SCRIBE: The design of a large-scale event notification infrastructure*

Antony Rowstron[1], Anne-Marie Kermarrec[1], Miguel Castro[1], and Peter Druschel[2]

[1] Microsoft Research,
7 J J Thomson Avenue, Cambridge, CB3 0FB, UK.
antr,anne-mk,mcastro@microsoft.com
[2] Rice University MS-132, 6100 Main Street,
Houston, TX 77005-1892, USA.
druschel@cs.rice.edu

Abstract. This paper presents Scribe, a large-scale event notification infrastructure for topic-based publish-subscribe applications. Scribe supports large numbers of topics, with a potentially large number of subscribers per topic. Scribe is built on top of Pastry, a generic peer-to-peer object location and routing substrate overlayed on the Internet, and leverages Pastry's reliability, self-organization and locality properties. Pastry is used to create a topic (group) and to build an efficient multicast tree for the dissemination of events to the topic's subscribers (members). Scribe provides weak reliability guarantees, but we outline how an application can extend Scribe to provide stronger ones.

1 Introduction

Publish-subscribe has emerged as a promising paradigm for large-scale, Internet based distributed systems. In general, subscribers register their interest in a topic or a pattern of events and then asynchronously receive events matching their interest, regardless of the events' publisher. Topic-based publish-subscribe [1–3] is very similar to group-based communication; subscribing is equivalent to becoming a member of a group. For such systems the challenge remains to build an infrastructure that can scale to, and tolerate the failure modes of the general Internet.

Techniques such as SRM (Scalable Reliable Multicast Protocol) [4] or RMTP (Reliable Message Transport Protocol) [5] have added reliability to network-level IP multicast [6, 7] solutions. However, tracking membership remains an issue in router-based multicast approaches and the lack of wide deployment of IP multicast limits their applicability. As a result, application-level multicast is gaining popularity. Appropriate algorithms and systems for scalable subscription management and scalable, reliable propagation of events are still an active research area [8–11].

Recent work on peer-to-peer overlay networks offers a scalable, self-organizing, fault-tolerant substrate for decentralized distributed applications [12–15]. Such systems

* Appears in the proceedings of 3rd International Workshop on Networked Group Communication (NGC2001), UCL, London, UK, November 2001.

offer an attractive platform for publish-subscribe systems that can leverage these properties. In this paper we present Scribe, a large-scale, decentralized event notification infrastructure built upon Pastry, a scalable, self-organizing peer-to-peer location and routing substrate with good locality properties [12]. Scribe provides efficient application-level multicast and is capable of scaling to a large number of subscribers, publishers and topics.

Scribe and Pastry adopt a fully decentralized peer-to-peer model, where each participating node has equal responsibilities. Scribe builds a multicast tree, formed by joining the Pastry routes from each subscriber to a rendez-vous point associated with a topic. Subscription maintenance and publishing in Scribe leverages the robustness, self-organization, locality and reliability properties of Pastry. Section 2 gives an overview of the Pastry routing and object location infrastructure. Section 3 describes the basic design of Scribe and we discuss related work in Section 4.

2 Pastry

In this section we briefly sketch Pastry [12]. Pastry forms a secure, robust, self-organizing overlay network in the Internet. Any Internet-connected host that runs the Pastry software and has proper credentials can participate in the overlay network.

Each Pastry node has a unique, 128-bit nodeId. The set of existing nodeIds is uniformly distributed; this can be achieved, for instance, by basing the nodeId on a secure hash of the node's public key or IP address. Given a message and a key, Pastry reliably routes the message to the Pastry node with a nodeId that is numerically closest to the key, among all live Pastry nodes. Assuming a Pastry network consisting of N nodes, Pastry can route to any node in less than $\lceil log_{2^b} N \rceil$ steps on average (b is a configuration parameter with typical value 4). With concurrent node failures, eventual delivery is guaranteed unless $\lfloor l/2 \rfloor$ nodes with *adjacent* nodeIds fail simultaneously (l is a configuration parameter with typical value 16).

