should be able to deliver γ bps to the receivers. Therefore, the total load generated in the network ($\gamma_t(h)$) is

$$\gamma_t(h) = h \cdot A(\gamma) + (n_G - h) \cdot B(\gamma) , \tag{6}$$

where $A(\gamma)$ and $B(\gamma)$ are the loads generated by a single group for which the root node is and is not a sender, respectively. Specifically, by following the same guidelines used to derive equations 2 and 3, $A(\gamma)$ and $B(\gamma)$ can be written as follows:

$$\begin{aligned} A(\gamma) &= \gamma \cdot k\sqrt{n} \cdot n_R \cdot n_S + (n_S - 1) \cdot \gamma \cdot k\sqrt{n} \\ B(\gamma) &= \gamma \cdot k\sqrt{n} \cdot n_R \cdot n_S + n_S \cdot \gamma \cdot k\sqrt{n} \end{aligned} \tag{7}$$

At the saturation point, the total load generated on the network, given by Equation 6, has to meet the available bandwidth of the network Γ. Therefore, the Scribe saturation throughput, conditioned to the fact that for h groups the root node is a sender, can be obtained from Equations 6 and 7 after simple algebraic manipulations:

$$\gamma_S^{n_G}(h) = \frac{\Gamma}{k\sqrt{n} \cdot (n_G \cdot n_R \cdot n_S + n_G \cdot n_S - h)} \tag{8}$$

It is also easy to show that the probability of having h groups for which the root node is a sender is distributed according to a Binomial distribution with parameters n_G and $p \triangleq n_S/(n_R + 1)$, i.e.:

$$p(h) = \binom{n_G}{h} \cdot p^h \cdot (1-p)^{n_G - h} \ . \tag{9}$$

Therefore, the average value of the Scribe saturation throughput can be evaluated as follows:

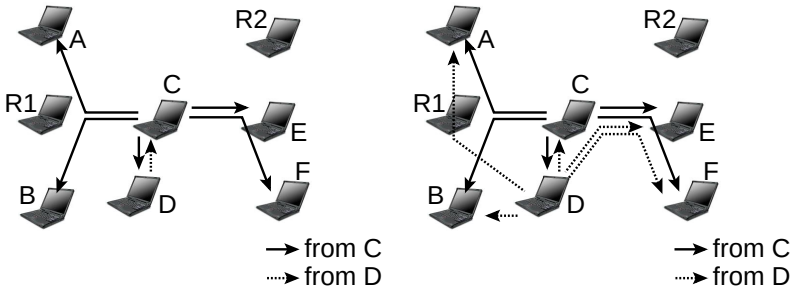
$$\gamma_S^{n_G} = \sum_{h=0}^{n_G} \gamma_S^{n_G}(h) \cdot p(h) \ . \tag{10}$$

Providing a closed form for Equation 10 is not practical. Therefore, in the following analysis we use the constructive method given by Equation 10 to evaluate the values of $\gamma_S^{n_G}$. As a final remark, note that the expressions of Equations 5 and 10 evaluated for $n_G = 1$ coincide with the expressions provided by Equations 1 and 4, respectively.

D. Model Validation

In order to validate our model we ran a set of experiments on the mesh topology described in Section V-C. Instead of using the real p2p systems, we replicated the set of UDP flows between the senders and the receivers that would have been generated by Pastry and CrossROAD to deliver the messages generated by Scribe and XScribe, respectively. Just as an example, Figure 6 shows the set of flows in the case of Scribe and XScribe when the senders were nodes C and D. Traffic on the UDP flows was generated by using the `netperf` tool, version 2.4.1 [39]. Replacing the p2p systems with the equivalent set of UDP flows generated through `netperf` gives us a more controllable environment, allowing us to precisely investigate the scalability bounds of Scribe and XScribe nevertheless.

We tested the system under increasing number of senders, between 2 and 6 (the maximum possible in our mesh topology). In every case we ran a set of experiments by increasing the load each sender generated on its flows, and we measured the throughput at the receivers. The configuration of an experiment is therefore defined by the number of senders and by the load generated by each sender on each flow. We ran each configuration 10 times, each run lasting 100 seconds. Results presented



(a) Scribe (b) XScribe

Fig. 6. Example of flows' pattern

hereafter are the average over the 10 runs. The 95% confidence interval was almost always within 2% of the average value, and was about 15% of the average value just in a few cases.

