CONSTRUCTING EFFICIENT SELF-ORGANISING APPLICATION LAYER MULTICAST OVERLAYS

A THESIS SUBMITTED TO

THE UNIVERSITY OF KENT

IN THE SUBJECT OF COMPUTER SCIENCE

FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

By

Su Wei, Tan

October 2005

Abstract

Application Layer Multicast (ALM) is an alternative to IP multicast, which has yet to achieve a widespread deployment in the Internet. ALM places multicast primitives directly in the multicast members, i.e. end systems, which use an overlay topology on top of the physical network for multicasting. The overlay consists of unicast connections between the members, hence bypasses the need for multicast support at the infrastructure level. The overlay structure used is a key factor that determines the efficiency of an ALM solution.

This thesis investigates efficient techniques to build both low cost (i.e. low resource usage) and low delay ALM trees. We focus on self-organising distributed proposals that use limited information about the underlying physical network, limited coordination between the members, and construct overlays with bounded branching degree subject to the bandwidth constraint of each individual member.

This work begins with a detailed simulation evaluation of existing ALM proposals chosen from different classes. This has resulted in enhancements to some existing proposals as well as a set of observations that could be used to assist future development of ALM proposals. As part of the evaluation, we devise a simple centralised greedy heuristic for creating low diameter degree-bounded mesh overlays, for benchmarking the class of distributed proposals. With the insights collected from the evaluation effort, we develop proposals for a distributed heuristic to build low delay delivery trees for both one-to-many and many-to-many multicast. For one-to-many multicast, we propose MeshTree, which is based on the observation that distributed delay optimisation can be trapped by the greedy problem and delay-cost trade-off. MeshTree addresses the problems by embedding the delivery tree in a degree-bounded mesh containing many short links. For many-to-many multicast, we consider a multiple shared trees approach to strike a balance between the performance and quality trade-off of the conventional single shared tree and multiple source-specific trees approaches. We show that both our proposals perform well compared to existing proposals. Both of these proposals are based on a mesh-based overlay creation and maintenance framework which we have developed. The framework offers quick failure recovery mechanism to address the inherent dynamic characteristic of the ALM system.

Acknowledgements

First of all, I must acknowledge the Computing Laboratory at Kent, and the University of Multimedia (Malaysia) for funding this work.

I would like to thank my thesis supervisor, Dr. Gill Waters, for her guidance, support and patience. I am also grateful to Dr. John Crawford, for many insightful sessions of discussion on multicast. He is in practice my second supervisor. Similar words of gratitude go to Prof. Peter Linington, who replaces Gill as my supervisor after her retirement, for reading and commenting on my final draft. He has no doubt added a touch of extra confidence to the outcome of this three year long research.

I would also like to thank my thesis committee member, Dr. David Shrimpton, for his valuable comments during several occasions.

There are friends at Kent that made this journey a joyful one. In particular, I thank Rodolfo, Chris, Gift, Ben, Philip, Verapol and Gansen for their precious friendships.

Studying abroad has never been an easy experience. I am indebted to my brother and sisters, for their unconditional supports. Knowing that they are there to well look after my parents, it enables me to focus primarily on my research. Finally, I thank my loving wife Peijun, who makes this all the more meaningful.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet came to life as a communication medium for the research community. In its early days, the Internet was mainly used to exchange data between research laboratories. The birth of world wide web in the early 90s proved to be a killer application that pushed the technology to a wider audience. This triggered a rapid growth of the Internet, and advances in the underlying technologies. The arrival of broadband facilitated cheaper and better access to the Internet by millions of households. More and more bandwidth-intensive applications, such as broadcast of audio and video contents, are now possible over the Internet.

A popular broadcast event (e.g. live concert of a popular pop star, or sport event like a world cup final) can attract an audience of millions. Today, the data exchange service provided by the Internet is mainly based on unicast, i.e. point-to-point between two computers. However the broadcast event requires a point-to-multipoint service. With the current unicast service, the data source needs to send a copy of data to each of the recipients. In other words, if the broadcast involves a million recipients, the source would have to repeatedly transmit the same packet a million times. Naturally, this injects redundant traffic into the network, as well as overloading the data source. The network or IP (Internet Protocol) layer multicast offers an efficient transmission mechanism for point-to-multipoint delivery. With IP multicast, the data source only sends one copy of data which is then replicated and forwarded as necessary by the network routers to the recipients. Each physical link will see only a single copy of the data.

In general, multicast is useful for applications that involve communication between multiple parties, i.e. group communication. Examples of such applications are video conferencing, distance learning, multi-party games, distributed simulation, etc. It is clear that this diverse set of applications requires a different kind of support from the underlying system. As described by Shi [86], we can roughly classify the key requirements as follows: the amount of data to be delivered (bandwidth requirement); timeliness of their delivery (latency requirement); the reliability of their delivery (reliability requirement); the number of participants that send data (multi-source requirement); the number of recipients to be reached (scalability requirement); and the frequency of members joining or leaving the group (dynamics requirement). Table 1.1 provides a summary of these requirements for some of the example applications. Undoubtedly, this diverse set of requirements challenges the unicast-based services, and calls for native multicast support in the Internet.

Unfortunately, despite over 15 years of active research, a global-scale deployment of IP multicast infrastructure has yet to be seen. This is due to a number of issues such as multicast address allocation, interdomain multicasting, security and the difficulty in providing higher level functionality. Owing to this, in recent years, the research community has revisited the case of providing multicast services at the application layer. This approach is often termed application layer/level multicast, end-system or end-host multicast. In this thesis, we will refer to it as Application Layer Multicast (ALM).

In ALM, multicast functionality is implemented by end systems. The end systems are organised into a multicast overlay topology, and deliver data over the overlay edges which are unicast connections. This thesis investigates efficient techniques for creating low cost and low delay ALM overlays. We are particularly interested in practical distributed techniques.

In the next section, we discuss IP multicast, including its history, problems faced and potential solutions. Section 1.2 explores ALM along with its advantages and challenges with respect to IP multicast. In Section 1.3, we briefly state the contributions of this work. The organisation of the remainder of this thesis is given in Section 1.4.

	Multi-source	Scalability	Dynamics	Bandwidth	Latency	Reliability
Video Conferencing	all	small	low	medium	critical	no
Distance Learning	one or few	medium	low	medium	critical	no
Multi-party Games	all	large	high	low	critical	yes
Distributed Simulation	all	large	low	high	depends	yes
Internet TV/Broadcast	one	huge	high	high	critical	no

Table 1.1: Application characteristics for group communications [86]

1.1 IP Multicasting

Early multicast support was constrained within a single local area network (LAN) domain. It was not until the late 80s that Deering and Cheriton [25] introduced multicast for internetworks and extended LANs. This marks the beginning of IP multicast.

