Διπλωματική Εργασία
Μεταπτυχιακού Διπλώματος Ειδίκευσης

«On the performance of Application-level Multicast Routing»

Δαγαλάκη Ευσεβία
Επιβλέπων: Ξυλωμένος Γεώργιος

ΑΘΗΝΑ, ΙΟΥΛΙΟΣ 2008
# Table of Contents

1 Introduction ...................................................................................................1  
  1.1 Motivation .............................................................................................1  
  1.2 Objectives ............................................................................................2  
2 Peer-to-peer systems ....................................................................................7  
  2.1 Peer-to-peer Applications ...................................................................7  
  2.2 Distributed Hash Tables ....................................................................8  
  2.3 Pastry ..................................................................................................10  
3 Publish-subscribe and multicast .................................................................15  
  3.1 Publish-subscribe ................................................................................15  
  3.2 IP Multicast .........................................................................................15  
  3.3 Application-level Multicast .................................................................17  
  3.4 Scribe ...................................................................................................18  
4 Evaluating multicast protocols ................................................................... 23  
  4.1 Evaluation framework ....................................................................... 23  
  4.2 Alternative Multicast Schemes .......................................................... 25  
  4.3 Evaluation of Application-Level Multicast ........................................ 25  
5 Simulation Framework ............................................................................... 29  
  5.1 OMNeT++ .......................................................................................... 29  
  5.2 OverSim ..............................................................................................30  
6 Implementation .......................................................................................... 35  
  6.1 OverSim classes and message types .................................................. 35  
  6.2 Code Modifications and Extensions .................................................. 40  
7 Evaluation ................................................................................................... 47  
  7.1 Experimental Setup ........................................................................... 47  
  7.2 Path Ratio ............................................................................................ 49  
  7.3 Node Stress ........................................................................................ 52  
8 Conclusions ................................................................................................. 57  
9 Future Work ............................................................................................... 59  
10 References ...............................................................................................61
List of Figures

Figure 1: Scenario where routers do not participate in Pastry and Scribe...........3
Figure 2: Scenario where routers participate in Pastry and Scribe. .................4
Figure 3: Routing a message from node 65a1fc with key d46a1c....................11
Figure 4: Modular Architecture of OverSim .................................................32
Figure 5: The Global Observer class............................................................35
Figure 6: The ALM Message class...............................................................36
Figure 7: The IPv4 Underlay class...............................................................37
Figure 8: The Pastry Modules class.............................................................38
Figure 9: The Multicast Scribe class.............................................................38
Figure 10: The modified ALM Message class..............................................40
Figure 11: Structure of InfoList.................................................................41
Figure 12: Control Flow for a join request (part 1).....................................41
Figure 13: Control Flow for a join request part 2)....................................42
Figure 14: Cumulative distribution of overlay vs. direct path ratio .............50
Figure 15: Number of Children Tables per Scribe node............................54
Figure 16: Total Number of Children Table Entries per Scribe node.........54
List of Tables

Table 1: ALM Message Types ................................................................. 36
Table 2: Underlay Module Types ........................................................... 37
Table 3: Pastry Module Types ............................................................... 38
Table 4: Scribe Message Types ............................................................. 39
Table 5: Example of run file ................................................................. 48
Table 6: Path Ratio for 6 groups ............................................................ 49
Table 7: Path Ratio for 12 groups .......................................................... 49
Table 8: Path Ratio for 60 groups .......................................................... 49
Table 9: Scenario 3: mean values for all runs and all cases ......................... 50
Table 10: Scenario 5: mean values for all runs and all cases ....................... 51
Table 11: Number of children tables (6 groups) ...................................... 52
Table 12: Number of children tables (12 groups) ..................................... 52
Table 13: Number of children tables (60 groups) .................................... 52
Table 14: Total Number of Children Table Entries (6 groups) .................... 53
Table 15: Total Number of Children Table Entries (12 groups) .................. 53
Table 16: Total Number of Children Table Entries (60 groups) ................. 53
Abstract

Despite its great success, the current Internet architecture faces several problems such as the non-cooperative behavior of end-users and the heterogeneity of network and user devices. Spam mail and denial of service (DoS) attacks are examples of non-cooperative behavior. Increasing demand of real-time and multimedia applications with high Quality of Service (QoS) remains a challenge for Internet Service Providers (ISPs) who need to provide users with differentiated services due to the heterogeneity of network and user devices. Moreover, ISPs should provide services that are scalable to a large number of users. To meet these challenges, optimal routing is necessary.

An innovative approach to overcome the shortcomings of the current Internet is the publish-subscribe (pub/sub) model. In pub/sub systems senders publish events and receivers subscribe to the publications they want to receive. Multicast routing is a bandwidth-saving solution that provides scalable dissemination of data and seems ideal for pub/sub systems where each publication may be of interest to numerous subscribers. It was first implemented in the network level as IP Multicast, but it has not experienced large-scale deployment. Overlay-based routing has recently gained attention through file sharing applications as it provides scalability, self-organization and fault-tolerance; it can also provide multicast without network support.

The objective of this work is to study the overlay routing process and to measure how optimal application-level multicast trees are. The Scribe application-layer multicast protocol is studied and used for simulation purposes. Scribe is built on top of the Pastry object location and routing substrate. Scribe multicast trees are examined under different topology scenarios which are differentiated with respect to the number of routers (access and backbone) that participate in the overlay. The metrics used are the overlay versus direct path ratio of the paths from each recipient to the rendezvous point of the tree and the node stress. The overlay versus path ratio is the ratio between the number of physical hops induced by the overlay routing process and the number of hops on the optimal physical path. The node stress is the number of children tables per Scribe node and the total number of children entries in all children tables for a Scribe node.
Acknowledgements

I wish to thank my examiners, Professor George Zulowiński and Professor George Polyzos, as well as the members of the laboratory of the Department of Economics of the University of Athens, who accepted me as a member of their research group.

During the meetings of the laboratory members, I had the opportunity to develop my research interests through group discussions and information on important research programs. I also thank them for encouraging me to actively participate in a very interesting research project, PSIRP, which was indeed the motivating factor for my thesis work.

Finally, I wish to thank the children of the laboratory, especially Basil Kemerlē and Pantelēs Frangoudē, and naturally Ntīno Katsarō, who with their kindness and positive attitude helped and encouraged me to create a very friendly working atmosphere and happy cooperation.

Eusêbía Dagalakē
1 Introduction

1.1 Motivation

The shortcomings of the current Internet architecture are more and more visible for end-users. The network accepts anything that the sender wants to send and will make a best-effort to deliver it to the receiver. The source for many problems lies in the false assumption that all users are cooperative and benign. Spam mail and denial of service attacks are examples of non-cooperative behavior. Furthermore, novel real-time and multimedia applications demand high quality of service. Internet Service Providers must meet the challenge to provide a large variety of such bandwidth-demanding applications and provide users with high quality of service as well as differentiated service for customers with different needs and different price sensitivity – some customers are willing to pay more for higher grade services.

To meet these challenges, ISPs are expanding their network infrastructures and resources to be able to support demanding applications and a growing number of customers. For performance optimization of an IP network, optimal routing must be accomplished. Multicast is an efficient solution for such applications as it results in bandwidth savings that make it an attractive service mainly to sources and administrators of low capacity domains. It also offers scalable dissemination of data and minimizes delay among participants.

IP Multicast is a solution that provides efficient group communication and maintains an efficient utilization of the available bandwidth. However, it has not yet experienced a large-scale deployment. It faces difficulties related to security issues, the need for special support from network devices, management problems as well as lack of an efficient multicast Congestion Control (CC) scheme [6]. Moreover, tracking membership remains an issue in router-based multicast approaches such as IP Multicast. For this reason, application-level multicast has attracted interest in the past few years as a feasible solution for applications that need multicast-based functionality.

An innovative approach is the publish-subscribe based paradigm to inter-networking. Senders publish events and receivers subscribe to the publications they want to receive. In a publish-subscribe system, only
publishers need to have names. Publish-subscribe is similar to group communication, hence it leverages multicast routing. However, there are many challenges to overcome, such as achieving rendezvous between publishers and subscribers, routing for finding (near) optimal forwarding paths and forwarding of data packets.

A number of overlay-based routing algorithms have been proposed for distributed pub-sub systems. An application-layer overlay network is implemented on top of the network layer and is intended to provide useful features such as fast deployment time, resilience and fault-tolerance. An overlay routing algorithm leverages network locality to provide additional services such as object location, storage, and synchronization services in an efficient and scalable manner. Moreover, overlay networks can support more complex network functionality on top of the IP routing functionality.

*Distributed Hash Tables* (DHTs) are an example of overlay-based communication. They provide a scalable, self-organizing and fault-tolerant substrate for decentralized distributed applications. The greatest challenge in large-scale peer-to-peer networks is the design of efficient algorithms for object location and routing within the network. Chord, CAN, Pastry and Tapestry are widely known DHT overlays. Scribe is an example of publish-subscribe system implemented on top of an overlay network and is based on the rendezvous point routing model. It is currently implemented on top of Pastry to support topic-based publish-subscribe applications. It leverages Pastry’s locality properties to build near optimal multicast trees.

### 1.2 Objectives

Past studies of application-layer multicast schemes concentrated on the quality of the multicast trees produced when only the end nodes participated in multicast routing. As overlay networks are becoming more popular however, it can be envisioned that the routers themselves may participate in this type of overlay based multicast routing if this is deemed to be a valuable service to the end users that they are serving. Adding routers to the overlay obviously increases network density, therefore membership load and network traffic load are distributed over more nodes, hence reducing node stress and
link stress, respectively. Moreover, the increased network density, together with the locality properties of Pastry routing, make it more possible for Scribe nodes to route join messages to a closer node with respect to the proximity metric than if there were no overlay routers. Consequently, Scribe forms paths that are closer to the underlying topology and the overlay versus direct path ratio decreases as well.

This is easier to see in the example depicted in Figures 1 and 2. Figure 1 corresponds to a topology where routers do not participate in the overlay or the implementation of multicast routing. There are two backbone routers (1.2.0.1, 1.3.0.1) and four access routers (1.4.0.1, 1.5.0.1, 1.6.0.1, 1.7.0.1). Each access router is responsible for an access network comprised of some overlay terminals (end hosts). The blue clouds indicate the access networks, the black lines correspond to direct links while the red lines correspond to overlay links, red circles indicate the overlay terminals that participate in a multicast group and green lines form the physical path from a subscriber to the rendezvous point of the group. In the figure, a multicast group of 4 nodes is depicted. The Node with IP ‘1.7.0.2’ is the root of the multicast tree for this group.

Figure 1: Scenario where routers do not participate in Pastry and Scribe.
According to the overlay connections, the number of physical hops induced by the overlay path from subscriber ‘1.6.0.2’ to the root is ten and the sequence of overlay hops for this overlay path is 1.7.0.2 to 1.5.0.2, with five intermediate physical hops, and 1.5.0.2 to 1.6.0.2 with five intermediate physical hops. The overlay versus direct path ratio is 2.5 (10/4) which is relatively high.

Figure 2 depicts the same network topology but now routers participate in the overlay and in multicast routing. We have added two overlay routers, the access router with IP ‘1.7.0.1’ and the backbone router with IP ‘1.2.0.1’. The multicast tree has changed because end hosts routed join messages to routers as they are closer to them with respect to the proximity metric. Here the number of physical hops induced by the overlay path from subscriber ‘1.6.0.2’ to the root of the tree ‘1.7.0.2’ is only four and the ratio is 1 which is ideal. It is therefore self-evident that participation of routers in the overlay and in multicast routing can greatly improve the performance of multicast routing.

