11. Application-Layer Multicast

Kostas Katrinis, Martin May (ETH Zurich)

11.1 Why Multicast on Application Layer

Since the early days of the Internet, extending routing capabilities beyond point-to-point communication has been a desired feature, mostly as a means of resource discovery. The limited size of the Internet at that time permitted the technique of broadcasting a single packet to every possible node. With its growth, Internet-wide broadcasting became increasingly expensive which imposed constraining the scope of broadcast packets to end points that expressed interest in receiving packets of a specific service (Selective Broadcast [612]). This was in fact the first attempt to offer indirectly a group communication service over the Internet. Still, Deering's seminal work [161] is regarded as the groundwork in Internet-wide group communication. In his host-group model, Deering also specified the extensions to unicast routing protocols required to support many-to-many communication. Since the first appearance of the host-group model, a lot of research and standardization effort has been put into ubiquitous *multicast* natively implemented on routers (network-layer service). Despite years of ongoing work, deployment of native IP-multicast still remains poor [170] and this has motivated the search for alternative approaches. However, it has been shown that the design choice of offering multicast at the network layer violated the end-to-end argument [531]. Specifically, the end-to-end argument suggests that it is only worth pushing a service to lower layers of a system if, by doing so, the returned performance/cost improvement outweighs the increased complexity of implementing the service at the network layer. Clearly, network-layer multicast and, in particular, its de facto Any-Source Multicast standard (the Any-Source Multicast model is implemented in the PIM-SM/MBGP/MSDP protocol suite) is violating the end-to-end principle, for many of its deployment problems require the realization of the service at the network layer (e.g., source discovery, inter-domain routing). Alternatively, one of the newly arisen multicast proposals tries to more closely adhere to the end-to-end argument by shifting the service implementation to higher in the protocol stack, namely to the application layer.

Application-Layer Multicast (**ALM**) does not assume any support from the network, i.e., it assumes just an elementary unicast forwarding service. All specific group communication functionality including group management (integrating new members into an existing group or coping with member departures), multicast tree formation, and packet replication, is moved to the application layer. In particular, ALM systems organize the participating

R. Steinmetz and K. Wehrle (Eds.): P2P Systems and Applications, LNCS 3485, pp. 157-170, 2005.

nodes in the service as an overlay network; the end-to-end IP paths among each node pair constitute the links of the overlay network. Group information and forwarding state, stored in the overlay nodes, are then used for packet replication over the overlay links and to deliver data packets originating from any source to the entire group.

11.2 Design Aspects and Taxonomy

The ultimate goal of an ALM scheme is to build an efficient overlay network and thus to maximize the service quality received by the service consumers (group members). The QoS (Quality of Service) parameters of interest may vary, depending on the requirements of the group communication application utilizing the ALM service. For instance, an interactive teleconferencing application would require an overlay optimized for stringently minimized delivery delays and high bandwidth, whereas a TV streaming service could tolerate larger overlay delays, but demands low loss and high bandwidth.

To date, numerous application-layer systems have been proposed to realize multicast. Most of these schemes are based on the same fundamental design choices and those characteristics are used here to classify the different proposals. A principal characteristic refers to routing organization: many ALM overlays first build a richly connected graph termed *mesh* (therefore, the general approach is referred to as *mesh-first*) and subsequently construct the distribution tree as a subgraph of the mesh (ESM [307] and ALMI [482]). In contrast, other ALM schemes first build a spanning tree on the overlay nodes (*tree-first* approach) and then enrich the tree by adding additional control links (Yoid [228], HMTP [638]).

A second classification characteristic is associated with the type of overlay used. Many ALM schemes implement multicast on top of a structured overlay network like CAN 8.3, Pastry 8.2, or Chord 8.1 by defining a virtual space and positioning the overlay nodes in it. Direct IP routing is abstracted to routing in the virtual space and accordingly, overlay multicast is abstracted to one- or many-to-many communication across the nodes in the virtual space. Representative examples of this class of ALM systems are CAN multicast [506] and Scribe [109]. Note that still the majority of ALM proposals is based on unstructured overlays, using explicit routing mechanisms.