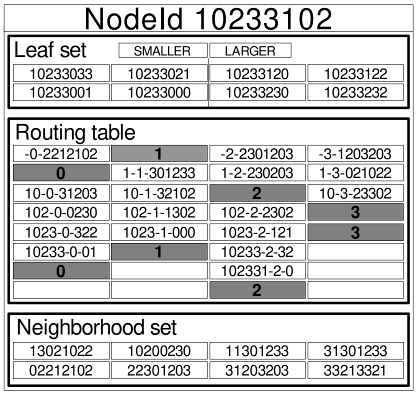

Fig. 1. State of a hypothetical Pastry node with nodeId 10233102, $b = 2$. All numbers are in base 4. The top row of the routing table represents level zero. The neighborhood set is not used in routing, but is needed during node addition/recovery.

The tables required in each Pastry node have only $(2^b - 1) * \lceil log_{2^b} N \rceil + 2l$ entries, where each entry maps a nodeId to the associated node's IP address. Moreover, after a node failure or the arrival of a new node, the invariants in all affected routing tables can be restored by exchanging $O(log_{2^b} N)$ messages. In the following, we briefly sketch the Pastry routing scheme. A full description and evaluation of Pastry can be found in [12].

For the purposes of routing, nodeIds and keys are thought of as a sequence of digits with base 2^b. A node's routing table is organized into $\lceil log_{2^b} N \rceil$ rows with $2^b - 1$ entries each. The $2^b - 1$ entries in row n of the routing table each refer to a node whose nodeId matches the present node's nodeId in the first n digits, but whose $n + 1$th digit has one of the $2^b - 1$ possible values other than the $n + 1$th digit in the present node's id. The uniform distribution of nodeIds ensures an even population of the nodeId space; thus, only $\lceil log_{2^b} N \rceil$ levels are populated in the routing table. Each entry in the routing table refers to one of potentially many nodes whose nodeId have the appropriate prefix. Among such nodes, the one closest to the present node (according to a scalar proximity metric, such as the delay or the number of IP routing hops) is chosen in practice.

In addition to the routing table, each node maintains IP addresses for the nodes in its *leaf set*, i.e., the set of nodes with the $l/2$ numerically closest larger nodeIds, and the $l/2$ nodes with numerically closest smaller nodeIds, relative to the present node's nodeId. Figure 1 depicts the state of a hypothetical Pastry node with the nodeId 10233102 (base 4), in a system that uses 16 bit nodeIds and a value of $b = 2$.

In each routing step, a node normally forwards the message to a node whose nodeId shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the present node's id. If no such node is found in the routing table, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node, but is numerically closer to the key than the present node's id. Such a node must be in the leaf set unless the message has already arrived at the node with numerically closest nodeId or its neighbor. And, unless $\lfloor |l|/2 \rfloor$ adjacent nodes in the leaf set have failed simultaneously, at least one of those nodes must be live.

2.1 Locality

Next, we discuss Pastry's locality properties, i.e., the properties of Pastry's routes with respect to the proximity metric. The proximity metric is a scalar value that reflects the "distance" between any pair of nodes, such as the number of IP routing hops, geographic distance, delay, or a combination thereof. It is assumed that a function exists that allows each Pastry node to determine the "distance" between itself and a node with a given IP address.

We limit our discussion to two of Pastry's locality properties that are relevant to Scribe. The first property is the total distance, in terms of the proximity metric, that messages are traveling along Pastry routes. Recall that each entry in the node routing tables is chosen to refer to the nearest node, according to the proximity metric, with the appropriate nodeId prefix. As a result, in each step a message is routed to the nearest node with a longer prefix match. Simulations show that, given a network topology based on the Georgia Tech model [16], the average distance traveled by a message is less than 66% higher than the distance between the source and destination in the underlying Internet.