**Figure 2: Scenario where routers participate in Pastry and Scribe.**

Considering the suboptimality of overlay based multicast routing, it is
important to assess the possible improvements that may be achieved by introducing routers into the overlay. Therefore, the goal of this thesis is to answer the following questions:

- How optimal are multicast trees created by Scribe, if we consider as optimal the trees that would be used by IP Multicast?
- Does the introduction of routers in Pastry and Scribe lead to better multicast trees, in the sense of reduced delays and lower node stress?
- What is the fraction of network routers that should participate in Pastry and Scribe in order to produce better multicast trees?

All these questions will be answered by using a fully fledged open source simulation environment, in contrast to previous studies of multicast trees created by Pastry and Scribe that employed simple custom simulators.
2 Peer-to-peer systems

2.1 Peer-to-peer Applications

A peer-to-peer (p2p) system is a distributed system with no central authority where network responsibilities are equally distributed among participating nodes. In p2p systems all nodes have identical capabilities and responsibilities and all communication is symmetric. In other words, a pure p2p network does not have the notion of clients or servers, only equal peer nodes that may act as both client and server. This model differs from the client-server model where the communication is to and from a central server. There are two basic applications of p2p networks:

- File Sharing: Each participating node permits other nodes to retrieve some of its local files. Therefore, the system must provide a lookup function to locate the node that has the requested file and retrieve it.

- Distributed Storage: Each participating node acts as a file server that permits other nodes to read and write its files. Therefore, the system must be able to locate the node that is responsible for storing a specific file (or part thereof) as well as for the reading and writing of the file.

In file sharing networks, files are only available to the nodes that own the files and can be retrieved only from them. On the other hand, in distributed storage systems, files are created from system clients who may or may not be involved in their storage. Both applications however face a common problem: given a key \( k \), how to locate the node that owns the data that are relate to \( k \). In a file sharing network, the key could be the file name and the data related to the key could be the address of the node from where the file can be retrieved. In a distributed storage system, the key could be the file name and the data could be the address of the node (or the addresses of the nodes in case a file is stored in multiple replicas) from where the file can be accessed for reading or writing. If the distributed storage system distributes file groups to several servers in order to increase reliability and system efficiency, the key could be the filename or the number of the file group that we are interested in, and the data could be the address of the node that offers this file group.

In a large p2p system, the number of nodes and keys could be very large. In
this chapter, we will see how a p2p system can be structured so that the lookup process for keys and the data related to the key will be efficient. Utilization of the data is out of the scope of this chapter, as this is dependent on the application. It should be noted that problems related to the utilization of the data retrieved by a key are simpler than the ones related to the key lookup, because the number of nodes involved for the utilization of specific data is small, whereas the number of nodes involved in key lookup equals the number of nodes in the system which could be large enough.

### 2.2 Distributed Hash Tables

Distributed hash tables (DHTs) are decentralized distributed systems offering a lookup service similar to a hash table for mapping name-value pairs. All participating nodes in a DHT are equally responsible for the maintenance of the DHT. Consequently, DHTs may scale to a large number of nodes, handling effectively node arrivals, departures and failures. DHTs are applicable in many areas and in particular they can encourage the design of more complex services, such as distributed file systems, peer-to-peer file sharing and content distribution systems, cooperative web caching, multicast, anycast, domain name services and instant messaging.

DHTs were originally introduced to address the limitations of peer-to-peer systems such as Napster, Gnutella and Freenet which targeted file sharing across different peers in the Internet. These systems differed in how they found the requested files. In Napster, there was a central entity that was responsible to route requests to those peers that had the answers. The drawback of this approach was that the central entity was a single point of failure. Gnutella used flooding to send the request to everyone in the network which was obviously less efficient than Napster. Finally, Freenet employed key based routing, associating each file with a key. Files with similar keys were clustered on a similar set of nodes, and queries could be routed unnecessarily to nodes that did not have the answers. Also Freenet could not guarantee that data would be found.

DHTs use a more structured key based routing procedure to maintain decentralization, efficiency and provide guaranteed lookup results. While they
support exact-match rather than keyword search, this functionality can be layered on top of a DHT. CAN, Chord, Pastry and Tapestry are four widely known DHTs in the networking community area.

A DHT consists of the following components:

- keyspace: the set of the available keys, such as the set of 160-bit strings
- keyspace partitioning: the keyspace is partitioned among participating nodes, so that each node owns a part of the keyspace
- overlay network: this network connects the participating nodes, allowing them to easily find the owner of a key in the keyspace

The storage and retrieval procedures proceed as follows. To store a file, the file’s name is hashed and a key $k$ is produced. Then a message $\text{put}(k, \text{data})$, where data is the file’s content, is sent to all overlay nodes until it reaches the node that is responsible for the key $k$ according to the keyspace partitioning. The pair $(k, \text{data})$ is stored in that node. For another node to retrieve the data, the node must hash again the name of the file and send a message $\text{get}(k)$ to all overlay nodes until it reaches the node that is responsible for the key.

Most DHTs use consistent hashing to map keys to nodes. The key space partitioning is done according to a distance function $d(k_1, k_2)$, which defines the distance between two keys $k_1$ and $k_2$. This distance is not related to geographical distance or network latency. Each node is assigned a unique key, called its identifier (ID). A node with ID $i$ is responsible for all the keys for which $i$ is the closest ID according to the distance function $d$. Consistent hashing ensures that the removal or addition of a node affects only the set of keys owned by nodes that have adjacent IDs, in contrast to traditional hash tables where removal or addition of an entry affects the entire keyspace and may lead to total remapping. Removal or addition of a node practically means a change of ownership which corresponds to bandwidth-intensive movement of objects stored in the DHT from one node to another. Therefore, a key challenge in DHTs is to minimize this reorganization in order to efficiently support high rates of churn (node arrival and failure).

For routing purposes, each participating node of a DHT maintains a set of
On the performance of Application-level Multicast Routing

links to its neighbors. These links form an overlay network of the participating
nodes. All DHT topologies have the essential property that a node either owns
a key $k$ or has a link to a node that is closer to $k$. Thus, a message is routed to
the owner of the key $k$ by making greedy choices. More specifically, at each
step, the message is forwarded to the neighbor whose ID is closer to $k$. If there
is no such neighbor, then the current node is the owner of the key. This
routing is also known as key based or content based routing.

There is a trade-off between node degree and route length in DHTs. Route
length with respect to the maximum number of hops in any route must be low,
so that requests complete quickly. In addition, the maximum number of
neighbors of any node (the node degree) must be low, so that the maintenance
overhead is not excessive. However, having shorter routes requires higher
maximum degrees. The most common choices for node degree and route
length are the following:

- Degree $O(1)$, route length $O(\log n)$
- Degree $O(\log n)$, route length $O(\log n / \log \log n)$
- Degree $O(\log n)$, route length $O(\log n)$
- Degree $O(n^{1/2})$, route length $O(1)$

2.3 Pastry

Pastry [1] is a generic peer-to-peer object location and routing system based
on self-organizing overlay network of nodes connected to the Internet. Pastry
is completely decentralized, fault-resilient, scalable and reliable, while
maintaining good route locality properties. Pastry is intended to support the
construction of peer-to-peer Internet applications like global file sharing, file
storage, group communication and naming systems.

A Pastry node can be any computer connected to the Internet that runs the
Pastry software. Each node routes client requests and may interact with
various applications. Each Pastry node is assigned a unique 128-bit ID
whenever it joins the Pastry system. This ID is used to locate the node in a
circular ID space, which ranges from 0 to $2^{128}-1$. Nodes are randomly
assigned IDs and this ensures an even distribution of IDs in the network. IDs
and keys can be thought of as a sequence of digits with base $2^b$, where $b$ is a configuration parameter, typically set to 4, that indicates the maximum number of bits of the key prefix matched to a given ID. Pastry routes a message with a key $k$ to the node whose ID is numerically closest to $k$.

![Figure 3: Routing a message from node 65a1fc with key d46a1c.](image)

The routing procedure is as follows. In each routing step, the node forwards the message to the node whose ID shares with $k$ a prefix that is at least one digit (or $b$ bits) longer than the prefix that $k$ shares with the current node’s ID. If no such node exists, then the current node forwards the message to a node whose ID shares with $k$ a prefix that has the same length with the prefix it shares with the current node’s ID but is numerically closer to $k$ than the current node’s ID. Figure 2 shows the path of an example message. In order to support this routing procedure, each node must maintain some state.

Each Pastry node maintains three main structures: a routing table, a neighborhood set and a leaf set. The routing table is organized in $\lceil \log_{2^b} N \rceil$ rows with $2^b - 1$ entries each. The $2^b - 1$ entries of row $n$ refer to nodes whose IDs share with the current node’s ID 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 of the current node’s ID. The neighborhood set contains the $M$ closest nodes to the current node according to the proximity metric. It is useful in maintaining locality properties as described below. The leaf set contains the $|L|/2$ nodes with numerically closest smaller IDs and the $|L|/2$ nodes with numerically closest larger IDs. The node that received a message with key $k$, first checks if the key falls within the range of IDs covered by its leaf set, otherwise it uses the
Routing performance experiments showed that in a network of \( N \) nodes, a node routes a message with key \( k \) to the numerically closest node in less than \([\log_2 N]\) steps. In case of concurrent node failures, delivery is guaranteed unless \([|L|/2]\) nodes with adjacent IDs fail simultaneously.

When a new node arrives in the Pastry network, it needs to initialize its state (neighborhood set, leaf set and routing table) and inform the other nodes of its presence. Let us assume that the new node has a nearby Pastry node, \( A \). In order for the new node to join the Pastry network, it calculates its ID, \( X \), usually according to a hash function, and then asks \( A \) to route a message that has its ID as the key. The message is routed to the node, \( Z \), with the numerically closest ID. All nodes that form the path from \( A \) to \( Z \) send their state tables to \( X \) as a response to the join request. The new node initializes its state according to these tables, and then informs any nodes that need to be aware of its arrival. As \( A \) is a nearby node to \( X \) according to the proximity metric, node \( X \) creates its neighborhood set according to \( A \)'s set. In addition, as \( Z \) has the closest ID to \( X \), \( Z \)'s leaf set is used as a basis for the construction of \( X \)'s leaf set.

The procedure of creating the routing table has as follows. In the general case, the IDs of \( X \) and \( A \) share no common prefix. Let \( A_i \) denote the \( i \)th row of \( A \)'s routing table, keeping in mind that row zero of a routing table refers to a node that its ID shares zero digits with the current node’s ID. Thus \( X_0 \) could be initialized with \( A_0 \). Appropriate values for \( X_1 \) can be obtained from \( B_1 \) where \( B \) is the first node in the path from \( A \) to \( Z \). Since the node ids of \( X \) and \( B \) share the same first digit. Similarly, \( X \) obtains the values for row \( X_2 \) from \( C_2 \) of node \( C \), which is the second node in the path from \( A \) to \( Z \), and so on. After the initialization of its state, node \( X \) sends its state to the nodes in its neighborhood set, leaf set and routing table, so that they may update their state. The total cost of a join in terms of messages exchanged is \( O(\log_2 b N) \).