IP multicast is based on an *open* service model: there is no mechanism that restricts the hosts from creating a *group*, receiving data from or sending data to a group. Each multicast group is identified by a 32-bit class-D multicast address (range from 224.0.0.0 to 239.255.255.255). To receive data from a multicast group, a host must first join the group. To do so, the host contacts its edge router using the Internet Group Management Protocol (IGMP) [34]. Once the host joins the group, it will receive all data addressed to the group, regardless of the identities of the sources. The data sources do not need to be members of the group. In addition, group members can join and leave the group at will, with no need to notify other members of, or senders to, the group. In short, an IP multicast group is not managed. Similar to its unicast counterpart, IP multicast datagrams use best-effort delivery and are inherently unreliable.

Multicast-capable routers participate in a multicast routing protocol which manages the connections between the routers. Typically, the connections are in the form of a single shared tree or a set of source-rooted trees. This depends on the routing protocol used. For example, DVMRP (Distance Vector Multicast Routing Protocol) [25], MOSPF (Multicast extension for Open Shortest Path First routing protocol) [65] and PIM-DM (Protocol Independent Multicast — Dense Mode) [1] create separate trees rooted at each source while CBT (Core Based Tree) [5] and PIM-SM (Protocol Independent Multicast — Sparse Mode) [31] create a single shared tree.

The interest in IP multicast took off with the creation of the MBone — the multicast backbone [30]. MBone began its life as an overlay network which interconnected islands of multicast-capable LANs using unicast tunnels, whose end points are workstations that run the `mrouted` routing daemon. `Mrouted` is capable of receiving unicast-encapsulated multicast packets, and forwarding the packets to appropriate out-going interfaces computed by DVMRP. In March 1992, the MBone achieved a remarkable milestone by carrying the first Internet audiocast from an Internet Engineering Task Force (IETF) meeting in San Diego to 20 sites.

DVMRP creates multicast trees rooted at each of the data sources. Each tree is built in a *flood* and *prune* manner. Specifically, a source transmits a packet to its edge router, which in turn replicates and forwards the copies on all out-going interfaces. When a router receives such a packet, it performs a reverse path forwarding (RPF) [24] checking to decide if the packet will be discarded or forwarded. If the packet is received from the interface that the router uses to reach the source, the router will forward the packet to all other interfaces. Packets arriving from other interfaces will be discarded silently. Each router periodically uses the IGMP to discover the existence of group members in its local network. If there is no group member, a router will transmit a “prune” message towards the source on the RPF interface. An intermediate router forwards the prune message along the path towards the source if it receives prune messages on all its interface except the interface towards the source. Owing to this, every router needs to keep state for each existing multicast group, regardless of whether the router itself actually belongs to the group. DVMRP is also known as a dense mode protocol, as it assumes a dense availability of members where pruning occurs infrequently. Other protocols that can also be classified as dense mode are MOSPF and PIM-DM. Dense mode protocols are not scalable due to the high volume of broadcast traffic generated and state information needs to be maintained at each router.

The rapid growth of the MBone has called for the development of a new class of protocols — sparse mode protocols — which are designed to work well when the members are sparsely distributed. Sparse mode protocols require the edge routers with group members to explicitly join in the multicast trees, as opposed to the dense-mode's flood and prune mechanism. Two popular examples are CBT [5] and PIM-SM [31], with PIM-SM seeing a wider deployment [2]. Both CBT and PIM-SM build a tree rooted at a special node, which is called the *core* in CBT and the *Rendezvous Point* (RP) in PIM-SM. For each multicast group, CBT uses a bidirectional shared tree where packets can be originated from any point of the tree; while PIM-SM uses a unidirectional shared tree where packets are first sent to the RP, which in turn delivers the packet down the multicast tree. PIM-SM allows the edge routers to switch to the source rooted trees when the perceived traffic exceed a certain threshold. The sparse mode protocols generally improve the scalability when compared with dense mode protocols.

The original MBone was built as a flat topology. Its continuous growth has resulted in problems such as large routing state and difficulties in management. Consequently, the multicast community has began to deploy hierarchical, interdomain multicast routing.

Current interdomain multicast routing is based on the following set of protocols: MBGP[1]/PIM-SM/MSDP. MBGP (Multicast extension for Border Gateway Protocol (BGP)) [12] provides a set of multicast extensions for the unicast-based BGP [79] so as to separate unicast and multicast policies for interdomain routing. PIM-SM manages trees for multicast members within each domain. In order to allow members to join to a group with sources located in remote domains (with remote RPs), the group-to-RP mapping must be advertised to all edge routers in other domains. This is done by MSDP (Multicast Source Discovery Protocol) [33] which distributes this mapping and announces sources to RPs in different domains, using a flooding mechanism.

The MBGP/PIM-SM/MSDP protocol suite is viewed as a near term solution, due to some scalability concerns over the flooding mechanism used in MSDP. In addition, MSDP also introduces long join latency and is not suitable for an environment with highly dynamic membership [2]. In the near future, the Border Gateway Multicast Protocol (BGMP) [56] is expected to provide interoperability between multicast routing protocols in different domains.

BGMP creates bidirectional shared trees between domains. The success of BGMP depends on a collision-free address allocation scheme. Currently, two models are being considered by the IETF: (i) a static address allocation and assignment scheme called GLOB [64]; and (ii) the Multicast Address Allocation Architecture (MAAA) [94] which consists of a set of protocols for allocating addresses dynamically. Multicast address allocation is difficult as the address space provides no geographical or topological meaning. The static nature of GLOB is inherently not scalable. On the other hand, the MAAA protocol set is complex, and potentially not scalable [27]. More importantly, it does not provide a solution to the problem of address starvation if multicast becomes a popular interdomain service.

[1] MBGP is also known as BGP4+.

The address starvation problem is addressed by the next version of IP, IPv6. IPv6 drastically increases the address space (for both unicast and multicast)[2] at the expense of changing the IP packet structures. Thus, it requires changes to the routing infrastructure, and is expected to be deployed in the Internet in an incremental manner. An extended time period may be needed before it fully replaces the current version of IP.