Let us assume that two nodes within distance d from each other route messages with the same key, such that the distance from each node to the node with nodeId closest to the key is much larger than d. The second locality property is concerned with the "distance" the messages travel until they reach a node where their routes merge.

Simulations show that the average distance traveled by each of the two messages before their routes merge is approximately equal to the distance between their respective source nodes. These properties have a strong impact on the locality properties of the Scribe multicast trees, as explained in Section 3.

2.2 Node addition and failure

A key design issue in Pastry is how to efficiently and dynamically maintain the node state, i.e., the routing table, leaf set and neighborhood sets, in the presence of node failures, node recoveries, and new node arrivals. The protocol is described and evaluated in [12].

Briefly, an arriving node with the newly chosen nodeId X can initialize its state by contacting a nearby node A (according to the proximity metric) and asking A to route a special message using X as the key. This message is routed to the existing node Z with nodeId numerically closest to X. X then obtains the leaf set from Z, the neighborhood set from A, and the ith row of the routing table from the ith node encountered along the route from A to Z. One can show that using this information, X can correctly initialize its state and notify nodes that need to know of its arrival, thereby restoring all of Pastry's invariants.

To handle node failures, neighboring nodes in the nodeId space (which are aware of each other by virtue of being in each other's leaf set) periodically exchange keep-alive messages. If a node is unresponsive for a period T, it is presumed failed. All members of the failed node's leaf set are then notified and they update their leaf sets to restore the invariant. Since the leaf sets of nodes with adjacent nodeIds overlap, this update is trivial. A recovering node contacts the nodes in its last known leaf set, obtains their current leaf sets, updates its own leaf set and then notifies the members of its new leaf set of its presence. Routing table entries that refer to failed nodes are repaired lazily; the details are described in [12].

2.3 Pastry API

In this section, we briefly describe the application programming interface (API) exported by Pastry which is used in the Scribe implementation. The presented API is slightly simplified for clarity. Pastry exports the following operations:

route(msg,key) causes Pastry to route the given message to the node with nodeId numerically closest to key, among all live Pastry nodes.
send(msg,IP-addr) causes Pastry to send the given message to the node with the specified IP address, if that node is live. The message is received by that node through the deliver method.

Applications layered on top of Pastry must export the following operations:

deliver(msg,key) called by Pastry when a message is received and the local node's nodeId is numerically closest to key among all live nodes, or when a message is received that was transmitted via *send*, using the IP address of the local node.
forward(msg,key,nextId) called by Pastry just before a message is forwarded to the node with nodeId = nextId. The application may change the contents of the message or

the value of nextId. Setting the nextId to NULL will terminate the message at the local node.

In the following section, we will describe how Scribe is layered on top of the Pastry API. Other applications built on top of Pastry include PAST, a persistent, global storage utility [17, 18].

3 Scribe

Any Scribe node may create a *topic*; other nodes can then register their interest in the topic and become a *subscriber* to the topic. Any Scribe node with the appropriate credentials for the topic can then publish events, and Scribe disseminates these events to all the topic's subscribers. Scribe provides a best-effort dissemination of events, and specifies no particular event delivery order. However, stronger reliability guarantees and ordered delivery for a topic can be built on top of Scribe, as outlined in Section 3.2. Nodes can publish events, create and subscribe to many topics, and topics can have many publishers and subscribers. Scribe can support large numbers of topics with a wide range of subscribers per topic, and a high rate of subscriber turnover.

Scribe offers a simple API to its applications:

create(credentials, topicId) creates a topic with topicId. Throughout, the credentials are used for access control.
subscribe(credentials, topicId, eventHandler) causes the local node to subscribe to the topic with topicId. All subsequently received events for that topic are passed to the specified event handler.
unsubscribe(credentials, topicId) causes the local node to unsubscribe from the topic with topicId.
publish(credentials, topicId, event) causes the event to be published in the topic with topicId.