To handle node failures, neighboring nodes in the node ID space, which are in each other’s leaf set, periodically exchange keep-alive messages. If a node is unresponsive for a period \( T \), then it is presumed failed, or departed without warning. In case a node notices that a neighbor in its leaf set has failed, it
contacts the live node with the largest index on the side of the failed node and asks for its leaf set. Since the leaf sets of nodes with adjacent node IDs overlap, the update is trivial. The obtained leaf set contains nodes with nearby IDs not presently in the leaf set of the local node. Among these nodes, the appropriate one is chosen to replace the failed node.

In case a node detects that a node in its routing table has failed, the routing procedure forwards the message to another node. However, the failed entry must be replaced. Let us suppose that the failed entry is $R^{dl}$ where $l$ refers to the row and $d$ to the entry in the row $l$. The local node contacts a node in the same row that contains the failed entry and asks for its entry for $R^{dl}$. If no such entry exists, then the local node contacts a live node in the next row and asks its entry for $R^{dl}$. This procedure will eventually find an appropriate node to replace the failed entry.

Pastry seeks to exploit proximity in the underlying Internet by routing through the shortest path possible. Pastry's notion of network proximity is based on a scalar proximity metric, such as the number of IP routing hops or geographic distance among a given pair of Pastry nodes. Pastry assumes that the application provides a function that allows a Pastry node to determine the distance of the node with a given IP address to itself according to the proximity metric. The choice of the proximity metric depends on the desired qualities of the resulting overlay (low delay, high bandwidth, low network utilization). Proximity metrics could be the round-trip time (minimum of a series of pings), bandwidth (using packet pair techniques), the number of IP routing hops (measured using traceroute) or some combination of the above.

Pastry generally aims to minimize the distance between a node and the nodes in its routing table. This is computationally expensive in a large system and requires $O(N)$ communication. However, Pastry can ensure that the routing table entries are close but not the closest in $O(\log_2 N)$. More specifically Pastry ensures the following:

**Proximity invariant:** Each entry in a node $X$'s routing table refers to a node that is near $X$, according to the proximity metric, among all live Pastry nodes with the appropriate ID prefix.
Based on this proximity invariant, three locality properties are derived:

- **Total distance traveled:** This property, also known as the short routes property, has to do with the total distance messages travel along Pastry routes in terms of the proximity metric. Pastry ensures that in each routing step, the message is sent to the nearest node with the longest prefix match. Simulations show that the average distance traveled by a message is between 1.59 and 2.2 times the corresponding underlying distance between the source and the destination.

- **Local Route Convergence:** Assuming that two nearby nodes sending two messages with the same key, this property refers to the distance traveled by the messages before their paths merge. More precisely, this property claims that the distance traveled is equal to the distance between the two source nodes.

- **Locating the nearest replica:** This property says that if replicas of an object are stored on $k$ nodes with adjacent IDs, Pastry messages requesting the object first reach the node that is closer to the client.
3 Publish-subscribe and multicast

3.1 Publish-subscribe
The publish-subscribe paradigm is an important candidate for asynchronous communication between entities in a distributed system. It is intended for large-scale, Internet-based distributed systems. Publish-subscribe is very similar to group communication: subscribing is equivalent to becoming a member of a group, while publishing is equivalent to sending to a group. A publish-subscribe system consists of two main entities, the publishers and the subscribers. Entities that are interested in certain event conditions form a group for that pattern of events by subscribing to the group. These are the subscribers. A publisher for a group is any entity that notifies (-publishes) subscribers of events that match their registered interests. Such timely notification of customized information is of great importance for many distributed applications such as consumer event notification systems, mobile alerting systems, etc.

Publish-subscribe systems can be categorized into three types based on the expressiveness of the subscriptions they support [2]. In topic-based and subject-based schemes, events are classified and labeled by the publisher to belong to predefined set of subjects. Each subject is assigned to a multicast group, leveraging group-based multicast techniques for event delivery. In content-based pub-sub schemes, subscribers can choose filtering criteria along multiple dimensions, using thresholds and conditions on the contents of the message, rather than being restricted to a predefined set of subjects as in topic-based and subject-based schemes. Filtering imposes difficulties because different events may satisfy a wide variety of groups which are differentiated by the filtering conditions. This imposes challenges for efficient matching of events to subscriptions as well as for efficient event delivery. As a result, topic-based systems are considered more suitable for Internet-scale applications.

3.2 IP Multicast
IP Multicast provides the point-to-multipoint delivery of IP datagrams which is a necessary requirement for group communication applications on the
Internet. The original IP Multicast service model was designed without assessing commercial requirements and has a complex architectural design. As a consequence, IP Multicast experienced little deployment from ISPs and carriers in the past few years, and researchers aimed to design more scalable approaches for group communication that can fulfill market requirements.

More specifically, the IP Multicast service model cannot fulfill some crucial commercial requirements which are:

- Group management, including group creation authorization, receiver authorization and sender authorization.
- Distributed multicast address allocation.
- Security, including protection against attacks on multicast routes and sessions, and support for data integrity mechanisms.
- Support for network management.

The reason for these is that IP Multicast offers an open group service model. There is no restriction for hosts and users to create groups, join groups or send/receive data from groups. In other words, there is no access control. Multicast datagrams, like IP datagrams, are best-effort and reliable. Each multicast group has a Class D multicast address. In order to join multicast groups and receive data from these groups, hosts must contact their routers using the Internet Group Management Protocol (IGMP). However, anyone can send to a group without becoming a receiver (non-member senders) and senders cannot reserve addresses or prevent other senders from using the same address. The number of receivers is dynamic and the state of each entity (sender, receiver or both) is unknown. In other words, there is no real IP multicast group management.

The connections between the routers that form the multicast spanning tree are maintained by multicast routing protocols. There have been several multicast routing protocols that are grouped in four categories [14]:

- Protocols that build multicast spanning trees composed of shortest paths from the sources or to the sources. Examples of this category are the Distance Vector Multicast Routing Protocol (DVMRP), Multicast...
Open Shortest Path First (MOSPF) and Protocol Independent Multicast Dense Mode (PIM-DM).

- Protocols that build multicast spanning trees consisting of shortest paths from a known central core, also called a rendezvous point (RP), with all sources in the session sharing the same spanning tree. Examples are PIM Sparse Mode (PIM-SM), Core-Based Trees (CBT), and Ordered CBT (OCBT).

- Protocols that build bi-directional shared trees in which packets from each source are disseminated along the tree starting from any point. Examples are CBT, OCBT and the Border Gateway Multicast Protocol (BGMP).

- Protocols that build unidirectional shared trees where packets are sent first to the core, which then sends packets down the multicast spanning tree to all the recipients of the session. PIM-SM is an example.

### 3.3 Application-level Multicast

Application-level Multicast (ALM) schemes provide group communication at the application layer by using the unicast transport facilities offered by Transport Control Protocol (TCP) and User Datagram Protocol (UDP). Group membership, addressing and multicast routing are now responsibilities of the end hosts, whereas routers were responsible for these operations in IP Multicast. Application specific intelligence can be added to support efficient multicast services, increased reliability and higher QoS. The greatest advantage is that the network itself is released from these responsibilities and only needs to provide the basic stateless, unicast, best-effort delivery. The decoupling of multicast from unicast routing offers easier deployment and flexibility. However, a number of challenges have to be addressed which are related to routing, efficiency, reliability and scalability.

One challenge of an ALM protocol is scalability, also a major drawback of IP Multicast, even though it was one of the main reasons for which multicast was initially conceived. An overlay multicast solution will be acceptable as long as it provides comparable or at least not much worse performance than IP Multicast. As underlying networks are highly dynamic, overlay multicast must
quickly adapt to network changes and react efficiently to these changes so as not to degrade performance. Robustness is also a great challenge due to multicast group dynamics and network topology changes created by node failures or churn behavior of the overlay nodes. Finally, overlay multicast must provide reliability and security, unlike IP Multicast that does not provide these features.

The main disadvantages of ALM schemes is that they impose higher delay and less efficient utilization of the available bandwidth compared to IP multicast. This is due to the fact that a packet in an overlay network may cross the same physical link several times before reaching the target recipient. Another factor that must be taken into account is that overlay nodes must exchange refresh messages with neighboring overlay peers in order to manage the overlay network. This imposes extra overhead control traffic in the overlay network. Considering however that IP multicast is not really available, these disadvantages must also be weighed against the fact that ALM approaches are easily available to any application that needs to use group communication.

### 3.4 Scribe

Scribe [3] is a large-scale event notification infrastructure for topic-based publish-subscribe applications. It is built on top of Pastry and leverages Pastry’s reliability, self-organization and locality properties. Scribe provides efficient application-level multicast and is scalable to a large number of subscribers, publishers and topics. Scribe uses Pastry to create a topic (group) and to build an efficient multicast tree for the dissemination of events to the topic’s subscribers.

A Scribe system is a network of Pastry nodes, where each node runs the Scribe application software. Any Scribe node can create a topic and other nodes can register their interest in this topic by becoming members of the topic. Any Scribe node with the appropriate credentials for the topic can then publish events related to the topic and Scribe will disseminates these events to the topic’s subscribers. Scribe uses best-effort dissemination of events and does not guarantee ordered delivery. However reliability can be implemented on top of Scribe. The Scribe software provides two methods, *forward* and *deliver*
which are invoked by Pastry whenever a message arrives, the former when the node is an intermediate stop towards the final destination, the latter when the node is the final destination itself. The possible message types in Scribe are SUBSCRIBE (to a topic), CREATE (a topic), UNSUBSCRIBE (from a topic) and PUBLISH (an event to a topic).

Each topic has a unique topic ID. The Scribe node with node ID numerically closest to the topic ID acts as the *rendezvous* (RV) point for that topic. The RV point forms the root of the multicast tree associated to the topic. To create a topic, a Scribe node asks Pastry to route a CREATE message to the RV point of the topic. The Scribe deliver method adds the topic to the set of topics that this node is aware of. The topic ID can be the hash value 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), which ensures a uniform distribution of topic IDs. Combining this with the uniform distribution of Pastry's node IDs, this ensures an even distribution of topics across nodes.

Scribe creates a multicast tree per topic rooted at the RV point of the group to disseminate events along the tree. The multicast tree is created using reverse path forwarding, since the shortest paths in the overlay towards the RV point are merged. Each node in the multicast tree is a forwarder for the group. A forwarder may or may not be a subscriber to the group. Each forwarder maintains a children table that contains an entry (IP address and node ID) for each of its children in the routing table.

When a Scribe node wants to subscribe to a topic, it asks Pastry to route a SUBSCRIBE message with key equal to the topic ID. Pastry routes the message to the RV point of the group. At each node along the route Pastry invokes the Scribe forward method, which checks the list of topics to see if the current node is a forwarder. If so, it adds the node to its children table for this topic and the procedure stops. Otherwise, it creates an entry for this topic, accepts the node as a child and routes a SUBSCRIBE message to the next node along the route towards the RV point, thus becoming a forwarder.

When a node wants to unsubscribe from a topic, the node marks locally the topic as no longer required. If there are no children in its children table for this topic, is sends an UNSUBSCRIBE through Pastry to its parent. The
message proceeds recursively up the multicast tree, until a node is reached that still has entries in its children table for this topic after removing the departing child.

The subscriber management mechanism is very efficient because it is distributed across the nodes in the multicast tree. Each subscription is managed locally: the RV point does not handle all the subscription requests. Pastry’s randomization properties ensure that the multicast tree is balanced and that the forwarding load is evenly distributed across the nodes on the tree. Thus, Scribe can scale to a large number of topics and subscribers per topic.