1.1.1 IP Multicast Issues and Alternative Proposals

IP multicast is attractive as it is the best bandwidth saving technique to deliver a message to multiple destinations. In order to take advantage of this, it requires changes at the infrastructure level. This results in a chicken-and-egg problem: the Internet service providers are waiting for killer applications that drive the demand for multicast support, while the users or application developers are waiting for widely available multicast support for them to exploit the technology. Besides, there are also a number of outstanding issues that slow down the deployment pace of IP multicast.

- *Interdomain multicasting and address allocation*. A key to making multicast a universally available technology is the success of interdomain multicast routing. However, as described previously, the current solution (MBGP/PIM-SM/MSDP) is rather complex and has scalability concerns. On the other hand, the long term proposal (BGMP) requires a strict address allocation scheme, which is itself a complicated problem. Address collision can result in cross traffic between two different multicast sessions. This poses a serious inefficiency risk for multicast receivers as packets from other sessions must be processed and dropped.

- *Security concerns*. As mentioned in the previous section, IP multicast is based on an open model where the groups are not managed. In particular, any host can transmit data to any group. This lack of access control makes the network vulnerable to flooding attacks by malicious sources. While such an attack could happen in the unicast service, the fact that a single message will reach a large number of recipients could exacerbate the situation.

- *Scalability and complexity*. IP multicast requires routers to maintain state information for each multicast group. This has introduced additional complexity to the IP layer, and raised serious scalability issue. At the higher layer, IP multicast's best-effort property also imposes additional difficulties in providing features such as congestion control, flow control and security, compared to its unicast counterpart.

To reduce the complexities of MBGP/PIM-SM/MSDP and BGMP, as well as addressing additional multicast-related issues (e.g. security and management), the multicast community has looked at a new

[2] IPv6 uses 128 bits for both unicast and multicast addresses, compared to 32 bits used in current version of IP (IPv4).

class of multicast model — Root Addressed Multicast Architecture (RAMA) [2]. RAMA is based on the observation that most multicast applications have a single source or have an easily identifiable primary source. RAMA identifies a multicast group using the <primary source, multicast group address> mapping. As each host has a unique address, this solves the address allocation problem. Two RAMA proposals are Express Multicast [45] and Simple Multicast (SM) [73].

Express is specifically designed for single-source applications. It creates a unidirectional tree rooted at the source. The source-specific approach allows strict control of the data source, thus addresses the potential attacks from an arbitrary source. SM provides a many-to-many service over a bidirectional shared tree, rooted at the primary source. While both proposals address some of the above issues, they are still in the development state. Furthermore, both proposals require changes at the infrastructure level. In particular, Express requires the deployment of IGMP version 3, which is still under development; while SM requires changes to the packet header. Hence, a widespread adoption of these proposals may still take some time.

There are other alternative solutions for providing multicast service over the Internet. In [29], El-Sayed et al. provide an excellent survey of what they called Alternative Group Communication Service (AGCS) proposals. AGCS proposals can be classified into the following groups:

- Proposals that are based on a unicast/multicast reflector. In this category, end hosts with unicast-only service contacts a reflector. The reflector serves as a gateway between a multicast-capable network and a set of unicast hosts. An example proposal is mTunnel [71]. The main advantage of such proposals is simplicity. However, as a reflector potentially serves a large number of hosts, it creates traffic hot spots near the reflector, and may not be scalable.

- Proposals that are based on a specific group communication routing service. These proposals typically require changes to the underlying routing infrastructure. One example is XCast [14] which is designed for applications with a very small member set. XCast includes an explicit list of multicast destinations in each packet. The packets are delivered using the existing unicast routing service. By carrying the whole destination list, the routers are relieved from keeping multicast state information. However, it requires support from the routers so as to examine the packet header, and to create and forward copies as necessary to the destinations.

- Proposals that create an automatic overlay topology. This group refers to proposals that create self-organising multicast overlay directly at the end systems, i.e. the focus of this thesis. It will be discussed in a greater detail in the following text.

The well-known end-to-end arguments [83] suggest that, a functionality should be (i) pushed to higher layers if possible; (ii) unless implementing it at the lower layer can achieve larger performance benefit that out-weighs the cost of additional complexity at the lower layer. Conventionally, multicast is designed based on the second consideration, where multicast primitives are implemented in the routers (network layer). As IP multicast is still not widely available, the research community began to revisit the first consideration, that is to push the multicast functionality to a higher layer — the application layer. Thus was born the so-called Application Layer Multicast (ALM).

1.2 Application Layer Multicast

The principle concept behind ALM is to implement multicast primitives directly at the end systems. The end system could be an end host, a proxy server or an edge router. The end systems are organised into a logical overlay network, and multicast data using the overlay edges which are unicast connections. In this way, ALM bypasses the need for network layer multicast support, and hence offering a ready-to-deploy solution. We may recall that the MBone is also an overlay network. What differentiates the MBone from ALM is that the MBone is tightly integrated with IP multicast, and it has a static topology with occasional manual reconfiguration. ALM can work without native multicast support, and it is often referred to as a class of self-organised proposals which construct, improve and repair the overlays without manual intervention.

ALM proposals began to emerge in year 2000, with pioneers such as End System Multicast (ESM) [21], Scattercast [18] and Yoid [36]. Chu et al.'s ESM introduced Narada, a self-organising protocol that creates an ALM overlay directly at the end hosts. Narada is designed for small-scale many-to-many multicast applications. Chu et al. provide the first quantitative comparison of ALM with IP multicast and naive unicast transmission. Their results prove the case for ALM. Yatin Chawathe's Scattercast advocates an infrastructure support for ALM, where a set of proxies running ALM protocols are deployed in the network. The proxies are connected by ALM overlays, and the clients (end hosts) subscribe to nearby proxies. Scattercast uses Gossamer, which has a number of similarities with Narada, to construct the ALM overlays. Paul Francis's Yoid, on the other hand, focuses on the architectural aspect of ALM. In particular, Yoid's architecture consists of three protocols: Yoid Distribution Protocol (YDP), Yoid Identification Protocol (YIDP) and Yoid Tree Management Protocol (YTMP). These protocols work together to provide generic content distribution.

Following these works, more and more ALM proposals emerge each year. For examples, ALMI [72], HMTP [109], NICE [7], TBCP [62], switch-trees [43], Scribe [15], Zigzag [95], etc. The performance of ALM is highly related to the structure of the overlay used. Unsurprisingly, the majority of the proposals concentrate on strategies for building efficient ALM overlays. The aim of this thesis can be summarised as: to understand the strengths and weaknesses of existing proposals; hence to offer improvements and new proposals to better construct ALM overlays.