Scribe uses Pastry to manage topic creation, subscription, and to build a per-topic multicast tree used to disseminate the events published in the topic. Pastry and Scribe are fully decentralized, all decisions are based on local information, and each node has identical capabilities. Each node can act as a publisher, a root of a multicast tree, a subscriber to a topic, a node within a multicast tree, and any sensible combination of the above. Much of the scalability and reliability of Scribe and Pastry derives from this peer-to-peer model.

3.1 Scribe Implementation

A Scribe system consists of a network of Pastry nodes, where each node runs the Scribe application software. The Scribe software on each node provides the *forward* and *deliver* methods, which are invoked by Pastry whenever a Scribe message arrives. The pseudo-code for these Scribe methods, simplified for clarity, is shown in Figure 2 and Figure 3, respectively.

Recall that the forward method is called whenever a Scribe message is routed through a node. The deliver method is called when a Scribe message arrives at the node

```
forward(msg, key, nextId)
    switch msg.type is
        SUBSCRIBE :  if !(msg.topic ∈ topics)
                         topics = topics ∪ msg.topic
                         msg.source = thisNodeId
                         route(msg,msg.topic)
                     topics[msg.topic].children ∪ msg.source
                     nextId = null    // Stop routing the original message
```

Fig. 2. Scribe implementation of forward.

```
(1)  deliver(msg,key)
(2)      switch msg.type is
(3)          CREATE :       topics = topics ∪ msg.topic
(4)          SUBSCRIBE :    topics[msg.topic].children ∪ msg.source
(5)          PUBLISH :      ∀ node in topics[msg.topic].children
(6)                             send(msg,node)
(7)                         if subscribedTo(msg.topic)
(8)                             invokeEventHandler(msg.topic, msg)
(9)          UNSUBSCRIBE :  topics[msg.topic].children =
                                    topics[msg.topic].children - msg.source
(10)                        if (|topics[msg.topic].children| = 0)
(11)                            msg.source = thisNodeId
(12)                            send(msg,topics[msg.topic].parent)
```

Fig. 3. Scribe implementation of deliver.

with nodeId numerically closest to the message's key, or when a message was addressed to the local node using the Pastry *send* operation. The possible message types in Scribe are SUBSCRIBE, CREATE, UNSUBSCRIBE and PUBLISH; the roles of these messages are described in the next sections.

The following variables are used in the pseudocode: *topics* is the set of topics that the local node is aware of, *msg.source* is the nodeId of the message's source node, *msg.event* is the published event (if present), *msg.topic* is the topicId of the topic and *msg.type* is the message type.

Topic Management Each topic has a unique *topicId*. The Scribe node with a nodeId numerically closest to the topicId acts as the *rendez-vous point* for the associated topic. The rendez-vous point forms the root of a multicast tree created for the topic.

To create a topic, a Scribe node asks Pastry to route a CREATE message using the topicId as the key (e.g. route(CREATE,topicId)). Pastry delivers this message to the node with the nodeId numerically closest to topicId. The Scribe deliver method adds the topic to the list of topics it already knows about (line 3 of Figure 3). It also checks the credentials to ensure that the topic can be created, and stores the credentials in the topics set. This Scribe node becomes the rendez-vous point for the topic.

The topicId is the hash of the topic's textual name concatenated with its creator's name. The hash is computed using a collision resistant hash function (e.g. SHA-1 [19]), which ensures a uniform distribution of topicIds. Since Pastry nodeIds are also uniformly distributed, this ensures an even distribution of topics across Pastry nodes. A topicId can be generated by any Scribe node using only the textual name of the topic and its creator, without the need for an additional naming service. Of course, proper credentials are necessary to subscribe or publish in the associated topic.

Membership management Scribe creates a multicast tree, rooted at the rendez-vous point, to disseminate the events published in the topic. The multicast tree is created using a scheme similar to reverse path forwarding [20]. The tree is formed by joining the Pastry routes from each subscriber to the rendez-vous point. Subscriptions to a topic are managed in a decentralized manner to support large and dynamic sets of subscribers.