Scribe leverages the two locality properties of Pastry: the short paths traveled and the local route convergence. These properties ensure that the routes from the root to each subscriber are short with respect to the proximity metric. Moreover, subscribers of a topic that are close with respect to the proximity metric tend to be children of the same parent in the multicast tree. This reduces link stress because the parent receives a single copy of a PUBLISH message and forward copies to its children along short routes.

When a Scribe node wants to publish an event for a topic, if he is aware of the IP address of the topic’s rendezvous point, it sends directly the message to the RV point. Otherwise, it asks Pastry to route a PUBLISH message with key the topic ID of the topic to the RV point, asking the RV point to return its IP address to the publisher. As long as the RV point has received the PUBLISH message, it disseminates the event along the multicast tree. The reason why the PUBLISH message is sent first to the RV point and then disseminated along the tree, is to allow the RV point to perform access control.

The caching of the IP address of the RV point is an optimization in order to avoid repeated routing through Pastry. In case that the RV point fails, then the publisher can route the PUBLISH message through Pastry to find the new one. If the RV point has changed because a new node with node ID closer to the topic ID has arrived, and the publisher has sent the PUBLISH message to the IP address of the old RV point, then the old RV point forwards the message to the new one and asks the new RV point to forward its IP address to the publisher.
Scribe provides best-effort delivery of events but it provides a framework for publish/subscribe applications to implement stronger reliability guarantees such as reliable and ordered delivery of events. Scribe uses TCP to disseminate events between parents and their children and uses Pastry to repair the tree in case a forwarder fails. Failure of a node is detected through heart-beat messages. A child knows that its parent has failed if it has not received a heart-beat message from its parent. In that case it asks Pastry to route a SUBSCRIBE message with key the topic ID of the topic. Pastry routes the message to the new parent thus repairing the multicast tree. Heart-beat messages can be avoided when events are published frequently on a topic, because event messages can act as heart-beat messages. Children table entries must be refreshed periodically by an explicit message from the child stating its continued interest in the topic, else entries are discarded.

In order to recover from failure of the rendezvous-point, the state of the RV point, which identifies the topic-creator and the access list, can be replicated across the \( k \) closest nodes to the root node in the node ID space. It must be noted that the \( k \) closest nodes to the root are in the leaf set of the root node. If a root’s child detects its failure, it routes a SUBSCRIBE message through Pastry to the new root, thus repairing the multicast tree. This mechanism scales well because failure of nodes is detected by sending messages to a small number of nodes and recovery is accomplished locally. As a result, only a small number of nodes \( O(\log_2 N) \) is involved.
4 Evaluating multicast protocols

4.1 Evaluation framework

There are two main goals commonly associated with multicasting. The first is resource conservation due to link sharing: each message is transmitted only once over each link of the multicast distribution tree, regardless of the number of receivers served over this link. The second is the decoupling of each sender from the receivers: each sender can address a potentially huge number of receivers with a single message. In order for resources to be actually conserved however, a good multicast tree needs to be constructed. Optimal multicast routing is equivalent to finding the Steiner tree of a network: given an undirected graph G and a subset of nodes S of G, the Steiner tree is the minimal cost subgraph of G such that there is a path between each pair of nodes in S. The nodes that are not in S but participate in the connection of nodes in G are called Steiner nodes. Unfortunately, the Steiner tree problem is NP complete, therefore only approximations of it can be constructed within realistic time constraints, thus multicast routing is generally suboptimal.

In IP Multicast the multicast distribution trees are normally constructed by merging the underlying optimal unicast routes from the source to every receiver in the group; in some protocols it is the reverse optimal paths that are used, which are equivalent to the forward optimal paths only when links are symmetric. These trees, although often called optimal, do not actually minimize resource consumption: they are not Steiner trees. Considering that on the Internet only unicast routing is provided however, it is more realistic to compare alternative multicast schemes against IP Multicast trees consisting of shortest paths towards each receiver.

Multicast communication schemes can be characterized by multiple performance metrics, including the following [9]:

- **Link Stress**: the number of identical packets sent over a physical link.
- **Link Stretch**: the ratio of the delay between two nodes along the overlay topology to the delay between these nodes in the unicast path.
- **Resource Usage**: the sum of the delay-stress product over all participating links.
On the performance of Application-level Multicast Routing

- **Time to first packet**: the time required for a new member to start receiving data after it has joined a group.
- **Loss**: the average number of packet losses after an ungraceful failure of a single participating node.
- **Robustness**: is the ability to adapt to network changes.
- **Control Overhead**: the number of messages exchanged and bandwidth utilization required for topology maintenance.

For the simulations discussed and performed in this thesis, the metrics used focused on comparing multicast with unicast paths, as well as IP multicast with overlay multicast paths. Specifically, the metrics used in the simulations are defined as follows:

- **Delay Penalty**: If we assume that the distribution trees in IP Multicast consist of the shortest paths between the root of the multicast tree and each receiver, then the delay penalty expresses the increase due to the use of an overlay routing mechanism such as Scribe. Two metrics of delay penalty can be considered: RMD (Ratio Maximum Delay) measures the ratio between the maximum delay using Scribe and the maximum delay using IP Multicast, and RAD (Ratio Average Delay) measures the ratio between the average delay using Scribe and the average delay using IP Multicast.

- **Node Stress**: In order to evaluate the load in each node, we can measure the number of groups with non-empty children tables, and the number of entries in children tables in each node on the multicast tree. Ideally, a multicast scheme will evenly distribute forwarding load over all nodes such that each node is responsible for forwarding multicast messages for a small number of groups and only to a small number of nodes, thus scaling in terms of both group size and number of groups.

- **Link Stress**: Link stress here is defined as the number of packets sent over each link when a message is multicast to each group. It can be expressed by the total number of messages in all links, the mean number of messages per link and the maximum link stress. Ideally, this load will be evenly distributed across the links of a network.
4.2 Alternative Multicast Schemes

There are many options for creating multicast distribution trees on top of overlay routing schemes. The first choice is the overlay scheme to use; options include CAN [27], Chord [25], Pastry [1], and Tapestry [24]. These overlay schemes can be categorized in two rough types: overlays which use generalized hypercube routing, such as Pastry, Tapestry and Chord and overlays which use a numerical distance metric to route through a Cartesian hyperspace, such as CAN. In addition, there are two approaches to achieve application-level multicast on top of an overlay scheme, flooding, as in CAN-Multicast, and tree building, as in Scribe, Bayeux and Hermes.

In the flooding, or overlay per group, approach, a separate overlay network is created for each multicast group, leveraging the routing information maintained in order to broadcast messages within the overlay. A new node wishing to join a group, must first find the overlay associated to that group. In order to implement this lookup function in a scalable manner, a distributed name service is required. To implement this functionality, a separate global overlay network can be used to implement a distributed hash table. The way flooding is achieved depends on the underlying overlay network. In all cases, the basic idea is that each node must forward each messages to all its neighbors, keeping track of past messages in order to avoid forwarding duplicates. Several overlay specific optimizations can be exploited in order to reduce the number of duplicates.

In the tree building, or tree per group, approach, a single overlay network is used for all groups but a separate tree is constructed for each group, over which the multicast messages are disseminated. This form of application-level multicast leverages the object location and routing properties of the overlay network to create multicast distribution trees and use them for multicast message propagation, by automatically assigning each group to a specific node within the overlay that serves as its RV point.

4.3 Evaluation of Application-Level Multicast

The tree per group approach of Scribe, running on top of Pastry, has been experimentally evaluated in comparison with IP Multicast [3] in terms of
delay, node stress and links stress. The evaluation employed a custom discrete event simulator that only modeled the propagation delay over the physical links, but not queuing delays or packet losses. The network topology consisted of a 5050 routers generated by the Georgia Tech random graph generator using the transit-stub model. End hosts were randomly assigned to the routers according to a uniform distribution, with the Pastry and Scribe software executing only on the end hosts, not on the routers. IP Multicast was assumed to employ a tree formed by the merge of the shortest unicast paths from the source node to each destination node. Both IP Multicast and Scribe were evaluated with groups of different sizes, generated according to a Zipfian distribution. Experiments were also executed with a large number of groups of size 11, which is typical of Instant Messaging. Up to 1500 groups and 100,000 nodes at maximum were simulated. The results showed that Scribe scales well because it distributes children tables and children table entries evenly across the nodes. However the results also indicate that Scribe multicast trees are not so efficient for small groups because they are formed by long paths without branching, thus inducing high link stress compared to IP Multicast.

The flooding versus tree-based multicast approaches have also been evaluated, over both Pastry and CAN overlay networks, using the same custom discrete event simulator [4] and similar topologies generated by the Georgia Tech random graph generator. Again only end nodes executed the overlay and multicast code, while IP Multicast was again assumed to employ shortest path trees. The metrics used were delay compared to IP Multicast, both in terms of maximum and average delays (RMD and RAD), link stress in terms of the number of packets sent over each link and node stress, in terms of routing entries per node and received messages when members join the tree and when multicast messages are disseminated.

The results showed that per-group multicast trees have several advantages over flooding. In CAN, the use of per-group multicast trees instead of flooding results in more efficient network usage. More precisely, relative delay penalties are better by factors of 2-3. In Pastry, per-group overlays impose significant overhead for their creation, since during the join phase, a new node needs to discover other nodes in the network and create its routing table and
the leaf sets. These actions may involve finding out about and potentially contacting dozens of other nodes. On the other hand, in the per-group multicast trees case, where trees are built using a single overlay network, routing state already established in the existing overlay network can be used during the join phase. This reduces the overhead imposed by the join process. The only case where flooding can be more advantageous than per-group multicast trees is when traffic for the group is carried only by group members.

Comparing the two types of overlay network in the per-group multicast tree approach, the relative delay penalty (RDP) values obtained for Pastry are 20% to 50% better than the ones obtained for CAN. Average link stress was also 15% lower for Pastry, whereas the maximum link stress for CAN was 25% lower. Similarly, the maximum number of forwarding entries that had to be maintained for CAN was about one third that for Pastry, whereas the average was similar. However, using the bottleneck remover algorithm, where an overloaded node redistributes its children tables to its children, Pastry can reduce the maximum link stress and the maximum number of forwarding table entries to values as low as CAN’s, while maintaining its low delay penalty compared to CAN, meaning that Pastry generally outperforms CAN.
5 Simulation Framework

This chapter presents the simulation framework used in this thesis for the evaluation of application-level multicast routing, that is, OMNeT++ along with OverSim. The main reasons are ease of use, a highly flexible and modular architecture and an open-source code base. OMNeT++ and OverSim provide a hierarchical architecture. At the lowest level of the hierarchy there are the simpler modules that implement the desired behavior, implemented as C++ classes. The compound modules, higher in the hierarchy, are more complex and are composed of simpler modules and maybe other compound modules. Communication between the modules is simulated by the exchange of messages. The action of sending or receiving a message is considered a single event. Initialization parameters for these modules are defined in human readable configuration files. Simulation runs can be differentiated by changing the values for these parameters. OMNeT++ and OverSim are described in more detail in the following sections.

5.1 OMNeT++

OMNeT++ [7] is an object-oriented modular discrete event network simulator consisting of hierarchically nested modules. Modules communicate through message-passing. Messages are sent either directly to a destination or along a predefined path, through gates and connections. Modules can have their own parameters. Parameters can be used to customize module behavior and to parameterize the model's topology. Modules at the lowest level of the hierarchy encapsulate behavior. These modules are termed simple modules, and they are programmed in C++ using the simulation library. Modules that contain submodules are termed compound modules. The top level module is the system module which consists of submodules, which can also contain submodules themselves. The depth of module nesting is not limited, allowing the user to reflect the logical structure of the actual system.