Finally, some of the proposed ALM schemes build overlays solely with end user systems, thereby shielding their operation against abrupt departure or failure of overlay nodes (ESM, ALMI, HMTP). In contrast to those, other systems (like Overcast [322] and RON [26]) assume the existence of fixed nodes (equivalently termed proxies, replicators, or multicast service nodes in the bibliography) that are deployed inside the network and which are essentially responsible for routing in the overlay. In these latter systems,

end user nodes do not usually participate in packet replication and group management, and act only as service consumers.

In the next section, we follow one of the possible taxonomies: classifying ALM based on structured or unstructured overlays. For each category, we further distinguish two sub-categories with reference to the peculiarities of each category.

Finally, before delving into the specification details of ALM systems, it is important to identify the performance overhead and service costs introduced by moving multicast functionality to the application layer. Below is a list of metrics that are often used to evaluate any given ALM system, but that also represent a guideline for ALM design in general:

- *Relative Delay Penalty (RDP) or Stretch*: Strictly defined as the ratio of the one-way overlay delay of a node pair over the unicast delay among the same nodes in the identical direction. The goal here is to match routing proximity as closely as possible to underlying IP routing, reducing the resulting delay penalty.
- *Throughput*: Similarly to RDP, this performance metric measures the effective data throughput achieved for a single receiver (over time or on average).
- *Stress*: Stress is defined as the number of times the same packet traverses a specific physical link in either direction. Essentially, it quantifies the cost of moving the replication effort to end systems in terms of data bandwidth.
- *Control Overhead*: Number of control messages exchanged throughout an ALM session. This metric represents the cost in terms of control message exchanges.

11.3 Unstructured Overlays

11.3.1 Centralized Systems

The term "centralized" in the context of ALM design does not refer to data replication handled by a centralized entity; instead, it points out the design principle of a centralized entity handling group management and creation/optimization of the distribution tree. This is the approach taken in the Application Level Multicast Infrastructure (ALMI [482]).

The entire coordination task in ALMI is assigned to the "*session controller*", as shown in the architecture diagram (Figure 11.1). The session controller - residing either on a dedicated server of the service provider or at a group member node - exchanges point-to-point messages (dashed arrows) via unicast with every overlay node (drawn as a circular point). It is worth mentioning that the controller does not lie on the data path, i.e., is not part of the distribution tree (marked with bold, solid arrows), thus avoiding bottlenecks in data distribution.

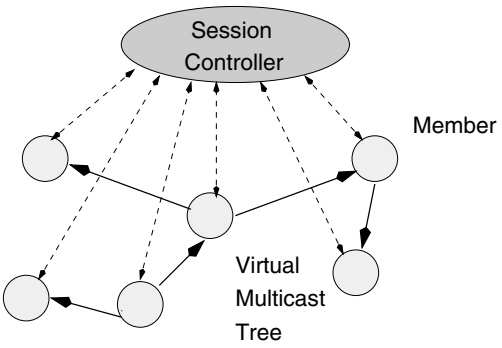


Fig. 11.1: Simplified ALMI architecture

A node that wants to join an ALMI session sends a *JOIN* message to the session controller. Note that the discovery of the session controller's location for a given session ID is beyond the scope of the system specification and realized by third party means, for example via a known URLs or e-mail notification. When the newly arrived node is accepted into the group, it receives a response containing its *member ID* (identifier in the overlay) and the location of a parent node to which it should append itself. Finally, the newly added node sends a *GRAFT* message to its parent and obtains in response the data ports for the two-way communication with its parent. Node departures are realized similarly by signaling the session controller with a *LEAVE* message.