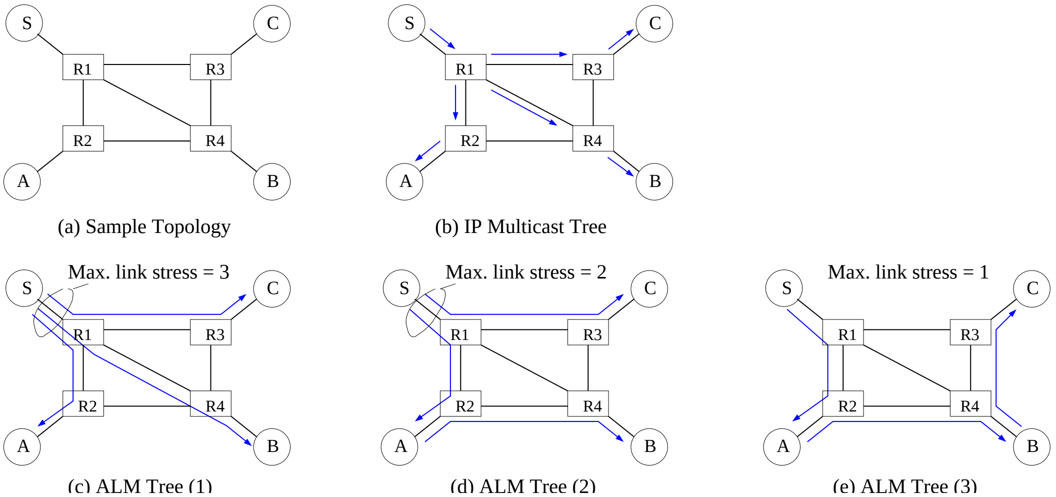


Figure 1.1: Contrasting ALM, IP multicast and naive unicast transmission

1.2.1 Comparing ALM, IP Multicast and Naive Unicast Transmission

In this section, we compare ALM, IP multicast and naive unicast transmission with the help of an example, and discuss the advantages and challenges of ALM.

To contrast ALM with network layer multicast, consider the sample topology in Figure 1.1 (a) where $R1$ to $R4$ are routers while S, A, B and C are end systems. Assume that S wishes to send data to all other nodes. Figure 1.1 (b) depicts the network layer multicast tree built by protocols such as DVMRP. We can see that the data is delivered to the receivers via the (reverse) shortest path tree rooted at S. Router $R1$ receives a single copy of the packet and forwards the replicated copies along the interfaces to $R2$, $R3$ and $R4$. From the figure, we can see that at most one copy of a packet is sent over any physical link. Also, the perceived delay at each recipient is as though the data were sent directly by unicast.

In ALM, data packets are replicated at end systems, and are delivered from one end system to another end system using a unicast connections. Collectively, these unicast connections interconnect the end systems into an overlay network. The resultant overlay can be in the form of a tree or a mesh which serves two purposes: as a control topology and as a topology for data distribution. Normally, the control topology is a mesh which provides redundant paths between the members. On the other hand, the data topology is usually a tree to ensure loop-free routing. Figure 1.1 (c), (d) and (e) shows three examples of ALM multicast trees on the sample topology. A naive implementation of ALM could degenerate to the unicast transmission as shown in Figure 1.1 (c). For this case, the physical link from S to $R1$ carries 3 copies of a transmission by S. This packet redundancy at a physical link is called link stress, which is defined as the number of identical copies of a packet carried by a *physical* link [21]. In other words, the naive unicast transmission could lead to link stress that is as high as the number of recipients on the link nearest to the source. Moreover, the data source needs to transmit as many copies of a packet as there are recipients. This could potentially overload the source.

We could build trees that have lower worst case physical link stress. For example, the trees in Figure 1.1 (d) and (e) have worst-case stress of 2 and 1 respectively (note that we consider a physical link between two nodes, e.g. A and $R2$, is consisting of two unidirectional links, $\langle A, R2\rangle$ and $\langle R2, A\rangle$). However, while these two trees improve link stress as compared to naive unicast transmission, they increase source-to-receiver delay. For instance, the tree in Figure 1.1 (e) shows that to get to C, a packet needs to be relayed from S to A, then from A to B, and finally from B to C.

The above examples illustrate two salient features in ALM: potential high link stress and longer end-to-end delay compared to the network layer multicast. Consequently, the quality of an ALM solution is often measured relative to the network layer solution. Also, the cost or network resources used by the overlay tree can be calculated as the sum of the delays of all *overlay links* in the tree [109]. Ideally, an overlay should have low delay, stress and tree cost.

1.2.2 ALM Advantages

Despite the potentially poorer data delivery quality, ALM has a number of advantages over network layer multicast. First of all, ALM provides a ready solution for deploying multicast services over the Internet. Unlike IP multicast, it allows application specific naming, and thus does not require a globally consistent naming scheme. By using a suitable overlay, ALM could avoid the high link stress and source overloading issues cause by naive unicast transmission. While ALM has emerged only in recent years, there have been a number of successful real-life applications. For example, the End System Multicast project [75] uses ALM for live events broadcasting and the Jungle Monkey project [76] focuses on distributed files sharing.

Until recently, the network layer multicast is only sparsely available in the form of islands of multicast network. ALM offers an opportunity to interconnect such multicast islands into a global multicast network. In fact, several projects such as Universal Multicast [110] and Broadcast Federation [19] are working towards this goal. Even in an IP multicast capable network, ALM can also be useful to offer service for groups with members that are sparsely available, or groups with a very small number of members. Maintaining many such groups with IP multicast could prove to be costly in terms of router state and the additional processing required.

By using unicast connections, ALM is transparent to the network layer. This helps to maintain the stateless nature of the underlying network. And, as ALM is based on unicast, it may be possible to leverage some well studied unicast solutions to achieve simple error, flow and congestion control in multicasting.

ALM overlays can also have a flexible structure. This allows novel structures to be used to simplify the maintenance of the overlay. For example, the Delaunay triangulation protocol [58] creates overlay formed by many Delaunay triangulations; NICE [7] builds overlays with multiple levels clusters. These overlays can be maintained with low overhead. In addition, application-specific optimisation objectives can be easily integrated with the overlay structure.

1.2.3 Challenges in Building Efficient ALM Overlays

We have seen that ALM offers several advantages over network layer multicast. Most importantly, it allows rapid deployment of multicast services over the Internet. However, there are several additional challenges faced by an application layer approach. Here we discuss issues related to building efficient distribution overlay structures.

