DOCTORAL DISSERTATION

Efficient Information Dissemination Systems

Anton Riabov

Columbia University

Department of Industrial Engineering
and Operations Research

March 2004

Acknowledgements

I would like to thank my advisors Prof. Daniel Bienstock and Prof. Jay Sethuraman for their help and support in my studies and research. I am also very grateful to my coauthors and mentors Li Zhang, Zhen Liu, Sambit Sahu, Joel L. Wolf, and Philip Yu at T.J. Watson IBM Research Center for their help in finding and understanding exciting practical research problems. This work would not have been possible without them.

Contents

Chapter 1

Introduction

This thesis presents new models for analysis of overlay-based information dissemination systems and proposes new algorithms that allow these systems to achieve optimal performance, or performance that is provably close to optimal. In this work we study optimization problems associated with finding spanning trees that achieve optimal delay or throughput subject to degree constraints. Although we formulate these problems and describe algorithms with Internet overlay networks in mind, our methods can be applied for performance optimization in other types of information networks where node degrees can not exceed preset maximum, dictated by real-world constraints.

1.1 Information Dissemination Systems

The digital information era has brought incredible advances in computing and communication technology, the advances that have made possible the advent of new methods of communication, such as web and email, and their penetration in all spheres of our life. Increasingly accessible electronic communication is changing the way businesses work and people live. The spread of the Internet reaches almost all parts of the world, and even in space astronauts can read and send email messages.

The most popular applications of these new communication technologies, including email and the web, are built upon reliable point-to-point connection-based data transmission provided by a family of network protocols called TCP/IP. Traditionally this communication method is referred to as *unicast,* to emphasize that each such transmission has only one destination. However, in many practical communication scenarios the same block or stream of data must be sent to more than one destination. Such applications as web seminars, video conferences, or online stock trading systems, use the network to push data streams from a source node to a set of subscribers. Increasingly available and widespread broadband Internet connections contribute to the growing interest in these new and important applications that can efficiently support group interaction. These new applications have the potential to shape the future of the Internet and change the way people communicate once again.

In this work we will use the term *Information Dissemination Systems* to describe systems that deliver individual copies of the same data from one source computer or a cluster of computers to client computers (*subscribers*) via the Internet. Although we concentrate on developing methods for efficient delivery of data from one source to multiple receivers, our results can be applied in cases where more than one group member can be a transmission source. For example, a straightforward extension to multiple sources is to construct a dissemination system for each source separately, and then use a combination of these systems, assigning each source to the corresponding dissemination system.

The typical applications that use group communication can impose different, and sometimes conflicting, requirements on the information dissemination system. Devising methods for designing efficient and reliable content dissemination systems that can satisfy these requirements is crucial for the development of the future generation of Internet communication.

The requirements typically presented to the content dissemination systems may demand to keep system performance above a fixed threshold, or may require the system to achieve best possible performance. Further, the requirements may impose constraints on reliability and other aspects of system behaviour.

Many applications, such as real-time streaming applications, including video-over-Internet broadcasts and video conferences, require *high throughput* and *low latency* of the dissemination system. Typically in streaming applications group membership changes are relatively infrequent, and transmitted data size is large, so that configuration time is insignificant when compared to transmission time. This property allows to use models that assume steady data flow in the network, that is not affected by configuration traffic and associated computation. Similarly, in cache synchronization applications and multiplayer video games group membership can be considered static.

Examples where static group membership does not hold can include peer-to-peer networks and news distribution systems, such as stock price update distribution. In news distribution systems very small news messages are delivered to large groups of subscribers, however amount of data sent to each particular user of the system is small.

Reliable delivery is another important requirement, which must be satisfied by the dissemination system in most of the described applications. Finally, in addition to throughput and latency requirements, which can be viewed as performance requirements, there is often also a requirement of *scalability*: the system must efficiently support communication within large groups, and small increase in group size should not lead to sharp decline in system performance.

1.2 Internet Infrastructure and Multicast

The Internet is a composition of a number of connected packet networks (subnets) supporting communication between host computers via Internet protocols. All protocols used in the Internet are based on the Internet Protocol (IP), which provides a basic data transport mechanism. The routers (or gateways) forward IP datagrams between subnets based on IP address specified in the datagram and a routing table stored in the router. During delivery IP datagrams may be damaged, lost, or arrive out of order. Applications running on the host computers rely on transport protocols for end-to-end communication. There are two transport layer protocols: Transmission Control Protocol (TCP) and User Datagram Protocols (UDP). TCP provides reliable connection-based data delivery. It can resequence packets if needed and adapt to changing network conditions by reducing or increasing transmission rate. UDP is a connectionless transport protocol, which does not guarantee reliable datagram delivery, but does not have the overhead associated with reliable communication.

IP address is a 32-bit number, which identifies destination of a datagram. A newer version of the Internet Protocol (IPv6) can use 48-bit numbers for addressing. Longer addresses allow to address more hosts uniquely, and therefore to support larger networks. However, these extended addresses are used in a very similar addressing scheme, and in this discussion we will refer to 32-bit version, which is commonly used in the current Internet. There are five classes of IP addresses: class A through E.

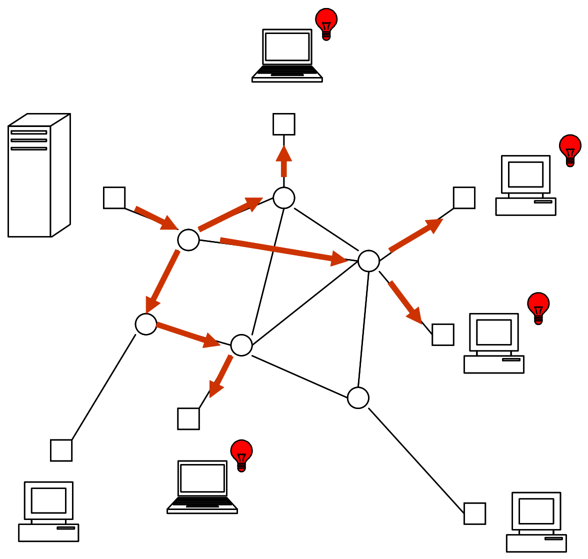


Figure 1.1: IP Multicast Tree.

Class of the IP address is defined by first bits of the address (address prefix). IP addresses of classes A, B and C identify hosts, and are used to specify targets of unicast communication. The distinction between classes A, B and C is no longer important. Class E address are reserved for experimental use. A special address 255.255.255.255 is used for broadcast: a datagram addressed to this address must be delivered to all hosts. And finally, class D addresses are used for *multicast*. A datagram sent to a multicast address must be delivered to all hosts belonging to a group of hosts, called a *multicast group*. Multicast group is identified by a 28-bit number encoded in class D address.

Multicast can be viewed as a middle point between unicast and broadcast: it implements one-to-many communication, as opposed to one-to-one in case of unicast or one-to-all in case of a broadcast. Multicast provides a more efficient method of group communication, than unicast. If the communication network is not a complete graph, sending the same data from the same source to several destinations can involve sending multiple copies of the same packet (but with different destinations) on the same link. This may not pose a problem for small groups and small transmission sizes, but in many applications limited bandwidth of the interface link connecting the source to the network may limit the size of the group. Multicast allows packets to follow a spanning tree reaching all destinations, while sending only one copy of a packet on each link, and creating a necessary number of copies at branching nodes.

Figure 1.1 shows an IP multicast tree in a network connecting computers and routers. Routers are shown as circles, and computers as squares. Distribution tree is shown by arrows, and routers can forward packets they receive to one or more destinations, as required by multicast routing.

Although a special class of IP addresses had been reserved for multicast from the beginning, multicast was being adopted very slowly. Up to this day availability of IP multicast in the Internet is limited: not all routers can interact to provide necessary multicast service. In particular routers must be able to forward datagrams to more than one destination, and to support required data structures in routing tables. A number of multicast routing protocols, such as PIM, DVMRP, CBT, MOSPF, have been developed. A survey of IP multicast methods can be found in [42]. For more details about Internet protocols and IP multicast see e.g. [22]. For discussions of IP multicast applications and challenges see [47].

There are several reasons why IP multicast is not currently widely used for information dissemination over the Internet:

1. *Limited number of groups.* Each multicast group must have a globally unique IP address. As a consequence, IP address range used by an application must be reserved in advance, and the number of groups that can coexist in the network is limited. This can result in service being denied to clients who attempt to establish a group when the allocated range of IP addresses is exhausted.

2. *Scalability.* Although multicast resolves the link congestion problem that exists in unicast case, discovery of group members and maintaining data structures in routing tables still presents challenges for large networks.

3. *Interdomain Routing.* Existing multicast routing protocols are deployed within isolated subnets within the Internet, and for large-scale multicast deployment it is required to connect the subnets via an interdomain multicast routing protocol. However, currently proposed protocols are considered too complicated and ineffective to be deployed [4].

4. *Reliability vs. Scalability.* IP multicast by itself does not provide reliable connection, and if reliability is desired, additional functionality must be implemented. However it has been shown that in such implementations throughput vanishes when group size increases [53, 16].

5. *Security.* IP multicast delivers a packet sent to multicast address to all destinations subscribed to the corresponding multicast group, while reducing traffic leaving the source to only one copy of the packet. Absence of access control makes IP multicast a target for Denial of Service (DoS) attacks.

1.3 Overlay Multicast and Performance Optimization

When it became apparent that IP multicast has very limited capacity as a means of information dissemination over Internet, an alternative called *overlay multicast* was proposed. This new technology is also referred to as *end-system multicast* or *application level multicast*. In overlay multicast the multicast distribution tree is formed using point-to-point connections between end systems (hosts). Standard Internet transport protocols (TCP or UDP) are used to provide transport service between end systems, and therefore the network routers are not required to support any of the IP multicast routing protocols. However the end systems are now required not only to receive the data, but to forward it to other end systems located deeper in the distribution tree. Overlay multicast removes the burden of multicast routing and packet replication from the routers, which allows to utilize existing Internet infrastructure for efficient information dissemination.

Figure 1.2 illustrates an implementation of overlay multicast in the network introduced in Figure 1.1. Arrows indicate unicast flows between end systems. An edge in the overlay network represents the path between the two nodes that it connects. While this path may traverse several routers in the physical network, this level of abstraction considers the path as a direct link in the overlay network.

This new technology has become an area of extensive research. Various studies have been conducted with the primary focus on the protocol development for efficient overlay tree construction and maintenance, such as Narada [19], Yoid [25], ALMI [46, 54], Host Multicast [64], NICE [9], Delauney graph [37], and [55, 56]. Some other work in peer to peer network area is also related to the tree construction in overlay multicast, see e.g. Chord [59] and CAN [48].

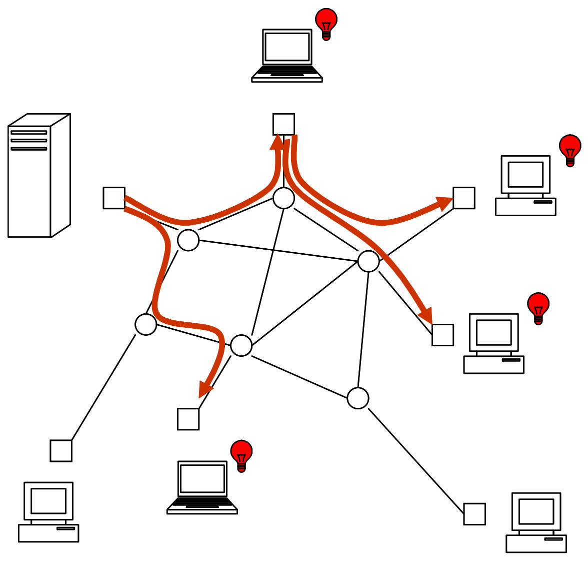


Figure 1.2: Overlay Multicast Tree.

Reliable multicast can also be implemented in overlay using point-to-point TCP connections. An examples of such systems include Overcast [33] and RMX [17]. The main advantage of such approaches is the ease of deployment. In addition, [17] argues that it is possible to better handle heterogeneity in receivers because of hop-by-hop congestion control and data recovery provided by TCP. Further, it has been shown that in contrast to reliable network-supported multicast, TCP-based overlay multicast is scalable [7, 6].

Performance characteristics of information dissemination system based on overlay multicast may vary depending on the choice of links that are used to build the multicast tree. Increasing the number of downstream connections for the source node, for example, may reduce latency of the system by reducing tree depth, but it may also reduce multicast throughput, since capacity of the link connecting the source node to the Internet must be shared between all outgoing links. Bandwidth sharing between point-to-point transmissions comprising the same multicast tree makes overlay multicast routing problem more difficult than its IP multicast counterpart, and hence wide range of performance optimization methods and models developed for IP multicast cannot be reused directly for overlay optimization.

In research papers that describe heuristics for optimal overlay routing, this throughput sharing phenomenon is modeled by imposing degree constraints on the nodes in multicast tree, see e.g. [39, 57, 54]. Degree constraints are also referred to as *fanout* constraints, due to the fact that multicast trees are directed, and degree constraints can be replaced by equivalent constraints on out-degree. In other words, it is assumed that if a certain fixed level of throughput must be achieved, the maximum number of streams that can be forwarded to downlinks by a node can be determined based on the capacity of access link connecting the node to the Internet. This model ignores capacity sharing between streams that may occur on the links inside the Internet, which is usually justified by high capacity of these links and efficient routing algorithms.

Problems of finding spanning trees that satisfy degree constraints and, optionally, optimize an objective function were also studied in a few research papers not directly related to overlay multicast. In [15] a numerical solution approach based on branch-and-bound method is proposed for the problem of finding minimum spanning tree which is subject to degree constraints. In [34] an approximation algorithm is proposed for the minimum diameter spanning tree problem. Finally, in [26] and [27] an approximation algorithm for finding a spanning tree of minimum degree is described. However, existing research of this topic does not provide ready solutions for optimal overlay routing problem, and further investigation of this class of problems is required.

1.4 Summary of Contributions

The main contribution of this dissertation is a family of efficient methods that can maximize scalability of an information dissemination system by configuring the system such that it achieves optimal or close to optimal performance. Our contributions lie in two areas related to large-scale information dissemination systems. First, we develop models and efficient methods for finding optimal overlay multicast trees with respect to throughput and latency. Second, we address the issue of forming multicast groups for the special case of information dissemination systems, where subscribers are interested in receiving only those messages that correspond to their interests – so called content-based publish-subscribe systems. We explain our results in more detail below.

Scalable and Reliable Overlay Multicast Architecture. We propose an architecture for reliable overlay multicast, which can recover from node failures and provide support for clients joining and leaving an ongoing multicast session. We evaluate scalability and reliability characteristics of our prototype implementation in the Internet. We demonstrate that reliable multicast overlays can be deployed on top of the current TCP/IP by adding a light set of application layer back-pressure mechanisms that guarantee both end-to-end flow control and reliability. We further show that architectures can be used for group communications of large sizes and still provide a group throughput that is close to that of a single point to point connection with these minimal guarantees.

Minimum Latency Routing. We describe an algorithm for constructing a multicast tree with the objective of minimizing the maximum communication delay (i.e. the longest path in the tree), while satisfying degree constraints at nodes. We show that the algorithm is a constant-factor approximation algorithm. We further prove that the algorithm is asymptotically optimal if the communicating nodes can be mapped into Euclidean space such that the nodes are uniformly distributed in a convex region. We evaluate performance of the algorithm using randomly generated configurations of up to 5,000,000 nodes.

Maximum Throughput Routing. We formulate the optimization problem of maximizing throughput in overlay multicast, and prove that it NP-hard. We further prove a bound on best achievable approximation factor. Next, we develop a constant factor approximation algorithm for the problem, which achieves throughput close to the best possible. Our experiments show that the problem of finding optimal routing can be very hard even on small networks, and cannot be solved by standard numerical solvers. We then develop an approach, based on integer programming, volume algorithm and cutting planes, that allows to find the exact optimal routing numerically, and perform numeric experiments.

Interest-Based Grouping. We consider efficient communication schemes based on multicast techniques for content-based publication-subscription systems. We propose to use clustering algorithms to form multicast groups. We devise new algorithms and adapt partitional data clustering algorithms for these content-based publication-subscription systems. These algorithms can be used to determine multicast groups with as much commonality as possible, based on the totality of subscribers' interests. These algorithms perform well in the context of highly heterogeneous subscriptions, and they also scale well. We demonstrate the quality of our algorithms via a set of simulation experiments.

1.5 Structure of the Thesis

This thesis is organized as follows. In Chapter 2 we introduce reliable overlay multicast architecture, and investigate scalability and reliability of the proposed system via experiments in the Internet. Chapter 3 studies minimum latency routing. We describe the model, and formulate an optimization problem that we call minimum radius spanning tree with degree constraints. We analyze existing work on mapping Internet delays to Euclidean space, and conclude that to solve the latency minimization problem it is sufficient to the minimum radius problem for the special case of Euclidean distances. We then describe a constant factor approximation algorithm. We prove that the algorithm constructs solutions that are asymptotically close to optimal if nodes are uniformly distributed in convex region. In Chapter 4 we study maximum throughput routing. We develop a constant factor approximation algorithm and a method for solving the problem exactly. Chapter 5 studies the problem of grouping subscribers based on interest, which arises in a special subset of information dissemination systems, namely content-based publish-subscribe systems. In this chapter approach the problem of grouping subscribers separately from the multicast routing problem due to its complexity. We propose and evaluate clustering heuristics for solving this problem. Finally, in Chapter 6 we summarize our results and propose directions for future work. Results presented in this thesis were published in [49, 50, 51, 7, 6].

Chapter 2

Reliable Overlay Multicast

2.1 Introduction

With the proliferation of broadband Internet access, end-system multicast becomes not only a practically feasible approach, but also an appealing alternative to the IP-supported multicast which has been experiencing deployment obstacles. In many applications, such as cache synchronization, peer-to-peer systems, and online learning reliable delivery is very important. In this chapter we will describe system architecture for reliable overlay multicast, and discuss reliability and scalability of proposed architecture.

Reliable overlay multicast can be implemented using point-to-point TCP connections. In Overcast [33], HTTP connections are used in between end-systems. In RMX [17], TCP sessions are directly used. The main advantage of such approaches is the ease of deployment. In addition, [17] argues that it is possible to better handle heterogeneity in receivers because of hop-by-hop congestion control and data recovery.

Two issues arise from this approach. The first one is the end-to-end reliability. In case of failure of a interior node of the overlay multicast tree, the nodes in the subtree rooted at the failed node need, on one hand, to be re-attached to the remaining tree and, on the other hand, to get TCP sessions established from where they are stopped. The former is referred to as the resiliency issue in the literature, which, in this context, consists in the detection of failures and in the reconstruction of trees. Very recently resilient architectures have become a hot topic. For example, in [10], a resilient multicast architecture was proposed using random backup links. While it is relatively easy to find nodes of re-attachment and thus to reconstruct the tree, it is not guaranteed that the TCP sessions can be restarted from where they are stopped due to the fact that the forwarding buffers of the intermediate nodes in the overlay network have finite size. It may happen that the packets needed by the newly established TCP sessions are no longer in the forwarding buffers.

The second issue that arises in reliable multicast using overlays is the scalability. There is a lack of understanding of the performance of TCP protocol when used in an overlay based group communication to provide reliable content delivery. Although studies in [17, 33] have advocated the usage of overlay networks of TCP connections, they do not address the scalability concerns, in terms of throughput, buffer requirements and latency of content delivery. In contrast, significant effort has been spent on the design and evaluation of IP-supported reliable multicast transport protocols in the last decade, see for example [24, 14, 36] and the references therein. It has been shown in various studies [53, 16] that for such IP-supported reliable multicast schemes, group throughput vanishes when the group size increases. Thus these schemes suffer from scalability issues.

Some preliminary results have been reported recently on the scalability issue of overlay based reliable multicast. In [60], the authors investigated this scalability issues while considering a TCP-friendly congestion control mechanism with fixed window-size for the point-to-point reliable transfer. Simulation results were presented to show the effect of the size of end-system buffers on the group throughput.

In recent work by F. Baccelli and others [7, 6] it was shown using max-plus algebra models that reliable overlay multicast based on point-to-point connections via TCP has scalable throughput in the sense that the group throughput is lower bounded by a constant independent of the group size.

In this chapter, we show that end-to-end reliability of multicast using overlays can be achieved with the native TCP back-pressure mechanism together with backup buffers. We call *back-pressure* the ability for the receiver of a TCP connection to stop packets being sent by the source, in case its receiving buffer is full (see more details in the next section). More specifically, we propose a simple end-system multicast architecture where the transfers between end-systems are carried out using TCP. The intermediate nodes have finite size forwarding buffers. There is also a backup buffer with finite size in each of these intermediate nodes storing copies of packets which are copied out from the receiver window/buffer to the forwarding buffer. These backup buffers are used when TCP connections are re-established for the children nodes after their parent node fails. Using theoretical investigations, experimentations in the Internet, and simulations of large networks, we show that such an architecture provides end-to-end reliability and can tolerate multiple simultaneous node failures, provided the backup buffers are sized appropriately. We also confirm via experiments in the Internet that the theoretical scalability results presented in [6], namely that the throughput of this reliable group communication is always strictly positive for any group size and any buffer size. The back-pressure mechanism of TCP allows not only reliable point-to-point transfers, but also scalable end-system multicast.

In our architecture, we also propose leave and join procedures which guarantee the reliability and scalability of the group communication. These considerations of leave and join events are particularly motivated by the increasing interest in using TCP for real time applications, (see studies on multimedia streaming over TCP [31, 40]). TCP is an appealing alternative of UDP for such applications due to a number of advantages such as fair bandwidth sharing, in-order delivery and passing through client imposed firewalls, which may only permit HTTP traffic.

The chapter is organized as follows. Section 2.2 describes the core of the problem and the multicast overlay architecture. Section 2.3 describes the algorithms ensuring overall reliability and gives a proof of their end-to-end reliability properties. Simulation results and Planet-Lab experiments aiming at showing the joint scalability and reliability properties of this kind of architecture are gathered in §2.4.

2.2 Architecture of Reliable Overlay Multicast

We consider the problem of reliable group communication where the same content has to be transported in an efficient way from a source to a set of users. The broadcasting of this content is made efficient via the setting of a multicast tree where each node of the tree duplicates the packets it receives from its parent node and sends them to all its child nodes. In contrast to native reliable IP multicast where the nodes of the tree are Internet routers and where specific routing and control mechanisms are needed, overlay multicast uses a tree where the nodes are end-systems and where the currently

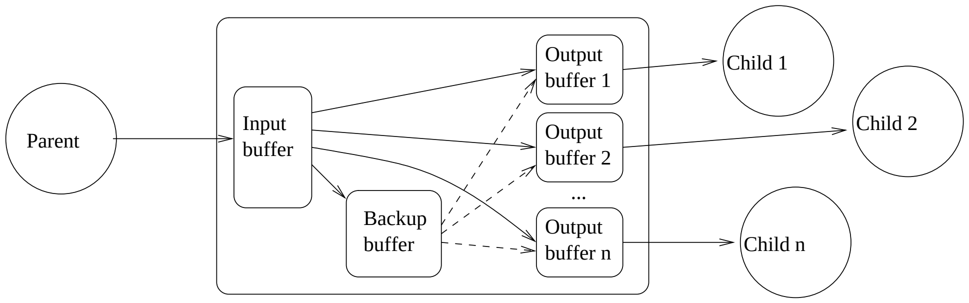


Figure 2.1: Reliable Overlay Multicast: Different buffers in a node.

available point to point connections between end-systems are the only requirement.

An edge in the overlay network represents the path between the two nodes that it connects. While this path may traverse several routers in the physical network, this level of abstraction considers the path as a direct link in the overlay network. The end-systems participate explicitly in forwarding data to other nodes in a store-and-forward way. The point-to-point communication between a parent node and a child node is carried out by TCP. As illustrated in Figure 1.2, after receiving data from its parent node, a node will replicate the data on each of its outgoing links and forward it to each of its child nodes in the overlay tree.

In such an overlay network, except for leaf nodes, all the other nodes, which store and forward packets, need to provision buffers for the packet forwarding purpose. On each node (except for the source node), there is an input buffer, corresponding to the receiver window of the upstream TCP, and, except for the leaf nodes, there are several output buffers, also referred to as forwarding buffers, one for each downstream TCP connections. There is also a backup buffer in each of these intermediate nodes storing copies of packets which are copied out from the input buffer (receiver window) to the output (forwarding) buffers. These backup buffers are used when TCP connections are re-established for the children nodes after their parent node fails. Figure 2.1 illustrates these buffering mechanisms. Throughout this chapter we shall assume that all these buffers have finite sizes $B_{\mathtt{IN}}$, $B_{\mathtt{OUT}}$, $B_{\mathtt{BACK}}$ (for, respectively, input buffer, output buffer and backup buffer).