Tree creation and maintenance are also tasks performed by the session controller. Given a performance metric of interest (e.g., delay), the controller computes locally a minimal spanning tree on the group members graph and assigns the upstream and downstream nodes to each overlay node in the distribution tree. Note that, unlike other ALM systems, ALMI builds a single shared tree with bidirectional links that is jointly used by all members for data distribution. Measurement data for the metric to be optimized is provided by each overlay node to the controller on a point-to-point basis. For this purpose, each overlay node actively probes every other node and reports the results to the controller. Obviously, this generates an $O(n^2)$ message overhead. To scale the monitoring service to larger groups, ALMI limits the degree of each node in the monitoring graph. Although this may initially lead to sub-optimal multicast trees, over time each node dynamically prunes bad links and adds new links to the monitoring topology, resulting in more efficient multicast trees.

Recapitulating, the centralization approach adopted by ALMI offers two primary advantages: high control over the overlay topology and ease of implementation. Moreover, as a side-effect of the first advantage, detection of malicious nodes is easier to realize because all control operations pass through

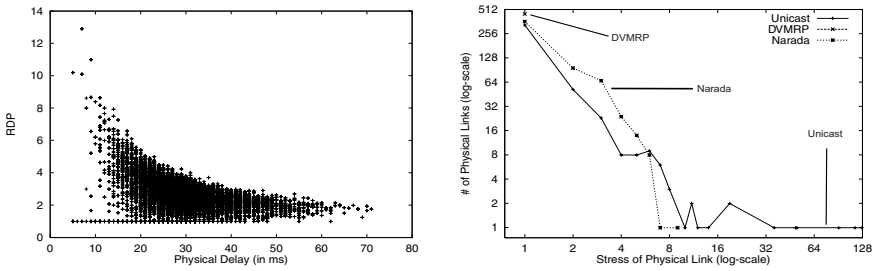
the session controller. On the other hand, ALMI is plagued with the scalability and dependability concerns of all centralized systems. While the first deficiency remains unresolved, ALMI tries to alleviate the negative effects of controller failures by introducing *backup controllers*. These synchronize periodically with the main controller's state and, in case of failure detection, one of the backup controllers replaces the session coordinator.

11.3.2 Fully Distributed Systems

Fully distributed application-layer multicast systems do not rely on a single (or a few) coordinating entities for routing organization and group management. Instead, each node maintains information about the overlay topology - either partly or entirely - and self-configures/self-adapts to changes in the overlay topology or the underlying network condition.

End System Multicast (ESM [307]) constitutes a fully distributed ALM scheme - and is one of the elementary studies on application layer multicast. ESM overlay networks utilize exclusively end systems, assuming only basic unicast IP service and thus avoiding any proxy or dedicated server infrastructure deployed within the network. Group management and packet replication among the end systems is managed by a protocol called *Narada*. *Narada* employs the mesh-first approach: overlay nodes first organize themselves in a redundant mesh graph. On top of this mesh, source-specific multicast trees are then created for data distribution. Consequently, service quality is at best as good as the quality of the mesh. Therefore, *Narada* attempts to ensure that the properties of the mesh meet the performance requirements of the application running over ESM. More precisely, a good mesh should comprise overlay links whose quality (e.g., delay, available bandwidth) is comparable to the quality of the IP path connecting the two endpoints of the overlay node pair. Second, it is desirable that each node has a limited fan-out, regulating the replication effort and stress at each overlay node.

Group management in *Narada* is equally distributed on all nodes. For the sake of robustness and efficiency of mesh maintenance, each overlay node keeps track of every group member. As ESM targets at small to medium sized multicast groups, scalability is not considered an important issue. If a node want to join the group it contacts a small number of arbitrary group members (the ID and location of these members is provided by a third party service not provided by the system, e.g., via URL or e-mail notification). The arrival of a new member is announced to all nodes by forcing each overlay member to broadcast periodic *heartbeat* messages ("refresh" messages) on the mesh. This leads ultimately to a situation where every overlay node is aware of the newly arrived node. Similarly, when a node decides to leave the group, the remaining group members detect its departure after they stop receiving heartbeat messages from the leaving node. Note that the latter may also



(a) RDP (b) Stress

Fig. 11.2: Delay performance of ESM and induced cost in terms of link stress

occur when the abrupt failure of a node causes the partitioning of the mesh. For this reason, when node A has not received any refresh message from node B for a given time period, it starts actively probing node B. If B responds, A creates an overlay link to B to repair the partition. Otherwise, it presumes the departure of B and ultimately deletes B from its group member list.