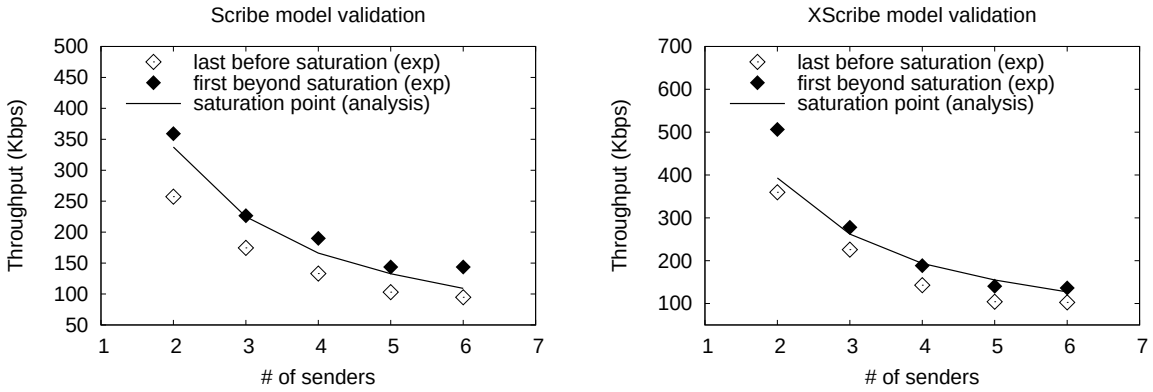
The outcome of an experiment (made up of 10 runs) was labelled as "good" if all the flows were able to correctly terminate the 10 runs. Under `netperf`, network overload manifests as a premature termination of some flows, which results in unavailability of the throughput measure at the receivers[4]. We labelled this type of experiments as "bad". Basically, "good" experiments represent cases in which the network was able to carry the load offered by all the senders, while "bad" experiments represent cases in which the network was overloaded (and the system was thus working beyond the saturation point).

TABLE II
VALIDATION PARAMETERS

Param	*Value*
k	0.67
n	8
n_R	5
n_S	2 to 6

Figure 7 shows the results from our testbed, compared with the saturation throughput predicted by the analytical model (i.e., by Equations 1 and 4). Table II shows the model parameter values used for the validation. The value of k (where $k\sqrt{n}$ is the average path length) has been derived based on the average path length of the flows in all the configurations. Note that this value is remarkably aligned with the theoretical value of $2/3$ predicted by [37]. The value of Γ was derived exploiting the analytical model presented in [40], which provides the capacity (i.e., the maximum throughput) of an 802.11 network with a variable number of saturated nodes within a unique carrier-sensing region. Specifically, from the logs of the validation experiments, we derived, in each configuration (i.e., for each number of senders), the number of active nodes in the network, and we computed the analytical value of Γ corresponding to that number of active nodes.

[4]In detail, nodes experiencing a throughput below the offered load become unable to correctly manage the timers used by `netperf` to schedule message transmissions, thus resulting in premature experiment termination.



(a) Scribe (b) XScribe

Fig. 7. Model validation

In Figure 7, for each number of senders, the white diamonds are related to the ”highest-load good experiment”, i.e. to the experiment with the highest offered load that resulted in a ”good” experiment. Specifically, the white diamonds are the average values of throughput measured at the receivers in the ”highest-load good experiment”. Likewise, black diamonds are related to the ”lowest-load bad experiment”, i.e. to the experiment with the lowest offered load that resulted in a ”bad” experiment. Recall that, due to `netperf` internals, in ”bad” experiments just a subset of the receivers were able to correctly complete the experiment, and thus to provide a meaningful throughput value. The black diamonds are the average values of throughput measured at those receiver that correctly completed the ”lowest-load bad experiment”.

Clearly, for a particular number of senders, the real saturation throughput lies somewhere between the white and black diamonds. Figure 7 actually validates our model, since the model predictions are, with a good degree of accuracy, within the region where the saturation throughput actually lies. Based on this result, in the following section we exploit the model to show that XScribe is able to significantly improve the saturation throughput with respect to Scribe, thus resulting in increased scalability with the load generated by applications.

VII. Evaluating Scribe and XScribe scalability bounds

A. Scalability with the number of senders and receivers

Figures 8 and 9 highlight the scalability properties of Scribe and XScribe as the number of senders (n_S) increases. Different curves are plotted for selected numbers of receivers “seen” by each sender (n_R). Note that the number of senders must be lower than the group size, i.e. curves have physical sense just up to $n_S = n_R + 1$. The k and Γ parameters are kept as in Table II. The number of nodes in the network (n) is scaled according to the multicast group size ($n_R + 1$) to keep the fraction of multicast members as in the experiments presented in Section V. Specifically, we set this fraction as $(n_R + 1)/n = \alpha = \frac{6}{8}$. Note that this represents a fairly dense configuration, in the sense that 75% of nodes (i.e., 6 over 8) are in the multicast group. Sensitiveness to the α parameter is investigated in

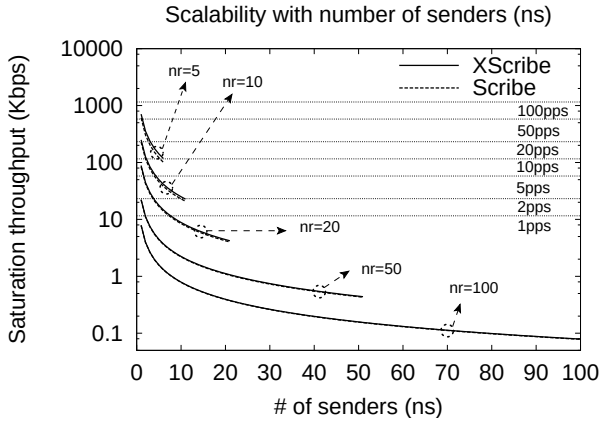


Fig. 8. Scalability with the number of senders (n_S).