Tree topology will have an effect on the performance of the group. If the depth of the tree is too big, the nodes deep in the tree will receive packets with long delay. On the other hand, if the (out)degree of the tree is too big, the downstream TCP connections will compete for the bandwidth of the shared links, especially that of the "last mile". In this chapter, we will not consider the tree construction optimization issue, which is studied in Chapters 3 and 4. Rather, we assume that the tree topology is given, and that the out-degrees (or fan-out) are bounded by a constant D.

From management perspective, the tree topology information needs to be stored and updated by the end-systems. Each node should at least have a partial view of the tree. Different architectures could be considered and implemented. In this work, we consider a simple, but not necessarily the most efficient, structure which consists in allowing each node to know its ancestor nodes and its entire subtree.

Notation We will consider several tree topologies, for which we introduce the following generic notation: we number end-systems by a pair (k, l) designing their location in the multicast tree. The first index k gives their distance to the root of the tree (or *level*). The second index l allows one to number end-systems with the same level. For the case of a complete binary tree, the end-systems with the same distance k from the source are labeled by numbers $l = 0, \ldots, 2^k - 1$.

The parent node of end-system (k, l) will be denoted $(k-1, \mathtt{m}(k, l))$. The child nodes of end-system (k, l) will be labeled $(k+1, l')$ with $l' \in \mathtt{d}(k, l)$. For a complete binary tree, $\mathtt{m}(k, l) = \lfloor \frac{l}{2} \rfloor$ and $\mathtt{d}(k, l)$ is $\{2l, 2l+1\}$.

2.2.1 Reliable Transfer and Forwarding via Back-Pressure

There can be three different types of packet losses in the overlay multicast: (1) losses that occur in the path in-between the nodes (sender and receiver); (2) losses due to input buffer overflow; (3) losses due to output buffer overflow. The first type of losses are recovered by the TCP acknowledgment and retransmit mechanisms. The second type of losses will not occur thanks to the back-pressure mechanism of TCP (RENO, New RENO or SACK). Indeed, the available space in the input buffer at the receiver node is advertised to the sender through the acknowledgments of TCP. The acknowledgment packet sent by the receiver of the TCP connection contains the space currently available in the receiver window. The sender will not send a new packet unless the new packet and those "in-fly" packets will have room in the receiver window. In addition, when the available input buffer space differs from the last advertised size by two Maximal Segment Size (MSS) or more, which can occur when packets are copied to the output buffers, the receiver sends a notification to the source via special packet. The last type is avoided in our architecture. Indeed, a packet will be removed from the input buffer only when it is copied to all of the output buffers. The copy process is blocked when an output buffer is full and is resumed once there is room for one packet in that output buffer. Thus, due to this "blocking" back-pressure mechanism, there will be no overflow at the output buffers.

2.2.2 End-to-End Reliability and Backup Buffers

With end-system multicast scheme, an important issue to address is resiliency, i.e., handling node failures and/or departures (possibly without prior notice). For this, one first needs to detect failures. Unfortunately TCP does not provide a reliable and efficient mechanism for detecting nodes that are not responding. Different methods can however be deployed for this purpose, e.g. heartbeat probes, keep-alive signals, etc. A heartbeat message is sent using UDP at regular time intervals to all neighbors of a node, and missing heartbeats from a neighbor indicate a node failure. The keep-alive messaging system can be established in a similar way. We will assume that one of such mechanisms is deployed. The comparison of their efficiency is out of the scope of this work.

Once a failure is detected, the tree needs to reconfigure in such a way that the subtrees rooted at the child nodes of the failed node are re-attached to the original tree. A new TCP connection is established for each re-attachment. In other words, we need to find "step-fathers" for the child nodes of the failed node. There is a variety of ways to reconfigure the tree. We shall present some in Section 2.3 and study their impact in Section 2.4. This reconfiguration is completed by propagating the information of new subtrees up to the root, and by propagating the ancestor chain information down to the newly reconnected subtrees. This tree topology information update process is initiated by the parent node and the child nodes of the failed node; whereas the ancestor chain information update process is initiated by the "step-father" nodes. In order to achieve end-to-end reliability, we need to ensure the data integrity while providing resiliency. In other words, we need to make sure that when these child nodes (of the failed node) are attached to the remaining tree, the new parent nodes have the data that are old enough so that these child nodes, as well as their offspring, receive the entire sequence of packets that the source sends out. For this purpose, we implement the backup buffers in the end-systems so that whenever a new TCP connection is established, the packets in the backup buffer of the sender will first be copied to the output buffer corresponding to this new connection. In this way, the sender starts with these packets that have smaller sequence numbers than those in the input buffer of the sender.

We will show in Section 2.3 that if the size of the backup buffer is large enough compared to those of input and output buffers, then the end-to-end reliability is guaranteed even if there are multiple simultaneous failures. More precisely, Let $B_{\mathtt{OUT}}^{\max}$ and $B_{\mathtt{IN}}^{\max}$ be the maximum sizes of output and input buffers, respectively. Then the backup buffers should be of size

$$B_{\mathtt{BACK}} \geq m \cdot (B_{\mathtt{OUT}}^{\max} + B_{\mathtt{IN}}^{\max}) + B_{\mathtt{OUT}}^{\max} \tag{2.1}$$

in order to tolerate m simultaneous failures. Under such a condition, the child nodes of the failed node can be re-attached to any of the nodes in the subtree rooted at the m-th generation ancestor of the failed node.

It is worthwhile noticing that this backup buffer architecture for the end-to-end reliability is very simple. In particular, it can be implemented at the application level and there is no need of searching for nodes possessing the packets with the right sequence numbers that these child nodes need.

2.2.3 Leave and Join Procedures

While convention wisdom suggests that UDP should be used for real time applications, there has been increasing interest in multimedia streaming over TCP, see e.g. studies in [31, 40]. TCP is an appealing alternative of UDP for such applications due to a number of advantages such as fair bandwidth sharing, in-order delivery. Moreover, TCP could pass through client imposed firewalls, which may only permit HTTP traffic. We envision that the end-system based reliable multicast architecture described above can be deployed for broadcast of live events.

Thus, we also consider leave and join procedures. The leave procedure can simply be the notification of the departure to the parent node and to the child nodes, followed by the disconnect of the corresponding TCP sessions. The tree can then be reconfigured as in the node failure situation. When a node joins the group, it can first contact the source node which will inform the new node who should be its parent node. The new node can then establish a TCP connection with the designated parent node. In order to guarantee the end-to-end reliability in the new tree, the source also notifies the new node about the constraints on the buffer sizes such that the input and output buffer sizes should not exceed $B_{\mathtt{IN}}^{\max}$ and $B_{\mathtt{OUT}}^{\max}$, respectively; and the backup buffer size should satisfy inequality (2.1). These leave and join procedures are completed by the topology information update process as described previously in the node failure case.

2.3 End-to-End Reliability

In this section we investigate into the end-to-end reliability issue. As we mentioned in Section 2.2, using TCP for point-to-point connections guarantees reliable transfers between the nodes of the group, but does not provide uninterrupted transmission in cases when a transit node suddenly stops functioning. Child nodes of the failing node must restore connection to the multicast group, and they should receive all stream packets. Throughout this chapter we shall assume that the source node never fails.

Avoiding interruption in packet sequence may not be trivial, especially for nodes distant from the root, since the packets that these nodes were receiving at the time of failure may have been already processed and discarded by all other group members, except for the failed node. We employ *backup buffers* to create copies of stream content which could be otherwise lost during node failure. Figure 2.1 illustrates our approach. While data is moved from the input buffer to the output buffers, a copy of data leaving input buffer is saved in the backup buffer. The backup buffer can then be used to restore packets which were lost during node failure.

We will show below that this end-to-end reliability can be achieved through the backup buffers, provided they are sized appropriately. We will formally derive a formula for the size of the backup buffer. We shall also present leave/join algorithms so as to keep the group throughput scalable.

2.3.1 Guarantee of the End-to-End Reliability

Definition of End-to-End Reliability. We define overlay multicast system to be *end-to-end reliable with tolerance to m failures*, if after removing *simultaneously* m nodes from the multicast tree and restoring connectivity, transmission can be continued, and all remaining nodes receive entire transmission in the same sequence. In other words, failure of m nodes does not lead to any changes in the sequence or content of the stream received at the remaining nodes. However recovering from failure may incur a delay, which is required to restore connectivity.

During the time when the system is recovering from m failures, it is not guaranteed to recover correctly from any additional failures. However if l, for some $1 \le l \le m$, failures occur, the system will be able to recover from additional $(m-l)$ failures even if the failures happen before the system has completely recovered. In such situations new failures occurring during recovery will increase total recovery time.

Let $B^{\max}_{\mathtt{OUT}}$ and $B^{\max}_{\mathtt{IN}}$ be the maximum sizes of output and input buffers in the system, respectively. A *backup buffer of order r* has size $(r \cdot (B^{\max}_{\mathtt{OUT}} + B^{\max}_{\mathtt{IN}}) + B^{\max}_{\mathtt{OUT}})$.

Failure Recovery Algorithm. We use the following simple algorithm to recover from failures. We shall call node (k', l') *surviving ancestor* of node (k, l), if the parent of node (k, l) did not survive the failure, and (k', l') is the first surviving node on the path from (k, l) to the source. Each disconnected end-system (k, l) must be reconnected to a node that belongs to the subtree of the surviving ancestor (k', l'). After connection is restored, the node (k', l') retransmits all packets contained in its backup buffer. Then it continues the transmission, reading from input buffer and writing to output buffer. Intermediate nodes on the new path from (k', l') to (k, l), as well as all nodes in the entire subtree of (k, l), must be able to ignore the packets that they have already received, and simply forward them to downstream nodes.

Theorem 2.3.1. *An overlay multicast system with backup buffer of size $(m \cdot (B^{\max}_{\mathtt{OUT}} + B^{\max}_{\mathtt{IN}}) + B^{\max}_{\mathtt{OUT}})$ is end-to-end reliable with tolerance to m failures.*

Proof: To estimate required backup buffer size, we first consider a chain of nodes $(k_1, l_1) \rightarrow (k_2, l_2) \rightarrow (k_3, l_3)$. Let $W^{(k_{i+1}, l_{i+1})}$ be the size of the receiver window on the TCP Connection (k_{i+1}, l_{i+1}), for i=1, 2. Suppose that node (k_2, l_2) fails. When failure is detected, node (k_3, l_3) will connect to node (k_1, l_1) and request it to re-send packets starting from packet number t+1, where t is the number of the last packet that node (k_3, l_3) received. The number of packets stored in input and output buffers at node (k_2, l_2), plus the number of packets 'in-fly' to and from node (k_2, l_2), is at most $(B^{\max}_{\mathtt{OUT}} + B^{\max}_{\mathtt{IN}})$.

This bound is guaranteed by TCP's choice of receiver window size: at most $W^{(k_2,l_2)}$ packets will be 'in-fly' to node (k_2, l_2), and $W^{(k_2,l_2)}$ does not exceed the amount of free memory in the input buffer of node (k_2, l_2). Similarly, at most $W^{(k_3,l_3)}$ packets will be 'in-fly' to node (k_3, l_3), but they are not removed from the output buffer of node (k_2, l_2), until (k_3, l_3) acknowledges that it has received the packets. Therefore the difference between the smallest packet number at node (k_1, l_1) and the highest packet number at node (k_3, l_3) does not exceed the sum of buffer sizes at node (k_2, l_2). During re-transmission the application at node (k_1, l_1) does not have access to the output socket buffer, and may need to re-transmit the contents of this buffer as well. Hence the total number of packets that need to be re-transmitted is bounded by $B^{\max}_{\texttt{OUT}} + (B^{\max}_{\texttt{OUT}} + B^{\max}_{\texttt{IN}})$, which is the size of an order 1 backup buffer.

If (k_2, l_2) has more than one child node, each of the child nodes will require at most $B^{\max}_{\texttt{OUT}}+(B^{\max}_{\texttt{OUT}} + B^{\max}_{\texttt{IN}})$ packets to be re-transmitted, and the same backup buffer of order 1 will provide all necessary packets.

If more than one failure occurs, and there is more than one failing node on the path from disconnected node (k, l) to it's surviving ancestor node (k', l'), the surviving ancestor node will need to re-transmit the contents of input and output buffers at all failing nodes on the path, plus the contents of output buffer at (k', l'). Since the number of failing nodes is bounded by m, we have proven the theorem. $\square$

Note that in our definition of tolerance of failures we used standard notion in the fault tolerance literature. The proof of Theorem 2.3.1 actually proves a much stronger result, which we state it as a corollary here:

Corollary 2.3.2. *An overlay multicast system with backup buffer of size* $(m \cdot (B^{\max}_{\mathit{OUT}} + B^{\max}_{\mathit{IN}}) + B^{\max}_{\mathit{OUT}})$ *is end-to-end reliable with tolerance to* m *simultaneous and consecutive failures in a chain of the tree.*

2.3.2 Handling Leave and Join and Restoring Connectivity

In practice a multicast system should allow nodes to leave and join the group during transmission. Leaving nodes can be handled by the failure recovery scheme. Several different strategies can be used for joining the group. A node joining the transmission may want to connect to a distant leaf node, which is processing packets of the smallest sequence numbers, so that the newly joined node can capture the most of transmitted data. However, if delay is an important factor, a joining node will try to connect to a node as close to the root as possible. In practice, the maximum number of down-links for each node is limited, due in particular to the last-mile effect, and not every node in the multicast group can accept new connections. Therefore the up-link node for new connection is chosen among "active" nodes, which have not yet exhausted their capacity.

The procedure for restoring connectivity after a failure is similar to the join procedure, but in this case the choice of nodes is further limited to the subtree of the surviving ancestor. In applications where communication delay is to be minimized, the goal is to maintain a tree as balanced as possible, subject to the degree constraint. We propose to use a greedy heuristic to restore connectivity. Our heuristic tries to minimize overall tree depth, subject to the degree constraint, by reconnecting longest subtrees to nodes that are as close to the root as possible. The algorithm description below is given for the case of one node failure, but the case of several failures can be handled as a sequence of single failures.

Algorithm GREEDY_RECONNECT

1. Suppose node (k, l) fails. Let $\mathcal{S}$ be the set of orphaned subtrees, rooted at childs of (k, l). Let $\mathcal{A}$ be the set of active nodes in subtree of $(k - 1, \mathtt{m}(k, l))$, but not in the subtree of (k, l).

2. Choose a node $(k+1, l') \in \mathcal{S}$ that has subtree of largest depth.

3. Choose a node $(p, q) \in \mathcal{A}$ that is closest to the source.

4. Connect $(k+1, l')$ to (p, q).

5. Update $\mathcal{S} \leftarrow \mathcal{S} \setminus \{(k-1, l')\}$ and add active nodes from subtree of $(k+1, l')$ to A.

6. If $\mathcal{S}$ is not empty, go to Step 2.

Depending on the objective function, other approaches can be considered. For example, if throughput is to be maximized, and last-mile links have limited bandwidth, then lower fanout allows to achieve higher throughput, and the optimal topology could be a chain. On the other hand, if delay is to be minimized, the optimal configuration would likely be the star where all the nodes have direct connections with the source node. Finally, if no specific goals are set, a random choice of uplink node (still subject to fanout constraints) is feasible.

2.4 Simulation and Experimental Results

2.4.1 Scalability and Reliability: Internet Measurements

In order to evaluate practicality of our models, we have implemented a prototype of TCP overlay multicasting system. We used Planet-Lab network, which gives access to computers located in universities and research centers over the world. Our implementation runs a separate process for each output and input buffer, which are synchronized via semaphores and pipes. As soon as data is read from input buffer, it is available for outgoing transmissions. A separate semaphore is used to ensure that data is not read from input socket, if it can not be sent to output buffers, which creates back-pressure. A dedicated central node was used to monitor and control progress of experiments.

Scalability Analysis. To analyze scalability of throughput, we constructed a balanced binary tree of 63 nodes (Figure 2.2) connected to the Internet. We started simultaneously transmissions in balanced sub-trees of sizes 15, 31 and 63 with the same source. Running experiments simultaneously allowed us to avoid difficulties associated with fluctuation of networking conditions. In this way, link capacities are always shared between trees of different sizes in roughly equal proportions across the trees. We measured throughput in packets per second, achieved on each link during transmission of 10MB of data. Throughput of a link was measured by receiving node. In Table 2.1 we summarize group throughput measurements for 3 different tree sizes and 3 different settings for output buffer size. Group throughput is computed as the minimum value of link throughput observed in the tree. Similarly to our simulations presented above, size of each packet is 200 bytes, size of the input buffer is equal to 50 packets, and size of the output buffer is variable. Output buffer size is given in packets.

One can observe that the group throughput changes very little in the group size. This is consistent with the simulation results reported above, although as is quite expected, the absolute numbers are different.

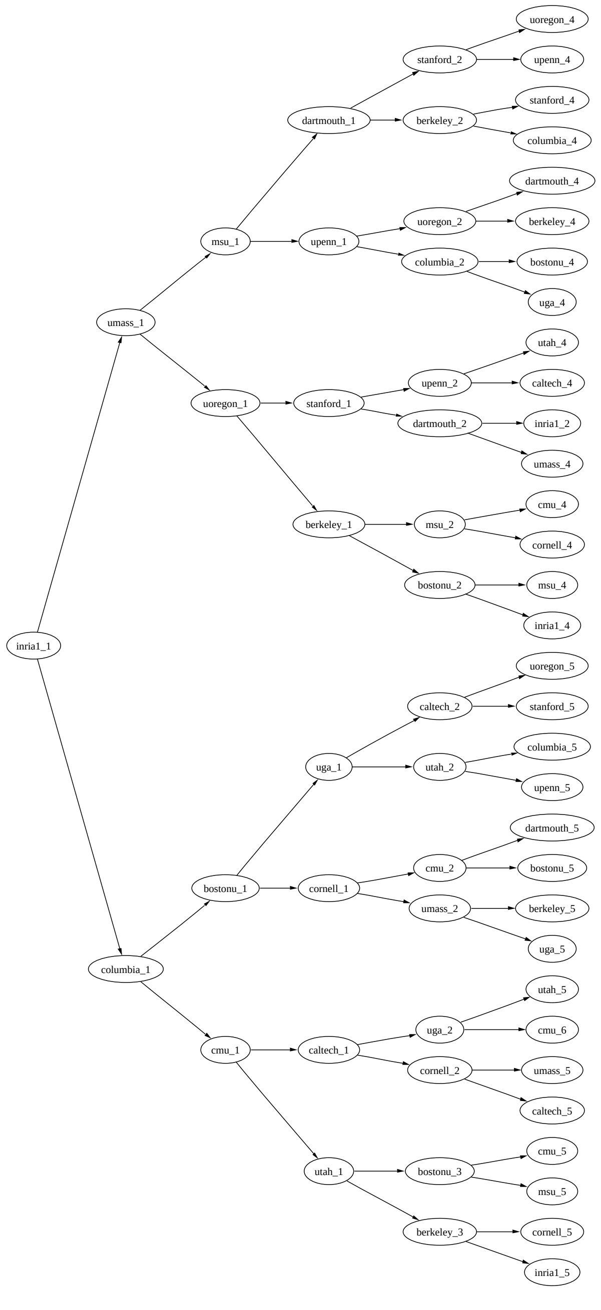


Figure 2.2: Multicast tree consisting of 64 PlanetLab nodes.

Table 2.1: Scalability Experiments in Planet-Lab: Throughput in Pkts/sec

Group size:	15	31	63
Buffer=50 Pkts	95	86	88
Buffer=100 Pkts	82	88	77
Buffer=1000 Pkts	87	95	93

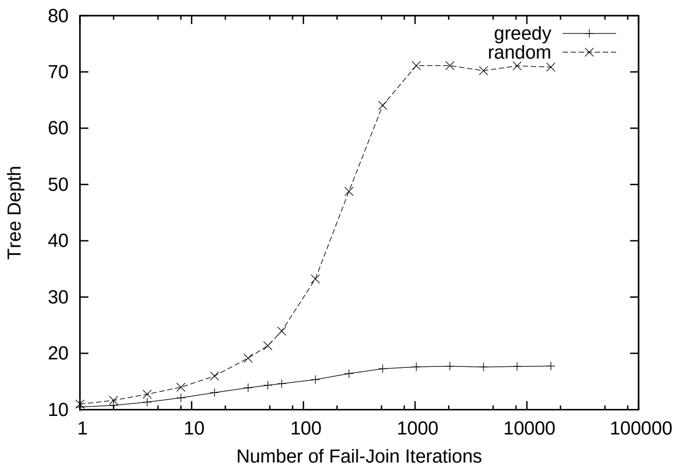
Reliability Analyses. To verify our approach to recovery after failures, we implemented a failure-resistant chain of 5 nodes running on Planet-Lab machines. During the transmission of 10 megabytes of data, two of 5 nodes fail. The failures are not simultaneous, and the system needs only to be resistant to one failure. In this experiment we limit both input and output buffer size to 50 packets. As in the previous experiment, size of each packet is 200 bytes (MSS=100 bytes). Our failure recovery algorithm needs a backup buffer of size 150 in this case. We have performed 10 runs of this experiment and measured group throughput, reconnection time and the number of redundant packets that are retransmitted after the connection is restored. Recall that in our architecture, the packet sequence numbers do not need to be advertised during the re-attachment procedure. Thus the child nodes of the failed node may receive duplicated packets after the connections are re-established. These redundant transmissions can impact the group throughput.

In our implementation, the failing node closes all its connections, and failure is detected by detecting dropped connections. After the failure is detected, the orphaned child node listens for an incoming connection from the surviving ancestor. We measure the interval between the time when failure is detected, and the time when connection is restored. This time interval is measured separately at the two participating nodes: surviving parent (P), and child (C). The results of our measurements are summarized in Table 2.2. The average reconnection time in seconds and number of retransmitted packets per one failure are given per one failure. The average group throughput is given per experiment. In these experiments, the average number of retransmitted packets is about half of the backup buffer size. The TCP sessions are re-established in a few seconds, in the same order as the TCP timeout. As the failure detection can be achieved in a few seconds as well, our experiment results show that the entire procedure of failure detection and reconnection can be completed in a few seconds.

Table 2.2: End-to-End Reliability Experiments in Planet-Lab

	min	average	max
Throughput (Pkts/sec)	49.05	55.24	57.65
# of Retransmitted Packets	34	80.5	122
Reconnection time (C)	0.12	3.53	5.2
Reconnection time (P)	0.27	3.81	5.37

Scalability vs. Reliability. Simulation results presented above have shown that when there are no failures, the larger the buffers the more scalable the group throughput is. However, with larger buffers, the backup buffer size has to be increased proportionally in order to guarantee the end-to-end reliability. The above experiment showed that when failures do occur, the redundant transmissions will be increased as a consequence of larger backup buffers. These redundant transmissions will in turn reduce the group throughput. We here investigate into this issue. We consider a chain of 10 nodes and we generate 2, 4 and 6 failures (in a sequential way, so that the system just need to tolerate 1 failure). Table 2.3 reports the throughput measurements obtained with these settings and with different output



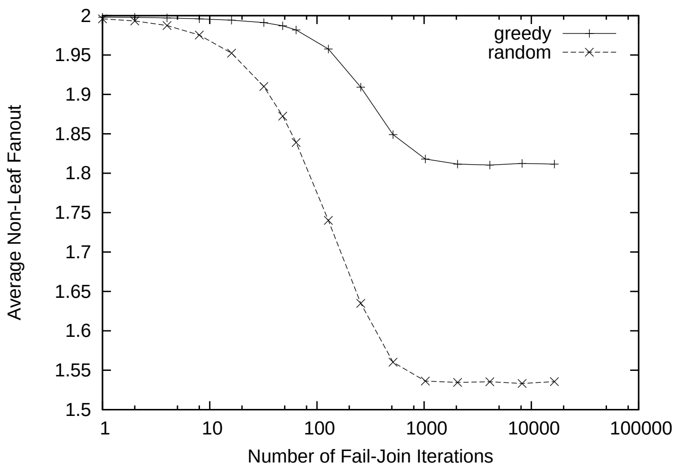


Figure 2.3: Evolution of tree depth (left) and evolution of average degree of non-leaf nodes (right).

buffer sizes. The backup buffer size is set to the input buffer size and twice the output buffer size. It is interesting to see that when the buffer sizes increase, the group throughput can actually decrease. While we cannot make general claims based on these observations alone, these experiments do show that the throughput monotonicity in buffer size no longer holds in the presence of failures. The more frequent the failures are, the more severe (negative) impact large buffers would have on the group throughput.

Table 2.3: Scalability vs. End-to-End Reliability. Throughput in KB/s

	buf=50	buf=200	buf=500	buf=1000
2 failures	25.6	26.8	45.2	31.5
4 failures	29.2	28.8	36.4	27.2
6 failures	30.9	28.8	30.8	24.0

2.4.2 Simulation of Tree Evolution

To complement the simulations and experiments presented above, we further developed a discrete-event simulator to simulate the evolution of tree topology with failures and recovery under different algorithms. In particular, we evaluate the heuristics presented in Section 2.3 for the tree reconstruction.