- *Quality of data delivery*. In ALM, data is delivered using unicast from one end system to another end system. As discussed previously, this results in redundant data traffic and prolonged end-to-end delay.

- *Capacity constraint and heterogeneity*. ALM requires the end systems actively to take part in data distribution. Recently, Saroiu et al. [84] conducted a measurement study on the end systems participating in two popular peer-to-peer file sharing systems: Napster and Gnutella [40]. Their results show that the end systems are highly heterogeneous in terms of bandwidth capability. In particular, the end systems access the Internet with a variety of access technologies, e.g. dial-up, Cable, DSL, T1 or T3. Overall, the available bandwidth for a large number of end systems (about 70%) is less than 3 Mbps. As each data connection consumes some bandwidth (depending on the application, see Table 1.1), an end system could only support a certain number of other systems. In other words, the branching degree (or fan-out) of an overlay delivery structure has to be restricted. It is important that the overlay must honour the degree limitation for each individual node.

- *Robustness*. End systems are usually more susceptible to problems like system failure than the network routers. In addition, the end systems may join or leave the group at will. As the overlay is directly formed from these systems, the overlay structure is necessarily changing over time. An ALM solution should quickly adapt to these changes in a robust manner.

- *Limited topology knowledge*. Unlike network routers, end systems have little or no knowledge about the underlying network topology. Such knowledge however is the key to building efficient overlays. In order to obtain information about the interesting network metrics (e.g. delay, bandwidth) between the nodes, end-to-end measurement techniques are often used. However, each measurement process may consume substantial network bandwidth, and will affect the scalability of the solution (see below).

- *Scalability*. In order to be useful for large-scale applications, an ALM solution needs to be scalable. The scalability is closely related to how an overlay is constructed and how it is maintained. In particular, the overlay construction and maintenance should not require global coordination among the members, and should work with limited knowledge of the network.

1.3 Thesis Contributions

This thesis focuses on practical distributed solutions for generating ALM trees that have low delay and low cost. A solution is practical in the sense that it addresses the previously mentioned challenges. More precisely, it needs to create and maintain a degree-bounded overlay subject to the bandwidth constraints at each individual node; it should work with minimum coordination between the members and limited knowledge of the underlying network topology; also, it needs to be responsive to the changes in the overlay membership. The focus of this thesis is the efficiency of the overlay structures used for data delivery. Upper-level services such as the reliability of data traffic, security or congestion control are beyond the scope of our work.

The contributions of this thesis can be summarised as follows.

- A detailed comparative study of some existing ALM overlay construction proposals using simulation. The proposals considered optimise either tree cost or delay, and they encompass a wide variety of overlay creation strategies. The comparison includes the performance in both one-to-many and many-to-many data delivery models. Unlike other work in similar areas, we perform an in-depth analysis of the overlay construction process of the various proposals. By analysing the strengths and weaknesses of these protocols, we identify enhancements to some existing proposals (e.g. switch-trees [43] and TBCP [62]), as well as directions for our own proposals. As a by-product, we developed a simple yet flexible and extensible simulator (called `ALMSim`) for our evaluation.

- A simple centralised heuristic for the minimum delay degree-bounded overlay mesh creation problem, which is NP-complete. Source-specific trees can be obtained from the mesh with a shortest path algorithm, e.g. Dijkstra's algorithm [23]. We refer to the algorithm as GreedyMesh, and use it as a benchmark in our evaluation of many-to-many ALM proposals. GreedyMesh may also be useful for creating overlays for small-scale delay-sensitive applications. We are not aware of any other algorithms that attempt to generate low diameter degree-bounded meshes.

- A distributed mesh-based framework which provides basic procedures for creating and maintaining a degree-bounded overlay tree. Its mesh-based approach offers fast and robust failure recovery. The framework is generic and can be used to improve the robustness of some existing ALM proposals.

- A proposal for a distributed algorithm called MeshTree, which creates low delay, degree-bounded overlay trees. MeshTree is inspired by two issues, namely the greedy problem and delay-cost trade-off that we observed in existing distributed proposals. Trees created by MeshTree are useful for real-time single-source applications with large receiver sets. For example, critical event notification (e.g. synchronisation and update data), and real-time stock quotes and updates (e.g. Yahoo!

MarketTracker [106]).

- A proposal for a distributed algorithm that uses multiple shared trees for multi-source multicasting. The proposal challenges the convention of using either a single shared tree or a set of source-specific trees. It strikes a balance between these two approaches in terms of delay performance and message overhead. The proposed solution is suitable for large-scale interactive multi-source applications. For example, distributed simulation, distance learning or multi-party gaming (see Table 1.1). For these applications, low latency delivery is of paramount importance as old information is quickly invalidated by newer information.

1.4 Thesis Structure

The rest of this thesis is organised as follows.

Chapter 2 begins with a broad discussion of ALM and related work. This includes the various system architectures, the service models considered in ALM overlay construction and a survey of prior ALM solutions. We also discuss related areas such as ALM in mobile networks, Internet distance measurement services and peer-to-peer file sharing.

Chapter 3 presents the system model, assumptions and the simulation design, including topologies, multicast member selection, performance metrics and the `ALMSim` simulator, used throughout the thesis.

In Chapter 4, we introduce the GreedyMesh algorithm. We compare it with a number of centralised algorithms. Some of these algorithms will be used as benchmarks for our evaluation of distributed solutions in later chapters.

In Chapter 5, we conduct a detailed performance comparison of a number of distributed ALM overlay construction proposals discussed in Chapter 2. We choose techniques that cover a wide range of overlay creation and maintenance strategies. The strengths and weaknesses observed help in the development of our own proposals.

Chapter 6 introduces the mesh-based framework for creating and maintaining a degree-bounded overlay tree in a distributed manner. The framework is important as it is the basis of our proposals in later chapters. To illustrate the working of the framework, we apply it to a degree- and delay-bounded, low cost tree creation problem.

In Chapter 7, we study the degree-bounded, low delay trees creation problem, which is useful for applications that require low source-to-receivers delay. By analysing the limitations of some existing distributed proposals, we propose MeshTree, and compare it against proposals that performed well in our comparison study in Chapter 5.

Chapter 8 investigates the multiple shared trees approach for many-to-many multicasting. This is motivated by the observation that the single shared tree approach is scalable but yields trees with poor end-to-end delay, while the source-specific trees approach has good delay but poor scalability. We compare the proposal with other existing work, and also examine its quality and overhead trade-off.