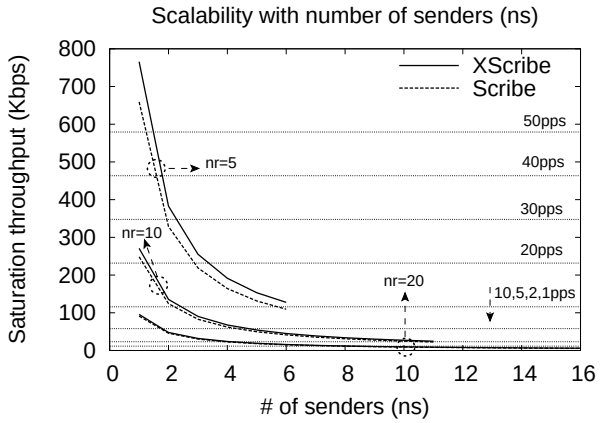


Fig. 9. Scalability with the number of senders (n_S). Focus on small-size networks.

Section VII-B. Finally, horizontal lines in the curves show representative loads as generated by senders of the Whiteboard Application presented in Section V.

First of all, it is worth noting that Scribe and XScribe support fairly well small to medium size groups. For groups with up to 20 receivers, there is a quite large region in which both systems are able to deliver more than 1 pps to all the receivers. For pretty small groups ($n_R = 5$) the systems are able to deliver even more than 50 pps to the receivers. According to the traffic characterisation presented in [13], this means that in that region Scribe and XScribe can support multicast applications generating a fairly high amount of traffic, such as p2p interactive games. On the other hand, similar applications cannot be supported when the size of the multicast group grows beyond 20. As shown by Figure 8, in these cases just applications generating light loads (i.e., below 1pps) can be efficiently supported. This is actually not a too severe limitation. As noted in [13], a large class of data-intensive multicast applications (i.e., multiplayer games) are expected to generate *small* groups. Thus, Scribe and XScribe are possible options in these scenarios. Note also that results plotted here are quite pessimistic, since our model assumes that all the nodes in the network are in the same Carrier Sensing range. Clearly,

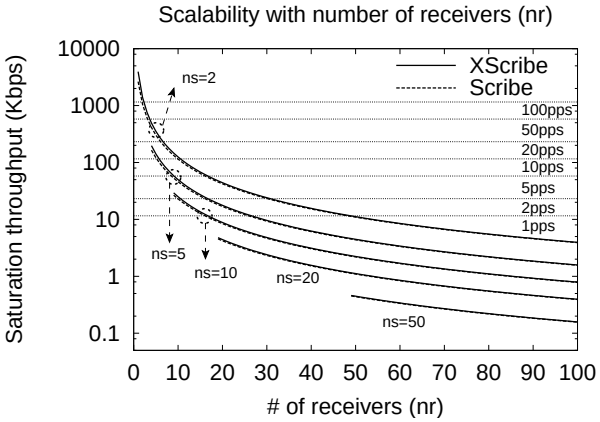


Fig. 10. Scalability with the number of receivers (n_R).

this assumption becomes less accurate as the number of nodes in the network (and the group size) increases.

Figure 9 also allows us to better understand the results presented in Section V. Specifically, those results were obtained with $n_S = 2$ and $n_R = 5$. In this case the saturation throughput is around 30 pps. Therefore, the results discussed in Section V tell that XScribe is able to significantly reduce the average delay in non overloaded configurations (below 30 pps). Beyond the saturation point (i.e., beyond 30 pps) XScribe works significantly better than Scribe does, it is thus able to mitigate the effect of network overloading more efficiently than Scribe. Figures 10 and 11 show the scalability with the number of receivers. Essentially, they confirm that Scribe and XScribe are both able to support very high application-level loads (even beyond 100 pps) in case of small-scale groups and a limited number of senders.

Finally, Figure 12 and 13 show the proportional increase of the saturation throughput achieved by XScribe with respect to Scribe. Specifically, they plot $(\gamma_{XS} - \gamma_S)/\gamma_S$, where γ_{XS} and γ_S are defined by Equations 1 and 4, respectively. The proportional advantage of XScribe over Scribe is slighthly dependent on the number of senders n_S (Figure 12), while it is fairly sensitive to the size of the group n_R (Figure 13). It is interesting to note that XScribe is able to increase the saturation throughput up to 50% with respect to Scribe. The advantage of using XScribe reduces as the number of receivers grows, but it still remains not negligible even for medium-size groups (i.e., for n_R between 10 and 20).