Starting with a balanced binary tree of 1023 nodes, we uniformly choose a failing node, apply random or greedy heuristic to restore connectivity, and add the node back using best-join. The tree must remain binary, joins are only allowed at nodes with out-degree less than 2. We measure the length of longest path and average degree of non-leaf nodes. The two methods used for restoring connectivity are GREEDY_RECONNECT and a randomized procedure, that reconnects orphaned subtrees to randomly chosen nodes with out-degree less than 2.

The results are presented in Figure 2.3. The plots show average tree depth and inner node fanout over 500 runs. We observe that GREEDY_RECONNECT helps to maintain significantly lower tree depth, and higher inner node degree, compared to the trivial approach that chooses active nodes randomly.

2.5 Conclusions

Our first conclusion is that reliable multicast overlays can be deployed on top of the current TCP/IP by adding a light set of application layer back-pressure mechanisms that guarantee both end-to-end flow control and reliability. A second important observation concerns the fear that as more and more TCP connections get interconnected in such a multicast overlay, some slow down experienced by distant connections might propagate to the root via back-pressure, leading the group throughput to vanish as the number of end-system grows. We have shown that such a fear has no grounds, provided all point to point connections that are used within the overlay offer minimal quality guarantees. Such architectures can be used for group communications of arbitrarily large sizes and still provide a group throughput that is close to that of a single point to point connection with these minimal guarantees.

Surprisingly, this conclusion holds true even in the case of moderate input and output buffers. Moderate buffers even seem to be a good tradeoff within this context: they allow more efficient recovery mechanisms in case of failures and, according to our simulation results, they do not affect too severely the group throughput if not too small. An optimal buffer size offering a good compromise between throughput and reliability could in principle be advertised to the group.

The next steps will consist in a more complete specification of the overlay architecture parameters.

Chapter 3

Minimum Radius Spanning Tree With Degree Constraints

3.1 Introduction

In simplest multicast scenario, a dedicated source host delivers information to a group of receiving hosts. Overlay multicast is implemented in application layer, and all the data is transmitted via unicast delivery supported in underlying network. Because of bandwidth limitations, it may not be possible to simultaneously send data from source to each receiving host via unicast. An implementation of overlay multicast uses receiving hosts to forward information to other receivers. If the data stream intensity does not change, it is natural to assume that each participating host has a fixed bound on the number of hosts it can communicate to. This kind of bandwidth capacity constraints translate into degree constraints on the nodes of the multicast tree. In this case, to initiate overlay multicast, one needs to construct a degree-constrained spanning tree in a complete graph, where the nodes correspond to the hosts, and the edges correspond to the unicast communication paths.

An important practical problem in this context is to construct a multicast tree, which minimizes the largest communication delay observed by receiving hosts during multicast. Various studies have been conducted with the primary focus on the protocol development for efficient overlay tree construction and maintenance, such as Narada [19], Yoid [25], ALMI [46], Host Multicast [64], NICE [9], Delauney graph [37]. Some other work in peer to peer network is also related to the tree construction in application level multicast, see e.g. Chord [59] and CAN [48]. Most of such studies have been experimental in nature, see e.g. [18, 19, 17, 33, 46, 20]. In particular, Chu et al [18] use a heuristic called *Bandwidth-Latency* to build the multicast overlay tree. This heuristic, described in more detail in [61], selects paths by choosing those with the greatest available bandwidth (i.e., maximum possible fanout).

We note that this tree construction problem corresponds to a graph-theoretic problem of constructing a rooted spanning tree of minimum radius with degree constraints. This problem is known in literature. The famous Travelling Salesman Problem [5] is a special case, but in general the degree-constrained spanning tree problem is harder than the TSP.

In [58], and later [57] and [54], the authors describe an NP-hard *minimum diameter, degree-limited spanning tree problem (MDDL)*, and propose heuristics for solving it. In the minimum-diameter version they consider, the objective is to minimize largest communication delay between any pair of participating nodes. However, the quality of heuristic solution observed in simulations described in [58] decreases, as the number of nodes increases.

In [39], Malouch *et al.* introduce the radius minimization version, where the distance to the root is minimized. The authors prove that the problem in general is NP-hard, and show that in the special case of unit node-to-node delays the problem can be solved optimally in polynomial time. For the case of general distances a set of heuristics is described.

Another approximation algorithm for the radius minimization problem was proposed in a recent work by Könemann, Levin and Singha [34]. The algorithm produces a spanning tree with diameter within factor of $O(\sqrt{\log n})$ for a graph of n nodes, if distances between nodes satisfy triangle inequality. However this worst-case bound, similarly to that derived in [58], increases with the size of the network.

In this chapter we assume that each node can be mapped to a point in Euclidean space, and node-to-node delays can be approximated by Euclidean distances between these points. Under this assumption, we describe an algorithm for constructing a degree-constrained spanning tree, and show that it arrives at asymptotically optimal solution. The asymptotic optimality result holds if points are uniformly distributed inside a convex region in Euclidean space, and at least 2 outgoing links are allowed at each node. This result easily extends to non-uniform distribution case, with the only requirement that density function is strictly more than some constant $\epsilon > 0$ inside the convex region, and is zero everywhere else.

The method of mapping hosts to points in Euclidean space, and delays to Euclidean distances, is often used in analysis of overlay networks. For example, [57] and [37] use geographical locations of computers to create a mapping of hosts to the two-dimensional plane. The advantage of this method is that no actual network delays need to be measured to construct the mapping, and subsequently the multicast tree. Another approach, proposed in work by Global Network Positioning group [44], achieves higher accuracy by measuring some of the delays, and mapping hosts into Euclidean spaces of dimension 3 and above. In our work, we assume that the mapping has already been done, for example, using one of the methods above. We will concentrate on constructing degree-constrained spanning tree with minimal radius.

Our approach to the problem is similar in spirit to that used for other routing problems in Euclidean space, for example vehicle routing and traveling salesman problem [5]. The region covering destination points is divided in smaller sub-regions (cells) with a regular grid, and then the problem is solved within these small sub-regions. The size of grid cell is then used to estimate worst-case length of the path contained in the cell. However, in our algorithm we are constructing a routing tree that satisfies degree constraints, rather than a path connecting the end points. Therefore, we use a grid and connection methods significantly different from those used in previously studied geometric problems.

We organize our presentation in four parts. First, we present a simple constant-factor approximation algorithm for solving the problem in Euclidean space. We use constant-factor algorithm as a subroutine of the asymptotically optimal algorithm, to connect points inside cells of a polar grid. Next, we describe our asymptotically optimal algorithm for the special case of out-degree at least 6 at each node, and points uniformly distributed in a two-dimensional disk, and prove asymptotic optimality. We follow by describing how to extend the algorithm to work in higher dimensions, with general degree constraints, and with general convex regions. Finally, we analyze algorithm performance using simulation.

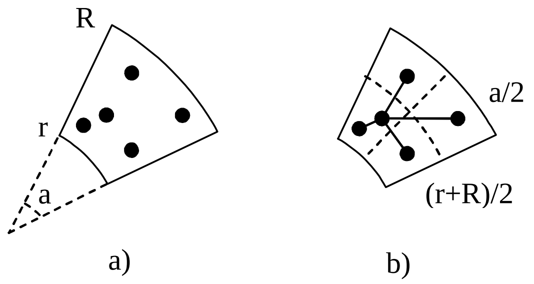


Figure 3.1: Constant factor approximation algorithm.

3.2 Constant Factor Approximation

Before we can describe the asymptotically optimal algorithm, which is the main focus of this chapter, we need to introduce a subroutine for connecting points within cells of a polar grid. This subroutine in itself is an approximation algorithm. It creates a valid degree-constrained spanning tree for a given set of points in Euclidean space. The length of the longest path in the tree is within a constant factor of the best solution among all the possible degree-constrained spanning trees. This constant approximation factor is independent of the number of points in the region. Although it is easy to describe a version of the algorithm for a square, we will describe a polar version, which is more suitable for this setting.

Consider ring segment shown on Figure 3.1 a), with inner radius r, outer radius R, and angle a. Suppose all the points are contained within this segment. Assume that the source point, which is the root of the tree, is also specified. The algorithm proceeds recursively as following:

Bisection Algorithm

1. Divide the segment into 4 sub-segments, by splitting it with an arc of radius $(R + r)/2$ and a ray dividing angle a into two halves.
2. Pick a representative point in each non-empty sub-segment, such that its' radius in polar coordinates is closest to the radius of the source node. Connect the source to all the representatives. See Figure 3.1 b).
3. Repeat the procedure within each non-empty sub-segment, to connect the points inside the sub-segment, using the representative point as a local source.

The algorithm constructs a spanning tree, in which each node has at most 4 children. We observe that each path always moves monotonously along the radius axis. The steps along the angle axis at each level can be bounded by the angle of the sub-segment. Therefore, the length of each path l_p can be bounded from above using the triangle inequality as follows:

$$\begin{aligned} l_p &\leq \max(R-q, q-r) + Ra + Ra/2 + Ra/4 + \ldots \leq \\ &\leq \max(R-q, q-r) + 2Ra, \end{aligned} \tag{3.1}$$

where q is the radius of the source node.

We will now show that this algorithm can be used to construct a constant factor approximation for a given set of nodes. We first construct a ring segment to cover all these points. We pick the center of the ring to be very far, so that the angle a is small, ($\sin a > 5/6a$), and both R and r are large, such that $r > 0.6R$. Pick R and r such that $R - r$ can not be reduced, without leaving some nodes out of the ring segment. Similarly, assume that a can not be reduced. Then, since any path must connect to extreme nodes, and using triangle inequality, it is easy to see, that for the optimal longest path OPT the following holds:

$$OPT \geq \max(R - q, q - r),$$

$$OPT \geq r \sin a \geq {}^1\!/_2 Ra.$$

Combining this with (3.1), for any tree path p, we obtain:

$$l_p \leq 5 \cdot OPT,$$

and therefore this algorithm can be used to produce solutions within constant factor of the best possible.

It is not difficult to modify the algorithm to produce a spanning tree with out-degree 2. To do this, during each recursive call, connect the source to two points from the same segment. Points should be chosen to have radius closest to the source. Then each of the two points can be used to connect 2 of the 4 sub-segments, so that all sub-segments are connected. In this case, upper bound on the solution doubles the angle term, since on each level of the path we now use 2 links instead of one:

$$l_p \leq \max(R - q, q - r) + 4Ra. \tag{3.2}$$

Theorem 3.2.1. *The Bisection Algorithm provides a solution within a factor of 5 optimal for the minimum radius problem when maximum out-degree is restricted to be 4. The approximation factor becomes 9 if the maximum out-degree is restricted to be 2.*

3.3 Asymptotically Optimal Algorithm

In this section we describe our hierarchical algorithm to recursively build multicast trees. We will prove that it is asymptotically optimal. To simplify the presentation, we will make several assumptions. These assumptions will be lifted in the next section.

We assume that the n points corresponding to the communicating hosts are uniformly distributed inside a disk of radius 1 and the source is located at the center of the disk. We assume each node can forward transmission to at least 6 down-stream links. The main idea of the algorithm is to divide the disk into a hierarchy of smaller and smaller grid cells. The algorithm builds a tree based on the hierarchy to connect the points in the grid cells. At high level, our grid partitioning algorithm proceeds in three stages:

Algorithm Polar_Grid

1. Creates a grid of equal area cells, covering the disk.
2. Connects the cells, using cell representatives, and forming a core network.
3. Connects points within the cells, using the constant factor approximation algorithm.

We next describe the details of each step of the algorithm in the following subsections. We then evaluate the performance of the algorithm and prove the asymptotic optimality results.

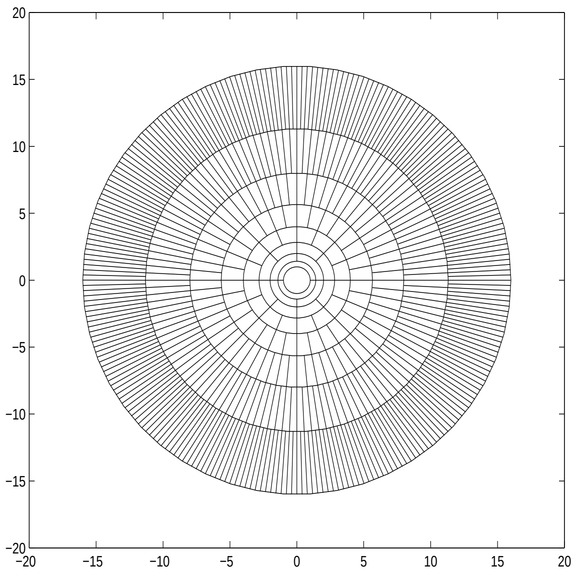


Figure 3.2: Dividing the disk into ring segments of equal area.

3.3.1 Constructing the polar grid.

First, the algorithm creates a polar grid covering the unit disk. This grid must have the following properties:

1. All cells of the grid have the same area.
2. Cells are organized in rings. Each containing ring has twice more cells than the ring immediately inside it.
3. There is at least one point in each cell of the grid, except for the cells in the outermost ring.

For a fixed number of rings k, we construct the grid by dividing the unit disk using k circles with the same center, and radius

$$r_i = 1/\sqrt{2}^{k-i}, \quad 0 \leq i \leq k-1. \tag{3.3}$$

We further divide each ring i into 2^i equal segments, such that each cell segment on level i is aligned with 2 segments on level $i+1$ (see Figure 3.2).

Since the radius of ring i is chosen such that $r_i = \sqrt{2}r_{i-1}$, the area of disk bounded by circle i is twice the area of disk bounded by circle $(i-1)$. If we imagine that there are two cells inside circle 0, then it is easy to see that for each i, circle i contains twice more cells than circle $(i-1)$, and therefore property 1) holds.

Given a set of points, we can choose the number of rings k as large as possible, such that property 3) is satisfied. In the analysis section we will show that k increases as the number of points n increases.

3.3.2 Connecting the cells.

According to property 3) of the grid, each ring segment contains at least one point (except for the outermost segments). We can choose a point within each segment to be the representative of the segment. If there is more than one point in a segment, choose the point that is closest to the center on the inner arc of the segment. Cell representatives are connected in a binary tree, rooted at the source node in the center of the unit disk. Each representative is connected to two representatives of next ring cells, aligned with its cell. The outermost ring cells that do not have any points are ignored.

3.3.3 Connecting remaining points within cells.

Finally, in each cell that contains more than one point, we run the constant factor algorithm described in the previous section. The algorithm connects all the remaining points, and the distribution tree is completely constructed.

The constant factor algorithm requires out-degree 4, and additional out-degree 2 can be used at the representative node to connect to next level cells, and therefore the resulting spanning tree will have maximum out-degree 6. We will improve on this estimate in the next section.

3.3.4 Lemmas.

In order to prove asymptotic optimality of the solution, we need to show that k increases as a function of number of nodes n. To do this, we need to introduce the following two lemmas.

Lemma 3.3.1. *If each of n balls is uniformly and independently assigned to one of n^α buckets (for some fixed α), the probability $p_\alpha(n)$ of having at least one empty bucket, after assignment is complete, satisfies*

$$p_\alpha(n) \le n^\alpha e^{-n^{1-\alpha}}. \tag{3.4}$$

Proof. The probability of having at least one bucket empty is bounded from above by the sum of probabilities of having each of the buckets empty. Therefore,

$$p_\alpha(n) \le n^\alpha \left(1 - \frac{1}{n^\alpha}\right)^n.$$

Note that $1 - x \le e^{-x}$ for any x, and inequality (3.4) follows. □

Corollary below immediately follows from the lemma.

Corollary 3.3.2. *If $\alpha < 1$, then $p_\alpha(n) \to 0$ as $n \to \infty$.*

Since we are interested in deriving an asymptotic result, Corollary 3.3.2 would suffice for our analysis. However we would like to know the value of α that can gives meaningful results even for small n. The following lemma gives insight into possible values of of α.

Lemma 3.3.3. *If $\alpha \le {}^1/_2$, then $p_\alpha(n) \le e^{-1}$ for all $n \ge 1$.*

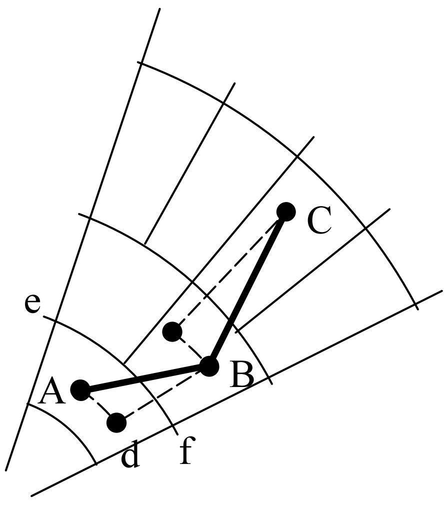


Figure 3.3: Proof of upper bound on longest path.

Proof. Consider $f_\alpha(x) = x^\alpha e^{-x^{1-\alpha}}$. Assume that $0 < \alpha < 1$ and $x \geq 0$. Observe that in this case $f_\alpha(x)$ is a concave function of x. By taking derivative, we can show that it reaches its maximum at

$$x^*_\alpha = \left(\frac{\alpha}{1-\alpha}\right)^{1/(1-\alpha)}.$$

Notice that x^*_α is increasing in α and $x^*_{1/2} = 1$. Therefore if $\alpha \leq {}^1\!/_2$, the maximum is attained at some $x^*_\alpha \leq 1$, and hence for $x \geq 1$ function $f_\alpha(x)$ is non-increasing. Furthermore, for any α, $f_\alpha(1) = e^{-1}$. The lemma follows from equation (3.4), i.e. $p_\alpha(n) \leq f_\alpha(n)$. $\square$

Therefore, since in k-ring grid there are 2^{k+1} cells, with high probability we can say that if we require at least one point in each cell,

$$\sqrt{n} \leq 2^{k+1},$$

and therefore,

$$k \geq {}^1\!/_2 \log_2 n. \tag{3.5}$$

In our analysis we will assume that n is sufficiently large, and $k \geq 1$.

3.3.5 Solution analysis.

We can now evaluate solution quality based on the uniform distribution assumptions. It's easy to see that, as the number of nodes n increases, the lower bound of the optimal solution cost (the longest distance from disk center to any point) approaches 1 from below. To complete the proof, we need to show that an upper bound on solution obtained by the algorithm approaches 1 from above.

Any path P in the constructed spanning tree consists of two parts: the sub-path p connecting cell representatives, and the sub-path q between the points in the last cell, constructed by the constant factor approximation algorithm:

$$l_P = l_p + l_q.$$

Making use of (3.1), we can write

$$l_q \leq \max(R - q, q - r) + 2Ra,$$

for some R, r, a, q, defined by the last cell of path P.

Using polar version of the triangle inequality, the length of the path can be bounded from above by computing radius and arc components separately. The upper bound path follows cell boundaries. For example, in Figure 3.3, the length of AB is less than $Ad + dB$, and arc Ad can be upper-bounded by arc ef. The total length of all the ray segments (similar to dB) is at most 1 – the radius of the disk. The $\max(R - q, q - r)$ component of l_q can be included in this estimate as well, since we pick least-radius point to be cell representative.

Thus,

$$l_P \leq 1 + 2Ra + S_k, \tag{3.6}$$

where S_k is the sum of arc lengths for inner $(k - 1)$ circles of k-ring grid.

Let Δ_i be the length of an cell-side arc of circle i:

$$\Delta_i = 2\pi \frac{1}{\sqrt{2}^{k-i}} \cdot \frac{1}{2^i} = \frac{2\pi}{\sqrt{2}^{k+i}}, \quad 0 \leq i \leq k.$$

In our estimate of S_k only the inner arcs are involved, i.e. arcs 1 through k-1. Hence,

$$S_k = \sum_{i=1}^{k-1} \Delta_i = \frac{2\pi}{\sqrt{2}^{k+1}} \cdot \frac{1 - 1/\sqrt{2}^{k-1}}{1 - 1/\sqrt{2}}.$$

Recall that Ra in (3.6) is an arc length as well, for some ring j:

$$Ra \leq \Delta_j.$$

We can rewrite (3.6) as following:

$$l_P \leq 1 + 2Ra + S_k \leq 1 + 2\Delta_j + S_k. \tag{3.7}$$

We can show that the right-hand side of inequality (3.7) approaches 1 from above, as n approaches infinity. Here's the precise argument. Both Δ_j and S_k are infinitesimal as k goes to infinity. For any arbitrary small $\epsilon > 0$, there exists a K such that when $k > K$, the quality of the solution l_p is less than $1 + \epsilon/2$. Based on Corollary 3.3.2, for any arbitrary small $\delta > 0$, there exists an N_1, such that when $n > N_1$, the probability of having at least one point in each cell is larger than $1 - \delta/2$. It is also easy to show that there exists an N_2, such that when $n > N_2$, the probability of having a point in the ring between the circle of radius $1 - \epsilon/2$ and the unit circle is larger than $1 - \delta/2$. This implies the minimum radius is at least $1 - \epsilon/2$. Therefore, with probability at least $1 - \delta$, when $n > \max\{N_1, N_2\}$, the minimum radius is at least $1 - \epsilon/2$, and at the same time there is at least one point in each of the grid cells, which implies $l_p < 1 + \epsilon/2$. Under this condition, the length of the longest path in this tree is within ϵ plus the value of the optimal solution. This completes the proof for the asymptotic optimality of Algorithm Polar_Grid:

Theorem 3.3.4. *For any small $\epsilon, \delta > 0$, there exists an N such that when the number of points n is larger than N, with probability greater than $1 - \delta$, the length of the longest path in the tree produced by Algorithm Polar_Grid is within ϵ plus the optimal solution.*

3.4 Generalization

In the previous section, in order to simplify presentation, we assumed that points are uniformly distributed inside a disk, the out-degree of at least 6 is allowed, and the points belong to a two-dimensional space. All these assumptions are not essential for the end result. The algorithm can be adjusted accordingly to remove these constraints. We describe the necessary changes in this section.

3.4.1 Out-Degree 2

There is a version of the asymptotically optimal algorithm, in which at least out-degree 2 must be allowed at every node. In other words, it is possible to construct a binary tree with the same asymptotic optimality property.

We have discussed how to adjust the constant factor approximation algorithm in Section 3.2. A few changes have to be made to the algorithm that chooses and connects cell representatives. In each cell, three cases are possible:

1. There is only one point in the cell. Make it a cell representative, and use it to connect to the two cells in the next ring.
2. There are two points in the cell. Choose a point closest to the center of the disk as the cell representative. Connect the representative directly to the other point. Then connect the second point to the two cells in the next ring.
3. There are three or more points in the cell. Choose the closest to center point as the cell representative. Pick one of the remaining cells to be the center for connecting other points in the cell. Choose another point for connecting cells in the next ring. The two special points are connected directly to the representative point.

The asymptotic analysis and the constant factor approximation analysis are very similar. The only difference is that the contributions from the arcs need to be doubled. This is because now, two links are used in each cell, instead of one link. Since this contribution is infinitesimal for large n, the constant multiplier can be ignored, and the same proof holds.

3.4.2 Higher Dimensions

The algorithm can be adjusted to work in dimensions higher than two. The most important component of the proof is the creation of the polar grid, which satisfies properties 1)-3). The grid can be created similarly, in polar coordinates, by splitting d-dimensional sphere into segments. Radius of each subsequent ring should equal to previous ring radius, multiplied by $\sqrt[d]{2}$ (so it has double volume), and each cell is split into two, with alternating splitting axis. Although details of equal volume split become tedious, a similar proof can be constructed.

Nodes	Rings	Core	Delay	Dev	Bound	CPU Sec
100	3.61	1.53	1.852	0.20	7.18	0.002
500	5.26	1.22	1.420	0.08	4.92	0.01
1,000	6.06	1.13	1.302	0.05	4.09	0.02
5,000	8.01	1.00	1.142	0.02	2.65	0.08
10,000	8.97	0.99	1.102	0.02	2.20	0.17
50,000	11.00	0.94	1.049	0.01	1.61	0.96
100,000	11.98	0.95	1.034	0.00	1.43	2.01
500,000	14.00	0.92	1.016	0.00	1.22	11.06
1,000,000	15.00	0.93	1.012	0.00	1.15	22.99
5,000,000	17.00	0.91	1.005	0.00	1.08	132.34

Table 3.1: Experiment Results: Trees of Out-Degree 6.

Nodes	Rings	Core	Delay	Dev	Bound	CPU Sec
100	3.61	2.21	2.634	0.31	10.74	0.0015
500	5.26	1.61	1.876	0.15	6.96	0.01
1,000	6.06	1.40	1.622	0.11	5.66	0.02
5,000	8.01	1.12	1.285	0.04	3.44	0.08
10,000	8.97	1.06	1.202	0.03	2.76	0.17
50,000	11.00	0.98	1.095	0.01	1.88	1.02
100,000	11.98	0.97	1.067	0.01	1.63	2.13
500,000	14.00	0.93	1.031	0.00	1.32	11.84
1,000,000	15.00	0.94	1.022	0.00	1.22	24.52
5,000,000	17.00	0.91	1.009	0.00	1.11	142.08

Table 3.2: Experiment Results: Trees of Out-Degree 2.