The mesh constructed with *Narada* is heavily influenced by randomizing effects, such as link failures after node departures, additions of bad quality "emergency" links during partition repair, or additions of arbitrary links during node arrivals (recall that a newly arrived node creates arbitrary links to a few bootstrap nodes). Also, throughout the session, the varying conditions of the network substrate may render a previously good mesh formation obsolete. Due to the reasons mentioned above, ESM employs periodic re-evaluation of the constructed mesh. This re-constructed mesh is achieved autonomically by each member actively probing its mesh links and adding new, or dropping existing, links. Various heuristics are used for evaluation of the utility of link additions/drops, incorporating the effects of a potential overlay reconfiguration on the entire group [307].

Finally, data delivery in ESM follows the reverse-path forwarding [150] concept: a node n_i that receives a packet from source S via a mesh-neighbor n_j forwards the packet, if and only if n_j is the next hop on the shortest path of n_i to S. If that condition holds, n_i replicates the packet to all of its mesh-neighbors that use n_i as their next hop to reach S. The forwarding algorithm requires each node to maintain an entry to every other node in its routing table, which does not only contain the next overlay hop and the cost associated with the destination, but also the exact path that leads to it. This extra information is also used for loops-avoidance and count-to-infinity problems.

Figure 11.2(a) plots the delay penalty with regard to native unicast delay obtained with an ESM overlay. The data originate from a simulation of the

Narada protocol, incorporating 1024 nodes, 3145 links and a group size of 128, where the out-degree of each node was limited to 3 to 6 neighbors. The plot conveys that *Narada* minimizes the delay penalty experienced by node pairs with high unicast delay, whereas the penalty increases for small unicast delay values. The fairly large RDP values for node pairs with very low unicast delays are justified by the fact that even a small suboptimal configuration of the overlay topology magnifies the tiny delay of nodes residing close by to a large penalty. Still, the effective delay among these node pairs adheres to real-time requirements. In addition, the histogram in Figure 11.2(b) compares the physical link stress in ESM against two common alternatives: naive unicast and IP multicast using DVMRP [161]. Intuitively, native multicast is optimal in terms of stress (stress 1 across all links). It is worth mentioning that ESM manages to limit the maximum link stress to 9. In contrast, native unicast leads to a longer tail, loading few links with stress greater than 9, and worse, transmitting the same packet 128 times over a single link.

Summarizing, End System Multicast builds self-organizing overlays and therefore provides increased robustness with minimal configuration effort. As it does not require any support from network internal application-layer entities, it constitutes a ready-to-deploy solution. Furthermore, beyond the system specification presented herein, *Narada* can be optimized against various QoS metrics, such as throughput or node out-degree [306]. Finally, an asset of the entire ESM endeavor is that the system prototype has already been used for various Internet broadcasts, spanning multiple continents connecting home, academic and commercial network environments [305]. One of the limitations of ESM is its inability to scale to large group sizes, mainly because of the volume of routing information that needs to be exchanged and maintained by each node. A major concern common to all host-based ALM solutions is service interruption due to abrupt node failures. This failure type however, is one of the major open issues in peer-to-peer research in general.

11.4 Structured Overlays

Shortly after the introduction of multicast at the application layer, scaling to large group sizes (i.e., thousands of nodes) became a major concern in ALM research. Scalability is difficult to achieve with unstructured overlays - in the centralized case because of the bottleneck at the central point of control (as in all centralized systems) and in the distributed case owing to the large amount of state and control traffic required to establish a global view of the entire overlay topology (node clustering in those schemes may further increase the scalability of the distributed model). Instead, building a multicast service on top of a structured overlay provides increased scalability. In short, a *structured overlay* is an application-layer network whose nodes form a virtual space. Part III in this book contains a thorough review of

structured overlays and provides insight into issues pertaining to their design and performance.