Scribe nodes that are part of a topic's multicast tree are called *forwarders* with respect to the topic; they may or may not be subscribers to the topic. Each forwarder maintains a *children table* for the topic containing an entry (IP address and NodeId) for each of its children in the multicast tree.

When a Scribe node wishes to subscribe to a topic, it asks Pastry to route a SUBSCRIBE message with the topic's *topicId* as the key (e.g. route (SUBSCRIBE,topicId)). This message is routed by Pastry towards the topic's rendez-vous point. At each node along the route, Pastry invokes Scribe's forward method. Forward (lines 3 to 8 in Figure 2) checks its list of topics to see if it is currently a forwarder; if so, it accepts the node as a child, adding it to the children table. If the node is not already a forwarder, it creates an entry for the topic, and adds the source node as a child in the associated children table. It then becomes a forwarder for the topic by sending a SUBSCRIBE message to the next node along the route from the original subscriber to the rendez-vous point. The original message from the source is terminated; this is achieved by setting nextId = null, in line 8 of Figure 2.

Figure 4 illustrates the subscription mechanism. The circles represent nodes, and some of the nodes have their *nodeId* shown. For simplicity $b = 1$, so the prefix is matched one bit at a time. We assume that there is a topic with *topicId* 1100 whose rendez-vous point is the node with the same identifier. The node with *nodeId* 0111 is subscribing to this topic. In this example, Pastry routes the SUBSCRIBE message to node 1001; then the message from 1001 is routed to 1101; finally, the message from 1101 arrives at 1100. This route is indicated by the solid arrows in Figure 4.

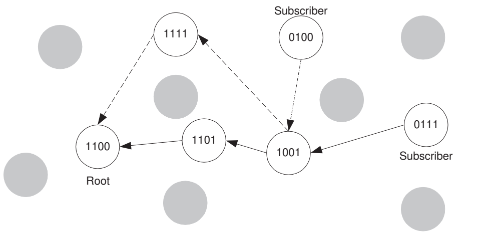

Fig. 4. Base Mechanism for Subscription and Multicast Tree Creation.

Let us assume that nodes 1001 and 1101 are not already forwarders for topic 1100. The subscription of node 0111 causes the other two nodes along the route to become forwarders for the topic, and causes them to add the preceding node in the route to their children tables. Now let us assume that node 0100 decides to subscribe to the same topic. The route that its SUBSCRIBE message would take is shown using dot-dash arrows. Since node 1001 is already a forwarder, it adds node 0100 to its children table for the topic, and the SUBSCRIBE message is terminated.

When a Scribe node wishes to unsubscribe from a topic, a node locally marks the topic as no longer required. If there are no entries in the children table, it sends a UNSUBSCRIPTION message to its parent in the multicast tree, as shown in lines 9 to 12 in Figure 3. The message proceeds recursively up the multicast tree, until a node is reached that still has entries in the children table after removing the departing child. It should be noted that nodes in the multicast tree are aware of their parent's nodeId only after they have received an event from their parent. Should a node wish to unsubscribe before receiving an event, the implementation transparently delays the unsubscription until the first event is received.

The subscriber management mechanism is efficient for topics with different numbers of subscribers, varying from one to all Scribe nodes. The list of subscribers to a topic is distributed across the nodes in the multicast tree. Pastry's randomization properties ensure that the tree is well balanced and that the forwarding load is evenly balanced across the nodes. This balance enables Scribe to support large numbers of topics and subscribers per topics. Subscription requests are handled locally in a decentralized fashion. In particular, the rendez-vous point does not handle all subscription requests.

The locality properties of Pastry (discussed in Section 2.1) ensure that the network routes from the root to each subscriber are short with respect to the proximity metric. In addition, subscribers that are close with respect to the proximity metric tend to be children of a parent in the multicast tree that is also close to them. This reduces stress on network links because the parent receives a single copy of the event message and forwards copies to its children along short routes.