3.4.3 General Convex Region

Proving asymptotic optimality for a circle (sphere), with source in the center, implies asymptotic optimality in any convex region with arbitrary source placement inside the region. The algorithm constructs the smallest ring covering all points and centered at the source, and proceeds similarly as the circle case. The analysis is very similar. In this case, the lower bound on longest path approaches the outer ring radius from below.

3.5 Experiments

In this section, we provide some experimental results to illustrate the quality, running time and other properties of our heuristic algorithms, for problems of different sizes. For each size of the problem, we have generated 200 random sets of points, uniformly distributed inside the unit disk. We computed the average maximum delay and other parameters of solution trees. We have tested both the out-degree 6 and out-degree 2 versions of the algorithm. We have also evaluated performance of three-dimensional version of the algorithm for connecting points uniformly distributed inside unit sphere. We used an Intel Pentium II 400 Mhz computer with 128 megabytes of RAM to run our experiments.

All the data obtained in our experiments on unit disk is shown in Tables 3.5 and 3.5. Column one contains n, the number of nodes to be connected. The Second column, “Rings”, is the average value of k – the number of rings for this problem size. Column 3, “Core”, contains the average *core delay* – the longest portion of the path between cell representative nodes. Column 4, “Delay”, shows the average longest delay observed in the solution tree. Column 5, “Dev”, displays the standard deviation of the longest delay. The lower bound on the delay is close to 1, so the closer delay is to 1, the better. “Bound” columns show the value of the upper bound given by equation (3.7), evaluated at $j = 0$. The reason to pick $j = 0$ is because $\Delta_0 \geq \Delta_j$ for all j. In the formula for upper bound, the coefficient of Δ_j should be doubled for out-degree 2 trees. Finally, the “CPU Sec” column contains the computation times.

To illustrate our results, we have included a set of plots, based on data shown in Tables 3.5,3.5. The results demonstrate that the algorithm converges very quickly.

Figure 3.4 shows the maximum sender-to-receiver delay, together with the delay bound and the core delay. The horizontal axis representing the number of nodes, is in logarithmic scale. This is the case for this plot, as well as for plots 3.5 and 3.6. The bound used in the analysis of the algorithms significantly over-estimates the delay for problems with a small number of nodes. This bound becomes better and better as the number of nodes increases. The difference between the core and the total delay is not reducing. This is because the difference depends on the radius of the outermost ring, which remains constant as the number of nodes increases.

Figure 3.5 combines the plots on Figure 3.4, and compares the maximum delay for degree 2 and degree 6. The delay overhead of degree 2 trees is almost 2 times the overhead of degree 6 trees. This is intuitive, since there is the same relationship between the bounds on the lengths of the paths. As the number of nodes increases, the degree of each particular node becomes less and less important, and the two curves all converge to the best possible delay of one.

Figure 3.6 shows how the number of rings, k, in the grid created by the algorithm changes with the number of nodes, n. The node axis is again in logarithmic scale. The points follow almost a straight line. This indicates that there is a logarithmic dependence, which is implied by (3.5).

Figure 3.7 shows how running time of the program increases with the increase in the number of nodes. The small insert plot shows the details for problems with nodes between 100 and 10,000. Although most likely our straightforward implementation of the algorithm can be improved, and in practice running time will depend on hardware and software environment used, the plot allows us to evaluate the general trend. In our experiments we observed that running time increases almost linearly, which makes it possible to run the algorithm for networks with very large sizes.

Indeed, during the assignment of points to cells of the grid our algorithm inspects each point only once, which requires $O(n)$ operations. Then, the bisection algorithm has to divide ring segments, and enumerate points within each segment. For m points submitted to the bisection algorithm, it will create at most m non-empty segments, and in the worst case the number of operations at this stage can be estimated as $O(m^2)$, since each point may be inspected during processing of each segment. But since the distribution of points is uniform, with high probability total running time of our algorithm will be linear in n, which can be intuitively explained by the following argument. Since points are distributed uniformly between cells, the average number of points in each cell is $1/2^k$. Our experiments confirm that the relationship between k and n stated in (3.5) holds, and k is a logarithmic function of n (see Figure 3.6). Because of this relationship, the number of points per cell on average remains constant, independent of n, and running time of bisection in each cell is also a constant. Since we require at least one point to be contained in each cell, the total number of cells, and therefore the

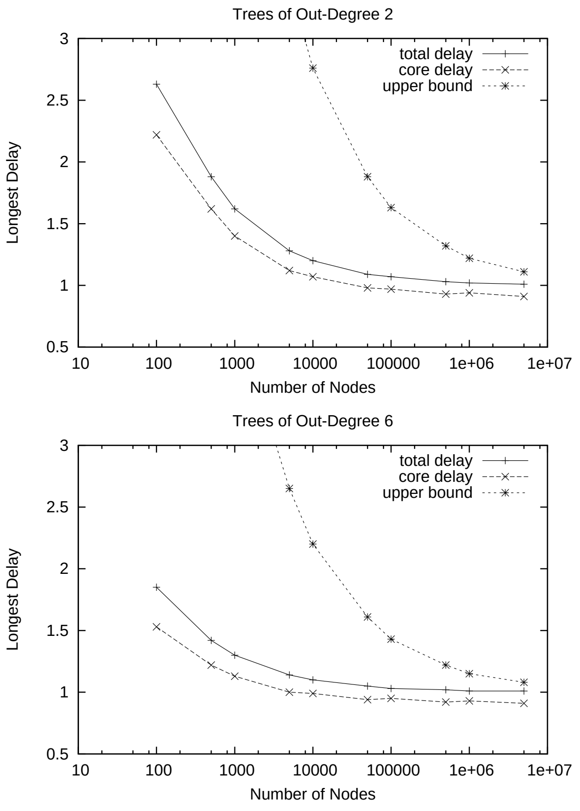


Figure 3.4: Average maximum delay compared to bounds.

total number of calls to bisection procedure is at most $O(n)$, leading to overall number of operations $O(n)$.

Finally, on Figure 3.8 we demonstrate algorithm convergence results in three-dimensional unit sphere. Similarly to unit disk case, we run 200 experiments for each problem size, and computed average longest path length. In three dimensions, straightforward extension of our algorithm builds a tree of out-degree 10, in which each cell representative node uses 2 links to connect to cells in the next ring, and at most 8 links are used to connect to points inside the cell, using bisection algorithm. As in two dimensions, we modify the algorithm to construct trees of out-degree no more than 2. In both cases, the longest path length converges to the lower bound of 1.

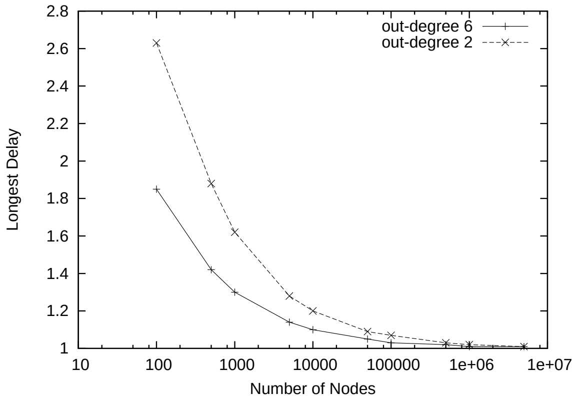


Figure 3.5: Comparison of average maximum delay for out-degrees 2 and 6.

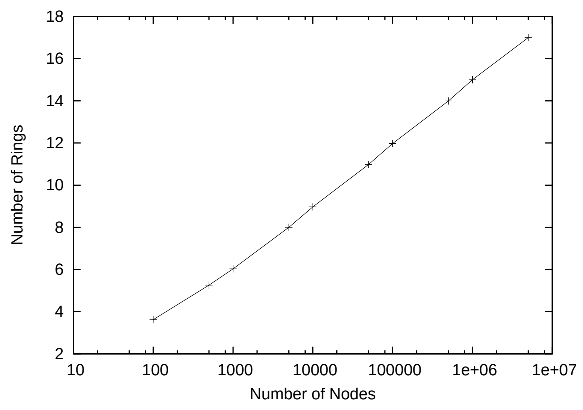


Figure 3.6: Average number of rings in polar grid.

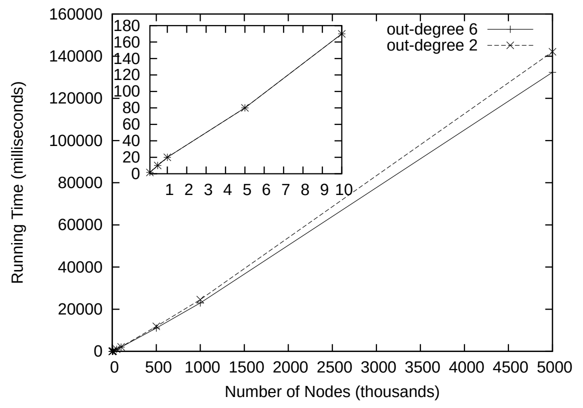


Figure 3.7: Algorithm running time.

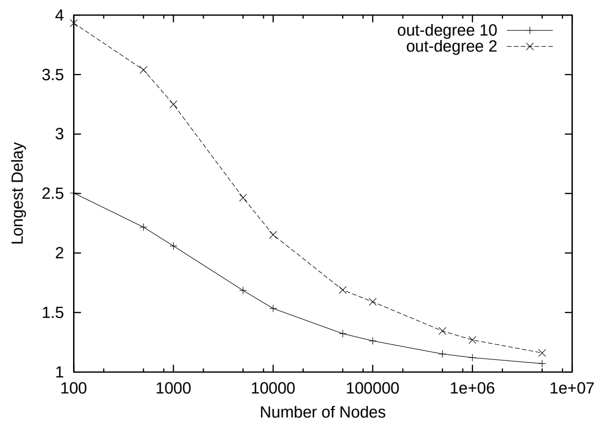


Figure 3.8: Average maximum delay in three-dimensional unit sphere.

Similarly to longest path results on unit disk, shown on Figure 3.5, in three dimensions the difference between out-degree 2 and out-degree 10 trees becomes less noticeable as the number of nodes increases. Although asymptotic optimality holds in any multi-dimensional Euclidean space, Figure 3.8 shows that for the same number of nodes largest delay in 3 dimensions is higher, than in 2 dimensions. This can be explained by the increase in the average distance between uniformly distributed points, as dimensionality of unit sphere increases and number of points remains constant.

3.6 Conclusion

We investigate the problem of constructing an overlay multicast tree, that minimizes largest sender-to-receiver delay, and satisfies bandwidth constraints by limiting out-degree of nodes in the tree. We approach the problem by creating a mapping of communicating hosts to points multi-dimensional Euclidean space, that approximates unicast communication delays between hosts to distances between points, using methods described in [44] or [57] and [37].

In this setting, we describe a simple bisection algorithm, that constructs a tree with maximum delay within a constant factor of optimal for any set of nodes. Next, we assume that communicating points are randomly distributed inside two-dimensional disk, centered around the sender. We describe another approximation algorithm, which uses the bisection method as a subroutine in each cell of a polar grid, and prove that it creates a tree with maximum delay that asymptotically approaches the best possible, as the number of nodes increases, and with maximum out-degree 6. This result implies that as the number of communicating hosts grows, it becomes possible to construct better and better overlay multicast trees. In particular, for many remote leaf nodes communication delay from the source will decrease.

Next, we describe how to extend our grid-based algorithm to construct trees of out-degree no more than two, to work with points in more than two dimensions, and to work with general convex regions, not limited to spheres. We show that asymptotic optimality is preserved during all modifications we make.

Finally, we implement the algorithm, and perform simulation experiments to analyze its performance. Experiment results show that in practice the algorithm converges even faster, than it is predicted the theoretic bound we derived. Further, the experiments confirm that running time of the algorithm grows almost linearly with the number of nodes, and the algorithm is reasonably scalable.

Our algorithm can also be applied to the minimum diameter version of the problem, described in [58] and other papers. The bisection algorithm provides a larger factor approximation for the minimum diameter problem. But asymptotic optimality of our algorithm can not be guaranteed for any convex region in Euclidean space, although the result still holds for points uniformly distributed in a sphere. To construct an optimal solution in a sphere, an artificial root node should be chosen among nodes closest to the sphere center. In general convex regions the algorithm will only find a tree with delay within factor of 2 of the optimal as the number of nodes becomes large.

Since for all mapping methods there is usually a discrepancy between Euclidean distances and actual transmission delays, it is interesting to see how well the algorithm performs in practice. We leave this for future work. Also we note that in practice there is an interest in a decentralized version of the algorithm.

Chapter 4

Throughput Maximization in Overlay Multicast

4.1 Introduction

In many applications it is necessary to construct an overlay multicast network which allows to send a stream of data from a specified source node to all other participating nodes with maximum throughput. Any node can send data to any other node, but throughput for each pair of nodes is limited by the throughput of the path in the network connecting the two nodes. In many cases, however, the structure of the network is unknown. There are several strategies for simplifying the constraints. In our model we assume there are two main factors affecting throughput between nodes of the system. There is a network link throughput for each pair of nodes, that constitutes a bound on any communication between these nodes. There is also an uplink throughput bound for each node, imposed by last-mile connection of the node to the network. We will also refer to it as node capacity. Uplink throughput is shared by all outgoing, and possible incoming transmissions. We assume that there is no bandwidth sharing network links, and the maximum capacity of the link can be achieved by any stream using the link. Maximizing throughput in this model is an NP-hard problem. We derive approximability bounds, develop approximation algorithms and numerical methods for solving this problem. The model and outline of one of approximation algorithms are also described in paper by Baccelli, Chaintreau, Liu, Riabov and Sahu [7].

Our results on approximability bounds and approximation algorithms are summarized in the following table (where c_e denotes link capacity, and c_v denotes node capacity):

Problem	Approximability Bound	Approximation Factor
Duplex Channel	1/2	2/3
Separate Channel	1/2	$\min\left\{1/2,\ (\min_v c_v) / \left(\min_u \max_{e\in\delta(u)} c_e\right)\right\}$
Restricted Separate Channel	1/2	1/2

The rest of this chapter is organized as follows. We start our analysis in Section 4.2 by developing an approach for solving a sub-problem of finding a degree-constrained spanning tree with non-uniform degree constraints. In Sections 4.3 and 4.4 we describe duplex channel and separate channel variants of throughput maximization problem, derive approximability bounds and describe approximation algorithms. In Section 4.5 we describe methods for computing upper bounds on achievable throughput. In Section 4.6 we describe methods for solving the separate channel version of problem numerically using integer programming and a modification of volume algorithm. We present results of our numeric experiments in Section 4.7. Finally, we summarize our results and outline directions for future work in Section 4.8.

4.2 Degree-Constrained Spanning Tree Problem

We will start our analysis of maximum throughput problem by describing methods for solving the problem of finding a spanning tree satisfying degree constraints in a general graph. More formally, given an undirected graph $G = (V, E)$ one must find a spanning tree T such that for each vertex $v \in V$ degree of v in T is at most b_v, or prove that such a tree does not exist. Degree bounds b_v are positive integers, which are given as a part of problem instance. We will refer to this problem as *Degree-Constrained Spanning Tree (DCST).* Note that if we have a method for solving DCST, we can use it to verify whether a fixed throughput value θ can be achieved in our multicast system, and to build a tree that achieves this throughput. We will see in the following sections that there are only polynomially many possible boundary throughput values, and therefore an efficient algorithm for DSCT can easily be used for solving throughput maximization problem.

DCST is NP-hard (Hamiltonian Path is a special case), and therefore finding an approximate solution is the best one could hope for. We are not aware of any literature that tackles this problem in its pure form. However, the problem of finding a minimum spanning tree, which satisfies degree constraints (*Degree-Constrained Minimum Spanning Tree, DCMST*), has been analyzed by Caccetta and Hill [15]. Naturally, minimizing sum of edge weights in a tree subject to degree constraints is a harder problem than simply finding any tree which satisfies the constraints. Caccetta and Hill propose a set of heuristics and branch-and-cut algorithm for solving DCMST. We will return to their results in the section dedicated to solving the problem numerically.

The problem of finding *Minimum Degree Spanning Tree (MDST)* is closely related to DCST. In particular, if the degree constraints are uniform, i.e. all $b_v = b$ for some fixed positive integer $b \geq 2$, it is enough to solve MDST in graph G and compare maximum degree of the solution to uniform bound b to decide whether there exist a solution satisfying this bound or not. The same solution tree represents the solution to DCST if the bound is satisfied, otherwise DCST is infeasible. Fürer and Raghavachari developed the classic $\Delta^* + 1$ approximation algorithm for MDST [26, 30]. The algorithm finds a tree which has largest degree at most one more than largest degree in optimal tree Δ^*.

In this section we extend analysis by Fürer and Raghavachari and describe an algorithm for approximately solving DCST with non-uniform degree constraints. Our algorithm either constructs a tree which violates degree constraint at each node by at most 1, or proves that it is not possible to satisfy the constraints.

4.2.1 Witness Set Theorem

To assist the analysis of algorithm performance, we prove the following theorem that allows us to establish a bound maximum difference of node degree in a spanning tree from optimal under special conditions. This theorem is based on a generalization of the witness set analysis developed in [26].

Suppose we are given a graph $G = (V, E)$. Further suppose there exists a spanning tree T^* in G, satisfying degree constraints $\{b_v\}$. Assume we have constructed another spanning tree T. Let d_v be the degree of node v in T. Under these conditions, the following theorem holds:

Theorem 4.2.1. *Let S and B be two subsets of V, $S \cap B = \emptyset, S \neq \emptyset$, such that for some $k \geq 0$:*

$$\forall v \in S : \quad d_v = b_v + k + 1,$$

$$\forall v \in B : \quad d_v = b_v + k.$$

Let $\mathcal{F}$ be the forest generated from T by removing all nodes in $S \cup B$. Then, if there are no edges of G between different trees in $\mathcal{F}$, it follows that the only possible value of k is $k = 0$.

Proof. The theorem requires that $k \geq 0$, we will prove that $k \leq 0$. The total degree of nodes $S \cup B$ in T is

$$\sum_{v \in S \cup B} d_v = \sum_{v \in S \cup B} b_v + |S|(k+1) + |B|k.$$

Since T is acyclic, it has at most $|S \cup B| - 1$ edges with both endpoints in $|S \cup B|$. Therefore the total number of edges of T with at least one end in $S \cup B$ is at least

$$\sum_{v \in S \cup B} b_v + |S|(k) + |B|(k-1) + 1.$$

Each tree of $\mathcal{F}$ must be connected with exactly one edge to $|S \cup B|$, and therefore the number of edges with only one endpoint in $S \cup B$ gives us a lower bound on the number of trees $|\mathcal{F}|$:

$$|\mathcal{F}| \geq \sum_{v \in S \cup B} b_v + |S|(k-1) + |B|(k-2) + 2.$$

The total number of edges in any spanning tree connecting $|\mathcal{F}|$ sets of nodes and each node in $|S \cup B|$ is at least $|\mathcal{F}| + |S \cup B| - 1$. Each of these edges has to have at least one endpoint in $S \cup B$, since there are no edges between nodes in different trees of $\mathcal{F}$. Therefore, in any spanning tree the sum of degrees of nodes in $S \cup B$ has to be at least

$$|\mathcal{F}| + |S \cup B| - 1 \geq \sum_{v \in S \cup B} b_v + |S|k + |B|(k-1) + 1.$$

This includes the tree T^*, and therefore the following inequality must hold:

$$\sum_{v \in S \cup B} b_v + |S|k + |B|(k-1) + 1 \leq \sum_{v \in S \cup B} b_v$$

Since $S \neq \emptyset$ this inequality can only be satisfied if $k \leq 0$. □

Note that if all conditions of the theorem are satisfied for some $k \geq 1$, node set $S \cup B$ can be used to prove that there does not exist a spanning tree satisfying degree constraints $\{b_v\}$. Following terminology used in [26], we will call set $S \cup B$ a *witness set.*

Algorithm DCST_APPROX

1. Input: Graph $G = (V, E)$, spanning tree T, degree constraints $\{b_v\}_{v \in V}$.

   Let $h(v) = \emptyset$ for every vertex v.
   Let d_v be the degree of vertex v in tree T.
   Let $k = \max_{v \in V}(d_v - b_v) - 1$. If $k < 0$, output T and exit (T solves DCST problem exactly).

   Color vertices of graph G according to the amount of constraint violation:

   - v is colored *red*, if $(d_v - b_v) = k + 1$;
   - v is colored *yellow*, if $(d_v - b_v) = k$;
   - v is colored *green* otherwise.

   Let $\mathcal{F}$ be the set of connected components formed from T by removing *red* and *yellow* nodes.

2. Find an edge $e = (i, j) \in E$ between *green* nodes i and j belonging to different components of $\mathcal{F}$.

   If no such edge exists, output T and exit. If $k > 0$, The set S of *red* nodes and the set B of *yellow* nodes and Theorem 4.2.1 can be used to prove that the DCST problem is infeasible.

3. Consider the unique path P between i and j in T. If there is a *red* node v on path P, run EDGE_EXCHANGE(T, h, v, e). End of iteration: repeat starting from Step 1.

4. Else, for every *yellow* node v on path P change color to *green* and assign $h(v) \leftarrow \{e\}$. Combine components corresponding to i and j in $\mathcal{F}$. Go to Step 2.

Figure 4.1: DCST_APPROX: 2-approximation algorithm.

4.2.2 Approximation Algorithm for DCST

Out approximation algorithm (DCST_APPROX, Figure 4.1) is very similar to the algorithm in [26]. It starts with any spanning tree T in graph G. Any simple algorithm, such as Kruskal's algorithm for constructing minimum spanning tree [35] with unit weights can be used. Algorithm proceeds in iterations. Each iteration produces a valid spanning tree. During each iteration the tree is modified such that the number of nodes with maximum degree constraint violation is reduced by one. The algorithm stops, when the number of nodes with maximum violation can not be reduced, and either produces a witness set proving that the degree constraints cannot be satisfied, or Theorem 4.2.1 can be used to prove that the tree produced by the algorithm violates degree constraint at each node by at most one.

During each iteration the algorithm makes one or more edge exchange operations, adding a new edge to tree T and removing one of the edges to break the loop. If new edge and the edge being removed do not share a vertex, as a result degree of two nodes increases by one, and degree of two other nodes decreases by one. If the two edges have a common vertex, degree of that vertex is not changed, two other edge endpoints change their degree. This procedure is used to decrease the number of vertices with maximum degree violation, by reducing degree of one such vertex by one.

Subroutine EDGE_EXCHANGE$(T, h(\cdot), v, (i,j))$

1. Remove from T one of the two edges of the path connecting i and j in T, and incident to v.
2. Introduce edge (i,j) in T to restore connectivity.
3. If $h(i) \neq \emptyset$, run EDGE_EXCHANGE$(T, h, i, h(i))$;
   If $h(j) \neq \emptyset$, run EDGE_EXCHANGE$(T, h, j, h(j))$.

Figure 4.2: EDGE_EXCHANGE subroutine.

4.2.3 Analysis of the Algorithm

We need to prove that the algorithm terminates in polynomial time, and when it terminates the solution it outputs violates degree constraints by at most one.

At each iteration, the algorithm reduces the number of nodes with maximum violation. Maximum violation of a degree constraint can not be more than $n = |V|$, and therefore total number of iterations is bounded by n^2. At each iteration, the algorithm processes an edge between two different components, merging the components. Therefore no edge can be processed twice, and each edge processing procedure involves polynomially many operations, which implies that algorithm termination time is bounded by a polynomial in n.

It is easy to see that when the algorithm terminates, the set of *red* nodes S and the set of *yellow* nodes B satisfy the conditions of Theorem 4.2.1, and can be used to either prove that the DCST problem is infeasible or that $k = 0$. If $k = 0$ degree constraints are violated by at most one at each vertex.

To complete the proof, we need to show that edge exchange procedure in Step 3 reduces the number of nodes with violation $k+1$ by one. The edge exchange is organized so that it decreases degree of the red node on last path P by one, and it may only increase by at most one the degrees of nodes with violation no more than $k-1$.

4.3 Duplex Channel Problem

In duplex channel configuration the bandwidth of uplink channels is shared between incoming and outgoing streams. Therefore node capacity is shared between the incoming stream and all outgoing streams.

Definition 4.3.1. Formally, the *duplex channel throughput maximization problem (DCTMP)* is defined as following. Let $G = (V, E)$ be a complete undirected graph. We are given capacity constraints: node capacity $c_v \geq 0$ for each node $v \in V$, edge capacity $c_e \geq 0$ for each edge $e \in E$. Find a spanning tree $T = (V, E_T)$ in graph G which maximizes throughput function $\theta(T)$:

$$\theta(T) = \min\left\{\min_{e \in E_T} c_e, \min_{v \in V} \frac{c_v}{|\delta_T(v)|}\right\},$$

where $|\delta_T(v)|$ is the degree of node v in tree T.