Two approaches have been taken towards extending the routing infrastructure of a structured overlay to support multicast and these differ primarily in the manner they implement packet replication:

1. Directed flooding of each message across the virtual space.
2. Forming a tree, rooted at the group source used to distribute messages from a specific source to the group members (tree leaves).

In the following, we review one representative system from each of the two categories and end with a comparison of the two.

11.4.1 Flooding-Based Replication

A Content-Addressable Network (CAN [505] [504] and Section 8.3) utilizes a d-dimensional Cartesian coordinate space that is dynamically split into distinct coordinate zones (finite coordinate intervals of the space) and each zone is allocated to an overlay node according to a hash function. Two nodes in the space are neighbors if their zones overlap along d-1 dimensions and differ in exactly one dimension. Point-to-point routing is then accomplished by simply looking up the neighbor's set at each node and greedy-forwarding towards the node with coordinates closer to the destination's coordinates. New nodes join the system by contacting one of the well-known (e.g., retrievable via a DNS query) bootstrap nodes, and then the requests are forwarded to the destination node from which they claim a share of its space. Similarly, node departures result in merging the freed coordinate space with the zone of a neighboring node.

Multicast in CANs [506] is achieved by forming a "mini" CAN consisting of the CAN nodes that are members of the multicast group. Given a multicast group identified by G, the identifier is hashed to a coordinate space (x, y). The node responsible for the hashed coordinate space becomes the bootstrap node of the new "mini" CAN. The rest of the group members follow the usual CAN construction process to join the new multicast CAN. It is worth mentioning that as each node in the CAN has maximally $2d$ neighbors (where d stands for the CAN's dimensions), the per node multicast state requirements are independent of the number of multicast sources in the group.

Replication of messages to enable delivery to multiple destinations is realized as follows:

1. The source of the group forwards a message to all neighbors.
2. A node that receives a message from a neighboring node across dimension i forwards the message to all its neighbors across dimensions $1..(i-1)$ and to its neighbors across dimension i, but only in the opposite direction from which it received the message.

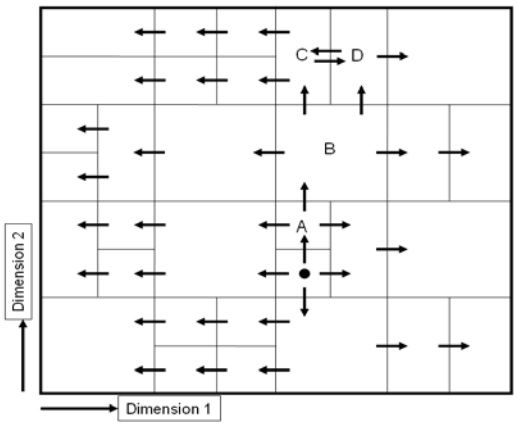


Fig. 11.3: Multicasting in a sample two-dimensional CAN

3. If a message has already traversed at least half of the distance from the source across a particular dimension, then the message is not forwarded by a receiving node.
4. Each node caches the sequence numbers of received messages and discards messages it has already forwarded.

Intuitively, rules 1 and 2 ensure that a message reaches all dimensions of the virtual space and additionally that the message is flooded to all nodes in one single dimension. Rule 3 prevents the flooding process from looping. A sample one-to-many communication in a two-dimensional CAN is depicted in Figure 11.3, where the listed forwarding rules are applied on a hop-by-hop basis to flood a message to all members of the CAN. Note that if the coordinate space is not perfectly partitioned, then a node may receive multiple copies of the same packet. This is particularly true for nodes C and D in the flooding example presented in Figure 11.3. [506] specifies enhancements to the elementary forwarding process for avoiding, but not entirely eliminate, an important fraction of duplicates.