Event dissemination Publishers use Pastry to locate the rendez-vous point of a topic. If the publisher is aware of the rendez-vous point's IP address then the PUBLISH message can be sent straight to the node. If the publisher does not know the IP address of the rendez-vous point, then it uses Pastry to route to that node (e.g. route(PUBLISH, topicId)), and asks the rendez-vous point to return its IP address to the publisher. Events are disseminated from the rendez-vous point along the multicast tree in the obvious way (lines 5 and 6 of Figure 3).

The caching of the rendez-vous point's IP address is an optimization, to avoid repeated routing through Pastry. If the rendez-vous point fails then the publisher can route the event through Pastry and discover the new rendez-vous point. If the rendez-vous point has changed because a new node has arrived, then the old rendez-vous point can forward the publish message to the new rendez-vous point and ask the new rendez-vous point to forward its IP address to the publisher.

There is a single multicast tree for each topic and all publishers use the above procedure to publish events. This allows the rendez-vous node to perform access control.

3.2 Reliability

Publish/subscribe applications may have diverse reliability requirements. Some topics may require reliable and ordered delivery of events, whilst others require only best-effort delivery. Therefore, Scribe provides only best-effort delivery of events but it offers a framework for applications to implement stronger reliability guarantees.

Scribe uses TCP to disseminate events reliably from parents to their children in the multicast tree, and it uses Pastry to repair the multicast tree when a forwarder fails.

Repairing the multicast tree. Periodically, each non-leaf node in the tree sends a heartbeat message to its children. When events are frequently published on a topic, most of these messages can be avoided since events serve as an implicit heartbeat signal. A child suspects that its parent is faulty when it fails to receive heartbeat messages. Upon detection of the failure of its parent, a node calls Pastry to route a SUBSCRIBE message to the topic's identifier. Pastry will route the message to a new parent, thus repairing the multicast tree.

For example, in Figure 4, consider the failure of node 1101. Node 1001 detects the failure of 1101 and uses Pastry to route a SUBSCRIBE message towards the root through an alternative route. The message reaches node 1111, which adds 1001 to its children table and, since it is not a forwarder, sends a SUBSCRIBE message towards the root. This causes node 1100 to add 1111 to its children table.

Scribe can also tolerate the failure of multicast tree roots (rendez-vous points). The state associated with the rendez-vous point, which identifies the topic creator and has an access control list, is replicated across the k closest nodes to the root node in the nodeId space (where a typical value of k is 5). It should be noted that these nodes are in the leaf set of the root node. If the root fails, its immediate children detect the failure and subscribe again through Pastry. Pastry routes the subscriptions to a new root (the live node with the numerically closest nodeId to the topicId), which takes over the role of the rendez-vous point. Publishers likewise discover the new rendez-vous point by routing via Pastry.

Children table entries are discarded unless they are periodically refreshed by an explicit message from the child, stating its continued interest in the topic.

This tree repair mechanism scales well: fault detection is done by sending messages to a small number of nodes, and recovery from faults is local; only a small number of nodes ($O(log_{2^b} N)$) is involved.

Providing additional guarantees. By default, Scribe provides reliable, ordered delivery of events only if the TCP connections between the nodes in the multicast tree do not break. For example, if some nodes in the multicast tree fail, Scribe may fail to deliver events or may deliver them out of order.

Scribe provides a simple mechanism to allow applications to implement stronger reliability guarantees. Applications can define the following upcall methods, which are invoked by Scribe.

forwardHandler(msg) is invoked by Scribe before the node forwards an event, msg, to its children in the multicast tree. The method can modify msg before it is forwarded.
subscribeHandler(msg) is invoked by Scribe after a new child is added to one of the

node's children tables. The argument is the SUBSCRIBE message.
faultHandler(msg) is invoked by Scribe when a node suspects that its parent is faulty. The argument is the SUBSCRIBE message that is sent to repair the tree. The method can modify msg to add additional information before it is sent.