To summarise, results presented so far show that Scribe and XScribe saturation points drop as either the number of senders or the number of receivers increase. This is actually due to the fact that both γ_S and γ_{XS} decrease as $1/n_R$ and $1/n_S$. The interesting feature is that for small- and medium-size groups (i.e., for n_S less than 10 and n_R less than 20) the saturation throughput is above 1 pps, and can even be greater than 100 pps. This means that, in such network scales, these systems are able to support a fairly large range of multicast applications. Furthermore, XScribe is able to increase the saturation throughput with respect to Scribe. Specifically, the increase is in the range between 5% and 50% in

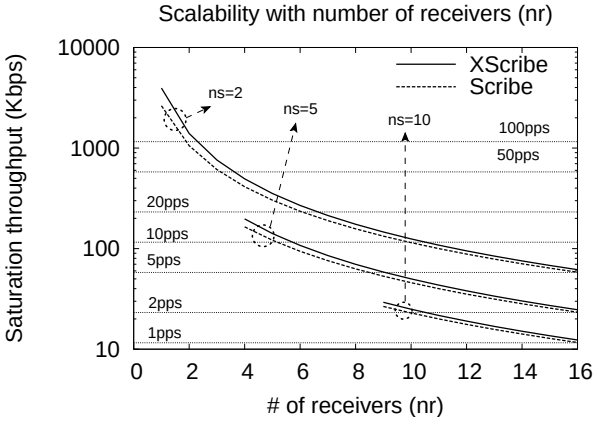


Fig. 11. Scalability with the number of receivers (n_R). Focus on small-size networks.

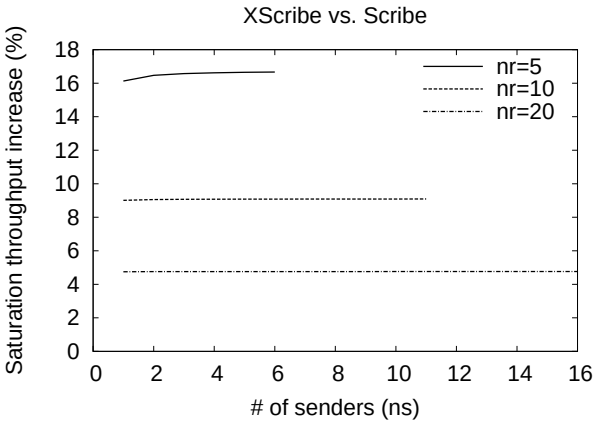


Fig. 12. Advantage of XScribe with respect to Scribe as a function of n_S.

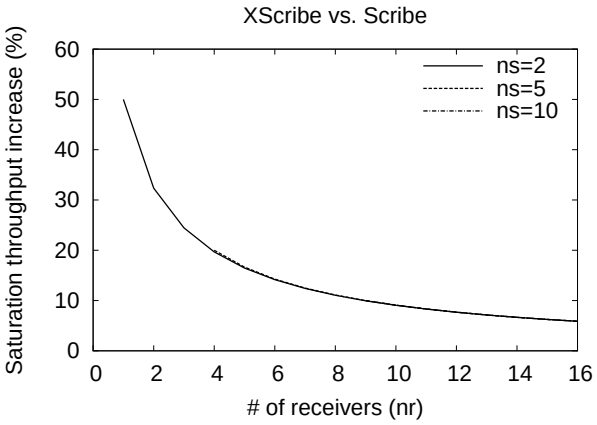


Fig. 13. Advantage of XScribe with respect to Scribe as a function of n_R.

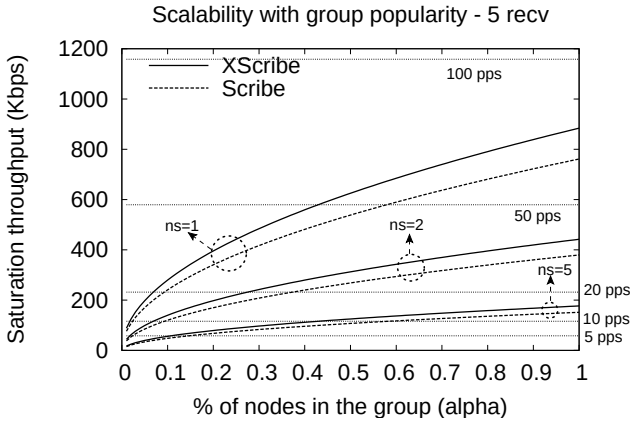


Fig. 14. Scalability with the group popularity, for 5 receivers.