11.4.2 Tree-Based Replication

Pastry [527] is a self-organizing overlay network of nodes, where each node routes requests and interacts with local instances of one or more applications. Each node in the Pastry overlay is assigned a 128-bit identifier, the *nodeId*. The nodeId determines the node's position in a circular nodeId space, ranging from 0 to 2^{128-1}. Message forwarding in Pastry is performed as follows: A node with nodeId id_1 intending to reach a destination node with nodeId id_n,

forwards the message to the node with nodeId id_2, where a prefix of id_2 matches in more than b bits the identifier id_n compared with the match of id_1. If no such match exists, the node forwards the message to the node with the numerically closest nodeId. It can be proven that the described routing scheme always converges [527]. Forwarding decisions, as well as mapping of nodeIds to IP addresses, is accomplished using routing state maintained at each node. A more detailed presentation of the Pastry system can be found in section 8.2, which discusses thoroughly the routing table structure (section 8.2.2), node bootstrapping (section 8.2.4), failure handling (section 8.2.4) and proximity matching from the geographical/IP-level proximity to the identifier space (section 8.2.5).

Scribe [109] builds a large-scale, fully decentralized, many-to-many distribution service on top of a Pastry infrastructure. Multicast distribution is essentially core-based, using a Pastry node as the rendezvous point. Group members "join" the tree routed at the well known rendezvous point, while group sources send multicast data directly to the core. Scribe exposes the following simple API calls to group communication applications:

- **Create(credentials,topicId)**: creates a group identified by a unique topicId, which is the result of hashing a textual description of the group's topic concatenated with the nodeId of the creator node. Credentials are used for applying access control to group creation.
- **Subscribe(credentials, topicId, eventHandler)**: commands the local Scribe instance to join the group identified by the topicId, resulting in receiving multicast data of a particular group. Arriving group data are passed to the specified event handler.
- **Unsubscribe(credentials, topicId)**: causes the local node to leave the specified group.
- **Publish(credentials, topicId, event)**: used by group sources to communicate an event (i.e., multicast data) to the specified group.

An application intending to create a group uses Scribe's "create" API call. Then Scribe passes a CREATE message using the topicId and credentials specified by the application to the local Pastry instance. The message is routed to the node with the nodeId numerically closest to the topicId. The receiving node checks the credentials, adds the topicId to the locally maintained groups and becomes the rendezvous point (RP) for the group. Adding a leaf to the tree rooted at the RP is a receiver-initiated process: Scribe asks Pastry to route a SUBSCRIBE message using the relevant topicId as the message destination key. At each node along the route towards the RP, the message is intercepted by Scribe. If the node already holds state of the particular topicId, it adds the preceding node's nodeId to its *children table* and forwards the message to the RP. If state does not exist, it creates a children table entry associated with the topicId and again forwards the message towards the RP. The latter process results in a reverse-path forwarding [150] distribution tree

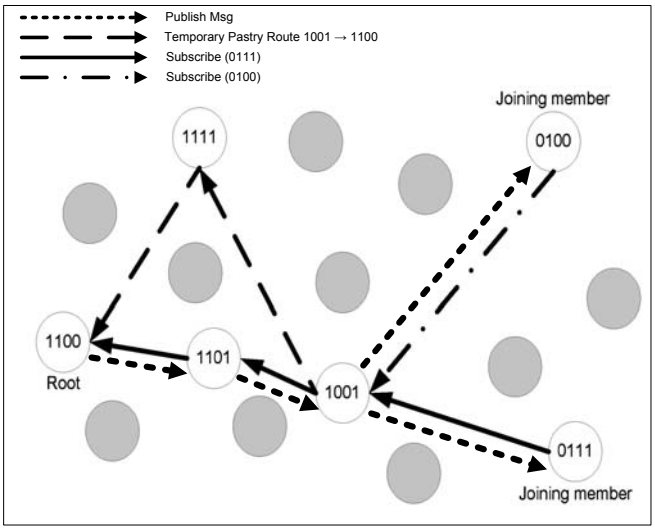
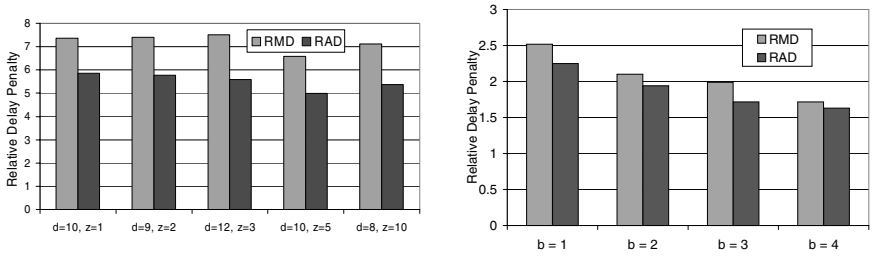