The model’s structure is described in OMNeT++’s NED language. The NEtwork Description (NED) language facilitates the modular description of a
network, which consists of a number of component descriptions (channels, simple/compound module types). Each module is an instance of a module type which describes the module and serves as a component for more complex module types.

Connectivity between modules is established through input and output gates within a single level of the module hierarchy. Connections can be assigned parameters in order to facilitate the modelling of communication networks. More specifically, connections are optionally described by three parameters: propagation delay, bit error rate and data rate. The user can specify other parameters as well as define custom link types. Data rate is used to model transmission delay and subsequently queuing delay. More specifically, given the data rate and the length of the connection, the transmission time for a message can be calculated. The message reserves the out gate for a period equal to its transmission time. In case other messages are sent to the same out gate, their transmission will be delayed as if the gate had an internal queue for the messages waiting to be transmitted. The Omnet++ class library provides the appropriate methods to implement this. Moreover, packet loss can be easily implemented in Omnet++. Code could be added in the destination model to throw away the received message according to a probabilistic distribution function, as if it had never been received.

OMNeT++ provides the user with human readable configuration files to initialize parameters for modules. In addition, OMNeT++ provides various statistic collections and visualization tools for results analysis. Moreover, the simulator supports parallel and distributed simulation with the multiple instances communicating via MPI, as well as support for network emulation via interfaces with real networks and the ability to use real networking code inside the simulator.

5.2 OverSim

OverSim [8] is an open-source overlay network simulation framework based on OMNeT++. It supports several models for structured (e.g. Chord, Kademlia, Pastry) and unstructured (e.g. GIA) p2p protocols. It was designed to support some special properties that existing simulators do not support,
such as heterogeneity of access networks and the high mobility rate of user terminals. More specifically, OverSim satisfies the following requirements:

- **Scalability**: ability to run simulations with a large number of nodes in a reasonable amount of time
- **Flexibility**: Ability to simulate both structured and unstructured networks, to specify simulation parameters in a human readable configuration file, and to support node mobility, node failure and malicious behavior of nodes.
- **Underlying Network Modeling**: provides underlay topologies that can be configured to have realistic bandwidths, packet delays and packet losses, as well as alternative network models to simulate large scale networks.
- **Reuse of simulation code**: Ability to handle and assemble real network packets and to communicate with other implementations of the same overlay protocol in order to enable researchers to validate simulation results by comparing them to the results from real-world test networks like PlanetLab.
- **Statistics**: provides statistical information such as sent, received and forwarded data per node, packet delivery status and packet hop count.
- **Documentation**: Provides a user manual and an API reference for using and extending the simulator with other overlay protocols.
- **Interactive Visualizer**: provides a GUI in order to visualize both the underlay and overlay topology in a customizable way.

OverSim has a modular architecture. An overview of this architecture is illustrated in figure 4. The basic features of OMNeT++ that were used for the design of OverSim are its modularity, the message generator and visualization features. In the following, we describe the modules of this architecture.
OverSim provides three underlying network models: Simple, SingleHost and INET. In the Simple underlay, there is a global routing table used to send messages directly from one network node to another. Messages are delayed either by a constant period of time or according to the node’s distance. Each node is placed in an Euclidean space and is assigned to a logical access network which is characterized by bandwidth, access delay and packet loss, in order to simulate heterogeneous access networks. This model is the most scalable one. The SingleHost model was developed to support the reuse of overlay protocol implementations without code modifications in real networks, like PlanetLab. Each OverSim instance emulates a single host that can communicate with other OverSim instances over existing networks like the Internet.

The INET model, also referred as Ipv4Underlay, is derived from the INET framework of OMNeT++ which supports simulation models of all network layers. INET model is ideal for the construction of complete backbone structures. Code for hash table based routing and interface caches was added to the INET Framework for faster forwarding of data packets, so as to support large scale networks. INET provides access routers, backbone routers. It also provides a module TunOutRouter that extends the INET model with similar
On the performance of Application-level Multicast Routing

techniques as in the SingleHost model, in order for a router to act as a gateway to the real network. All underlying network models have a consistent UDP/IP interface to the overlay protocols. Hence, overlay protocols can run on top of each underlay model.

OverSim supports structured as well as unstructured p2p overlay protocols. The major capability of the OverSim Overlay module however is that it facilitates the implementation of new overlay protocols by implementing and integrating in the simulator some common functionalities of overlay protocols. The common functionalities of overlay protocols are:

- **Overlay message handling** (RPC and statistical data): it deals with timeouts and packet retransmissions in case of packet loss and with the collection of message related statistical data.

- **Generic lookup function**: it provides a generic recursive and iterative lookup interface. Overlay protocols have to implement only a method to query the local routing table for the closest node in the overlay topology. The lookup function also supports the simulation of malicious node behavior.

- **Support for the visualization of the overlay topology**: the GUI of OMNeT++ eases the debugging of overlay protocols by showing transferred messages in detail. It also allows altering internal states during run time, like the overlay routing table. Additionally, OverSim gives developers the ability to draw arrows between overlay nodes in order to visualize both overlay and underlay topologies and to be able to demonstrate the effects of topology adaptation mechanisms.

- **Bootstrapping support**: OverSim provides A GlobalObserver model that mainly acts as a BootstrapOracle which provides a new node that wants to join the overlay with the address of an already existing node in the Overlay. It can be additionally used for collection of global statistics as well as global parameters usage and user-defined functions.

These common functions save developers from rewriting error-prone code and also make overlay protocols more comparable. For instance, overlay protocols show similar behavior if lookup messages have to be retransmitted due to packet loss. For the communication of overlay protocols and
applications there exists a common API. A new overlay protocol must provide at least a *key-based routing* (KBR) interface to the application.

OverSim supports a variety of applications that rely on key-based routing. There is a KBRTes application that sends periodically test messages to random overlay keys or nodeIDs and records message delay and hop count. The latest version of OverSim also includes an implementation of the application-level multicast protocol Scribe, as well as other application-level functionalities.
6 Implementation

This chapter presents the implementation details of our simulation framework. The first section describes the underlay and application classes and message types of OverSim used in our simulations, while the second section describes the code modifications and extensions we made.

6.1 OverSim classes and message types

OverSim provides the ‘Common’ package of classes which implement the common functionality of overlay protocols. In the following, we describe the implementation details of the classes of the Common package that are closely related to our implementation. The ‘OverlayKey’ class provides a common overlay key class that represents topic identifiers and node identifiers with base $2^{16}$. The ‘GlobalObserver’ module is responsible for the bootstrapping procedure and for maintaining global state (statistics, parameters). It contains simpler modules that implement the desired behavior and are implemented as C++ classes. The classes that we are interested in are the ‘BootstrapOracle’ and ‘GlobalParameters’ classes. The following diagram shows the usage relationships between modules:

![Diagram of Global Observer class](image)

**Figure 5: The Global Observer class.**

The BootstrapOracle maintains a structure called PeerHashMap which maintains information for peers that participate in the overlay. A peer can be any instance of an overlay host. Each peer is associated to a bootstrapEntry structure which contains instances of TransportAddress and PeerInfo classes. The TransportAddress class implements a common transport address. It covers the complete node information, like IP-address and port. The PeerInfo class provides additional underlay specific information associated with a
certain transport address. We are interested in the `moduleId` attribute of `PeerInfo` class which refers to the identifier of the underlay module that corresponds to the related peer. There is a `BootstrapOracleAccess` header file which gives access to the `BootstrapOracle` module.

`GlobalParameters` is responsible for storing global simulation parameters. There is a `GlobalParametersAccess` header file which gives access to the `GlobalParameters` module.

In addition, the Common package provides a ‘`CommonMessages`’ class that implements the common message types for overlay protocols. The `ALMMessage` type is the base message for ALM communication. Several other message types are derived from this base message type. The inheritance hierarchy is illustrated in the following figure; the table below records the functionality of each message type.

![Figure 6: The ALM Message class.](image)

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ALMCreateMessage</code></td>
<td>Create a multicast group</td>
</tr>
<tr>
<td><code>ALMDeleteMessage</code></td>
<td>Delete a multicast group</td>
</tr>
<tr>
<td><code>ALMSendMessage</code></td>
<td>Subscribe to a multicast group</td>
</tr>
<tr>
<td><code>ALMLeaveMessage</code></td>
<td>Leave a multicast group</td>
</tr>
<tr>
<td><code>ALMMulticastMessage</code></td>
<td>Send a multicast message to all group members</td>
</tr>
<tr>
<td><code>ALMAnycastMessage</code></td>
<td>Send a message to a (random) member in the group</td>
</tr>
</tbody>
</table>

| Table 1: ALM Message Types |

The `Ipv4Network` of OverSim is a network that connects backbone and access
networks. The *Ipv4Underlay* module is comprised of other modules (compound or simple) as illustrated by the following diagram:

![Diagram](image)

**Figure 7: The IPv4 Underlay class.**

The functionality of each module is recorded in the table below:

<table>
<thead>
<tr>
<th>Underlay Module Type</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>AccessRouter</em></td>
<td>Connects the access nets to the network backbone</td>
</tr>
<tr>
<td><em>GlobalObserver</em></td>
<td>Contains the bootstrapOracle and the globalStatistics module</td>
</tr>
<tr>
<td><em>Ipv4UnderlayConfigurator</em></td>
<td>Configurator of the Ipv4Underlay</td>
</tr>
<tr>
<td><em>OverlayAccessRouter</em></td>
<td>Access router that participates in the overlay</td>
</tr>
<tr>
<td><em>OverlayRouter</em></td>
<td>Router in the overlay network</td>
</tr>
<tr>
<td><em>TunOutRouter</em></td>
<td>Acts as a gateway to the real network</td>
</tr>
</tbody>
</table>

**Table 2: Underlay Module Types**

The *Ipv4UnderlayConfigurator* is responsible for setting up the Ipv4Network. It periodically adds overlay nodes to the network in the initialization phase, with the interval set by the *initialMobilityDelay* parameter, adds/removes/migrates overlay nodes after init phase periodically with the interval set by *targetMobilityDelay* parameter with the probabilities defined by the parameters *creationProbability*, *removalProbability*, and *migrationProbability*.

*PastryModules* is the module that implements the Pastry KBR overlay. It is comprised of simpler modules which implement the desired behavior as described in [1]. The following diagram shows usage relationships between
these modules.

![Pastry Modules class](image)

**Figure 8: The Pastry Modules class.**

Each overlay host runs an instance of the Pastry class. The following table describes the usage of each module:

<table>
<thead>
<tr>
<th>Pastry Module Type</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pastry</td>
<td>The main module of the Pastry implementation</td>
</tr>
<tr>
<td>PastryRoutingTable</td>
<td>Contains the routing table of the Pastry implementation</td>
</tr>
<tr>
<td>PastryLeafSet</td>
<td>Contains the leafset of the Pastry implementation</td>
</tr>
<tr>
<td>PastryNeighborhodSet</td>
<td>Contains the neighbourhood set of the Pastry implementation</td>
</tr>
</tbody>
</table>

Table 3: Pastry Module Types

*MulticastScribe* is the compound module that implements the application-level multicast protocol Scribe. It contains the simple module *Scribe* which is implemented as a C++ class and provides all the necessary functionality, message forwarding and delivery methods, handles join/leave requests, handles multicast messages, etc.