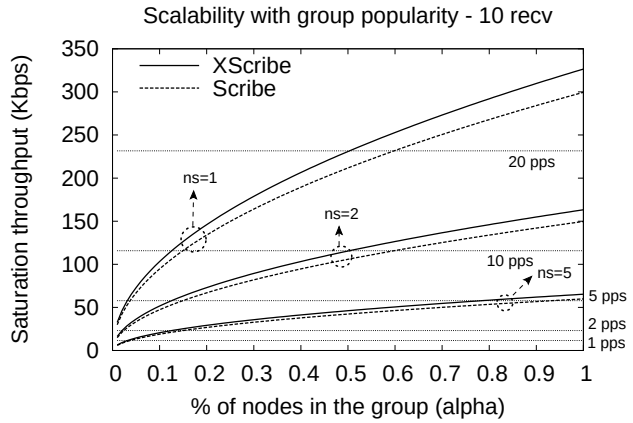


Fig. 15. Scalability with the group popularity, for 10 receivers.

the cases we have highlighted. This confirms that XScribe generates quite less traffic on the network. The XScribe advantages is more pronounced for small sets of receivers. This is because the additional traffic generated by Scribe is due to carrying all multicast messages from the senders to the root before delivering them to the receivers. Clearly, this cost is less amortised when the number of receivers is small.

B. Sensitiveness to the multicast group popularity

To analyse the systems' scalability properties with respect to the fraction of nodes belonging to the multicast group, we consider three representative cases for the number of receivers (i.e., $n_R = 5, 10, 20$), and for the number of senders (i.e., $n_S = 1, 2, 5$). In the previous section we have shown that in all these cases Scribe and XScribe are able to reasonably support multicast applications, i.e., the saturation throughput is higher than 1 pps. Recall also that results presented in the previous section are obtained in case 75% of the nodes in the network belong to the multicast group.

Figures 14, 15, and 16 show the saturation throughput in all the representative cases, as functions of

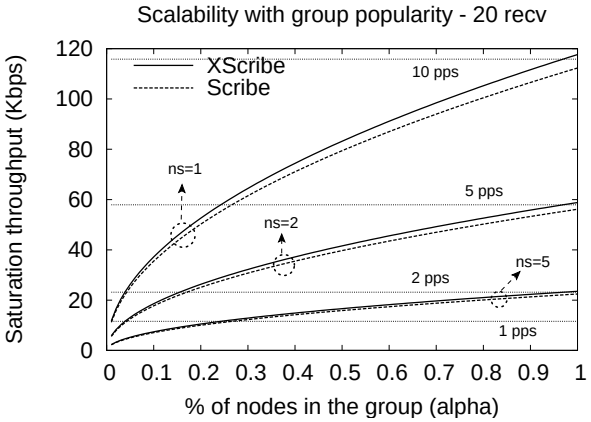


Fig. 16. Scalability with the group popularity, for 20 receivers.

the multicast group popularity. The group popularity is actually represented by the α parameter, which is defined as the ratio between the number of nodes in the group, and the number of nodes in the network, i.e., $\alpha = (n_R + 1)/n$. Note that in each plot the number of receivers is fixed. Therefore, lower values of α represent larger networks or, from a complementary standpoint, sparser groups. Clearly, the saturation throughput decreases as the group becomes more and more sparse, i.e., as the value of α decreases. This is because the average length of the paths between senders and receivers increases with the network size, specifically as $O(\sqrt{n})$. Thus, the network becomes more congested as its size increases, because messages have to be forwarded many times. From Equations 1 and 4 it can be derived that both γ_{XS} and γ_S are proportional to $\sqrt{\alpha}$. Therefore, the decrease of the saturation throughput with low α values is not steep at all, and the saturation throughput drop sharply just for very sparse groups, i.e., for α values below 10%. Clearly, the specific usability bounds depend heavily on the number of senders and receivers, as can be noted by comparing the plots in the three figures. However, from the shape of the curves we can conclude that, for a fixed group size, Scribe and XScribe performance degrade fairly gracefully as the network size increases, unless for very sparse groups. It is also worth pointing out that the saturation throughput remain greater than 1 pps for fairly sparse groups even in the worst scenario we have considered here. Specifically, note that the curve related to 5 senders crosses the 1 pps line in Figure 16 at about $\alpha = 0.3$.

As far as the sensitiveness to α, we do not plot the proportional advantage of XScribe over Scribe, because it does not depend on α. Actually, the proportional advantage of XScribe over Scribe is defined as $(\gamma_{XS} - \gamma_S)/\gamma_S$, and γ_{XS} and γ_S are proportional to $\sqrt{\alpha}$.