For example, an application can implement ordered, reliable delivery of events by defining the upcalls as follows. The forwardHandler is defined such that the root assigns a sequence number to each event and such that recently published events are buffered by the root and by each node in the multicast tree. Events are retransmitted after the multicast tree is repaired. The faultHandler adds the last sequence number, n, delivered by the node to the SUBSCRIBE message and the subscribeHandler retransmits buffered events with sequence numbers above n to the new child. To ensure reliable delivery, the events must be buffered for an amount of time that exceeds the maximal time to repair the multicast tree after a TCP connection breaks.

To tolerate root failures, the root needs to be replicated. For example, one could choose a set of replicas in the leaf set of the root and use an algorithm like Paxos [21] to ensure strong consistency.

4 Related work

Like Scribe, Overcast [22] and Narada [23] implement multicast using a self-organizing overlay network, and they assume only unicast support from the underlying network layer. Overcast builds a source-rooted multicast tree using end-to-end bandwidth measurements to optimize bandwidth between the source and the various group members. Narada uses a two step process to build the multicast tree. First, it builds a mesh per group containing all the group members. Then, it constructs a spanning tree of the mesh for each source to multicast data. The mesh is dynamically optimized by performing end-to-end latency measurements and adding and removing links to reduce multicast latency. The mesh creation and maintenance algorithms assume that all group members know about each other and, therefore, do not scale to large groups.

Scribe builds a multicast tree on top of a Pastry network, and relies on Pastry to optimize route locality based on a proximity metric (e.g. IP hops or latency). The main difference is that the Pastry network can scale to an extremely large number of nodes because the algorithms to build and maintain the network have space and time costs of $O(log_{2^b} N)$. This enables support for extremely large groups and sharing of the Pastry network by a large number of groups.

The recent work on Bayeux [11] is the most similar to Scribe. Bayeux is built on top of a scalable peer-to-peer object location system called Tapestry [13] (which is similar to Pastry). Like Scribe, it supports multiple groups, and it builds a multicast tree per group on top of Tapestry but this tree is built quite differently. Each request to join a group is routed by Tapestry all the way to the node acting as the root. Then, the root records the identity of the new member and uses Tapestry to route another message back to the new member. Every Tapestry node (or router) along this route records the identity of the new member. Requests to leave a group are handled in a similar way.

Bayeux has two scalability problems when compared to Scribe. Firstly, it requires nodes to maintain more group membership information. The root keeps a list of all group members, the routers one hop away from the route keep a list containing on average $\frac{S}{b}$ members (where b is the base used in Tapestry routing), and so on. Secondly, Bayeux generates more traffic when handling group membership changes. In particular, all group management traffic must go through the root. Bayeux proposes a multicast tree partitioning mechanism to ameliorate these problems by splitting the root into several replicas and partitioning members across them. But this only improves scalability by a small constant factor.

In Scribe, the expected amount of group membership information kept by each node is small, as the subscribers are distributed over the nodes. Additionally, group join and leave requests are handled locally. This allows Scribe to scale to extremely large groups and to deal with rapid changes in group membership efficiently.

The mechanisms for fault resilience in Bayeux and Scribe are also very different. All the mechanisms for fault resilience proposed in Bayeux are sender-based whereas Scribe uses a receiver-based mechanism. In Bayeux, routers proactively duplicate outgoing packets across several paths or perform active probes to select alternative paths. Both these schemes have some disadvantages. The mechanisms that perform packet duplication consume additional bandwidth, and the mechanisms that select alternative paths require replication and transfer of group membership information across different paths. Scribe relies on heartbeats sent by parents to their children in the multicast tree to detect faults, and children use Pastry to reroute to a different parent when a fault is detected. Additionally, Bayeux does not provide a mechanism to handle root failures whereas Scribe does.