![MulticastScribe class](image)

**Figure 9: The Multicast Scribe class.**
Scribe package provides two additional classes: \emph{ScribeMessages} and \emph{ScribeGroup}. \emph{ScribeMessages} class provides Scribe-protocol-specific message types to facilitate the implementation of Scribe. The table below records the Scribe message types:

<table>
<thead>
<tr>
<th>Scribe Message Type</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>\emph{ScribeDataMessage}</td>
<td>To send data to multicast group</td>
</tr>
<tr>
<td>\emph{ScribeJoinCall}</td>
<td>To send a join request</td>
</tr>
<tr>
<td>\emph{ScribeJoinResponse}</td>
<td>To send a response to a join call</td>
</tr>
<tr>
<td>\emph{ScribeLeaveMessage}</td>
<td>To leave a multicast group</td>
</tr>
<tr>
<td>\emph{ScribePublishCall}</td>
<td>To publish an event to a multicast group</td>
</tr>
<tr>
<td>\emph{ScribePublishResponse}</td>
<td>To send a response to a publish call</td>
</tr>
<tr>
<td>\emph{ScribeSubscriptionRefreshMessage}</td>
<td>To state a continued interest in a topic</td>
</tr>
<tr>
<td>\emph{ScribeTimer}</td>
<td>To send timer events</td>
</tr>
</tbody>
</table>

\textbf{Table 4: Scribe Message Types}

The \emph{ScribeGroup} class encapsulates the information of a Scribe multicast group relevant to a member node. More precisely, each instance of the \emph{ScribeGroup} class maintains information about the \emph{OverlayKey} identifier of the group it refers to, the rendezvous point of the group, the parent of the node that owns this \emph{ScribeGroup} instance, a list of its children for this group and whether this node is a forwarder and/or a subscriber for the group. In other words, the \emph{ScribeGroup} class facilitates the implementation of the Scribe principle that membership information is distributed over Scribe nodes, with each Scribe node being responsible for forwarding and duplicating packets for a part of the group and join/leave messages being handled locally. There is no global knowledge of the membership information, only local.

Each node that implements Scribe runs the \emph{Scribe} class. Each Scribe node maintains a map structure named ‘\emph{GroupList}’ that maintains local information about all groups this node is responsible for. More specifically, it maps the \emph{OverlayKey} identifier of each group to a \emph{ScribeGroup} instance which contains local information for this group relative to this node.
6.2 Code Modifications and Extensions

In the GlobalParameters module, we have added global parameters to control the application level multicast functionality. These parameters are targetMulticastGroupNum, targetJoinNum, which define the target number of multicast groups to be created and the target number of joins for each multicast group, and currentMulticastgroupNum, currentJoinNum which keep the current value of the above parameters. We have additionally implemented setter/getter methods to access/modify these parameters.

We have also extended the CommonMessages to include one more application level message type, named ALMCompleteMessage. ALMCompleteMessage is used to indicate that the target number of multicast groups has been created and the target number of joins per group has been completed.

![ALMMessage](image)

**Figure 10:** The modified ALM Message class.

As the OverSim implementation of Scribe provides only local membership information, we implemented a map structure named InfoList which maintains global information for all multicast groups created. InfoList maps each OverlayKey associated to a group to a structure named ConnectList that shows the connections between children and their parent in the multicast tree related to the group and whether a node is a recipient for the group. The structure of InfoList is illustrated in the figure below:
As shown in the figure, each entry in the ConnectList refers to a child of the group which is referred to by its IP address. The child is related with the IP address of its parent in the multicast tree; it is also noted whether this child is forwarder or/and subscriber for the group. InfoList is refreshed each time a join request is handled. The control flow for a join request is illustrated below:

When an overlay host wants to join a Scribe multicast group it sends an ALMSSubscribeMessage to the lower tier which contains the OverlayKey.
identifier of the group. The control then flows to the `handleUpperMessage` method of Scribe class which then invokes the `subscribeToGroup` method. This method updates the `GroupList` and `InfoList` structures with local and global information relative to this join request. The pseudo code of `subscribeToGroup` that updates the InfoList is recorded in the following table:

```
IF entry for group with groupId does not exist in InfoList
  Create entry for groupId in InfoList
ENDIF
IF entry for this node does not exist in ConnectList of groupId entry
  Create entry for this node
ENDIF
IF this node is not already a subscriber for group with groupId
  Increment join counter for group with groupId
  Make this node a subscriber for group with groupId
ENDIF
IF this node is not already a forwarder for group with groupId
  Set its parent to UNSPECIFIED for group with groupId
  Send ScribeJoinCall to find a parent for group with groupId
ELSE if this node is already a forwarder for group with groupId
  This node has already a parent for group with groupId
ENDIF
```

If this node is not already a forwarder for this group, a `ScribeJoinCallMessage` is sent to find a parent for this group. When a Scribe node receives a join call, either the `HandleJoinCall` method or the `forward` method is invoked by Pastry to handle the join call. Both methods then invoke `handleJoinMessage` to handle the join request.

![Diagram](image)

**Figure 13: Control Flow for a join request part 2**
The pseudo code of *handleJoinMessage* that updates the *InfoList* is shown in the following table. The source node of the join call could be either a subscriber that routed the join call to this node through Pastry or could be a forwarder of the join call of some Scribe node until the message reaches a node that is already a member of the group.

```plaintext
IF entry for this node does not exist in ConnectList of the group join call refers to
    Create new entry for this node and set recipient status to false because this
    node will become a forwarder for the source node of join call
ENDIF
IF group is new or no parent is known
    Set parent of this node to UNSPECIFIED
    Send join to parent
ELSE group is not new
    This node has already a parent for the group join call refers to
ENDIF
Update the entry for the source node of join call by setting its parent to be this node
```

When the target number of multicast groups have been created and the target number of joins per group has been completed, an *ALMCompleteMessage* is sent from upper tier to lower tier. As long as all join call messages have been handled and there is no unspecified parent for a child in InfoList, the evaluation can start. The *measureNodeStress* method is invoked for each Scribe node that is of underlay module type *OverlayHost* to measure the stress imposed on this node. We are interested only in the stress imposed on end hosts. The pseudo code of *measureNodeStress* is shown below:

```plaintext
FOR all groups in GroupList of this node
    IF the children table of the group is non-empty
        Increment children table counter by one
        Increment children table entries counter by the number of children of
        this node for this group
    ENDIF
ENDFOR
Collect children table counter in a statistic collection
Collect the children table entries counter in a statistic collection
Take samples, min/max/mean value, standard deviation and variance of statistic collections
```

The children table counter maintains the number of non-empty children.
tables for which this node is responsible. Children table entries counter maintains the total number of entries in all children tables for which this node is responsible.

We additionally invoke the method `measurePathRatio` which measures the overlay versus direct path ratio of the paths from all recipients to the rendezvous-point for all multicast groups in `InfoList`. The overlay versus direct path ratio is the ratio between the number of physical hops induced by the overlay routing process of Scribe and the number of physical hops in the direct physical path. In order to count the physical hops of a path, for each multicast group in `InfoList`, a subgraph of our initial network graph is created which consists of all routers (access, backbone, overlay, if any exist) and only the overlay host modules that are members of the multicast group.

To implement this functionality, we use the `cTopology` C++ class of OMNeT++ framework. This class gives the possibility to create a sub graph of the original network graph which will include only selected modules. The modules that will be included are selected either by module type, or by a parameter’s presence and its value or with a user-supplied Boolean function.

The `extractFromNetwork` function of the `cTopology` class creates a topology of selected modules according to a user-supplied Boolean function. Our Boolean function is named `selectMulticastMembers` and selects all the underlay router modules (‘AccessRouter’, ‘Router’, ‘OverlayAccessRouter’, ‘OverlayRouter’) that participate in the initial network graph as well as the OverlayHost modules that are members of the multicast group.

An OverlayHost module “is” (refers to) a member of a multicast group if there exists a member for this group in `InfoList` that has the same IP address as the one associated to the module identifier of the module. The IP address of a module is obtained via the method `getPeerAddressById` of the `BootstrapOracle` module which we have implemented. `getPeerAddressById` returns the IP address associated with the module identifier passed as argument. The pseudo code is shown in the table below:
Moreover, the cTopology class can calculate shortest paths between nodes in the topology to support optimal routing. More specifically, it provides a computationally inexpensive algorithm (Dijkstra’s) to find shortest paths from all nodes to a target node. The algorithm stores the results in the cTopology object. In the simplest case, all edges are assumed to have the same weight. cTopology offers a method `distanceToTarget` that returns the distance related to a shortest path in terms of the weight assigned to the edges. In the unweighted case, it returns the number of hops to the target node.

When the topology for a multicast group has been created, we calculate the shortest paths between any pair of nodes in the topology by invoking the `unweightedSingleShortestPathsTo` method of the cTopology class. This method measures the distance between a target node and all other nodes of the topology with respect to the number of physical hops.

We maintain a map structure named `DistList` to store the results of step 2. `DistList` maps the IP address of a target node to another map structure `HopsList` which maintains the distance of all other nodes to the target node. More precisely, `HopsList` maps the IP address of a node to a distance, which is the distance of this node to the target node in terms of physical hops. `measurePathRatio` is concisely described by the following pseudo code:

```plaintext
FOR each group in InfoList
    Create a topology to represent the group(all routers, only end hosts that are members of the group)
    FOR each node of the topology
        Set the current node to be the target node
        Find shortest paths from all other nodes of the topology to the target node with respect to the number of physical hops
        Store the distance of each shortest path in the DistList
    ENDFOR
    Invoke overlay2directPathRatio method for this group
ENDFOR
```
The method *overlay2directPathRatio* calculates the path ratio between the number of physical hops induced by Scribe and the number of physical hops in the direct physical path. In order to compute the number of physical hops on an overlay path, we must sum up the physical hops in each intermediate overlay link from a recipient to the rendezvous point. As it was previously mentioned, *ConnectList* maintains information about the members of a group, whether they are forwarders or/and subscribers, and their connection to their parent. So, according to the *ConnectList*, we start from the recipient, take each intermediate overlay link, and recursively sum up the distance of the two involved nodes of each link with respect to the number of hops. Each distance is retrieved from the *DistList*. The recursive process stops when we reach the rendezvous point. The distance of the direct path is also retrieved from *DistList*. Below, the pseudo code for *overlay2directPathRatio* is presented. The recursive procedure for computing the overlay_path_hops is omitted for text clarity.

```plaintext
overlay2directPathRatio ( distList, connectList )
FOR each node in the connectList
    IF the node is a subscriber of the group
        overlay_path_hops = the number of physical hops induced by the overlay path from the subscriber to the rendezvous point of the multicast group
        direct_path_hops = the number of physical hops in the direct path from the subscriber to the rendezvous point
        ratio = overlay_path_hops / direct_path_hops
        Collect the value of ratio in a statistic collection
    ENDIF
ENDFOR
Collect the mean value of ratio observed in the group in a statistic collection
Take samples, min/max/mean value, standard deviation and variance of the last statistic collection
```
7 Evaluation

This section presents results of simulation experiments evaluating the performance of Scribe under different topology scenarios which are differentiated by the fact that routers may or may not participate in maintaining the overlay and Scribe. The experiments compare the performance of Scribe to IP Multicast in two ways: the overlay versus direct path ratio and the stress on each node.