Finally, Chapter 9 concludes this thesis with a summary of the thesis contributions and suggestions for future work.

Chapter 2

Background

In the previous chapter, we introduced the concept of Application Layer Multicast (ALM). This chapter serves to provide further details on ALM, as well as to discuss some related research areas.

This chapter consists of three parts. In the first part (Section 2.1 – 2.3), we discuss several design issues in creating ALM overlays, which range through system architectures, optimisation objectives and service models. In the second part (Section 2.4 – 2.6), we review several representative ALM proposals, focusing on various aspects of building efficient ALM overlays. The third part (Section 2.7) describes other related research areas: ALM in mobile networks, Internet distance measurement and peer-to-peer networking. Section 2.8 concludes this chapter.

2.1 ALM System Architectures

The study of ALM has often centred around two basic system architectures: (i) end systems only; and (ii) a hybrid of end systems and network layer multicast.

As described in Chapter 1, an end system can be an end user's machine or a more powerful proxy server. Hence, we can further classify this architecture based on the type of end system that actively takes part in the overlay construction. This results in a pure end host-based architecture and a proxy-based architecture.

In the end host-based architecture, an ALM protocol can be directly implemented at the end users' machines. Examples of proposals based on this approach are Narada [21], HostCast [57] and Banana Tree Protocol [42]. The main advantage of this approach is that no specific infrastructure machines need to be installed in the networks, and it therefore provides instant deployment at low monetary cost. However, there is a trade-off between the deployment cost and the data delivery quality. Poor delivery quality may be expected as the users' machines normally have low access bandwidth (e.g. dial-up users). These systems are also more prone to failure problems which affect the reliability of the overlay.

In the proxy-based architecture, an organisation that provides value-added services deploys proxies at strategic locations on the Internet. End hosts then attach themselves to these proxies to send and receive data. As specialised machines are used, this solution can provide better services, but at a higher cost. Scattercast [18], OMNI [9] and AMCast [86] are examples of such systems.

While the above proposals enable global deployment of multicast applications, they do not attempt to exploit the existing multicast infrastructure. Consequently, several proposals [19, 110] consider a hybrid architecture where end systems are used to interconnect islands of multicast networks. In this case, the end systems can be either an end host or a dedicated proxy. In [19], Chawathe and Seshadri propose Broadcast Federation, which uses specialised nodes called broadcast gateways (BG) to interconnect different broadcast networks (BN). Each of the BNs implements its own independent multicast protocol, such as the DVMRP [25], Core Based Tree (CBT) [5], Protocol Independent Multicast (PIM) [31], Express Multicast [45], Simple Multicast [73] or even an overlay multicast solution. The Universal Multicast proposed by Zhang et al. serves as a general framework that can work with various ALM protocols to build dynamic unicast tunnels to connect IP multicast islands. The key concept is to elect one or more designated members (DMs) from the multicast members in each IP multicast island. The DMs act as representative which interconnect the islands using an ALM protocol, e.g. HMTP or Narada.

2.2 Optimisation Objectives

The optimisation objectives of a solution are closely related to the upper-level applications. Following are some commonly considered objectives.

2.2.1 Bandwidth

Bandwidth-intensive applications such as video distribution require high-bandwidth delivery paths. In a study on quality of service unicast routing [101], Wang and Crowcroft define bandwidth as a concave metric, which means that the bandwidth of a path is determined by the bottleneck link (i.e. the link with the smallest bandwidth) in the path. This suggests that to achieve a high-bandwidth delivery tree, the bottleneck links need to be placed as close as possible to the leaves. In [22], Cohen and Kaempfer show that the problem of finding an overlay tree with the maximum bottleneck bandwidth is NP-complete. Jannotti et al. propose a distributed protocol called Overcast [48] that attempts to attain bandwidth-optimised trees.

Degree Constraints One difficulty in bandwidth optimisation is that end-to-end bandwidth measurement is an expensive operation. Normally, it requires some amount of data to be transferred from one node to another node for an extended period. A recent Internet traffic measurement study [13] suggests that congestion normally happens at the access network, rather than the backbone network. Hence, most existing techniques relate the bandwidth limitation of a node to the node's access bandwidth capability. In particular, given the source rate and the access bandwidth, a node can place a limit on the number of out-going flows in the delivery structure (see Chapter 3 for details). More precisely, the multicast tree needs to be degree-bounded based on each individual node's capacity constraint. All the proposals in this thesis follow this degree constraint requirement. Typically, degree constraint is considered with other objectives, such as tree cost and/or delay (Section 2.2.2 and 2.2.3).

Multiple Trees versus Single Tree Typically, data is delivered over a single multicast tree. Due to this, the forwarding load is only carried by the interior nodes in the tree, while the leaf nodes contribute no resources. Recently, several projects [16, 70] have begun to exploit the spare resources available at these non-contributing nodes. The main concept is to deliver the contents that are encoded into multiple sub-streams using techniques such as multiple description coding (MDC), and these sub-streams are then delivered over multiple trees formed by the members. In this manner, each node may become the interior node in one tree and become the leaf node in the other trees, hence the forwarding load is shared more evenly. More importantly, as the previously unused resources can now be used, the overall throughput is often higher than the single tree delivery mechanism.

This thesis considers the single tree delivery mechanism. While the multiple trees approach provides better throughput, it may not be applicable to all applications. For example, applications that require timely data delivery. With multiple trees, each receiver may need to wait for the arrival of data from all trees. This will delay the processing of the data. In addition, for applications with a low data rate, it is possible that the overhead of maintaining multiple trees is not worthwhile.

It is important to note that the multiple trees discussed here should not be confused with the multiple shared trees concept to be presented in Chapter 8. Here, the multiple trees concept is referring to the transmission of data from a data source to the receivers, using several trees simultaneously. While the multiple shared trees approach also creates more than one tree, a data source only selects one of these trees for data delivery.

2.2.2 Tree Cost or Network Resource Usage

In network layer multicast, tree cost is determined by the summation of individual link costs. The cost of a physical link could be an administratively configured value or the bandwidth cost, which is known by the multicast routers. In application layer routing, an overlay link is a unicast tunnel between two nodes, which spans across multiple physical links. For simplicity, the cost of an overlay link is often quantified as the summation of the delays of physical links it traversed, i.e. the end-to-end delay of the overlay link. This is because it is easier to estimate the distance between two end systems, rather than to accumulate cost between them. Throughout this thesis, we treat the delay of an overlay link as its cost. Corresponding to this, the overlay tree cost can be given as the summation of the delays of all overlay links in the tree:

$$overlay\ tree\ cost = \sum_{\forall e \in E_T} d(e) \tag{2.1}$$

where E_T is the set of overlay tree links and $d(e)$ is the link's delay. Usually, the round-trip delay between the nodes is used.