It can be shown that DCTMP is NP-hard, and it is NP-hard to find a tree which achieves more than 2/3 of the optimal throughput. The approximation algorithm for degree-constrained spanning tree problem can be used to find a tree which achieves throughput at least 1/2 of optimal. We prove these results in the following subsections.

4.3.1 Complexity Analysis

NP-hardness of DCTMP can be proven by constructing a reduction from Hamiltonian Path problem.

Theorem 4.3.1. *Approximating DCTMP within more than* $2/3$ *of optimal is NP-hard.*

Proof. Suppose there exists an algorithm A that constructs a solution with value more than $2/3\theta^*$ for any DCTMP instance with optimal value θ^*. We will show that this algorithm can be used to solve Hamiltonian Path problem to optimality. Suppose we are given an undirected graph $G = (V, E)$, and a pair of vertices $i \in V$ and $j \in V$. A solution to the Hamiltonian Path problem P_h is a path that starts at i, visits each vertex of G and terminates at j. The algorithm should determine whether such a path exists, and if it exists, output it.

To specify throughput maximization problem P_t we construct a complete graph $G' = (V, E')$ by adding to G edges as necessary to make it complete. The newly added edges are assigned capacity 0, and the original edges are assigned capacity 1:

$$c_e = \begin{cases} 0, & e \in E' \setminus E; \\ 1, & e \in E. \end{cases}$$

Node capacities for all nodes v in V are set to be $c_v = 2$ except i and j, and $c_i = c_j = 1$.

By construction, P_t has optimal solution with value 1 if and only if P_h has a solution, i.e. there exists a Hamiltonian path in G. Since $\theta(T)$ is equal to either capacity of one of graph edges or to a fraction of node capacity. Therefore for any spanning tree T in G' throughput value $\theta(T)$ can only be one of the following values: $\left\{1, \frac{2}{3}, \frac{2}{4}, \frac{2}{5}, \ldots, \frac{2}{n-1}, 0\right\} \cup \left\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots, \frac{1}{n-1}\right\}$. Therefore algorithm A, which by our assumption can find a solution with value more than 2/3 if there exists a solution with value 1, will in fact find a solution with value 1 in that case. Thus, A can find a solution to Hamiltonian Path problem, and hence it is NP-hard to find a solution for DCTMP which is guaranteed to be within more than 2/3 of optimal. □

4.3.2 Approximation Algorithm

Since there are polynomially many values that $\theta(T)$ can assume, it is possible to search for the best value, for example enumerating all possibilities. Once throughput value is fixed, the problem of maximizing throughput reduces to a degree-constrained spanning tree problem: edges that do not have enough capacity can be removed, and node capacities translate into degree constraints. DCST_APPROX algorithm can be used to find an approximate solution. By analyzing resulting approximation factor we obtain the following theorem.

Theorem 4.3.2. *There exists a polynomial time algorithm which, given an instance of duplex channel throughput maximization problem, constructs a spanning tree with throughput at least* $1/2$ *of optimal.*

Proof. Let θ^* be the optimal throughput value for DCTMP on graph $G = (V, E)$ and capacities $\{c_v\}, \{c_e\}$. Also let $G' = (V, E')$ be the graph obtained from G by removing edges with capacity less than θ^*: $E' = E \setminus \{e : c_e < \theta^*\}$. Degree of vertex v in the optimal tree can not exceed $b_v = \lfloor c_v/\theta^* \rfloor$.

Running DCST_APPROX on graph G' with degree constraints $\{b_v\}$ will find a spanning tree T, in which degree constraints are violated by at most one. Let θ be the throughput of tree T: $\theta = \theta(T)$, and let d_v be the degree of vertex v in T. If throughput of T is bounded by capacity of an edge, it immediately follows that $\theta = \theta^*$, and the proof is complete.

We only need to show that $\theta \geq \theta^*/2$, if throughput of T is bounded by capacity of some node v:

$$d_v = c_v/\theta.$$

Node capacity constraint at v must hold for θ^*, hence $b_v \leq c/\theta^*$. Therefore,

$$c_v/\theta = d_v \leq b_v + 1 \leq c/\theta^* + 1.$$

Since in any feasible tree $c/\theta^* \geq 1$, the following inequality holds:

$$c_v/\theta \leq 2c_v/\theta^*.$$

Therefore, $\theta \geq \theta^*/2$. □

Proposition below, which can be proven similarly to the theorem above, gives insight into the reasons behind the gap between best possible approximation factor and approximation factor achieved by our algorithm.

Proposition 4.3.3. *For the class of DCTMP problems in which for all nodes $c_v \geq 2\theta^*$ (i.e. uplink throughput does not force a node to become a leaf), the algorithm described in Theorem 4.3.2 finds a solution with throughput at least $2/3$ of the optimal.*

This factor remains the best possible for the restricted class of problems (since Hamiltonian Path construction remains valid). We do not know whether it is possible to improve our algorithm or to prove a tighter bound on approximation factor, but the latter seems more likely.

4.4 Separate Channel Problem

The difference between the separate channel and duplex channel variants of the problem lies in the bandwidth sharing scheme implemented in uplink channels. If there is no bandwidth sharing between incoming and outgoing streams, capacity constraint at the node only limits the outgoing bandwidth. This variant of the problem will be studied in more detail in the following sections.

Definition 4.4.1. The *separate channel throughput maximization problem (SCTMP)* is defined as following. Let $G = (V, E)$ be a complete undirected graph. Node $s \in V$ is a source node, which does not receive incoming transmissions. All nodes except s receive one incoming stream, and may send several outgoing streams. We are given capacity constraints: node capacity $c_v \geq 0$ for each node $v \in V$, edge capacity $c_e \geq 0$ for each edge $e \in E$. The problem is to find a spanning tree $T = (V, E_T)$ in graph G which maximizes throughput function $\theta(T)$:

$$\theta(T) = \min \left\{ \min_{e \in E_T} c_e, \ \min_{v \in V \setminus \{s\}} \frac{c_v}{|\delta_T(v)| - 1}, \ \frac{c_s}{|\delta_T(s)|} \right\},$$

where $|\delta_T(v)|$ is the degree of node v in tree T.

We also define a modification of SCTMP, which introduces an additional assumption that makes the problem more tractable.

Definition 4.4.2. A *restricted SCTMP (RSCTMP)* is a SCTMP in which capacity bounds for all vertices v and all edges e incident to v satisfy $\max_{e\in\delta(v)} c_e \leq 2c_v$.

This assumption is equivalent to assuming that incoming bandwidth of each node does not exceed two times the bandwidth of outgoing channel.

Similarly to the duplex channel variant, both SCTMP and RSCTMP are NP-hard, and it is NP-hard to find a tree which achieves more than 1/2 of the optimal throughput for either problem. The approximation algorithm for degree-constrained spanning tree problem can be used to find a tree which achieves throughput at least 1/2 of optimal solution of RSCTMP, which is the best possible. The approximation factor that can be guaranteed by the same algorithm for SCTMP depends on edge and node capacities and equals to $\min\big\{1/2,\ (\min_v c_v)\,/\,\big(\min_u \max_{e\in\delta(u)} c_e\big)\big\}$. We prove these results in the following subsections.

4.4.1 Complexity Analysis

Using a construction very similar to one used in duplex channel formulation, we can use reduction from Hamiltonian Path problem to prove NP-hardness of SCTMP and RSCTMP, and derive a bound on approximation factor.

Theorem 4.4.1. *Approximating RSCTMP within more than* $1/2$ *of optimal is NP-hard.*

Corollary 4.4.2. *Approximating SCTMP within more than* $1/2$ *of optimal is NP-hard.*

Proof of Theorem 4.4.1 is similar to the proof of Theorem 4.3.1, and we omit it. Corollary follows since RSCTMP is a special case of SCTMP problem.

4.4.2 Approximation Algorithm

The approximation algorithm for SCTMP (and RSCTMP) can be constructed based on DCST_APPROX, using the search technique used in duplex channel case. However analysis of algorithm performance differs, since it can happen that the node that was bounding in approximate solution had out-degree 1, and out-degree 0 in the optimal solution. If that is the case, the same bound on optimal solution can not be used, since the capacity constraint no longer applies at this node. Therefore we provide a complete proof for the following theorem.

Theorem 4.4.3. *There exists a polynomial time algorithm which, given an instance of SCTMP, constructs a spanning tree with throughput at least*

$$\min\left\{\frac{1}{2},\ \frac{\min_{v\in V} c_v}{\min_{u\in V}\max_{e\in\delta(u)} c_e}\right\}\cdot OPT,$$

where OPT is the optimal throughput value.

Proof. Let θ^* be the optimal throughput value for SCTMP on graph $G = (V, E)$, with source node $s \in V$ and capacities $\{c_v\}, \{c_e\}$. Also let $G' = (V, E')$ be the graph obtained from G by removing edges with capacity less than θ^*: $E' = E \setminus \{e : c_e < \theta^*\}$. Degree of non-source vertex $v \neq s$ in the optimal tree can not exceed $b_v = \lfloor c_v/\theta^* \rfloor + 1$. Degree of source node s is bounded by $b_s = \lfloor c_v/\theta^* \rfloor$.

Algorithm DCST_APPROX given graph G' with degree constraints $\{b_v\}$ finds a spanning tree T, in which degree constraints are violated by at most one. Let θ be the throughput of tree T: $\theta = \theta(T)$, and let d_v be the degree of vertex v in T. If throughput of T is bounded by capacity of an edge, it immediately follows that $\theta = \theta^*$, and the proof is complete.

If $\theta = c_s/d_s$, i.e. the capacity constraint is tight at the source node, it follows that $\theta \geq \theta^*/2$, as it is shown in Theorem 4.3.2.

Therefore we only need to show that $\theta \geq \theta^*/2$, if throughput of T is bounded by capacity of some non-source node $v \neq s$:

$$d_v = c_v/\theta + 1.$$

There are two possibilities: $d_v \geq 3$ and $d_v = 2$. It can not happen that $d_v = 1$, since leaf nodes are not affected by constraints on outgoing bandwidth.

If $d_v \geq 3$, it must be that $b_v \geq 2$ and capacity constraint at v must hold for θ^*, hence $b_v \leq c/\theta^* + 1$. Therefore,

$$c_v/\theta = d_v - 1 \leq b_v \leq c/\theta^* + 1.$$

Since we assumed that $b_v \geq 2$, we have that $c/\theta^* \geq b_v - 1 \geq 1$, and it follows that $c_v/\theta \leq 2c_v/\theta^*$. Therefore, $\theta \geq \theta^*/2$.

Suppose $d_v = 2$. If $b_v = 2$, the argument above still goes through, and $\theta \geq \theta^*/2$.

So far we have proven that our algorithm achieves approximation factor of 1/2. However, the remaining case may cause us to reduce the approximation factor. Let $d_v = 2$, and $b_v = 1$. Node v is no longer subject to capacity constraint at the optimal solution, and it can not be used to obtain the bound. Nevertheless optimal tree must traverse at least one edge leading to every node u, since optimal tree T^* is connected, and therefore

$$\theta^* \leq \min_{u \in V} \max_{e \in \delta(u)} c_e. \tag{4.1}$$

Since $d_v = 2$ and node constraint at v is tight for θ, we have that $\theta = c_v \geq 0$. Hence inequality (4.1) is equivalent to

$$c_v \theta^* \leq \theta \min_{u \in V} \max_{e \in \delta(u)} c_e,$$

and therefore

$$\theta \geq \frac{c_v}{\min_{u \in V} \max_{e \in \delta(u)} c_e} \cdot \theta^* \geq \frac{\min_v c_v}{\min_{u \in V} \max_{e \in \delta(u)} c_e} \cdot \theta^*.$$

□

Corollary 4.4.4. *There exists a polynomial time algorithm which, given an instance of RSCTMP with optimal throughput value OPT, constructs a spanning tree with throughput at least $OPT/2$.*

Proof. The following inequalities, which are valid for the restricted problem, together with Theorem 4.4.3 prove this Corollary:

$$\frac{\min_{v \in V} c_v}{\min_{u \in V} \max_{e \in \delta(u)} c_e} \geq \min_{v \in V} \frac{c_v}{\max_{e \in \delta(v)} c_e} \geq \frac{1}{2}.$$

□

4.5 Upper Bounds

To evaluate quality of solution produced by approximation algorithms and to narrow search range for the cutting algorithm described in the next section we use two methods, that allow us to compute an upper bound on maximum achievable throughput. Both methods operate with the graph $G_\theta = (V, E_\theta)$ obtained from $G = (V, E)$ by fixing throughput value θ and removing edges that have capacity less than θ. Capacity constraints at nodes of G translate into degree constraints in G_θ. Let d_v^θ be maximum allowed degree of vertex v in spanning tree of graph G_θ. Now the problem is to find whether there is a spanning tree of G_θ that satisfies these degree constraints. If G_θ is not connected, the answer is trivial, but in general this problem is NP-hard. In the following subsections we propose two heuristics that prove negative answer to this question. Before heuristics are applied, we assume that values d_v^θ are adjusted such that $d_v^\theta \leq |\delta_\theta(v)|$ for all $v \in V$, where $|\delta_\theta(v)|$ is the degree of vertex v in graph G_θ.

4.5.1 Total Degree Bound

The total degree bound is based on the following condition.

Proposition 4.5.1. *If $\sum_{v\in V} d_v^\theta < 2(|V| - 1)$, there does not exist a degree-constrained spanning tree in G_θ with degree constraints $\{d_v^\theta\}$.*

Proof. The total number of edges in any spanning tree of G_θ has to be $|V| - 1$, and each edge has two endpoints. Hence, for any spanning tree total degree of vertices should be at least twice the number of edges:

$$\sum_{v\in V} d_v^\theta \geq 2(|V| - 1).$$

□

4.5.2 Witness Set Bound

In computing the witness set bound we use the witness sets we used earlier in proving constant factor for our approximation algorithm. Definition of witness sets was introduced in [26] and used for deriving lower bound for minimum degree spanning tree problem.

Definition 4.5.1. Subset of vertices $S \subseteq V$ is called a *witness set* for degree constraints $\{d_v^\theta\}$ and graph G_θ, if removing S and adjacent edges from G_θ separates the graph into N_S connected components, and

$$\sum_{v\in S} d_v^\theta < N_S + |S| - 1.$$

Proposition 4.5.2. *If there exists a witness set S for degree constraints $\{d_v^\theta\}$ and graph G_θ, a degree-constrained spanning tree in this graph can not be constructed.*

Proof. Since removing S from V decomposes the graph into N_S components that have no edges between them, the only way to connect the components is with edges adjacent to nodes of S. Therefore at least $N_S + |S| - 1$ edges are needed for connecting nodes of $|S|$ and the remaining components, and each of these edges has to have at least one endpoint in S. Hence the following condition must hold for the spanning tree to exist:

$$\sum_{v \in S} d_v^\theta \geq N_S + |S| - 1,$$

which contradicts definition of the witness set. □

We do not have an efficient way of discovering witness sets, and it is probably NP-hard to do. Therefore we use a heuristic to discover witness sets. We sort the nodes by degree constraints in increasing order, and verify witness set condition for all combinations of first 10 nodes in this list. As experiment results show, this approach can help discover witness sets in cases where total degree bound is not tight.

4.6 Solving Throughput Maximization Problem Numerically

In this section we develop a method for solving separate channel version of throughput maximization problem (SCTMP) numerically using integer programming and the Volume Algorithm. We analyze performance of our method in the following section dedicated to numerical experiments.

Since throughput in the system can always be increased until it is bounded by capacity constraint at either a node or an edge, there is a finite number of possible solution values. More precisely, the optimal throughput value is selected from the set of threshold values $\mathcal{H} = \{c_v/k : v \in V, k \in Z \cap [1, n]\} \cup \{c_e : e \in E\}$; therefore, $|\mathcal{H}| \leq (n^2 + n(n-1)/2)$. We can use approximation algorithm described in section 4.4 to obtain a lower bound on throughput $\underline{\theta}$. Upper bound $\overline{\theta}$ can be computed by relaxing constraints on throughput of individual nodes and solving the problem, which becomes equivalent to the minimum spanning tree, using one of polynomial time algorithms, for example the classic Kruskal's algorithm [35, 3].

We evaluate threshold values one by one, either using binary search or linear enumeration. For each threshold value we formulate an integer problem, solution to which gives us the routing tree. Using an integer programming solver, we determine whether a solution exists for given fixed throughput value. However, complete formulation of the integer program is too large to be solved by generic IP solver. In our experiments even an LP relaxation of complete formulation for a problem with 96 nodes could not be solved by CPLEX [21], which failed after working on the problem 2 days. We have developed an incremental approach based on cutting planes. Starting with a relaxation that has a small number of constraints, we detect and add violated constraints, and resolve the modified problem, until no violations can be detected. This method allows us to obtain an integer solution for the same problem instance with 96 nodes in less than 10 seconds.

Our algorithm starts with a very loose relaxation of the integer program. On each iteration current LP formulation is solved, and if violated cuts can be detected, at cuts of one type is added to the formulation, and the problem is solved again. On the next iteration the algorithm will attempt to discover violations starting with the next cut type first. When no new violated cutting inequalities can be added, the algorithm switches problem type to integer program, requiring $x_e \in \{0, 1\}$, and proceeds with iterations, but now invoking IP solver instead of LP solver.

4.6.1 Initial Relaxation

For each edge e of the complete undirected graph G we define a zero-one variable x_e, which indicates whether the edge will be used in the multicast tree. Let θ be throughput of the tree. For every nonempty subset of nodes $S \subseteq V$, let $E(S)$ be subset of edges in E that have both endpoints in S: $E(S) = \{e = (i,j) : \ e \in E, \ i,j \in S\}$. Then, the problem can be formulated as the following non-linear integer program:

$$
\begin{aligned}
& \max \theta \\
\text{Subject to} \quad & \\
& x_e \theta \leq c_e && \forall e \in E \\
& \Big(\sum_{\forall e \in \delta(v)} x_e - 1 \Big) \theta \leq c_v && \forall v \in V \\
& \sum_{e \in E} x_e = n - 1 \\
& \sum_{e \in E(S)} x_e \leq |S| - 1 && \forall S \subset V \\
& x_e \in \{0,1\} && \forall e \in E
\end{aligned}
$$

The last set of inequalities (subtour elimination constraints) together with the equality constraint guarantee that $\{x_e\}$ define a spanning tree [38], and the first two inequalities enforce capacity constraints. The problem can easily be rewritten as an equivalent linear integer program, if we divide capacity constraints by θ and introduce new variable $y = \theta^{-1}$:

$$
\begin{aligned}
& \min y \\
\text{Subject to} \quad & && && (4.2) \\
& x_e - c_e y \leq 0 && \forall e \in E \\
& \sum_{e \in \delta(v)} x_e - c_v y \leq 1 && \forall v \in V \\
& \sum_{e \in E} x_e = n - 1 \\
& \sum_{e \in E(S)} x_e \leq |S| - 1 && \forall S \subset V \\
& x_e \in \{0,1\} && \forall e \in E
\end{aligned}
$$

To start solving the problem, we relax integrality and subtour elimination constraints, keeping the first three, and obtain the following LP relaxation:

$$\begin{aligned}
&\min y \\
\text{Subject to}\quad & \\
& x_e - c_e y \leq 0 && \forall e \in E \\
& \sum_{e \in \delta(v)} x_e - c_v y \leq 1 && \forall v \in V \\
& \sum_{e \in E} x_e = n - 1 && \\
& 0 \leq x_e \leq 1 && \forall e \in E
\end{aligned}$$

This LP has $2|E| + |V| + 1$ constraints, and can be quickly solved by most linear programming solvers.

4.6.2 Cutting Inequalities

Since optimal solution to the relaxed formulation is not necessarily a feasible solution to the original problem, we employ cut generation to get our bound closer to the optimal solution. In this section we define inequality classes used for avoiding infeasible solutions, and describe separation procedures for each class of constraints.

Subtour Elimination Constraints

The inequalities used in (4.2) represent the usual *subtour elimination constraints.* Subtour elimination inequality can be written for every subset of nodes:

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad \forall S \subset V \tag{4.3}$$

To find violated constraints, we use the following search procedure. Using the current solution x^0 we construct a maximum spanning tree. Then for every edge not in the tree, we check whether adding this edge to the tree forms a cycle that violates the subtour elimination constraint.

Minimum Cut Inequalities

Minimum cut inequalities do not make use of degree constraints, and are valid for any spanning tree. If x defines spanning tree in G, then for any cut C of graph G at least one tree edge must cross the cut:

$$\sum_{e \in \delta C} x_e \geq 1 \qquad \forall C \subset V \tag{4.4}$$

To find a violated inequality given a solution x^0 to the relaxation, for every non-source node t : $\{t \in V, t \neq s\}$ we solve maximum s-t flow problem in graph G, where capacity of each edge e is defined by x_e^0. If maximum flow for some node t is below 1, we find minimum s-t cut C. By minimum cut - maximum flow theorem x^0 violates (4.4) for cut C, and we can add this inequality to current LP. If for all t maximum flow is at least 1, none of the minimum cut inequalities are violated.

Similar approach was used in [41] for solving a similar the problem of finding an optimal connected subgraph with fixed degree constraints at nodes. The authors further propose the use of *multicut inequalities*, studied in [28], to strengthen the formulation. Multicut inequalities can be defined for any partition of vertex set $V = \bigcup_{i=1}^{p} W_i$, $W_j \cap W_i = \emptyset$ for $i \neq j$:

$$\frac{1}{2}\sum_{i=1}^{p}\sum_{e\in\delta(W_i)} x_e \geq p-1 \tag{4.5}$$

Inequalities (4.4) are multicut inequalities with $p-1$. It can be shown that in our formulation multicut inequalities are linear combinations of subtour elimination constraints and and the constraint on the sum of all x variables, $\sum_e x_e = n-1$. Multicut inequalities also express the witness set constraints, used in the proof of constant factor approximation.

Node Capacity Cuts

The node capacity inequality

$$\sum_{e\in\delta(v)} x_e - c_v y \leq 1 \qquad \forall v \in V$$

often can be strengthened by adding a copy, in which constant 1 is replaced with $b_e y$, where $b_e = \max_{e\in\delta(v)} c_e$. The inequality is valid, since if at least one link can be connected to v, it must be that $b_e y \geq 1$ due to edge capacity constraints. Inequality can be checked for every node and added to the formulation if necessary.

Bound Cuts

An upper bound on the objective allows us to round the capacity constraints. Suppose we are given an upper bound on y: $y \leq \bar{y}$. It can be obtained, for example, from the lower bound on the throughput, $\bar{y} = 1/\underline{\theta}$. Then, since $cy \leq c\bar{y}$ for any positive constant c, the following equations are valid for all integral solutions:

$$\begin{aligned} & x_e = 0 && \forall e \in E : c_e\bar{y} < 1 \\ & \sum_{e\in\delta(v)} x_e \leq \min\left\{1 + \lfloor c_v \bar{y} \rfloor, \left|\{e \in \delta(v) : c_e\bar{y} < 1\}\right|\right\} && \forall v \in V \end{aligned}$$

Leaf Cuts

In addition to the bound cuts, if we have an upper bound on the objective value $\bar{y} = 1/\underline{\theta}$, we can add disable a set of edges based on the following observation. Node v must a leaf in the optimal solution if

$c_v\bar{y} < 1$. In a network with more than two nodes an edge connecting two leaf nodes can never be used, and therefore corresponding variable should be forced to 0. Violations are detected by enumerating edges connecting leaf nodes. Naturally, for this class of cuts, as well as for bound cuts, quality of the cut depends on the quality of the upper bound.

Degree Constraint Rounding

To define stronger degree constraints we will introduce new variables for node degrees. Let z_i be degree of node i, $i = 1..n$. Given a solution x to our relaxation, we can compute corresponding z_i as a sum of edge variables for adjacent edges: $z_i = \sum_{j\neq i} x_{ij}$. Note that if x is integral, then z is integral as well, and for solution to be valid $z_i \geq 1$ for all i. Further, based on our initial LP relaxation, we have the following constraints on z_i:

$$\begin{aligned} & z_i - c_i y \leq 1 && 1 \leq i \leq n \\ & \sum_{i=1}^{n} z_i = n - 1 \end{aligned}$$

Adding the inequalities we obtain that for every i the following two inequalities must hold:

$$\begin{aligned} & z_i - c_i y \leq 1 \\ & z_i + \sum_{j\neq i} c_j y \geq 0 \end{aligned}$$

Assuming that we have a lower bound on y, $y \geq \underline{y}$, we can make use of mixed integer rounding to improve one of these two inequalities. Note that we can use current LP solution as a lower bound.

4.6.3 Volume Algorithm

In order to obtain strong bounds on maximum achievable throughput, we have implemented a version of the volume algorithm, proposed in [11]. Volume algorithm is a modification of sub-gradient method which allows to construct both primal feasible solution and Lagrangian dual solutions.