5 Conclusions

We have presented Scribe, a large-scale and fully decentralized event notification system built on top of Pastry, a peer-to-peer object location and routing substrate overlayed on the Internet. Scribe is designed to scale to large numbers of subscribers and topics, and supports multiple publishers per topic.

Scribe leverages the scalability, locality, fault-resilience and self-organization properties of Pastry. Pastry is used to maintain topics and subscriptions, and to build efficient multicast trees. Scribe's randomized placement of topics and multicast roots balances the load among participating nodes. Furthermore, Pastry's properties enable Scribe to exploit locality to build efficient multicast trees and to handle subscriptions in a decentralized manner.

Fault-tolerance in Scribe is based on Pastry's self-organizing properties. Scribes default reliability scheme ensures automatic adaptation of the multicast tree to node and network failures. Event dissemination is performed on a best-effort basis; consistent ordering of delivered events is not guaranteed. However, stronger reliability models can be layered on top of Scribe.

Simulation results, based on a realistic network topology model and presented in [24], indicate that Scribe scales well. It efficiently supports a large number of nodes, topics, and a wide range of subscribers per topic. Hence, Scribe can concurrently support applications with widely different characteristics. Results also show that it balances the load among participating nodes, while achieving acceptable delay and link stress, when compared to network-level (IP) multicast.

References

1. Talarian Corporation. *Everything You need to know about Middleware: Mission-Critical Interprocess Communication (White Paper)*. http://www.talarian.com/, 1999.
2. TIBCO. *TIB/Rendezvous White Paper*. http://www.rv.tibco.com/whitepaper.html, 1999.
3. P.T. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. Technical Report DSC ID:2000104, EPFL, January 2001.
4. S. Floyd, V. Jacobson, C.G. liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transaction on networking*, pages 784–803, December 1997.
5. J.C. Lin and S. Paul. A reliable multicast transport protocol. In *Proc. of IEEE INFOCOM'96*, pages 1414–1424, 1996.
6. S. Deering and D. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems*, 8(2), May 1990.
7. S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. The PIM Architecture for Wide-Area Multicast Routing. *IEEE/ACM Transactions on Networking*, 4(2), April 1996.
8. K.P. Birman, M. Hayden, O.Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, May 1999.
9. Patrick Eugster, Sidath Handurukande, Rachid Guerraoui, Anne-Marie Kermarrec, and Petr Kouznetsov. Lightweight probabilistic broadcast. In *Proceedings of The International Conference on Dependable Systems and Networks (DSN 2001)*, July 2001.
10. Luis F. Cabrera, Michael B. Jones, and Marvin Theimer. Herald: Achieving a global event notification service. In *HotOS VIII*, May 2001.
11. Shelly Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination. In *Proc. of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, June 2001.
12. Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Middleware 2001*, Heidelberg, Germany, November 2001.
13. Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.
14. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. ACM SIGCOMM'01*, San Diego, CA, August 2001.
15. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. of ACM SIGCOMM*, August 2001.
16. E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *INFOCOM96*, 1996.
17. Peter Druschel and Antony Rowstron. PAST: A persistent and anonymous store. In *HotOS VIII*, May 2001.
18. Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. ACM SOSP'01*, Banff, Canada, October 2001.

19. FIPS 180-1. Secure hash standard. Technical Report Publication 180-1, Federal Information Processing Standard (FIPS), National Institute of Standards and Technology, US Department of Commerce, Washington D.C., April 1995.
20. Yogen K. Dalal and Robert Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, 21(12):1040–1048, 1978.
21. L. Lamport. The Part-Time Parliament. Report Research Report 49, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, September 1989.
22. John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole. Overcast: Reliable Multicasting with an Overlay Network. In *Proc. of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, pages 197–212, October 2000.
23. Yang hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Proc. of ACM Sigmetrics*, pages 1–12, June 2000.
24. Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralized publish-subscribe infrastructure, September 2001. Submitted for publication. *http://www.research.microsoft.com/∼antr/scribe*.