7.1 Experimental Setup

OMNeT++ together with OverSim were used as the simulation platform to evaluate Scribe. The IPv4Network module of OverSim is used as the underlay layer because it allows the simulation of complete backbone structures. Pastry is the protocol used to construct the overlay and Scribe is the application-layer multicast protocol used for multicast routing. The simulations run on a network topology of 600 routers and 600 end hosts. End hosts and routers can run the code to maintain the overlays and implement Scribe. End hosts were randomly assigned to routers in the core with uniform probability.

The IPv4Network is hierarchical. There are 24 backbone routers and 576 access routers. We generated 10 different topologies using the same parameters but different random seeds. We ran all the experiments in all topologies. The results we present are the average of the results obtained with each topology. In order to compare the results for Scribe to IP Multicast, we used the shortest paths of the unicast routes from each recipient to the rendezvous point. Shortest paths are obtained via Dijkstra algorithm.

The number and size of groups varies: we create 6, 12 and 60 groups with 100, 50 and 11 joins/group, respectively. The minimum group size of eleven is typical of Instant Messaging applications, which is one of the targeted multicast applications. The members of each group were selected randomly with uniform probability from the set of Scribe nodes, using different random seeds for each topology. Distributions with better network locality could improve the performance of Scribe.
For each set of parameters, we run different scenarios which are differentiated by the percentage of overlay routers. We run five scenarios, where 0%, 10%, 25%, 50% and 100% of the routers participate in the overlay respectively. Parameters for each run are initialized in the configuration file. An example is illustrated in the following table:

```
[Run 40]
description = "Scribe Test(600 routers, 6 groups, 100 joins/group, no perc)"
network = IPv4Network
seed-0-ml = 1453732904

**.targetOverlayTerminalNum=600
**.globalParameters.targetJoinNum = 100
**.globalParameters.currentJoinNum = 0
**.globalParameters.targetMulticastGroupNum = 6
**.globalParameters.currentMulticastGroupNum = 0
**.globalParameters.terminalNum = 600
IPv4Network.backboneRouterNum=24
IPv4Network.overlayBackboneRouterNum=0
IPv4Network.accessRouterNum=576
IPv4Network.overlayAccessRouterNum=0
**.overlay.measureNetwInitPhase=true
**.overlay.collectPerHopDelay = true
**.overlayType = "PastryModules"
**.tier1Type = "MulticastScribe"
**.tier2Type="ScribeTestModule"
**.numTiers = 2
**.overlay.iterativeLookup=false
**.overlay.useCommonAPIforward=true
**.overlay.pastry.optimizeLookup=false
**.overlay.pastry.optimisticForward=false
**.overlay.pastry.avoidDuplicates=false
```

Table 5: Example of run file.

This table presents the parameters initialization for run 40. This run describes an IPv4Network of 600 routers, 24 backbone routers and 576 access routers, and 600 end hosts. There are no overlay routers in this run. The overlay type is PastryModules which indicates that all overlay hosts run Pastry. There are two tiers. Tier 1 is MulticastScribe which means overlay hosts implement Scribe and tier 2 is ScribeTestModule which is a test module of OverSim. The target number of multicast groups to be created is 6 and the target number of joins per group is 100. The seed for this run is 1453732904.
7.2 Path Ratio

The first set of experiments compares the number of physical hops on a path from a recipient to the rendezvous point of a multicast group using Scribe and IP Multicast. Scribe increases the number of physical hops induced by the overlay routing process relative to IP Multicast routing. To evaluate this penalty, we measured the overlay versus direct path ratio which is the ratio between the number of physical hops induced by the overlay routing process and the number of physical hops in the direct physical path. The average results for all runs are shown in below, with a confidence interval with alpha factor of 0.95:

<table>
<thead>
<tr>
<th>#overlay routers (%)</th>
<th>#groups</th>
<th>#runs</th>
<th>average max path ratio</th>
<th>average mean path ratio</th>
<th>confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>10</td>
<td>3,8098</td>
<td>3,153589</td>
<td>0,012071</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>10</td>
<td>3,1133</td>
<td>2,47009</td>
<td>0,00944</td>
</tr>
<tr>
<td>25</td>
<td>6</td>
<td>10</td>
<td>2,92225</td>
<td>2,579796</td>
<td>0,011888</td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td>10</td>
<td>2,307166</td>
<td>2,151667</td>
<td>0,00417</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td>10</td>
<td>2,613033</td>
<td>2,407114</td>
<td>0,00536</td>
</tr>
</tbody>
</table>

Table 6: Path Ratio for 6 groups

<table>
<thead>
<tr>
<th>#overlay routers (%)</th>
<th>#groups</th>
<th>#runs</th>
<th>average max path ratio</th>
<th>average mean path ratio</th>
<th>confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12</td>
<td>10</td>
<td>4,952592</td>
<td>3,62268778</td>
<td>0,017306</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>10</td>
<td>2,6896</td>
<td>2,095001</td>
<td>0,005966</td>
</tr>
<tr>
<td>25</td>
<td>12</td>
<td>10</td>
<td>3,821631</td>
<td>3,34820222</td>
<td>0,01067</td>
</tr>
<tr>
<td>50</td>
<td>12</td>
<td>10</td>
<td>1,947865</td>
<td>2,033283</td>
<td>0,001065</td>
</tr>
<tr>
<td>100</td>
<td>12</td>
<td>10</td>
<td>2,414466</td>
<td>2,295106</td>
<td>0,008005</td>
</tr>
</tbody>
</table>

Table 7: Path Ratio for 12 groups

<table>
<thead>
<tr>
<th>#overlay routers (%)</th>
<th>#groups</th>
<th>#runs</th>
<th>average max path ratio</th>
<th>average mean path ratio</th>
<th>confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>60</td>
<td>10</td>
<td>5,47979</td>
<td>2,69929667</td>
<td>0,022148</td>
</tr>
<tr>
<td>10</td>
<td>60</td>
<td>10</td>
<td>3,848317</td>
<td>2,27163333</td>
<td>0,015538</td>
</tr>
<tr>
<td>25</td>
<td>60</td>
<td>10</td>
<td>3,221381</td>
<td>2,38348</td>
<td>0,006388</td>
</tr>
<tr>
<td>50</td>
<td>60</td>
<td>10</td>
<td>2,748146</td>
<td>2,19508667</td>
<td>0,016199</td>
</tr>
<tr>
<td>100</td>
<td>60</td>
<td>10</td>
<td>2,785858</td>
<td>2,09619111</td>
<td>0,014564</td>
</tr>
</tbody>
</table>

Table 8: Path Ratio for 60 groups
Figure 14 shows the distribution of overlay versus direct path ratio. The y-value of a point represents the average mean path ratio observed in the target number of groups of each curve, relative to the percentage of routers that participate in the overlay and Scribe (x-value).

![Overlay vs Direct Path Ratio](image)

**Figure 14: Cumulative distribution of overlay vs. direct path ratio**

As shown in the figure, the overlay versus direct path ratio decreases with the participation of routers in the overlay. For 60 small groups, it reaches its lowest value when all routers participate in the overlay. It can be observed that for all cases (6, 12, 60 groups) there is an increase of the average path ratio for scenario 3 (25% overlay routers). In order to clarify why this happens, the following table presents the mean path ratio for all runs of scenario 3.

<table>
<thead>
<tr>
<th>run No</th>
<th>mean path ratio (6 groups)</th>
<th>mean path ratio (12 groups)</th>
<th>mean path ratio (60 groups)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4,20183</td>
<td>3,94939</td>
<td>2,47586</td>
</tr>
<tr>
<td>2</td>
<td>2,44656</td>
<td>3,18597</td>
<td>2,44662</td>
</tr>
<tr>
<td>3</td>
<td>1,80986</td>
<td>3,62094</td>
<td>2,39442</td>
</tr>
<tr>
<td>4</td>
<td>2,38608</td>
<td>4,83572</td>
<td>2,5299</td>
</tr>
<tr>
<td>5</td>
<td>2,63519</td>
<td>4,5075</td>
<td>3,48187</td>
</tr>
<tr>
<td>6</td>
<td>1,37258</td>
<td>2,42794</td>
<td>2,32558</td>
</tr>
<tr>
<td>7</td>
<td>4,76733</td>
<td>2,30514</td>
<td>1,44985</td>
</tr>
<tr>
<td>8</td>
<td>2,09464</td>
<td>3,77828</td>
<td>1,63409</td>
</tr>
<tr>
<td>9</td>
<td>2,06778</td>
<td>1,52294</td>
<td>2,71313</td>
</tr>
<tr>
<td>10</td>
<td>2,01611</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 9: Scenario 3: mean values for all runs and all cases**
The values noted in red show the maximum mean values observed for each case. For 6 groups, the mean path ratio is between 1.37 – 2.44 for most runs. However, the average mean path ratio is 2.57 which is much higher relative to single mean values. This is because the high red values have a lot of influence on the sum of mean values. Hence, the average value is higher than expected. The same holds for the cases of 12 and 60 groups. This also explains the slight increase of the average mean path ratio for scenario 5 (100% overlay routers) as can be observed from the following table which shows the mean values for scenario 5:

<table>
<thead>
<tr>
<th>run No</th>
<th>mean path ratio (6 groups)</th>
<th>mean path ratio (12 groups)</th>
<th>mean path ratio (60 groups)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.35875</td>
<td>2.81183</td>
<td>2.33439</td>
</tr>
<tr>
<td>2</td>
<td>1.97944</td>
<td>1.99569</td>
<td>2.10467</td>
</tr>
<tr>
<td>3</td>
<td>2.00472</td>
<td>2.38292</td>
<td>2.20189</td>
</tr>
<tr>
<td>4</td>
<td>1.76767</td>
<td>1.39153</td>
<td>1.99212</td>
</tr>
<tr>
<td>5</td>
<td>4.36908</td>
<td>2.34467</td>
<td>2.73644</td>
</tr>
<tr>
<td>6</td>
<td>2.92167</td>
<td>2.96025</td>
<td>1.80152</td>
</tr>
<tr>
<td>7</td>
<td>2.07975</td>
<td>4.08617</td>
<td>1.85182</td>
</tr>
<tr>
<td>8</td>
<td>1.77264</td>
<td>1.79514</td>
<td>1.81848</td>
</tr>
<tr>
<td>9</td>
<td>2.89542</td>
<td>1.83758</td>
<td>2.02439</td>
</tr>
<tr>
<td>10</td>
<td>2.922</td>
<td>1.34528</td>
<td></td>
</tr>
</tbody>
</table>

**Table 10: Scenario 5: mean values for all runs and all cases**

This table makes it clear that the mean path ratio for 6 groups is between 1.3-2 for most cases. Nevertheless, the high mean value 4.36 of run 5 (red pointed value) as well as the relatively high mean values for runs 6 and 9 result in a higher average mean path ratio. The same holds for 12 groups. For 60 groups, all runs present similar behavior, hence the result in the figure is not misquoted.