We introduce Lagrangian multipliers λ_v for node capacity constraints, and consider the following Lagrangian relaxation of integer program formulation (4.2) for the throughput maximization problem:

$$\min y + \sum_{v\in V} \left(\sum_{e\in\delta(v)} x_e - c_v y - 1 \right) \lambda_v$$

Subject to

$$\begin{aligned} & x_e - c_e y \leq 0 && \forall e \in E \\ & \{x_e\} \text{ defines a spanning tree in } G \end{aligned}$$

Or equivalently, simplifying the objective:

$$\min \sum_{uv\in E} (\lambda_u + \lambda_v)\, x_{uv} + \left(1 - \sum_{v\in V} c_v \lambda_v\right) y - \sum_{v\in V} \lambda_v \tag{4.6}$$

Subject to

$$x_e - c_e y \leq 0 \qquad \forall e \in E$$

$$\{x_e\} \text{ defines a spanning tree in } G$$

Given fixed λ, optimization problem (4.6) can be solved in polynomial time. If $\left(1 - \sum_{v\in V} c_v\lambda_v\right) < 0$ the problem is unbounded. Otherwise, optimal value of y is $y^* = 1/c_e$ for some edge e. If both y and λ are fixed, the problem reduces to Minimum Spanning Tree (MST) in a graph formed of edges with sufficient capacity, and is easily solvable. Therefore we can enumerate all possible values for y, solving MST for each one, and compare the results to find the optimal solution.

Since we know have an oracle that can find integral optimal solutions to the Lagrangian relaxation, we can apply the volume algorithm [11] by Barahona and Anbil. Here we briefly describe the algorithm, and refer the reader to [11] for more details. We will write capacity constraints of (4.2) in general form $Ax \leq b$. Our version of volume algorithm proceeds as follows.

Algorithm INT_VOLUME_ALGORITHM

1. Start with $\lambda = 0$ and x, y corresponding to a spanning tree. Let $t = 1$.
2. Compute $v^t = (b - Ax)^+$ and $\lambda^t = \lambda + sv^t$ for
$$s = f\frac{\bar{y} - y}{||v||^2},$$
where f is a number between 0 and 2. Solve the Lagrangian relaxation to obtain solutions λ^t, x^t and y^t.
3. If $y^t > y$ update $\lambda \leftarrow \lambda^t$ and $y \leftarrow y^t$.
Let $t \leftarrow t + 1$ and go to Step 2.

In the volume algorithm primal feasible solution is a convex combination of x^t obtained during several past iterations, with weights exponentially increasing with age of the iteration, such that most recent iteration has largest weight. In our case x^t are integer variables. Therefore every 100 iterations we run a search for a solution that achieves highest throughput and only uses edges that were used in at least one of x^t during K previous iterations. In our experiments $K = 16$. Also, during each iteration when solution x^t is obtained, we run our approximation algorithms starting with the tree given by x^t, trying to find a better solution.

4.7 Experiments

We evaluate our algorithms on several randomly generated instances. We generated random network topologies of up to 500 nodes using GT-ITM package (see [63] for more details). We add client nodes to each of the stub nodes of generated topology. Client nodes are connected to the uplink with a

Nodes	Volume	Cut	Extended Cut
12	0.02s	0.07s	0.08s
24	4.89s	0.37s	0.22s
37	(3.3% gap)	1.86s	0.47s
48	31.67s	0.02s	0.02s
57	(8% gap)	120.98s	2.59s
77	(13% gap)	11m:47s	7.87s
96	(10% gap)	(35% gap)	9.60s
389	(9% gap)	(43% gap)	(43% gap)

Table 4.1: Numerical Results: Running Time

Nodes	Heuristic	Witness Set Bound	Total Degree Bound
12	83.59%	100%	100%
24	83.59%	100%	100%
37	66.94%	100%	100%
48	100.00%	100%	145.95%
57	69.17%	100%	100%
77	66.94%	100%	100%
96	74.26%	100%	100%

Table 4.2: Numerical Results: Quality of Bounds at Root

single edge. For each edge capacity bound was randomly generated. Finally, we randomly chose n client nodes that are participating in application-level multicast. For each such node v capacity c_v is determined by capacity of connecting edge. We further need to define capacities of edges c_{uv}, which are determined as minimum capacity on shortest path between u and v in the generated network.

The results of our experiments are summarized in Tables 4.1 and 4.2. First column of both tables contains number of nodes in each problem instance.

In Table 4.1 we compare performance of the volume algorithm and two modifications of cutting algorithm. For those experiments in which optimal solution was found in less than an hour, time used for solving the problem is shown. If the problem still was not solved after 1 hour, table entry contains the best proven gap between lower and upper bound. First modification of cutting algorithm ("Cut" column) only uses subtour elimination and minimum cut inequalities, which makes it similar to the set of cuts used in branch-and-cut algorithm proposed in [15]. In the second modification ("Extended Cut") we enable leaf and degree cuts.

Table 4.2 shows upper and lower bounds obtained at the startup of the algorithm on the same problem instances that are shown in Table 4.1. In columns 2 to 4 upper and lower bound throughput value is shown in percents of optimal throughput. Heuristic solution typically achieves less than 100% of optimal throughput, except for the instance with 48 nodes, which was solved to optimality.

In all our experiments networks were constructed such that a tight bound can be obtained by computing the sum of node degrees in the graph for each possible threshold value. The only instance in which a tight bound is not obtained this way, a 48-node network, is solved to optimality by our heuristic. In that case a one-node witness set can prove a tight bound. This suggests that the cases that are not determined by node constraints are more easily solved by the heuristic, but further experiments are required to support this conjecture.

4.8 Conclusion

In this chapter we describe approximation algorithms, for which we prove worst-case performance guarantee, and a numerical method for finding exact optimal solution for the NP-hard problem of maximum throughput multicast routing. Experiments on simulated data show that the exact solution can be found in problems of reasonable size.

From the practical perspective, future work must include modifications that bring our model closer to implementation, and performance evaluation of a prototype implementation. In particular, our approximation algorithm can adapt to changing network conditions by performing several edge exchange iterations. Each iteration of the approximation algorithm can start with any valid spanning tree, and improve throughput incrementally via edge exchanges. Edge exchange can be performed without interrupting multicast session, and backup buffer mechanism described Chapter 2 can be used to prevent losses during edge exchange.

From theoretical perspective, it is of interest to attempt to close the gap between the approximability bound and worst-case bound achieved by our approximation algorithm. We believe that the approximability bound can be improved, but we leave that for future research.

Chapter 5

Multicast Group Membership

5.1 Introduction

Publication-Subscription systems (*pub-sub* for short) provide information on specific real-time *events* from *publishers* to interested *subscribers.* The subscribers express their interest in the form of multiple *subscriptions.* The publishers and subscribers may be located at arbitrary nodes in a distributed network.

Pub-sub systems can be characterized into two broad types based on the degree of generality, usability and personalization allowed to the subscribers. We focus in this chapter on the more sophisticated of these, namely *content-based* pub-sub. See, for example, the pioneering work of the Gryphon project [2, 8, 45], as well as NEONet [43] and READY [29]. Such systems differ from the more mature *subject-based* pub-sub systems in their generality, usability and flexibility. Borrowing extensively from the classic stock market example used by Gryphon, a subject-based pub-sub system would allow subscriptions based on broad criteria only. A subscriber might request all events related to IBM, for instance. Such a system is powerful but relatively inflexible. The subscriber might receive many publications involving IBM stock market events of little interest. On the other hand, a content-based pub-sub system would allow subscriptions which were based on the conjunction of potentially multiple predicates related to different attributes.

Expanding on the stock market scenario originally discussed by Gryphon, a content-based pub-sub system would allow subscriptions which are based on the conjunction of potentially multiple predicates concerning different attributes. The motivating Gryphon subscription example was based on three distinct attributes:

- The first was an equality predicate based on a character string referring to the stock name. *name*=IBM would be an example, though one could also imagine categories ("blue chip", for instance) being specified instead.
- The second could be one or two inequality predicates based on a two decimal numerical attribute referring to the stock price, such as $90.00 < \textit{price} \leq 110.00$. Alternatively one could normalize these by the price of the stock at the start of the day, so that a subscription might look for changes in price within 10 percent in a day.
- The third could be one or two inequality predicates based on an integer attribute referring to the volume: $\textit{volume} > 10{,}000$ might be an example. One could easily imagine translating these

volumes into dollars to ensure a common basis amongst differing stocks, so that one might look for volume whose monetary equivalent exceeds one million dollars.

A client with this (expanded) subscription would receive information about all trades of "blue chip" stocks whose price stays within a 10 percent range during the day and involves more than a million dollars in start-of-day units.

So content-based pub-sub is more general and much more personalizable than subject-based pub-sub. In general, we can assume that content-based pub-sub systems allow each predicate to be *range-based*, composed of *intervals* in the underlying domain of the predicate. (Because computers can handle only inherently finite and discrete attribute values, one can assume without loss of generality that all possible intervals are actually open on the left and closed on the right. This assumption allows the intervals to 'fit together' more cleanly.) By decomposing a subscription with multiple such ranges into multiple subscriptions consisting of single ranges we can see that it is sufficient only to consider intervals, albeit at a cost of more subscriptions. And even attributes such as name, not typically thought of as numerical, can be indexed and therefore linearized in some fashion. A predicate involving "blue chip"s can thus be decomposed into the union of several conjunctions.

This perspective allows us to think of a subscription as a half-open, half-closed aligned rectangle in space, each dimension corresponding to a different attribute. (The term *aligned* means that the rectangle is parallel to the various axes.) A published event becomes a point in the underlying space.

Clearly, the extra function involved with pub-sub systems comes at a price. It is technically challenging for a content-based pub-sub system to efficiently distribute the many publication events to the interested subscribers over the network. And it is technically challenging to do so in a manner which scales with the dimensionality of the underlying event space, the number of publishers and the number of subscriptions.

Due to complexity of optimization problems arising in pub-sub context, in this chapter we assume that multicast routing problem is solved independently, for example using one of the approaches we describe in Chapters 3 and 4. Hence, are two key dynamic problems that a content-based pub-sub system needs to solve:

1. One must *match* a given real-time event quickly to determine the set of relevant subscribers. This is the so-called *matching* problem.

2. One must decide to unicast, multicast and/or broadcast information about the event over the network efficiently to the matched subscribers (or possibly to a superset of those subscribers, if that is more algorithmically reasonable, to be filtered out as necessary). We shall call this the *distribution method* problem.

Both of these problems must be solved in real time as new events are published. However, there is a related, static "preprossessing" problem that should be solved in order to enable the real-time algorithms to function efficiently. Basically we must precompute a set of high quality multicast groups having as much commonality as possible, based on the totality of subscribers' interests. We shall call this the *subscription clustering* problem. This chapter will be focused on the clustering problem, whereas the matching problem is analyzed in more detail in a companion paper [50].

The Gryphon papers do tackle these problems in elegant ways. In [2] and [8], for instance, the authors consider the matching problem. The interested reader is referred to [23] for a detailed description of the matching algorithm, a performance analysis and a comparison with existing matching algorithms. In [8] and [45] the authors consider multicasting techniques. It seems fair to say, however, that the authors base their algorithms primarily on subscriptions in which each dimension is based on either equality or wild-card predicates. (A *wild-card* (*) indicates that the subscriber does not care about the value in that dimension.) While their algorithms will certainly work in full generality, we believe that they are optimized for their motivating predicate types.

However, the above mentioned earlier work of Gryphon project (see, for example, [45]) concluded that multicast mechanism does not provide substantial benefits in many pub-sub systems, and that given the multicast overhead, unicast and broadcast are sufficient for these systems. We believe that the conclusions would be very different depending on the network configurations, distribution of publications and subscriptions. In this work we consider larger networks with fewer number of subscriptions from each node. We think this setting could be closer to the real world environments. This leads to the potentially large advantage of forming multicast groups. We will evaluate the benefits of employing (both network-supported and application-level) multicast mechanisms. We shall show that the communication costs depends crucially on how to form the multicast groups.

Specifically, in this chapter:

- We examine the various assumptions in the Gryphon framework and investigate quantitatively different impacts from several aspects of the pub-sub frame work.
- We introduce a general framework that allows to adapt partitional data clustering algorithms for pub-sub systems in which subscriber preferences are described more generally than in previous work.
- We devise a number of new clustering algorithms and enhanced some others. Among the algorithms we now consider are *K-means*, an important variant called *Forgy K-means*, a hierarchical clustering algorithm based on *Minimum Spanning Trees (MST)*, the *Pairwise Grouping* algorithm and a variant called *Approximate Pairwise Grouping*, and finally a so-called *No-Loss* algorithm. (The no-loss strategy implies that a publication never needs to be filtered out in the dynamic stage: Every subscriber that receives a publication is indeed interested in that message. The other algorithms described in this paper do *not* have this property.) Our comparisons show new leaders among these algorithms, and our results are more robust and realistic.
- We evaluate our algorithms on a large network model. Our subscriptions are based on three different models of interest, and the same is true for our publication model. We analyze the effects of regional subscriber preferences relative to the network topology. We show that the conclusions in this paper depend dramatically on assumptions about the size and structure of the network.

We will consider two flavors of multicasting in this paper, network supported and application level multicast schemes. The interested reader is referred to [4] for a description of the various tradeoffs, see also [46] for a description of application level multicasting.

The remainder of this chapter is organized as follows. We introduce some notation in Section 5.2. In Section 5.3, we describe some preliminary investigations which illustrate potential impacts of communication algorithms on the communication costs. We then in Section 5.4 describe the algorithms for solving common interest clustering problems. In Section 5.5 we present the results of our experiments and comment on the relative performance of the algorithms. Finally, in Section 5.6 we summarize the results and discuss about future work. We believe that the many ideas for future work are indicative of the newness and importance of this area of research.

5.2 Problem Notation

In this section we define some key parameters we will use in our problem descriptions. Let Ω denote the publication event space. Each event being published within the system can be uniquely described with a single value ω such that $\omega \in \Omega$. Let N denote the number of dimensions (or attributes) in Ω, so that $\Omega \subseteq \mathbf{R}^N$. Let $p_p(x)$ be the probability that publications events are within set $x \subseteq \Omega$. Define the underlying network topology as an undirected graph $G = (V, E)$. Define the communication costs to be $c_e \geq 0$ for each edge $e \in E$. Let $V_P \subseteq V$ be the set of nodes containing publishers. Let $V_S \subseteq V$ be the set of nodes containing subscribers. Let N_S be the number of subscribers. In this chapter we shall assume that each subscriber $v_i \in V_S, i = 1, ..., N_S$, has a set of subscription preference expressed by (possibly infinite) rectangles $\mathcal{I}_i = \{b_{ij}\}_{j=1}^{r_i}$ associated with it. Each $b_{ij} \subseteq \Omega$ is an aligned rectangle in space Ω. We define $\mathcal{I} := \bigcup_{v \in V_S} \mathcal{I}_v$ to be the set including all subscription rectangles. We define $k := |\mathcal{I}|$ to be the number of subscriptions.

The size of the clustering problem is essentially determined by the dimension of the event space N and the number of subscriptions k. We are interested in algorithms that scale well with respect to these values.

Typically the number K of multicast groups available to the clustering algorithm is known in advance. In the case of network-supported multicast, this is the number of multicast IP addresses purchased. In the case of application level multicast this number is limited by the amount of memory and processing power of participating computers.

5.3 Preliminary Analyses

Earlier work from the Gryphon project (see, for example, [45]) has demonstrated that multicast mechanism does not provide substantial benefits in many pub-sub systems. Given the well-known structural and performance overhead for applying multicast, unicast and broadcast are sufficient for these systems, having little or no overhead and insignificant increase in the delivery cost. We examine the various assumptions in the Gryphon framework and investigate quantitatively different impacts from several aspects of the pub-sub frame work. The conclusions are very different depending on the network configurations, distribution of publications and subscriptions.

In our model, events are generated in 4 dimensions, with the first one corresponds to "regional attribute". When a publication event occurs, the publication is always set to the identification number of originating subnet ("stub") for this message. The *degree of regionalism* parameter is the probability that in a subscription this attribute equals to corresponding subnet number. Zero degree of regionalism corresponds to no regionalism, and degree 1 to absolute regionalism. Regional subscriptions in this table are generated with this parameter set to 0.4. Non-regional subscriptions have this parameter set to 0 during generation of subscriptions.

The other 3 attributes of events can take on integer values between 0 and 20, according to either uniform or Gaussian distribution. Preference on each parameter can be either a "don't care" parameter, denoted "*", which means that all values of this parameter are of interest, or the interval can be specified. In the uniform case, the probabilities of not having "*" in position 2 is 0.98, and then decreases with the rate of 0.78 for subsequent parameters, so parameter 3 preference is specified with probability $0.98 \cdot 0.78$, and parameter 4 with probability $0.98 \cdot 0.78^2$. If parameter preference is specified, then two random numbers between 0 and 20 are generated, sorted if needed, and assigned to the ends of the preference interval. For Gaussian distribution each of the 3 parameters preference can have value of "*" with probability q_1, can be a left-ended interval with probability q_2, a right-ended interval with probability q_3 and an interval with both ends with probability $(1 - q_1 - q_2 - q_3)$. If both ends of the interval are specified, the center of the interval follows a Gaussian distribution with parameters (μ_3, σ_3) and the length of the interval follows a Pareto-like distribution with a given mean. If the interval is one-ended, then the end of the interval is chosen from Gaussian distribution with parameters (μ_1, σ_1) and (μ_2, σ_2) for left-ended and right-ended intervals. The parameters in the experiment were chosen from the following table, to simulate stock name, price and trading volume:

Para	q_1	q_2	q_3	μ_1, σ_1	μ_2, σ_2	μ_3, σ_3	mean
2	0.1	0	0	8,2	10,2	9,6	1
2	0.15	0.1	0.1	8,1	10,1	9,2	4
2	0.35	0.1	0.1	8,1	10,1	9,2	4

The networks were generated by GT-ITM package (see [63] for more details) using transit-stub model with one transit block and the following parameters:

Node	Trans node	Stubs/trans node	Nodes in a stub
100	4	3	8
300	5	3	20
600	4	3	50

Figure 5.1 illustrates the topology of the network with 100 nodes. More details about the simulation framework can be found in section 5.5.1.

Tables 5.3 and 5.2 show the communication costs for broadcast, unicast and ideal multicast, where ideal multicast stands for the distribution scheme where a multicast group is formed for each publication event and is composed only of the subscribers interested in this event. The ideal multicast could thus use all the possible 2^{N_S} groups. We observe that for the same network, the difference in cost between the broadcast and ideal multicast is small for cases with a large number of subscriptions, and becomes larger as the number of subscriptions decreases. This gap can be as large as 4 times the ideal solution. Therefore, there is need to further investigate more sophisticated content delivery mechanisms.

The Gryphon framework has a 100 node network, with an average of 125 subscriptions for each of the 80 nodes. For such types of networks with a small number of nodes, each node having many subscriptions per publication, there is a very high probability that at least one of the subscriptions at this node includes this publication. Therefore, the number of nodes interested in this publication is either very high or very low. This means that the system needs to deliver the publication either to almost every node or to a very small set of nodes. We believe that this fact results in the conclusion by Gryphon that broadcast is sufficient for such systems. For larger networks with relatively fewer subscriptions, on the other hand, it is unlikely that publications need to be delivered to almost every node in the network. Multicast is most beneficial in this kind of setting which messages need to be delivered to only part of the network.

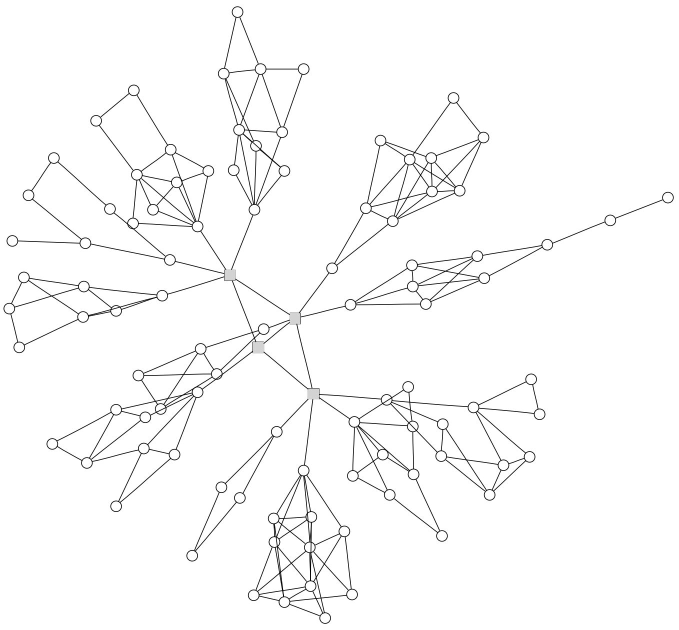


Figure 5.1: Network topology with 100 nodes.

Node	Sub'n	Dist'n	Unicast	Broadcast	Ideal
100	5000	uniform	31351	1430	1334
100	5000	gaussian	48805	1430	1415
100	1000	uniform	5846	1430	867
100	1000	gaussian	9513	1430	1134
100	80	uniform	750	1430	310
100	80	gaussian	548	1430	287
300	5000	uniform	38612	5079	3453
300	1000	uniform	8181	5079	867
300	350	uniform	3638	3880	1065
600	10000	uniform	92178	10235	6720
600	10000	gaussian	139020	10235	8212
600	5000	uniform	45320	10235	4820
600	5000	gaussian	69179	10235	6431
600	1000	uniform	5477	10235	1350
600	1000	gaussian	9408	10235	1823

Table 5.1: Potential advantage of multicast for case with degree 0.4 regionalism

Table 5.2: Potential advantage of multicast for case with no regionalism

Node	Sub'n	Dist'n	Unicast	Broadcast	Ideal
100	5000	uniform	50737	1430	1377
100	5000	gaussian	81779	1430	1428
100	1000	uniform	9409	1430	1039
100	1000	gaussian	16314	1430	1301
100	80	uniform	816	1430	328
100	80	gaussian	1580	1430	545
300	5000	uniform	61513	5079	4019
300	5000	gaussian	98735	5079	4751
300	1000	uniform	13384	5079	2026
300	1000	gaussian	21167	5079	2918
300	80	uniform	61513	5079	1113
300	80	gaussian	6113	5079	1598
600	10000	uniform	151270	10235	7993
600	10000	gaussian	232405	10235	9382
600	5000	uniform	73830	10235	6184
600	5000	gaussian	116952	10235	8000
600	1000	uniform	8276	10235	1791
600	1000	gaussian	14428	10235	2502

In Tables 5.3 and 5.2 the unicast and ideal multicasts are in general larger for the gaussian distributions than for the uniform. This is due to the fact that more nodes are likely to be interested in the publication events for the gaussian case, hence results in more communication cost. This shows that the publication distributions also affects the multicast benefits.

Furthermore, Tables 5.3 and 5.2 show that the communication costs for regional-specific subscriptions are smaller than the costs for non-regional subscriptions. More generally, the dependence of the subscriptions by different nodes can have a big impact on the multicast benefits. Consider independent subscriptions by the nodes for a particular publication. The nodes that are interested in this publication would be scattered around the network evenly with very high probability. The multicast benefit for delivering messages to such a scattered network would not be significant. On the other hand, if the subscriptions have a regional concentration, the interested nodes of a publication would very likely be more localized. It would not be surprising to observe substantial benefits from employing multicasts.

In addition, if the probability that each node subscribes to a given message is independent of other nodes, and there is no concentration of common interest in the event space, then it is difficult to form only a few number of groups and greatly improve communication efficiency – each of $(2^{N_S} - 1)$ possible multicast groups will be needed with equal probability.

The rest of this chapter considers larger networks with fewer number of subscriptions from each node. We think this setting is closer to the real world environments. This leads to the potentially large advantage of forming multicast groups. For such pub-sub systems, using broadcast to deliver messages would not be appropriate due to its large communication overhead. We will evaluate the benefits of employing multicast mechanisms. The communication benefits of multicast depends crucially on how to form the multicast groups. Due to the overhead of managing a large number of multicast groups, we need to consider forming a limited number of groups. We develop several algorithms for constructing multicast groups and evaluate their performance benefits. Algorithmic complexity is also a key factor for these real-time applications. We further study the cost benefit and running time trade-offs of these algorithm and discover good algorithms for practical applications.

Before going into details on the algorithms, it is important to describe the assumptions of our studies for the aforementioned reasons. First, we assume that the peaks in density of subscriptions follow peaks in density of the messages. It seems likely that multicast will not provide comparable improvements in communication cost without this assumption.

We further assume that the subscriber preferences are regional in the network topology. In our experiments, for example, stock name preference (mean of the distribution) was assigned according to the transit block in the network. In addition we assume subscriptions themselves are unevenly distributed in the network, with higher concentrations of interest in some areas, and lower in others.