The overlay tree cost also provides a simplified view of the network resource used by the overlay tree. The network resource usage is defined in [21] by Chu et al. as the sum of the product of all physical links' stress and the links' delays. We recall that an overlay link is composed of a series of physical links. Equation 2.1 adds up the delays of all overlay links. This effectively sums up the delays of the physical links for as many times as they are used in the overlay tree, i.e. the network resource used by the overlay tree. A low cost tree is suitable for delay insensitive applications such as bulk file transfer. The Host Multicast Tree Protocol (HMTP) [109] is designed for this type of applications.

The optimum solution for this problem is the minimum cost spanning tree, which can be calculated using the Prim's or Kruskal's algorithm [23]. However, if the delivery tree needs to be degree-bounded, the problem (i.e. degree-bounded minimum cost tree) is NP-hard [50].

2.2.3 Delay

Delay is important for applications that require timely delivery of their data. For example, streaming media, interactive multi-party network gaming, video conferencing and distance learning. We are interested in providing low end-to-end data delivery from the source(s) to all the recipients.

Obviously, a source-rooted shortest path tree offers the best delay performance. In an overlay, this degenerates to the naive unicast transmission where the source node is the central hub of a star topology. This, however, is impractical as it not only burdens the source, it also stresses the physical links close to the source. Unfortunately, problems associated with creating low delay degree-bounded trees are normally NP-hard.

Before we discuss some of the degree-bounded tree creation problems, we first define several ways of measuring the delay performance for an overlay structure (either a connected mesh or spanning tree).

- *Diameter*. The diameter for a connected overlay is the longest of all shortest path distances (via the overlay) between any pair of overlay members. If the overlay is in the form of a tree, the term *tree diameter* will be used. Tree diameter is used for shared tree where data can be originated from any point on the tree, i.e. in the case of many-to-many data delivery.

- *Root-diameter*. This is specific for a tree structure, which is the maximum shortest path distance (via the tree) from the tree root to any tree members. Root-diameter is used when the tree root is the sole data source, i.e. in the case of one-to-many data delivery.

- *Average delay*. This represents the overall average delay performance observed by the members. It is calculated as the ratio between the total delay observed by the members for each data source and the total number of source-receiver pairs.

For single-source applications, Li and Mohapatra [57] propose HostCast to create low root-diameter trees for delay-sensitive streaming media applications; in [54], Kostic et al. design ACDC to achieve trees which are root-diameter bounded, and have low tree cost; and in [9], Banerjee et al. consider the problem of minimising the average delay to the recipients in a proxy-based architecture. In terms of multi-source applications, Shi et al. consider a centralised algorithm for the problem of minimising the tree diameter. Distributed proposals, such as Narada [21] and Gossamer [18], that create per source trees are designed to achieve low average delay between the members. All these proposals build degree-bounded trees.

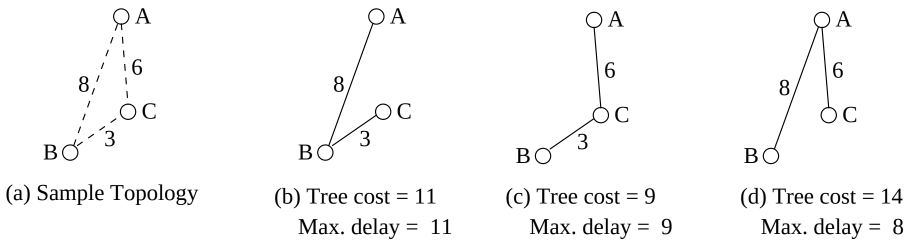
We are particularly interested in the following three NP-complete problems, which will be studied in detail in Chapters 4, 6 and 7 respectively.

1. *Minimum diameter degree-bounded subgraph problem*. The objective of this problem is to create a connected degree-bounded subgraph (mesh) which has minimum delay between the nodes. Degree-bounded source-specific trees can be obtained from the mesh for many-to-many data delivery. In Chapter 4, we devise a greedy heuristic called GreedyMesh for the problem. GreedyMesh is mainly used as our benchmark for many-to-many ALM proposals.

2. *Root-diameter- and degree-bounded, minimum cost tree problem*. This problem aims to find a degree-bounded tree that has the minimum tree cost, and has root-diameter that is within a given delay target. For this problem, we provide a distributed solution called dbMeshTree in Chapter 6.

3. *Minimum root-diameter degree-bounded tree problem*. In this problem, the objective is to obtain a degree-bounded tree that has the minimum root-diameter. It is thus useful for single-source applications that require timely delivery. In Chapter 7, we propose MeshTree, a distributed tree building proposal for this problem.

Triangle Problem An important issue in distributed tree cost and delay minimisation is the triangle problem [109, 57]. We explain this problem with Figure 2.1. In this figure, the value next to a link represents the link's cost. Figures 2.1 (b) – (d) depict three different ways to organise the nodes (A, B and C) into a tree rooted at A. The triangle problem is depicted by Figure 2.1 (b) which includes the longest path, A – B – C, in the configuration. Figures 2.1 (c) and (d) illustrate the minimum cost and the minimum delay (root-diameter) configurations respectively. For tree cost minimisation, it is easy to see that configuration (b) uses more network resources than configuration (c); for delay minimisation, configuration (b) also yields a higher root-diameter than configuration (d). Hence, it is crucial that a distributed solution can detect and overcome the ineffective triangle for its optimisation objective. Comparing configurations (c) and (d) suggests an interesting property: there still exists a trade-off between



(a) Sample Topology (b) Tree cost = 11 Max. delay = 11 (c) Tree cost = 9 Max. delay = 9 (d) Tree cost = 14 Max. delay = 8

Figure 2.1: Triangle problem

the tree cost and the delay measures. Indeed, our simulation results in later chapters confirm this simple observation.

2.2.4 Scalability

Scalability is a major concern for any large-scale applications such as multi-party network games, streaming media and distributed simulation. It is highly related to the approach taken to construct the overlay (e.g. centralised or decentralised), the structure of the overlay (e.g. tree or mesh) and the way that the overlay is maintained (i.e. the message overhead). Typically, a decentralised solution can scale better than a centralised one; and a tree having fewer links than a mesh is often viewed as a more scalable structure. However, the key to scalability is determined by the volume of state information and number of messages transmitted between the members. We are interested in solutions that can scale up to a reasonably large group size, e.g. thousands of members.