Therefore, it is clear that when routers run code to maintain the overlay and to implement Scribe, the path ratio decreases significantly. This can be explained because when adding routers in the overlay the overlay network density increases. This together with the locality properties of Pastry make it more possible for a node to route join messages to nodes that are closer with respect to the proximity metric, hence forming paths that are more closer to the
On the performance of Application-level Multicast Routing

7.3 Node Stress

In Scribe, end nodes are responsible for maintaining membership information and for forwarding and duplicating packets whereas routers perform these tasks in IP Multicast. The second set of experiments intends to figure out whether the participation of routers in the overlay and in the implementation of Scribe reduces node stress on each node. To evaluate the stress imposed by Scribe on each node, we measure the number of groups with non-empty children tables and the total number of entries in all children tables on each Scribe node. The following tables show the cumulative results for all runs:

<table>
<thead>
<tr>
<th>#overlay routers (%)</th>
<th>#groups</th>
<th>#runs</th>
<th>average max (#tables)</th>
<th>average mean (#tables)</th>
<th>confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>10</td>
<td>6</td>
<td>0.0795001</td>
<td>0.011896</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>10</td>
<td>6</td>
<td>0.0233333</td>
<td>0.008399</td>
</tr>
<tr>
<td>25</td>
<td>6</td>
<td>10</td>
<td>5.4</td>
<td>0.0168333</td>
<td>0.006518</td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td>10</td>
<td>4.2</td>
<td>0.0111667</td>
<td>0.004857</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td>10</td>
<td>3.6</td>
<td>0.0116667</td>
<td></td>
</tr>
</tbody>
</table>

Table 11: Number of children tables (6 groups)

<table>
<thead>
<tr>
<th>#overlay routers (%)</th>
<th>#groups</th>
<th>#runs</th>
<th>average max (#tables)</th>
<th>average mean (#tables)</th>
<th>confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12</td>
<td>10</td>
<td>12</td>
<td>0.13185178</td>
<td>0.024838</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>10</td>
<td>9.9</td>
<td>0.0355</td>
<td>0.016798</td>
</tr>
<tr>
<td>25</td>
<td>12</td>
<td>10</td>
<td>10.7</td>
<td>0.04462967</td>
<td>0.015097</td>
</tr>
<tr>
<td>50</td>
<td>12</td>
<td>10</td>
<td>6.7</td>
<td>0.0138333</td>
<td>0.013727</td>
</tr>
<tr>
<td>100</td>
<td>12</td>
<td>10</td>
<td>6.8</td>
<td>0.0165</td>
<td>0.010359</td>
</tr>
</tbody>
</table>

Table 12: Number of children tables (12 groups)

<table>
<thead>
<tr>
<th>#overlay routers (%)</th>
<th>#groups</th>
<th>#runs</th>
<th>average max (#tables)</th>
<th>average mean (#tables)</th>
<th>confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>60</td>
<td>10</td>
<td>60</td>
<td>0.39685189</td>
<td>0.099991</td>
</tr>
<tr>
<td>10</td>
<td>60</td>
<td>10</td>
<td>37.7</td>
<td>0.12444456</td>
<td>0.023652</td>
</tr>
<tr>
<td>25</td>
<td>60</td>
<td>10</td>
<td>25</td>
<td>0.073889</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>60</td>
<td>10</td>
<td>21</td>
<td>0.04240733</td>
<td>0.025987</td>
</tr>
<tr>
<td>100</td>
<td>60</td>
<td>10</td>
<td>24.6</td>
<td>0.05351856</td>
<td>0.01999</td>
</tr>
</tbody>
</table>

Table 13: Number of children tables (60 groups)
It must be pointed out that when routers do not participate in the overlay, in all cases (6, 12, 60 groups), the maximum number of children tables for all runs is equal to the total number of groups. In other words, there are Scribe nodes which are responsible for all groups. The participation of routers in the overlay reduces this maximum value. Indeed, in scenarios 4 and 5 (50% and 100% overlay routers), the maximum number of children tables reduces for all runs, with less than half of the runs having a maximum number of children tables equal to the total number of groups.

The following numbers consider the number of children table entries:

<table>
<thead>
<tr>
<th>#overlay routers (%)</th>
<th>#groups</th>
<th>#runs</th>
<th>average max (#entries)</th>
<th>average mean (#entries)</th>
<th>confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>10</td>
<td>321.1</td>
<td>1,079167</td>
<td>0,3687347</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>10</td>
<td>39.7</td>
<td>0,0891668</td>
<td>0,0354642</td>
</tr>
<tr>
<td>25</td>
<td>6</td>
<td>10</td>
<td>36</td>
<td>0,0771666</td>
<td>0,0587473</td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td>10</td>
<td>12.7</td>
<td>0,0253333</td>
<td>0,0275241</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td>10</td>
<td>14.2</td>
<td>0,033</td>
<td></td>
</tr>
</tbody>
</table>

Table 14: Total Number of Children Table Entries (6 groups).

<table>
<thead>
<tr>
<th>#overlay routers (%)</th>
<th>#groups</th>
<th>#runs</th>
<th>average max (#entries)</th>
<th>average mean (#entries)</th>
<th>confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12</td>
<td>10</td>
<td>302</td>
<td>1,12036889</td>
<td>0,380102</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>10</td>
<td>55.2</td>
<td>0,1196667</td>
<td>0,016798</td>
</tr>
<tr>
<td>25</td>
<td>12</td>
<td>10</td>
<td>50.5</td>
<td>0,11351851</td>
<td>0,10552</td>
</tr>
<tr>
<td>50</td>
<td>12</td>
<td>10</td>
<td>7.6</td>
<td>0,0153333</td>
<td>0,013727</td>
</tr>
<tr>
<td>100</td>
<td>12</td>
<td>10</td>
<td>13.1</td>
<td>0,027</td>
<td>0,010359</td>
</tr>
</tbody>
</table>

Table 15: Total Number of Children Table Entries (12 groups).

<table>
<thead>
<tr>
<th>#overlay routers (%)</th>
<th>#groups</th>
<th>#runs</th>
<th>average max (#entries)</th>
<th>average mean (#entries)</th>
<th>confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>60</td>
<td>10</td>
<td>358</td>
<td>1,33833222</td>
<td>0,341701</td>
</tr>
<tr>
<td>10</td>
<td>60</td>
<td>10</td>
<td>98.8</td>
<td>0,24555544</td>
<td>0,023652</td>
</tr>
<tr>
<td>25</td>
<td>60</td>
<td>10</td>
<td>30.5</td>
<td>0,08740756</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>60</td>
<td>10</td>
<td>26.8</td>
<td>0,0522222</td>
<td>0,025987</td>
</tr>
<tr>
<td>100</td>
<td>60</td>
<td>10</td>
<td>35</td>
<td>0,07185178</td>
<td>0,01999</td>
</tr>
</tbody>
</table>

Table 16: Total Number of Children Table Entries (60 groups).

It is remarkable that, in the case of 6 groups and 100 joins/group, the average maximum number of children table entries is 321 when there are no overlay
On the performance of Application-level Multicast Routing

routers, it reduces to 39 and 36 for 10% and 25% overlay routers, and finally reaches 12 and 14 for 50% and 100% overlay routers. Similar results are observed for 12 and 60 groups. Hence, the participation of overlay routers has a substantial effect on reducing the number of entries for which a node is responsible for forwarding and duplicating packets. The graphical representation of these results is illustrated in figures 15 and 16.

**Figure 15: Number of Children Tables per Scribe node**

As can be observed from the figure, the node stress is much higher for groups with smaller group size in scenario 1. This is because Scribe forms longer paths with no branching for small group sizes, hence more Scribe nodes are responsible for a large number of non-empty children tables.

**Figure 16: Total Number of Children Table Entries per Scribe node**

These results verify the fact that Scribe distributes evenly the load on each
Scribe node: each node is responsible for forwarding multicast messages for a small number of groups, and it forwards multicast messages only to a small number of nodes. Therefore, Scribe is scalable to a large number of groups and to a large number of joins per group. Furthermore, the participation of routers in the overlay and Scribe as forwarders, reduces significantly node stress on Scribe nodes. Indeed, node stress is dramatically reduced and is minimized when all routers participate in the overlay. This is because the density of multicast groups increases and, taken together with the even distribution of load on Scribe nodes, this makes the membership load and the load for forwarding and duplicating packets distributed over more nodes.
8 Conclusions

The emerging Internet has introduced the need for bandwidth-intensive applications with high quality of service demands. In addition, the number of end hosts interested in network services increases dramatically. Internet Service Providers expand their network infrastructures in order to serve a large number of users and offer them high quality applications.

The need to provide demanding applications to end hosts without degrading network quality and eliminating network resources has led to the construction of overlay networks. An overlay network is a network of peers that is built on top of the physical network. Peers are connected with virtual links, each of which corresponds to a path in the underlaying network that may be comprised of several physical links. The overlay networks were designed to support application-level functionality and decouple the network from the complexity of these applications.

Furthermore, the inefficiency and lack of scalability of unicast routing, has introduced a new routing scheme, the multicast routing. Multicast routing is intended for applications that need to support group-based functionality. IP Multicast is a solution for group-based applications, yet it has seen low deployment. Another feasible solution is application-level multicast. Application-level multicast is built on top of overlay structures.

Distributed Hash Tables (DHTs) are an example of overlay-based communication that provide a scalable, self-organizing and fault-tolerant substrate for decentralized distributed applications. Such systems offer an attractive platform for publish-subscribe systems. Publish-subscribe is a new trend for multicast-based communication in application level.

Scribe is an application-layer multicast protocol for topic-based publish-subscribe. It is built on top of Pastry, an overlay DHT that offers an object location and routing substrate for distributed applications. Scribe leverages Pastry’s locality properties to create multicast trees and route messages.

Till now, application-layer multicast built on top of overlay networks was intended to involve only end hosts who are interested in the application. In this work, we propose a different scenario where routers participate in the
overlay. This is opposite to the initial motive for the construction of overlays. However, our expectation was that participation of routers in the overlay will upgrade multicast routing process by creating more optimal multicast trees.

We conducted simulation experiments to evaluate the performance of our proposed multicast routing scenario. More specifically, we evaluated the Scribe application-level multicast protocol with respect to two metrics: the overlay versus direct path ratio and the node stress imposed on each node.

Our simulation results, based on the Ipv4Network topology model of OverSim simulator, indicate that the participation of routers in the overlay and in the implementation of Scribe as forwarders, significantly reduces both overlay vs. direct path ratio and node stress imposed on each Scribe node. Indeed, node stress is dramatically reduced and is minimized to almost zero when all routers participate in the overlay.

It is clear then that the participation of routers optimizes multicast functionality. Therefore, it is advantageous for routers to participate themselves in the overlay based multicast routing if this is deemed to be a valuable service to the end users that they are serving.
9 Future Work

Part of our future work is to evaluate the performance of multicast routing with respect to the link stress on each physical link in the presence of overlay routers in the network topology. Our expectation is that the participation of routers in the overlay and in the implementation of multicast routing will reduce the stress on each physical link. This is because network density increases, hence the network traffic load is distributed over more nodes. Consequently the number of packets sent over each link will be reduced.

Another future work plan which is of great importance is to verify our results in realistic networks. To accomplish this, we will import transit-stub topologies of the GT-ITM topology generator [11, 16] in our simulation platform by using BRITE [18] as an intermediate step in the translation of GT-ITM topologies into OMNeT++’s topology description language (NED). BRITE’s GT-ITM transit-stub topology parser has been extended to preserve the hierarchical network structure, which was until now flattened, and Omnet++’s BRITE-to-NED conversion patch has been extended in order to support this hierarchical structure and at the same time to be interpretable by the Oversim package, which requires a specific NED description structure for the underlying network topology.

Lastly, in our simulations we considered a static network topology where end hosts once joining a multicast tree, never leave or fail. However, in real scenarios, nodes exhibit churn behavior in the sense that they join or leave the network whenever they want. Therefore, it is interesting to introduce churn behavior in end hosts. Moreover, mobility of end hosts should be taken into consideration. Mobility could be simulated by removing a node from one access net and adding the node to another access net.
10 References


On the performance of Application-level Multicast Routing