Under these assumptions forming a limited number of groups using subscription clustering algorithms can potentially lead to large reduction of communication costs when compared to unicast and broadcast. While relatively restrictive, the assumptions still seem to be practical. Indeed, in many real-life pub-sub systems we would expect that events, in which more people are interested, are typically published more often, than the less interesting events. Also, we can expect the regionalism of subscriptions, with more concentration of users in certain parts of the network, and regionalism of interest, with interest distribution being different in different parts of the network. The model formally presented in section 5.5.1 follows the above assumptions.

5.4 Algorithms for Subscription Clustering

There are two distinct categories of subscription clustering algorithms that we present in the chapter. First category, the *Grid-Based* clustering algorithms, extends earlier work on subscription clustering in content-based pub-sub systems for the case of rectangular preference sets. Work by the Gryphon group [8] and by Katz *et al* [62] employ similar data algorithms for clustering of point subscriptions. In the following subsection we describe the framework that allow us to use data clustering algorithms for clustering subscriptions. In the following subsections we illustrate our approach by describing how four clustering algorithms: *K-means*, *Forgy K-means*, *MST*, and *Pairwise Grouping* can be used for subscription clustering.

The second category of subscription clustering algorithms presented in this section includes just one algorithm – the *No-Loss* algorithm. While grid-based algorithms sometimes can "lose" messages, sending them to subscribers who are not interested in the particular message, but only happen to have close interests, the No-Loss algorithm guarantees that all subscribers receiving a message are interested in it, thus avoiding redundant communication in the network.

Subscription clustering algorithms form multicast groups, as well as produce data structures that are used for matching events to multicast groups. The last subsection describes matching algorithms for the two categories of clustering algorithms.

5.4.1 Grid-Based Clustering Framework

Grid-based subscription clustering algorithms (or *cell clustering* algorithms) apply data clustering heuristics to the cells of a regular grid in event space Ω. Data clustering algorithms are widely used for grouping "similar" objects. Similarity of objects is determined based on the value of a distance function. In this subsection we define feature vectors and distance function for clustering, and describe the application of several data clustering algorithms to our model in the following subsections.

Feature Vectors. Each cell $a \subset \Omega$ has a subscriber membership vector $\mathbf{s}(a) \in \{0,1\}^{N_S}$ associated with it. By definition,

$$\mathbf{s}(a)_i := \begin{cases} 1 & \text{if there exists index } j \text{ s.t. } b_{ij} \cap a \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

In this vector non-zero elements correspond to subscribers interested in the cell.

The most commonly used strategy for partitional clustering is square-error minimization criterion, which amounts to minimizing sum of the squared Euclidean distances between feature vectors corresponding to objects over the entire set of objects. We use membership vectors as feature vectors of cells instead of using, for example, the coordinates of the cell center in event space Ω. Using coordinates in Ω for this purpose would lead to poorer solutions, since our goal is to create groups based on common interest, as opposed to *similar* interest. Grouping based on similar interest is considered, for example, in the work by Katz *et al* [62]. In our model, however, we assume that subscribers are only interested in events for which they have directly expressed interest, and there is a performance penalty to be paid when a message is delivered to a subscriber who not interested in it. On the other hand, when several subscribers are interested in the same events, possibly scattered over Ω, it is still logical to assign these subscribers to the same multicast group. Therefore we conclude that the comparison of sets of interested subscribers is more suitable for identifying common interest than comparison of coordinates in the event space.

Distance Function. Squared Euclidean distance between cells a and b can be computed as $d_e^2(a,b) = \sum_i \big(\mathbf{s}(a)_i - \mathbf{s}(b)_i\big)^2 = \sum_i \big(\mathbf{s}(a)_i \oplus \mathbf{s}(b)_i\big)$, where "$\oplus$" denotes a binary operator of exclusive *or*. In our model, the probability density function of publications can be taken into account in order to better characterize our objective. We define the distance function d as: $d(a,b) := p_p(a) \sum_{i \in V_S} \max\big\{[\mathbf{s}(a)_i - \mathbf{s}(b)_i], 0\big\} + \; p_p(b) \sum_{i \in V_S} \max\big\{[\mathbf{s}(b)_i - \mathbf{s}(a)_i], 0\big\}$. The value of $d(a,b)$ is the expected number of messages sent to subscribers, who are not interested in them, if the cells a and b are combined in one group. Note that we can similarly define membership vectors and distance functions for sets of cells, simply replacing cells a and b with sets of cells instead.

The function d characterizes *expected waste*, the notion first introduced in work by Gryphon group [8]. The objective of clustering in this formulation is to form groups in a way that minimizes expected waste. It is known that the heuristics used for clustering are not guaranteed to achieve a global optimum, but in practice the quality of the solution may be sufficient.

Implementation Notes. If for cells a and b it is true that $\mathbf{s}(a) = \mathbf{s}(b)$, then the cells can be combined in one group inducing zero expected waste. Our implementation during preprocessing stage scans continuous blocks of cells searching for repeating sets of subscribers, and joining the cells into *hyper-cells*, in other words sets of cells having the same membership vector.

Since the number of cells might become too high for the algorithm to handle, it would be helpful if we could somehow select the "most popular" cells for clustering, leaving the rest for unicast. Our implementation sorts hyper-cells (after the grouping step described above) by "popularity rating" $r(.)$ defined by $r(a) := p_p(a) \sum_{i \in V_S} \mathbf{s}(a)_i$ and keeping only a fixed number of cells, having the largest values of popularity rating.

Our experiments show that after some number of cells, the improvement gained from feeding more cells to the algorithm becomes negligible. In fact, the more cells are given to clustering algorithm, the worse the quality of solution becomes. This justifies the need for some implementation of outliers removal algorithms for detection of cells that have rather unique combination of subscribers. On the other hand, even without the outlier removal algorithm, clustering a large enough fraction of cells can lead to sufficiently good results.

5.4.2 K-Means and Forgy K-Means Clustering

The use of k-means clustering algorithm in pub-sub systems (with different objective, and therefore different feature vectors and distance functions) was proposed by Katz *et al* in [62]. We have studied the performance of original algorithm by McQueen and as well as the one of its variants by Forgy, see details in [32].

```
0. Form initial $K$ groups.
1. Re-assign each cell to a closest group.
2. Repeat step 1 until no cell can be moved.
```

Figure 5.2: K-Means Clustering.

Figure 5.2 shows generic pseudo-code for a k-means clustering algorithm for forming K multicast groups. In step 0, the initial partition is formed. For this purpose, K hyper-cells with the highest popularity rating $r(a)$ are chosen to be centroids of groups, and the rest of hyper-cells is assigned to closest groups, based on the expected waste distance function. In step 1 each of the hyper-cells is examined, and assigned to the "closest" cluster. The distance to the cluster is determined using the distance function, measuring the distance between the membership vector of the hyper-cell and the membership vector of the cluster. The K-means algorithm updates the membership vector of the cluster each time a hyper-cell is moved. The algorithm by Forgy updates membership vectors of all clusters after step 1 is finished. A hyper-cell cannot be moved to another cluster if it is the last hyper-cell in its current cluster.

K-means type algorithms are proven to converge to a local optimum in a number of iterations, and in practice they converge quickly. Nevertheless, the processing can be stopped after any iteration, resulting in a feasible partition into K groups. This also provides an easy way to accommodate changes in cell membership, simply running a number of re-balancing iterations, when new subscribers arrive or subscription rectangles are changed in some other way.

5.4.3 Pairwise Grouping

Pairwise grouping for clustering point interest (or *pairs* for short) was proposed in the work by Gryphon group [8], and we have extended these ideas to the case of interest rectangles. It is a top-down clustering algorithm, which starts with each hyper-cell assigned to its own cluster. If the total number of clusters is larger than the required number K, the two groups with the minimum distance to each other are chosen, and combined. The membership vector for this group becomes a combination of vectors of the joined groups. The process is repeated until no more than K groups are left. The algorithm is summarized on Figure 5.3.

An approximate version of this algorithm inspects a fraction $1/e$ of the total number of the group combinations during the search for best distance, stores the pair with the shortest distance from this fraction of groups, and terminates the search after that, if a smaller distance is found. This heuristic derives from a well-known solution to secretary problem ([52, p. 114]) for maximizing the chance of

```
0. Given $l$ cells, form $l$ groups, one cell in each.
   Group $g_i = \{a_i\}$, for each cell $a_i$, and $\mathbf{s}(g_i) = \mathbf{s}(a_i)$.
1. Find $i$ and $j$ such that $i \neq j$ and $d(g_i, g_j)$ is minimized.
   Reset $g_i \leftarrow g_i \cup g_j$, update $\mathbf{s}(g_i)$, and remove $g_j$.
2. Repeat step 1 until there are only $K$ groups left.
```

Figure 5.3: Pairwise Grouping.

choosing the maximum value, if the decision must be done immediately. The algorithm modified in this way works faster, but it may obtain a poorer solution.

5.4.4 Minimum Spanning Tree Clustering

The use of the minimum spanning tree (*MST*) for clustering was proposed by Zahn (1971). Suppose we have a graph G, in which nodes correspond to hypercells, and there is an edge between each pair of nodes i and j of length $d(i, j)$. We will say that each node in itself is a component. Processing the edges in non-decreasing order of length, if the edge connects different components, combine the components into one, and proceed to the next edge. Continue until there are K components left.

```
0. Given $l$ cells, form $l$ groups, one cell in each.
1. For each pair in the order of increasing distance:
2.    If the hyper-cells of the pair are not in same group,
         combine the groups corresponding to cells.
      Repeat step 2 (for next pair) until
      there are only $K$ groups left.
```

Figure 5.4: Minimum Spanning Tree Clustering.

This algorithm is similar to Kruskal's algorithm [35] for finding MST in graph G, except that this version stops when exactly K connected components are formed. Pairwise grouping proceeds in the similar fashion as the MST algorithm, but in the pairwise grouping case the distances are calculated between groups, not between cells. Therefore it is impossible to sort pairs by distance in advance, which in effect leads to greater running time of the pairs algorithm, compared to MST on the same data.

5.4.5 No-Loss Algorithm

The grid-based family of cell clustering algorithms works with cells of a regular grid, and each cell of the grid can be associated with one of the multicast groups. As a result, each subscriber whose interest overlaps with the cell, is assigned to the multicast group. Since interest rectangles are not aligned on cell borders, it is possible that an event sent to a multicast group will reach subscribers that are not interested in this particular event, as well as the ones for which this event was intended. The idea behind the *No-Loss* algorithm is to avoid this kind of wasted communication completely, forming multicast groups corresponding to areas aligned to interest rectangles borders. The algorithm (Figure 5.5) tries to find the "most popular" intersections of interest rectangles. The popularity (or *weight*) of an area in event space is measured by the number $|u(s)|$ of subscribers interested in this area multiplied by the probability of publication in the area: $w(s) = p_p(s)|u(s)|$.

```
0. Set of rectangles: $S := \mathcal{I}$;
   Rectangle weights $w(b_{ij}) := p_p(b_{ij})$ for each $b_{ij} \in S$;
   Subscriber node lists: $u(b_{ij}) := \{i\}$ for each $b_{ij} \in S$.
1. Sort elements of $S$ by $w$, such that:
     if $s_l, s_m \in S$ and $l < m$, then $w(s_l) > w(s_m)$.
2. Retain only first $T$ elements
   in sets $S$, $w$ and $u$, discarding the rest.
3. For each $b_{ij} \in \mathcal{I}$ and $s \in S$, such that $t := s \cap b_{ij} \neq \emptyset$,
   and $i \notin u(s)$ do:
     if $\exists\ r \in S$ such that $r \equiv t$,
       $u(r) \leftarrow u(r) \cup u(t)$; $w(r) \leftarrow p_p(r)|u(r)|$.
     else $S \leftarrow S \cup t$; $u(t) \leftarrow u(t) \cup \{i\}$; $w(t) \leftarrow p_p(t)|u(t)|$.
4. Repeat from step 2 at most $k$ times.
5. Sort $S$ as in step 1.
6. Form $K$ multicast groups corresponding to
   first $K$ elements of $S$, grouping subscribers
   according to $u(s_l)$ lists, $l = 1..K$.
```

Figure 5.5: No-Loss Algorithm

```
1. Given an event $\omega$, find the corresponding cell
   of the grid in the event space $\Omega$.
2. If the event is matched to a cell:
2.1. Denote the associated multicast group $\mathcal{G}$.
     Find the number (or proportion) of members of $\mathcal{G}$,
     interested in $\omega$.
2.2. If the number is above a predefined threshold:
2.2.1. Send the message to group $\mathcal{G}$.
     Else
2.2.2. Send it only to interested subscribers.
3. Else (if the event is not matched)
     Send it to the list of interested subscribers.
```

Figure 5.6: Matching for Grid-Based Algorithms

5.4.6 Matching Subscriptions To Events

Each time an event arrives in the system, it has to be matched to multicast groups formed by a clustering algorithm in order to find out how to deliver the message. Matching must done efficiently, since the delay caused by the matching algorithm directly affects the maximum throughput of the system. In this subsection we briefly introduce matching algorithms to illustrate how the data structures produced by subscription clustering algorithms is used in real-time, and refer the reader to our paper [50] for more detailed discussions.

Each group produced by a clustering algorithm can be described as a set of aligned rectangles in the event space Ω. Therefore the problem of matching an event ω to multicast groups can be reduced to the problem of searching among aligned rectangles in event space Ω for the rectangles that contain a given point ω. This general problem is most commonly solved using a special pre-built data structure called R^*-tree (see [12]). In order to achieve better performance a modification of R^*-tree algorithm, the S-tree algorithm described in [1] can be used instead.

Matching in Grid-Based Algorithms. Multicast groups formed by a grid-based algorithm are associated with cells of the grid in the event space. Each cell of the grid is either associated with one group or is not associated with any group. Therefore matching algorithm should find which cell the event falls into, and take different actions according to whether the cell is associated with a group or not. If there is no group associated with the cell, the message must be delivered using unicast. If there is an associated multicast group is found, the message is usually delivered via multicasting to this group. However, if we can determine how many of subscribers included in the matched group are actually interested in the message, we may be able to avoid unnecessary communication by sending the message via unicast only to those interested in it. This optimization can help to noticeably reduce communication. We further discuss the effects of this optimization, as well as present experiment results, in a paper [50]. Figure 5.6 summarizes the matching algorithm in pseudo-code.

Matching in No-Loss Algorithm. No-Loss interest clustering stage forms a list $A \subseteq S$, consisting of the first n elements of S in the order of decreasing density w. When an event e arrives, the matching algorithm in Figure 5.7 is applied. The algorithm (using an S-tree, for example) finds multicast groups, as well as individual subscribers that are not included in the groups, whose interest rectangle includes the message.

```
1. If $e \in s$ where $s$ is a rectangle, $s \in A$
     let $s$ be such that $\forall t \in A : e \in t, w(t) \leq w(s)$
     (i.e. it is the rectangle with greatest density).
     Send message to multicast group formed of $u(s)$.
     Send message via unicast to subscribers in
     $(V_S \setminus u(s))$ that are interested in $e$.
2. Otherwise send message via unicast
   to subscribers interested in $e$.
```

Figure 5.7: Matching for No-Loss Algorithm

5.5 Experiments

5.5.1 Experiment Model

These results have been obtained on 3 different models of subscription interest and message distribution, but on the same network consisting of 600 nodes and the same distribution of subscribers on this network. In all three models 1000 subscription rectangles were generated.

We adopted the GT-ITM package from Georgia Institute of Technology [63] to generate a network with six hundred nodes according to a hierarchical scheme. This tool generates a hierarchical topology with transit blocks on top, stubs in the middle and nodes at the bottom. In our experiments, we first generate three transit blocks, with an average of five transit nodes in each block. Each transit node is connected on average to two stubs, and each stub has an average of twenty nodes.

For a given network topology, we generate subscriptions for each node. We first generate one thousand subscriptions with a $\{40\%, 30\%, 30\%\}$ breakdown for the three transit blocks. Within each transit block, there is a Zipf-like distribution for the number of subscriptions between all the stubs connected to this transit block. Subscriptions are distributed according to another (common) Zipf-like distribution within each stub.

The generated interval subscriptions are of the form $\{bst, name, quote, volume\}$. The first field bst, which could stand for *buy, sell*, and *transaction*, takes value B, S and T with probabilities $0.4, 0.4$, and 0.2, respectively. The center of the interval for the *name* field follows a normal distribution, with mean centered around the points specific to transit block number $(3, 10$ and $17)$, and standard deviation of 4. The length of this interval also follows a Zipf distribution. The intervals for the *quote* and *volume* fields are generated according to the same parametric distribution with different parameters. This parametric distribution takes values as follows:

$$\begin{array}{rl} (-\infty, +\infty), & \text{with probability } q_0, \\ [n, +\infty), & \text{with probability } q_1, \text{ and } n \sim N(\mu_1, \sigma_1), \\ (-\infty, n], & \text{with probability } q_2, \text{ and } n \sim N(\mu_2, \sigma_2), \\ [n_1, n_2], & \text{otherwise, center of interval} \sim N(\mu_3, \sigma_3), \\ & \text{interval length follows a Pareto distribution.} \end{array}$$

The parameters are given in the following table:

	q_0	q_1	q_2	μ_1, σ_1	μ_2, σ_2	μ_3, σ_3	c, α
price	0.15	0.1	0.1	9, 1	9, 1	9, 2	4, 1
vol	0.35	0.1	0.1	9, 1	9, 1	9, 2	4, 1

The generation of the subscriptions are intended to mimic the real life scenario that people's interests in stocks are centered around the current prices, the popularity of the information for different stocks have a Zipf-like distribution, and the popularity of the participants also have a Zipf-like distribution.

The publications are the points in the subscription space, which are generated according to a mixture of multivariate normal distributions. The different peaks in the multivariate normal distributions represent the multiple hot spots where events are published more frequently. We studied three scenarios, which are mixtures of one, four and nine multivariate normal distributions. The means and standard deviations for the single mode multivariate normal distribution are $(1, 1), (10, 6), (9, 2), (9, 6)$ for each of the four dimensions. The four mode distribution is constructed by sampling independent mixtures of multivariate normal distributions in each dimension. The mean and standard deviations for the first and fourth dimensions are $(1, 1)$ and $(9, 6)$, respectively. The second dimension is a normal random variable with parameters $(12, 3)$ with probability 0.5, and with parameters $(6, 2)$ with probability 0.5. The third dimension is a normal random variable with parameters $(4, 2)$ with probability 0.5, and with parameters $(16, 2)$ with probability 0.5. Similarly, for the nine mode distribution, the parameters for the first and fourth dimensions remain the same. The third dimension is N(4,3) with probability 0.3, N(11,3) with probability 0.4 and N(18,3) with probability 0.3. The fourth dimension is N(4,3) with probability 0.3, N(9,3) with probability 0.4 and N(16,3) with probability 0.3.

It should be noted that this experimental framework is flexible enough to accommodate other probability distributions for the subscriptions and publications. In the following study, we constructed 1000 subscriptions for the network with 600 nodes generated by the GT-ITM package.

We performed experiments on the generated testbed to evaluate the performance of the different schemes for forming multicast groups under two multicast frameworks: network supported and application-level multicast schemes. Network supported multicast requires the network routers to have the multicast capabilities, to be able to recognize multicast groups, and forward the information to the proper members of the group. There are two types of multicast algorithms currently used in routers: *dense mode* and *sparse mode* multicast. The implementations differ in the amount of state information and in the structure of the routing tree. We assume the *dense mode* multicast where the routing tree is a shortest path tree rooted at publisher. The amount of state information is proportional to both the number of publishers and the number of groups. In recent years, the study of multicast mechanisms has focused on the application level multicast, which does not require full support at the network routers. The members of a multicast group communicates through unicasts. They form a minimum

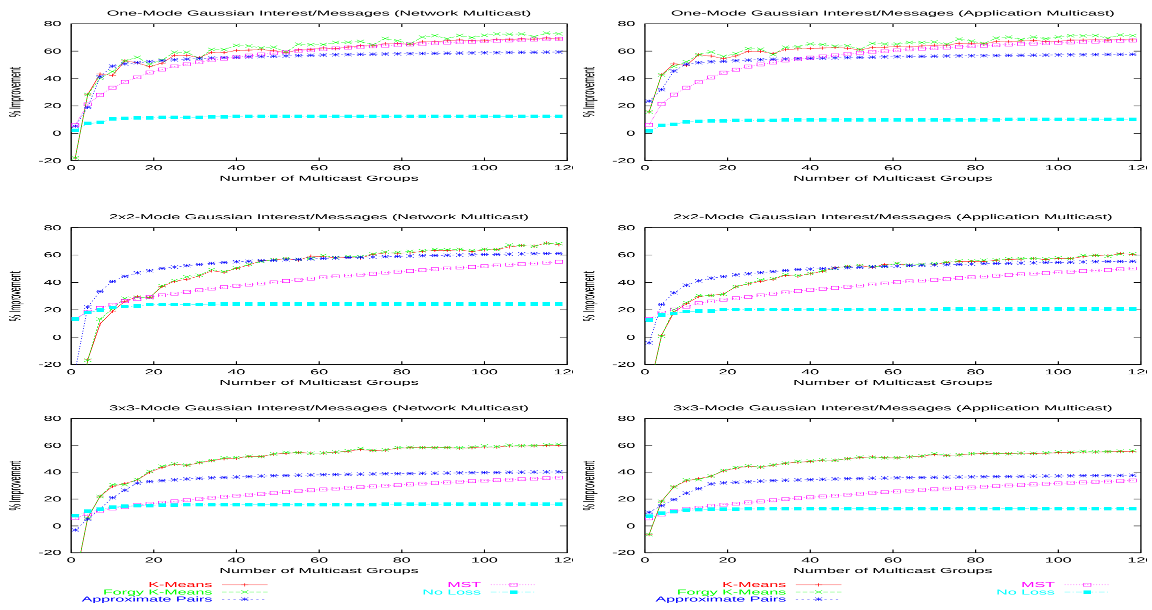


Figure 5.8: Algorithms Comparison.

spanning tree and forward the messages from one member to another through the minimum spanning tree. We also evaluated the impacts of the application level multicasts for the different algorithms.

5.5.2 Experiment Results

The following plots summarize simulation results, in which cost of communication was computed by summing up the edge costs (generated by GT-ITM package) on links on which communication takes place. Since absolute values of costs differ for different networks, we normalize the costs to make comparison easier. Thus the vertical axis in most plots shows "improvement percentage" over unicast. In other words, 0% communication cost improvement is achieved by using unicast to deliver each message. 100% cost improvement corresponds to the cost of delivering each message to a multicast group specially formed only of clients interested in this particular message, which is the best possible, and in the worst case requires as many as $O(2^{N_S})$ multicast groups. The goal of clustering algorithms is to get as close to this performance bound as possible, while using no more than K groups.

The absolute communication costs depend on different parameters. For the case of one-mode Gaussian subscription distribution, the unicast is 7139, the broadcast is 8536, and the ideal solution of network supported multicast is 1763.

Figure 5.8 shows how the communication cost changes as more groups become available for different algorithms. We are interested in algorithms that demonstrate monotone improvement, since intuitively when more groups can be formed we expect the algorithms to do a better job. Each plot is shown for network-level multicast and application level multicast. While application-level multicast results in slightly higher costs, the trend remains the same, and the algorithms that perform better under network multicast maintain their leadership under application-level multicast.

Performance of k-means is almost the same as one of Forgy k-means. Approximate pairs curve closely follows the curve of the pairs algorithm, as one can see in figure 5.11, so the latter is not shown in order to make plots readable.

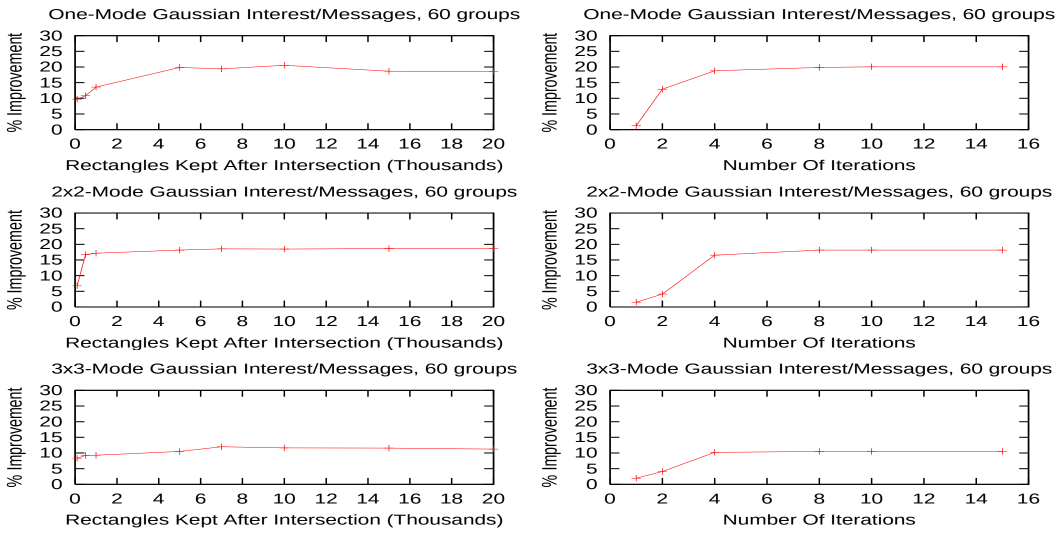


Figure 5.9: Effect of number of rectangles kept after intersecting and of number of iterations on no-loss algorithm.