C. Scalability with the number of groups

The last parameter we take into consideration is the number of multicast group, n_G. Specifically, Figures 17 and 18 show the saturation throughput of Scribe and XScribe as functions of n_G. Specifically, we draw three couple of curves, for three representative sets of the parameters (n_S, n_R). The three set of values for (n_S, n_R) have been chosen following the same principle used in Section VII-B, and are

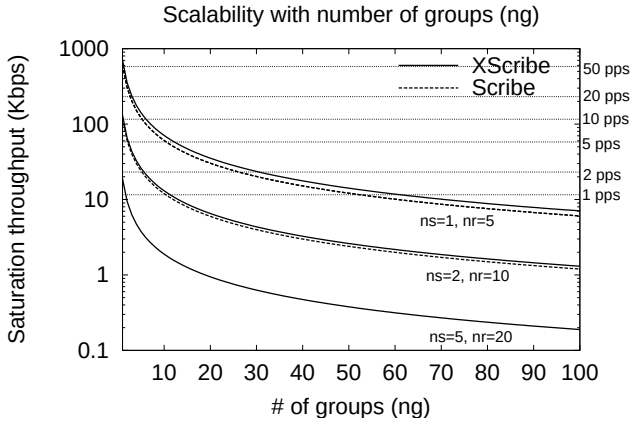


Fig. 17. Scalability with the number of groups.

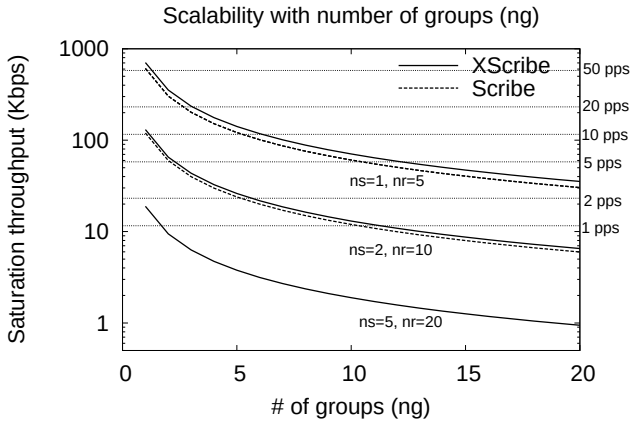


Fig. 18. Scalability with the number of groups, focus on small n_G values.

representative for small $(n_S = 1, n_R = 5)$, medium $(n_S = 2, n_R = 10)$, and slightly larger $(n_S = 5, n_R = 20)$ groups. The popularity of the multicast groups has been kept as in the experiments described in Section V, i.e., we set $\alpha = 0.75$.

Unfortunately, both system suffer from the increase of the number of groups. This might actually be a problem for a class of p2p applications (e.g. multiplayer games), which tend to generate several groups of small size. However, note that in the case of small groups, the saturation throughput remains well above 1 pps up to about 60 groups, and in the case of medium groups, the saturation throughput drops below 1 pps for more than 10 groups.

Finally, Figure 19 plots the proportional advantage of XScribe over Scribe as function of n_G. Interestingly, the proportional advantage is almost independent of n_G and, once again, mainly depends on the particular value of n_R.

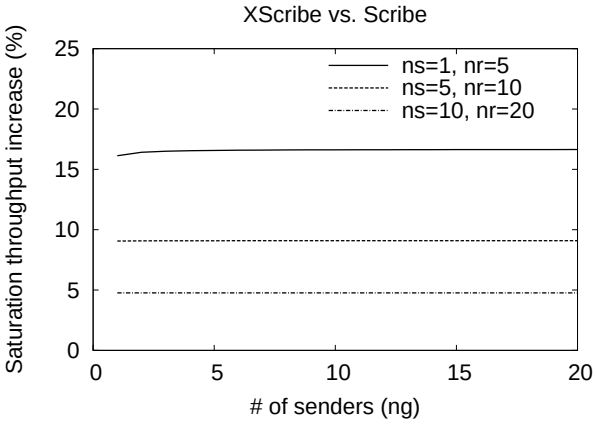


Fig. 19. Advantage of XScribe with respect to Scribe as a function of n_G.

D. Summary of the main results

In this section we have analysed the scalability bounds of both Scribe and XScribe with respect to a number of key system parameters. In conclusion, the saturation throughput of both systems drops significantly as either the number of senders (n_S), or the number of receivers (n_R), or the number of groups (n_G) increase, for a fixed network size. For a fixed group size, the saturation throughput decreases as the group becomes more and more sparse (i.e., as the popularity of the group, α, decreases). Despite these bounds, both Scribe and XScribe are possible candidates to support even p2p interactive gaming in small and medium size networks (up to 20 receivers and 10 senders), and for a reasonable number of groups (up to 60). In this region of the parameters' space, both systems are able to provide more than 1 pps to all the receivers, which can be seen as a minimum requirement to support this kind of applications (see [13]). Clearly, for less-demanding multicast applications, the range of applicability of these systems increases.

As expected, XScribe always outperforms Scribe, in the sense that it increases the saturation throughput for any given configuration of the system parameters. Quite interestingly, the proportional advantage of XScribe over Scribe slightly depends on the number of senders, on the number of groups, and does not depend at all on the group popularity. On the other hand, it is highly influenced by the number of receivers of the groups, and for few receivers the performance advantage can reach about 50%.