Fig. 11.4: Tree creation and data replication in a sample Scribe overlay.

(assuming path symmetry). Multicast sources address messages to the group by calling Scribe's "publish" primitive. This leads to a PUBLISH message - enclosing the group data - being forwarded towards the RP (using either Pastry or directly by using the IP address of the RP, if already discovered). As soon as the PUBLISH message reaches the RP node and provided that a) state of the specified topicId exists on the RP, and b) the credentials match, the RP "inserts" the message into the distribution tree by replicating it to all nodes in its children table. The same replication in the internal part of the tree assures that the PUBLISH message reaches all subscribed receivers. Finally, departures from the distribution tree are accomplished in a similar manner, using the "unsubscribe" API call provided by Scribe.

Figure 11.4 illustrates a sample Pastry overlay, where for the sake of presentation we only highlight nodes forming the Scribe distribution tree. In this example, we assume that the b parameter of Pastry is configured to $b = 1$, i.e., prefix matching is performed one bit at a time. We further assume that a group with $topicId = 1100$ has already been created. Because the RP node of a group is specified by locating the node with the numerically closest nodeId to the topicId of the group, in this example, node 1100 constitutes the RP. Let us assume that node 0111 is the first receiver that joins the group. For this, it asks Pastry to forward a SUBSCRIBE message to the RP. The message reaches its first "Pastry-hop", in this case node 1001. The SUBSCRIBE is intercepted by Scribe at node 1001, creating a children table entry with the nodeId 0111 as its single child and associating the table with the topicId. Finally, the message is forwarded towards the RP. En route to



(a) CAN (b) Scribe

Fig. 11.5: Relative delay penalty in various configurations of CAN multicast and Scribe.

the RP, the message accumulates state on the nodes it traverses, including the RP itself. As soon as the SUBSCRIBE message is treated at the RP, multicast data start flowing on the distribution tree (which in this case is a single node chain) towards node 1100. Now, let's look at how the arrival of a receiver in the "vicinity" of node 1100, namely of node 0100, affects the distribution tree. Following the common join process, node 0100 sends a SUBSCRIBE message towards the RP. The message is again delivered first to node 1001. Normally, the message would follow the path 1001-1111-1100 to reach the RP. However, since node 1001 already possesses state of the group 1100, it just adds node 0100 to its respective children table entry and terminates the SUBSCRIBE message.

11.4.3 Performance/Cost Evaluation

Selecting the more appropriate alternative between flooding and tree-based group communication is not trivial. Generally, the latency and bandwidth overhead for constructing per-group dedicated overlays are both greater with flooding mechanisms. On the other hand, flooding exclusively incorporates the members of a group into the group management and replication effort. As a result, they are considered appropriate for small groups. If this is a desired asset for the ALM service provider, flooding should be the mechanism of choice.