The algorithms were run with the following parameters. K-means and Forgy used 6000 rectangles and maximum of 100 iterations (usually the number of actual iterations was less than 20). Approximate pairs algorithm was using only 2000 rectangles. The time complexity graph in figure 5.11 shows that in this case time complexity is almost the same as the one of K-means. No-Loss algorithm was run with 5000 rectangles kept after intersection and 8 iterations. Figure 5.9 shows how the algorithm depends on these parameters. MST was run with 6000 rectangles.

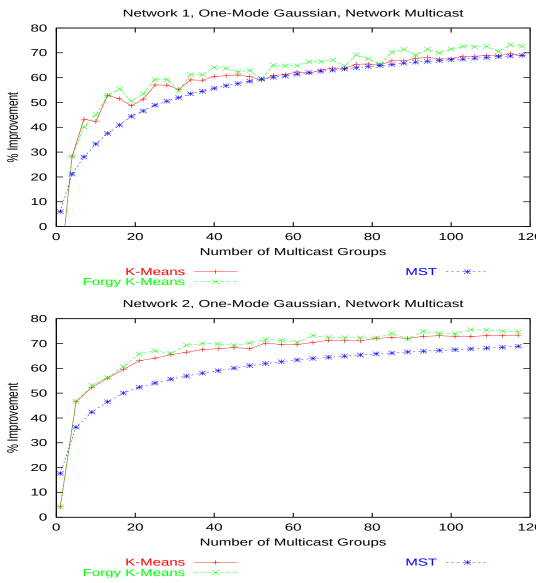


Figure 5.10: Performance of K-Means, Forgy and MST on 2 networks generated according to 1st model (one-mode Gaussian distribution).

Figure 5.10 shows that the trend of the algorithm performance does not depend greatly on the network topology. The left plot is taken from figure 5.8, and the right plot is obtained from simulation on a different network and subscription assignment generated according to the same parameters, but with different random seeds. While the grid-based clustering algorithms achieve local optimal solutions at best, in practice the solutions usually are good enough, reaching to 60% in this case.

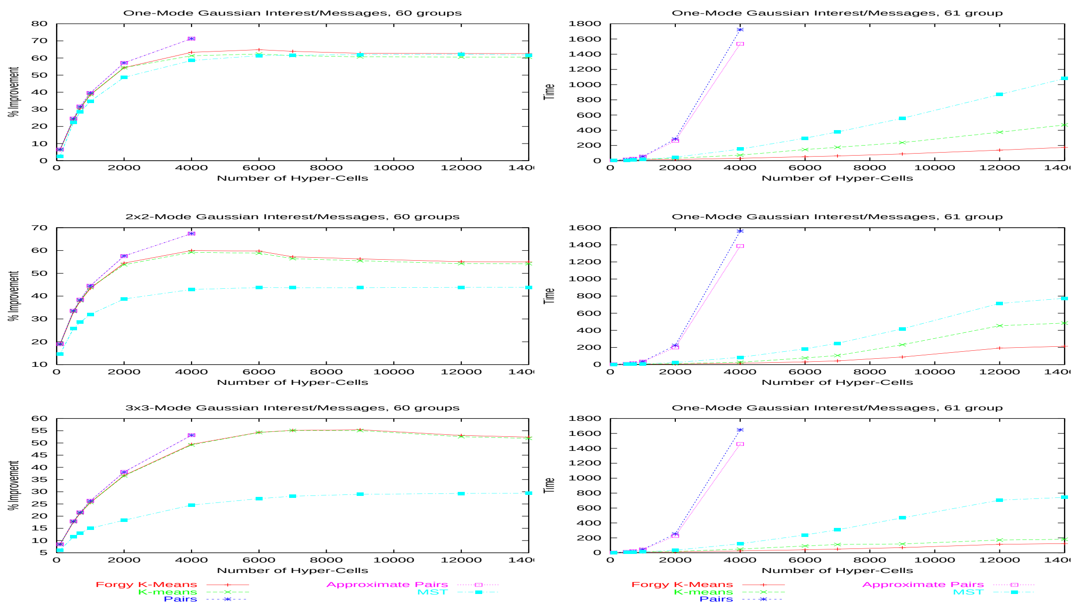


Figure 5.11: Effect of input data size on performance and running time of cell clustering algorithms.

Figure 5.12 combines the left and right plots of figure 5.11. Notice that Forgy and K-Means performance plots go down, when given more time. This only means that when more data is given to the algorithms. It consumes time without improving performance, It can even worsen the quality of solution, because the number of wasted messages becomes large. Data clustering algorithms usually make use of outlier removal techniques to avoid these problems. We leave the study of outlier removal effects for future work, simply noting here that the parameter that regulates the number of cells given to the clustering algorithm regulates its performance, as well as the time it takes to produce a solution, as shown in figure 5.11.

The results in figures 5.12 and 5.11 indicate that Forgy clustering should be preferred over k-means, since it gives better or comparable results faster. Cell-based clustering works well when the dimensionality of the event space is not too high, and the granularity of subscription interest is not too high. In this case these clustering methods should be preferred over No-Loss algorithm. We leave high-dimensional case for future study.

5.6 Conclusions and Discussions

We have considered the issue of efficient communication schemes based on multicast techniques for content-based publication-subscription systems. We have proposed and adapted clustering algorithms to form multicast groups for these content-based publication-subscription systems. These algorithms perform well in the context of highly heterogeneous subscriptions, and they also scale well. An efficiency of 60% to 80% with respect to the ideal solution can be achieved with a small number of multicast groups (less than 100 groups in our experiments).

Our experiments indicate that under several assumptions, which include high degree of regionalism of interest, and distribution of the interest close to the distribution of message parameters, the Forgy algorithm should be preferred for most purposes. Iterative clustering algorithms (K-means and Forgy) seem to be better suited for subscription dynamics, although other algorithms can be adapted as well. Hierarchical clustering algorithms (MST and Pairs) have worse performance than iterative clustering, but have the advantage of monotone improvement: when more multicast groups become available for the system to form, the new groups are formed by sub-division of existing ones. The analysis of influence of algorithm parameters on algorithm performance presented in this chapter must be helpful in practical implementation of the algorithms.

There are still many open issues to be addressed in the future. In what follows, we briefly describe a few of them.

1). Proposed algorithms can be adapted to make use of non-rectangular subscription interest sets by rounding the sets to appropriate shapes. While the no-loss algorithm relies on the rectangular interest set assumption, it is not very important to the other (grid-based) algorithms. The same grid data structures can be created without requiring the sets to be rectangles.

2). In many real-world scenarios each client is connected to an ISP via a single last-mile link. One simple variant of extending the transit-stub network topology [63] is the one in which higher costs are assigned to the last-mile links, since usually the last-mile links are the slowest and the most congested ones.

3). Evaluation of the algorithms on the real-world data would be very helpful for making decisions about implementation of the algorithms. For example, stock trading data can be used to simulate a stream of events coming into the system. However, information about the real structure of subscriptions is much harder to obtain.

4). In our experiments we did not simulate real network packets, implicitly assuming that there are no delays caused by congestion of network links. This is a reasonable assumption to make when the message size is small (1K or less). If the messages are of large sizes, a different type of communication cost evaluation must be used.

5). In reality clustering groups need to be constantly updated, since subscribers change their preferences, join and leave the network. Katz *et al* show that iterative clustering algorithms are well suited for dynamic changes in subscription structure [62]. Although a different type of distance measurement is used in that paper, the same iterative improvement strategy can be used to update the data structures of k-means and Forgy cell clustering algorithms.

6). In our model we have assumed that the matching of messages to groups or individual subscribers is done once: the first "intelligent" node that receives the message decides how to route it. In an alternative approach, described in several Gryphon project papers [2, 45], each intermediate node knows about the preferences of its neighbors, and matches each event against its specific data structures to find the neighbors to which the event must be forwarded next. This approach may save communication and matching time. However in practice dynamics of subscriptions require subscription changes to propagate quickly in the network, which makes this approach difficult to implement. Another related extension of pub-sub model requires the system to store messages at intermediate nodes, allowing off-line clients to retrieve information of interest when they connect to the system [13].

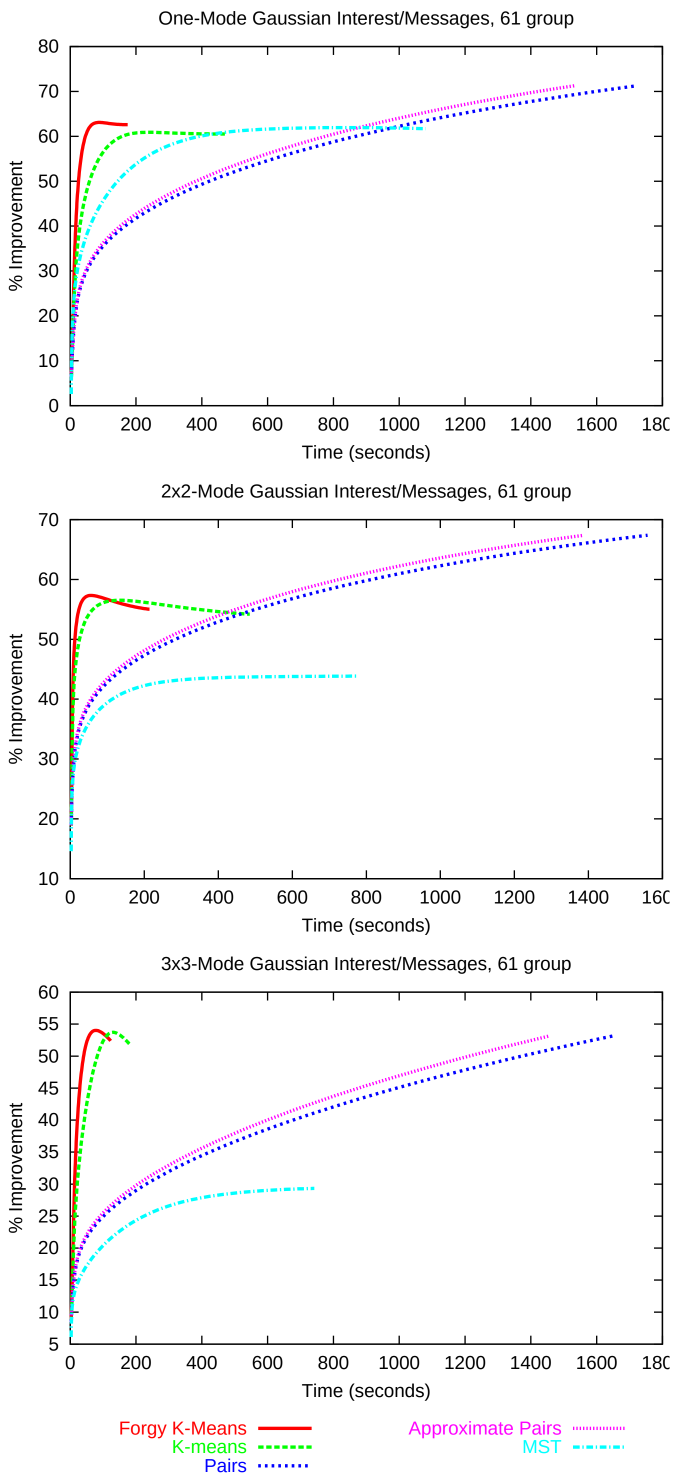


Figure 5.12: Quality of solution as a function of time.

Chapter 6

Conclusions and Future Work

With increasing capacity and availability of Internet connections inevitably grows interest in bandwidth-intensive applications, enabled by these changes. However due to the enormous size and the complex structure of the Internet, which consists of multiple interconnected sub-networks that are independently managed by different organizations, existing routing equipment cannot be updated quickly enough to provide most efficient use of existing resources. Multicast is an example of an efficient transport service, which cannot be implemented in existing infrastructure. Overlay multicast has emerged as universal solution. In addition to providing efficient transport for group communication, implementing multicast on application level allows to incorporate complex routing logic and sophisticated optimization algorithms, which require computational resources significantly beyond computing capacity of routers.

This dissertation describes methods that enable construction of reliable and scalable multicast systems. The main goal of our work was to achieve high scalability and efficiency of information dissemination using resources and technology currently available in the Internet.

6.1 Contributions

In this dissertation we addressed problems of finding optimal overlay routing, reliability and resiliency of overlay multicast systems, and grouping subscribers based on their interests.

Reliable Multicast Architecture. We show that reliable multicast overlays can be deployed on top of the current TCP/IP by adding a light set of application layer back-pressure mechanisms that guarantee both end-to-end flow control and reliability. We further show that such architectures are scalable, and can be used for group communications of large sizes and still provide a group throughput that is close to that of a single point-to-point connection.

Multicast Routing. We find that solving the problems of minimizing latency or maximizing throughput of overlay multicast routing is NP-hard. However, we develop approximation algorithms that allow to construct overlay multicast routing with throughput or latency within constant factor of optimal. Our algorithm for latency minimization requires that node-to-node delays are mapped to Euclidean space. With some mapping schemes described in literature this allows the algorithm to work without measuring delays for all pairs of participating nodes, which can be an important advantage in practice. For throughput minimization we develop methods based on volume algorithm and cutting planes that allow to solve the problem numerically.

Subscription Grouping. In a variant of information dissemination systems, called content-based publish-subscribe systems, multicast groups are not specified explicitly. Rather, group membership depends on the content of the message and on preferences specified at subscribing nodes. We propose algorithms that pre-configure multicast groups such that network congestion during message delivery is minimized.

6.2 Future Work

In our work we have addressed some of the important problems arising in overlay multicast applications. However, due to the complexity of these problems, there still are questions, answers to which will help to further improve performance of information dissemination systems. In the end of each chapter of this dissertation we discuss possible extensions of algorithms and analysis discussed in the chapter. Below we have selected research directions, that arise in connection with our work, and that we believe are most important for future development of efficient information dissemination systems.

1. The approximation algorithm for throughput maximization, described in Chapter 4, can be modified to adapt to changing network conditions. The algorithm proceeds in iterations, performing several edge exchange operations in each iteration. Edge exchange can be performed on a live system during broadcast, using techniques described in Chapter 2 to ensure uninterrupted transmission. This modification will allow the system to efficiently adapt to changing environment.

2. Trees are the simplest routing structures used for information dissemination. In particular, trees make it easy to ensure that transmissions are received in the same sequence by all nodes. Furthermore, each node in a multicast tree simply forwards several copies of the stream that it receives. However, more sophisticated structures spanning the overlay network may achieve better performance by utilizing network capacity more efficiently. In these networks end nodes must perform more complicated operations, forwarding only parts of the stream to downlinks.

3. Due to the fact that subscriber grouping problem is very complex by itself, we have analyzed it separately from the optimal routing problem. However, to achieve better performance it is necessary to consider the routing and grouping problems together, with the objective of minimizing network utilization, or with bandwidth constraints on edges. Given that both problems are NP-hard, when considered separately, the combined problem which has high practical importance for performance of publish-subscribe systems, is also very challenging.

4. The overlay multicast routing algorithms that we are proposing are centralized. In practice, however, some applications may require decentralized versions. Hence, it is of interest to develop decentralized algorithms for overlay routing.

5. Finally, in our analysis of approximation algorithms in several cases there is a gap between the best possible approximation factor and the approximation factor achieved by the algorithms. We believe that the approximability bounds can be improved, and, as usual with approximation algorithms, it poses an interesting and nontrivial question for future research.

Bibliography

[1] C. Aggarwal, J. Wolf, P. Yu, and M. Epelman. Using unbalanced trees for indexing multidimensional objects. *Knowledge and Information Systems*, 1:309–336, 1999.

[2] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. In *Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing (PODC '99)*, Atlanta, USA, May 1999.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows.* Prentice Hall, 1993.

[4] K. Almeroth. The evolution of multicast: From the mbone to inter-domain multicast to Internet2 deployment. *IEEE Network*, January/February 2000.

[5] S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[6] F. Baccelli, A. Chaintreau, Z. Liu, and A. Riabov. Achieving scalable and reliable multicast via back-pressured overlay networks. *Submitted to SIGCOMM 2004*, 2004.

[7] F. Baccelli, A. Chaintreau, Z. Liu, A. Riabov, and S. Sahu. Scalability of reliable group communication using overlays. In *Proc. of IEEE Infocom 2004*, 2004.

[8] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. Strom, and D. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the 19th International Conference on Distributed Computing Systems*, May 1999.

[9] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of ACM Sigcomm*, 2002.

[10] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient multicast using overlays. In *Sigmetrics*, 2003.

[11] F. Barahona and R. Anbil. The volume algorithm: Producing primal solutions with a subgradient method. Technical Report RC21103, IBM Research, New York, October 1997.

[12] N. Beckman, H. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An efficient and robust method for points and rectangles. In *Proceedings of the ACM SIGMOD Conference*, pages 322–331, May 1990.

[13] C. Binding, S. G. Hild, R. Hermann, and A. Schade. Intelligent messaging for the enterprise. Report RZ 3353 (#93399), IBM Research, 2001.

[14] C. Bormann, J. Ott, H.-C. Gehrcke, T. Kerschat, and N. Seifert. Mtp-2: Towards achieving the s.e.r.o. properties for multicast transport. In *Proc. of ICCCN 1994*, 1994.

[15] L. Caccetta and S.P. Hill. A branch and cut method for the degree-constrained minimum spanning tree problem. *Networks*, 37(2):74–83, March 2001.

[16] A. Chaintreau, F. Baccelli, and C. Diot. Impact of TCP-like congestion control on the throughput of multicast group. *IEEE/ACM Transactions on Networking*, 10:500–512, August 2002.

[17] Y. Chawathe, S. McCanne, and E. A. Brewer. Rmx: Reliable multicast for heterogeneous networks. In *Proceedings of IEEE Infocom*, 2000.

[18] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM'01*, San Diego, CA, August 2001.

[19] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM Sigmetrics*, June 2000.

[20] Akamai Corporation. Internet bottlenecks: The case for edge delivery services. Akamai whitepaper, 2000.

[21] ILOG CPLEX. `http://www.ilog.com/products/cplex/`.

[22] F. Baker et al. Requirements for IP version 4 routers. Request for Comments 1812, Internet Engineering Task Force, June 1995.

[23] F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Efficient matching for content-based publish/subscribe systems. Technical report, INRIA, `http://rodin.inria.fr/ pereira/matching.ps`, 2000.

[24] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *in IEEE/ACM ToN*, 5(6):784–803, December 1997.

[25] P. Francis. Yoid: Extending the internet multicast architecture. `http://www.icir.org/yoid/docs/yoidArch.ps.gz`, April 2000.

[26] M. Fürer and B. Raghavachari. Approximating the minimum degree spanning tree to within one from the optimal degree. In *Proc. of 3rd ACM-SIAM Symp. on Disc. Algorithms*, pages 317–324, 1992.

[27] M. Fürer and B. Raghavachari. Approximating the minimum-degree Steiner tree to within one of optimal. *J. Algorithms*, 12:409–423, 1994.

[28] M. Groetschel and C. L. Monma. Integer polyhedra associated with certain network design problems with connectivity constraints. *SIAM Journal on Discrete Mathematics*, 3:502–523, 1990.

[29] R. E. Gruber, B. Krishnamurthy, and Panagos. The architecture of the ready event notification service. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems Middleware Workshop*, 1999.

[30] D. S. Hochbaum (editor). *Approximation Algorithms for NP-Hard Problems*, B. Raghavachari. Chapter 7. Algorithms for Finding Low Degree Structures, pages 272–276. PWS Publishing Company, Boston, 1995.

[31] P. H. Hsiao, H. T. Kung, and K. S. Tan. Active delay control for tcp. In *Proc. of Globecom 2001*, San Antonio, TX, November 2001.

[32] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Engelwood Cliffs, New Jersey, 1988.

[33] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoe, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, October 2000.

[34] J. Könemann, A. Levin, and A. Sinha. Approximating the degree-bounded minimum diameter spanning tree problem. In *Proc. of APPROX 2003*, August 2003.

[35] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48–50, 1956.

[36] B.N. Levine and J.J. Garcia-Luna-Aceves. A comparison of reliable multicast protocols. *ACM Multimedia Systems*, 6(5):334–348, September 1998.

[37] J. Liebeherr and M. Nahas. Application-layer multicast with Delaunay triangulations. In *Global Internet Symposium, IEEE Globecom 2001*, November 2001.

[38] T. Magnanti and L. Wolsey. *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, chapter Optimal Trees, pages 503–615. North-Holland, Amsterdam, 1995.

[39] N. M. Malouch, Z. Liu, D. Rubenstein, and S. Sahu. A graph theoretic approach to bounding delay in proxy-assisted, end-system multicast. In *Tenth International Workshop on Quality of Service (IWQoS 2002)*, 2002.

[40] P. Mehra, A. Zakhor, and C. D. Vleeschouwer. Receiver-driven bandwidth sharing for tcp. In *Proc. of IEEE INFOCOM 2003*, 2003.

[41] C. N. Meneses, E. M. Macambira, and E. Uchoa. A branch-and-cut for the maximum degree-constrained connected subgraph problem. In *Proc. of the X Latin Iberian American Symposium of Operations Research and Systems (CLAIO)*, Mexico City, 2000.

[42] C. K. Miller. *Multicast Networking and Applications.* Addison-Wesley Pub Co, 1st edition, October 1998.

[43] NEONet. New era of networks. `http://www.neonsoft.com/products/NEONet.html`.

[44] T. S. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *INFOCOM'02*, New York, NY, June 2002.

[45] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman. Exploiting ip multicast in content-based publish-subscribe systems. In *IFIP/ACM International Conference on Distributed Systems Platforms*, New York, April 2000.

[46] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. Almi: An application level multicast infrastructure. In *Symposium on Internet Technologies*, March 2001.

[47] B. Quinn and K. Almeroth. IP multicast applications: Challenges and solutions. Request for Comments 3170, Internet Engineering Task Force, September 2001.

[48] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, August 2001.

[49] A. Riabov, Z. Liu, J. L. Wolf, P. S. Yu, and L. Zhang. Clustering algorithms for content-based publication-subscription systems. In *Proc. of ICDCS 2002*, Vienna, Austria, 2002.

[50] A. Riabov, Z. Liu, J. L. Wolf, P. S. Yu, and L. Zhang. New algorithms for content-based publication-subscription systems. In *Proc. of ICDCS 2003*, Providence, Rhode Island, 2003.

[51] A. Riabov, Z. Liu, and L. Zhang. Overlay multicast trees of minimal delay. In *Proc. of ICDCS 2004*, Tokyo, Japan, 2004.

[52] S.M. Ross. *Introduction to Probability Models.* Academic Press, San Diego, sixth edition edition, 1997.

[53] E. M. Schooler. *Why Multicast Protocols (Don't) Scale: An Analysis of Multipoint Algorithms for Scalable Group Communication.* Ph.d. dissertation, computer science department, California Institute of Technology, September 2000.

[54] S. Shi. *Design of Overlay Networks for Internet Multicast.* Ph.D. Thesis, Washington University in St. Louis, August 2002.

[55] S. Shi and J. Turner. Placing servers in overlay networks. Technical Report WUCS-02-05, Washington University, 2002.

[56] S. Shi and J. S. Turner. Multicast routing and bandwidth dimensioning in overlay networks. *IEEE JSAC*, 2002.

[57] S. Shi and J. S. Turner. Routing in overlay multicast networks. In *IEEE INFOCOM*, New York City, June 2002.

[58] S. Shi, J. S. Turner, and M. Waldvogel. Dimensioning server access bandwidth and multicast routing in overlay networks. In *The 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001)*, Port Jefferson, New York, June 2001.

[59] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160, Diego, California, United States, 2001.

[60] G. Urvoy-Keller and E. W. Biersack. A multicast congestion control model for overlay networks and its performance. In *NGC*, October 2002.

[61] Z. Wang and J. Crowcroft. Bandwidth-delay based routing algorithms. In IEEE Globecom'95, November 1995.

[62] T. Wong, R. Katz, and S. McCanne. An evaluation of preference clustering in large-scale multicast applications. In *Proceedings of IEEE Infocom 2000*, Tel Aviv, March 2000.

[63] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE Infocom 1996*, San Francisco, 1996.

[64] B. Zhang, S. Jamin, and L. Zhang. Host multicast: A framework for delivering multicast to end users. In *Proceedings of IEEE Infocom*, 2002.