VIII. Conclusions & Open Issues

In this work we have analysed the performance and limitations of p2p multicast systems for pervasive ad hoc networks. This is an interesting topic, because pervasive systems can benefit from efficient middleware services exploiting the p2p paradigm, and studying p2p multicast in this framework is thus important. We have considered Scribe, which is a p2p multicast system that has shown high performance over legacy wired networks, but it has several inefficiences in pervasive environments. Therefore, we have presented and evaluated XScribe, which is a simple cross-layer replacement for Scribe, designed for multi-hop ad hoc networks. XScribe takes a structure-less approach to multicast, and relies on the

networking structure already defined by the underlying layers instead of binding together multicast group members through network structures defined at the multicast level, such as trees or meshes. This is possible thanks to receiver awareness, i.e., in XScribe all the senders of a group know the group receivers. With respect to other solutions exploiting receiver awareness, XScribe implements this feature via a very efficient cross-layer mechanism. Both the experimental results shown in Section V and the analysis discussed in Section VII show that XScribe outperforms Scribe thanks to these design features. Specifically, XScribe increases the maximum throughput that the multicast system is able to deliver to multicast receivers and, even in overloaded conditions, reduces the performance penalties of Scribe in terms of delay and packet loss.

Despite these performance figures, there is still room for improvement. By design choice, XScribe adopts a pure p2p data delivery mechanism, in the sense that no information about the physical paths between senders and receivers is exploited during data delivery. This allowed us to stress the limits of such p2p delivery policies. However, the results in this paper clearly show that this policy actually limits the system scalability. Specifically, our analytical models show that, since a distinct message is delivered to each multicast receiver, the saturation throughput decreases as $1/n_R$ where n_R is the number of receivers "seen" by each sender. A possible fix to this point is including cross-layer optimisations also in the data-delivery phase. Specifically, possible portions of shared paths between senders and receivers could be exploited efficiently, since the same message could be transported just once along the shared portion of the path. Actually, this is a feature already implemented by traditional layer-3 multicast protocols. We think that integrating multicast features implementable at layer-3 (like sharing paths), and features related to the middleware layers (like exploiting subject-based routing) is the right balance to both achieve efficiency, and providing the applications with standard p2p multicast services.

IX. Acknowledgements

This work has been partly supported by the Italian Ministry for Research (MIUR) in the framework of the Project FIRB-PERF.

References

[1] M. Gerla, C. Lindemann, and A. Rowstron, "P2P MANETs - New Research Issues." [Online]. Available: http://drops.dagstuhl.de/portals/index.php?semnr=05152

[2] C. Lindemann and A. Rowstron, Eds., *ACM MobiCom MobiShare Workshop*, September 2006. [Online]. Available: http://www.mobishare.org

[3] P. Gupta and P. Kumar, "The Capacity of Wireless Networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, Mar. 2000.

[4] P. Gunningberg, H. Lundgren, E. Nordstroem, and C. Tschudin, "Lessons from Experimental MANET Research," *Ad Hoc Networks Journal*, vol. 3, no. 2, pp. 221–233, Mar. 2005.

[5] E. Borgia, M. Conti, F. Delmastro, and L. Pelusi, "Lessons from an Ad-Hoc Network Test-Bed: Middleware and Routing Issues," in *Ad Hoc & Sensor Wireless Networks, An International Journal*, Vol.1, Numbres 1-2, 2005.

[6] F. Delmastro, A. Passarella, and M. Conti, "Experimental Analysis of P2P Shared-Tree Multicast on MANETs: the Case of Scribe," in *Proc. of the Fifth IFIP Annual Mediterranean Ad Hoc Networking Workshop (MedHocNet 2006)*, Lipari, Italy, June 2006.

[7] F. Delmastro, "From Pastry to CrossROAD: Cross-layer Ring Overlay for AD hoc networks," in *Proc. of IEEE PerCom MP2P Workshop*, Kauai Island, Hawaii, Mar. 2005.

[8] M. Caesar, M. Castro, E. Nightingale, G. O'Shea, and A. Rowstron, "Virtual Ring Routing: Network Routing inspired by DHTs," in *Proc. of ACM SIGCOMM*, Pisa, Italy, September 11-15 2006.

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

[10] F. Delmastro, M. Conti, and E. Gregori, "P2P Common API for structured overlay networks: a Cross-Layer Extension," in *Proc. of MDC'06 Workshop*, Niagara Falls, Buffalo (NY), June 2006.

[11] 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," in *Proc. of INFOCOM*, San Francisco, CA, Apr. 2003.

[12] E. Borgia, M. Conti, and F. Delmastro, "MobileMAN: Design, Integration and Experimentation of Cross-Layer Mobile Multi-hop Ad Hoc Networks," *to appear in IEEE Communication Magazine, Ad Hoc & Sensor Network series*, 2006.

[13] B. Knuttson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-Peer Support for Massively Multiplayer Games," in *Proc. of INFOCOM*, 2004.

[14] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralised application-level multicast infrastructure," *IEEE JSAC*, vol. 20, no. 8, Oct. 2002.

[15] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-Level Multicast using Content-Addressable Networks," in *3rd International Workshop on Networked Group Communication*, November 2001.