A thorough quantitative comparison of the two mechanisms is presented in [110]. The paper does not confine the evaluation to flooding over CAN, and tree-multicast over Pastry (a.k.a. Scribe), but experiments further with other alternative combinations as well (e.g., tree-multicast using CANs). The evaluation is performed using a random topology of 5050 routers and 80000 end system nodes randomly assigned to routers. All end systems are part of a sin-

	Maximum Stress	Average Stress
CAN (d=10, z=1)	1958	3.49
CAN (d=9, z=2)	1595	3.27
CAN (d=12, z=3)	1333	2.93
CAN (d=10, z=5)	985	2.73
CAN (d=8, z=10)	631	2.69
Scribe (RAND)	1910.6	3.87
Scribe (TOP)	23999.4	3.90

Table 11.1: Maximum and average link stress in CAN and Scribe multicast. RAND corresponds to randomly assigning nodeIds to newly arriving Pastry nodes, whereas TOP denotes a topologically aware assignment of nodeIds, taking proximity into consideration.

gle multicast group with the same single source over all runs. Since there are various parameter sets possible for the instantiation of a CAN/Pastry overlay, the authors have experimented with various configurations: a) in CAN, by tuning the dimensions d of the Cartesian space and the maximum number of nodes allowed in each zone, and b) in Pastry, by tuning the parameter b (number of bits matched during per hop prefix matching).

A small excerpt of the results is illustrated in Figure 11.5 and Table 11.1. Figure 11.5 shows that tree-based multicast over Pastry is superior to CAN-flooding in terms of both maximum (RMD) and average (RAD) relative delay penalty, making the former more suitable for delay-sensitive applications (e.g., multi-point teleconferencing). In contrast, certain configurations of CAN-flooding manage to economize link utilization compared with Pastry, as outlined in Table 11.1. Consequently, for a non delay-critical application (e.g., TV broadcasting), CAN-flooding offers a cheaper solution. For more detailed comparisons, including further metrics and evaluation against a larger number of concurrent groups, please refer to [110].

11.5 Hot Topics

In this section we touch on various issues driving the future of application-layer multicast. Some of them are already part of ongoing work; others promise to unleash the full potential of multicast overlays.

Topology-Awareness:
Matching proximity in the overlay sense with the actual proximity of the underlying IP substrate is key in improving the performance (delivery delay, throughput) and cost (stress) of ALM. While this is a pure necessity in structured overlays [502], efforts are being spent on achieving a closer match in unstructured overlays as well [370].

Quality of Service:
Recently, the provision of (probabilistic) QoS guarantees in overlay communication [383] is receiving increasing interest. To name an example, an overlay node can take advantage of redundancy in overlay paths [539] to a given destination and alternate over the several path options according to path conditions. For instance, if node A is able to reach node B using two loss-disjoint paths P1 and P2, A is able to pick the path with the lowest loss rate to increase quality [538]. Similarly, multi-path routing together with redundant coding inside the overlay can be used in k-redundant, directed acyclic graphs to increase data throughput [643].

Multi-source Support:
Many of the existing ALM schemes employ source-specific multicast trees. Still, various multicast applications inherently have multiple sources (such as teleconferencing or online multi-player games). The trivial solution manifests itself by creating one tree per source; still, it is evident that this is not the most efficient solution in all practical cases. Alternatively, the overlay routing algorithm may take application semantics into consideration to provide economical multi-source support [341] (e.g., by creating trees on demand and applying tree caching).

Security:
Malicious node behavior can harm the performance and stability of an application-layer multicast overlay. For example, Mathy et al. showed in [401] the impact of malicious nodes reporting false delay measurement values. Clearly, shielding overlay networks with powerful cheating detection and avoidance mechanisms is an interesting challenge.

11.6 Summary

In the preceding sections, we have introduced a first set of interesting applications of peer-to-peer networks: application-layer multicast. In this field, peer-to-peer technology has helped to overcome the slow adaptation of multicast mechanisms at the network layer. While unstructured and centralized peer-to-peer systems allowed for fast deployment of those networks, the family of unstructured peer-to-peer networks offer *unlimited* scalability. The vast amount of ongoing work will unleash further improvements of the existing multicast systems and introduce new applications of peer-to-peer technology in other networking areas.