[16] 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," in *Eleventh International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2001.

[17] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High Bandwidth Multicast in Cooperative Environments," in *ACM SOSP*, October 2003.

[18] M. Ge, S. V. Krishnamurthy, and M. Faloutsos, "Application versus Network Layer Multicasting in Ad Hoc Networks: The ALMA Routing Protocol," *Elsevier Ad Hoc Networks Journal*, to appear.

[19] C. Gui and P. Mohapatra, "Overlay Multicast for MANETs Using Dynamic Virtual Mesh," *ACM/Springer Wireless Networks (WINET)*, to appear.

[20] E. Royer and C. Perkins, "Multicast Ad hoc On-Demand Distance Vector (MAODV) Routing," 2000. [Online]. Available: http://www3.ietf.org/proceedings/00jul/I-D/manet-maodv-00.txt

[21] S.-J. Lee, W. Su, and M. Gerla, "On-Demand Multicast Routing Protocol (ODMRP) for Ad Hoc Networks," 2000. [Online]. Available: http://www.cs.ucla.edu/NRL/wireless/PAPER/draft-ietf-manet-odmrp-02.txt

[22] S. Yang and J. Wu, *New Technologies of Multicasting in MANETS*, Y. Xiao and Y. Pan, Eds. Nova Publishers, 2005.

[23] M. Conti, E. Gregori, and G. Turi, "Towards Scalable P2P Computing for Mobile Ad Hoc Networks," in *Proc. of the second IEEE Annual Coneference on Pervasive Computing and Communications Workshops*, Orlando, Mar. 2004.

[24] L. Ji and M. Corson, "Explicit Multicasting for Mobile Ad Hoc Networks," *Mobile Networks and Applications*, vol. 8, pp. 525–549, 2003.

[25] J. Luo, P. Eugster, and J.-P. Hubaux, "Probabilistic reliable multicast in ad hoc networks," *Ad Hoc Networks*, vol. 2, pp. 369–386, 2004.

[26] A. Passarella and F. Delmastro, "Usability of Legacy P2P Multicast in Multi-Hop Ad hoc Networks: an Experimental Study," *EURASIP Journal on Wireless Communications and Networking*, accepted for publication.

[27] A. Passarella, F. Delmastro, and M. Conti, "XScribe: a Stateless, Cross-Layer Approach to P2P Multicast in Multi-Hop Ad hoc Networks," in *Proc. of ACM MobiShare*, September 2006.

[28] M. Castro, M. Costa, and A. Rowstron, "Debunking Some Myths About Structured and Unstructured Overlays," in *Proc. of NSDI*, May 2-5 2005.

[29] M. Conti, G. Maselli, G. Turi, and S. Giordano, "Cross Layering in Mobile Ad Hoc Network Design," *IEEE Computer*, Feb. 2004.

[30] E. Borgia, M. Conti, F. Delmastro, and E. Gregori, "Experimental comparison of Routing and Middleware solutions for Mobile Ad Hoc Networks: Legacy vs Cross-Layer approach," in *Proc. of the ACM SIGCOMM E-WIND Workshop*, Philadelphia, Aug. 2005.

[31] E. Borgia, M. Conti, and F. Delmastro, "MobileMAN: Integration and Experimentation of Legacy Mobile Multi-hop Ad Hoc Networks," *to appear in IEEE Communication Magazine, Ad Hoc & Sensor Network series*, 2006.

[32] F. Delmastro and A. Passarella, "An Experimental Study of P2P Group-Communication Applications in Real-World MANETs," in *Proc. of IEEE ICPS REALMAN 2005 Workshop*, Santorini, Greece, July 2005.

[33] (2005, Aug.) ACM SIGCOMM E-WIND Workshop. Philadelphia, PA. [Online]. Available: http://www-ece.rice.edu/E-WIND/

[34] FreePastry, Rice University. [Online]. Available: http://freepastry.rice.edu

[35] Optimized Link State Routing Protocol (OLSR): RFC3626. [Online]. Available: http://www.ietf.org/rfc/rfc3626.txt

[36] 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, no. 2, pp. 237–256, July 2005.

[37] J. Li, C. Blake, D. S. D. Couto, H. I. Lee, and R. Morris, "Capacity of Ad Hoc Wireless Networks," in *Proc. of ACM MobiCom*, 2001.

[38] F. Calì, M. Conti, and E. Gregori, "Dynamic Tuning of the IEEE 802.11 Protocol to Achieve a Theoretical Throughput Limit," *IEEE Transactions on Networking*, vol. 8, no. 6, pp. 785–799, December 2000.

[39] Netperf traffic generator. [Online]. Available: http://www.netperf.org/netoerf/NetperfPage.html

[40] G. Bianchi, "Performance Analysis of the IEEE 802.11 Distributed Coordination Function," *IEEE JSAC*, vol. 18, no. 9, pp. 1787–1800, March 2000.