2.3 Multicast Service Models

A tree is the natural structure for multicasting. By definition, it is loop-free. Hence, multicasting in a tree can be done by flooding: when a node receives a message from one of its tree links, it replicates and sends the messages out via the other tree links that emanates from it. Typically, a tree can be used as a unidirectional tree or a shared tree based on the service models under consideration: (i) *one-to-many*; or (ii) *many-to-many*.

The one-to-many model is used by single-sender applications such as file distribution or media streaming from a well-known source. In this model, a source rooted unidirectional tree that connects all the receivers is used. Examples of ALM solutions that are based on this model are Overcast [48], HostCast [57], and Zigzag [95].

The many-to-many model is for multi-sender applications such as video conferencing or multi-party network games. Three types of delivery tree are normally considered for this model: (i) source-specific trees; (ii) a bi-directional group shared tree; and (iii) a unidirectional group shared tree.

The source specific trees are a set of unidirectional trees each rooted at one of the data sources. The data items from each of the sources are mapped onto their respective trees. Examples of ALM protocols that use this type of delivery trees are NICE [7], Narada [21] and Gossamer [18]. In contrast, the bi-directional shared tree model uses only a single tree in data delivery. The tree is rooted at a well-known node that is typically called the core [5]. Protocols that consider this delivery mechanism are Yoid [36] and HMTP [109]. While the bi-directional shared tree allows all on-tree members to act as data senders, the unidirectional shared tree allows only the tree root to do so. In this case, the actual data source forwards the data to the tree root, which in turn forwards it onto the multicast tree. Overlay solutions that considered this type of tree are Bayeux [112] and Scribe [15]. The main reason to use such a tree is to impose strict access control [44] which is important for applications such as topic-based publish-subscribe applications [15].

Recently, Zappala et al. [108] examined the case for multiple shared trees for many-to-many network layer multicasting. In Chapter 8, we present our strategy which is based on this concept in the context of ALM.

2.4 General Working of ALM Overlay Construction

This section describes the working of a typical overlay construction technique. We then review some existing ALM proposals in the next two sections, with an emphasis on techniques to be examined in later chapters.

In theory, the overlay network can be viewed as a fully connected graph, as each node can reach every other node in the network via unicast connections. However, for practical reasons, only a small subset of the overlay links should be included in the ALM overlay. Hence, the basic functionality of an ALM overlay construction technique is to identify these links and maintain the connectivity of the overlay. The resultant overlay can be in the form of a tree or a mesh which serves two purposes: as a control topology and as a topology for data distribution. Normally, the control topology is a mesh which provides redundant paths between the members. On the other hand, the data topology is usually a tree to ensure loop-free routing. Following this, we use the term *overlay* to refer to the structure created and maintained by an ALM solution, rather than the complete overlay graph. In addition, we will use the term *tree-based* to refer to proposals that maintain only a tree structure (as both data and control topologies), while we use the term *mesh-based* for proposals that maintains a mesh structure.

Figure 2.2 illustrates the working of a typical overlay construction technique, which consists of two phases: (i) the joining phase; and (ii) the maintenance phase. The joining phase refers to the process where a newcomer is joining an overlay, i.e. growing the overlay. The maintenance phase manages the connectivity of the overlay. In the figure, a to g are existing members of a multicast session and x is a newcomer.

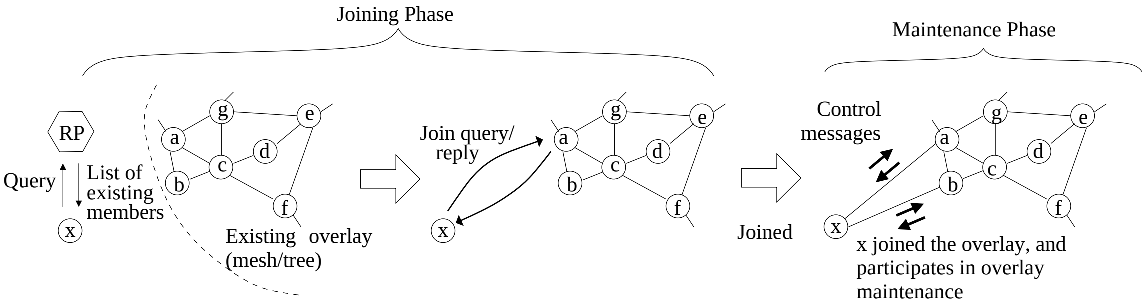


Figure 2.2: General working of an ALM overlay construction proposal

Typically, ALM protocols assume a well-known Rendezvous Point[1] (RP) [36] to bootstrap new members onto an existing session. The RP acts as a query server to provide existing members' information to newcomers. Optionally, the RP also participates in functions like access control and overlay partition healing. The identity of the RP can be obtained via an out-of-band mechanism, such as web publishing or email.

The overlay can be built in a centralised or a decentralised manner. In the centralised approach, the overlay creation and management are performed by a central controller. Newcomers can learn about the identity of the controller from the RP. The controller uses full knowledge of all members, and potentially the performance metrics (e.g. delay and bandwidth) between the members, to compute a high quality overlay. The computed overlay structure is then distributed to the members in the form of the neighbouring relationships (i.e. links) between the nodes. On receiving such information, the members will initiate connections to their assigned neighbours. Once attached to the overlay, the members enter the maintenance phase. In particular, they monitor the connections with their respective neighbours. Any changes to the neighbours' status (e.g. leave/fail) will be reported to the controller for it to reconfigure the overlay.

On the other hand, in the decentralised approach, the overlay creation and management are done by the members in a distributed fashion. Consider Figure 2.2: newcomer x first requests a list of members from the RP. From the given list, x then selects one or more members as joining targets and sends to each of them a joining request message. When a node, say a, receives a request message, it performs an admission control decision for the requesting node. The main decision criterion is whether it has spare capacity for a new link. Other application-specific criteria (e.g. optimisation objectives) may also be used. Once joined on the overlay, x enters the maintenance phase and begins to monitor the status of its neighbours. Any changes to the neighbours' status is normally handled by x itself (i.e. the affected member).

The next section discusses a number of centralised proposals, and Section 2.6 explores some decentralised proposals. Figure 2.3 provides a simple classification of the proposals to be discussed.

[1] Note that the similarity between the RP for ALM and the RP for PIM-SM [31] is only in their names.

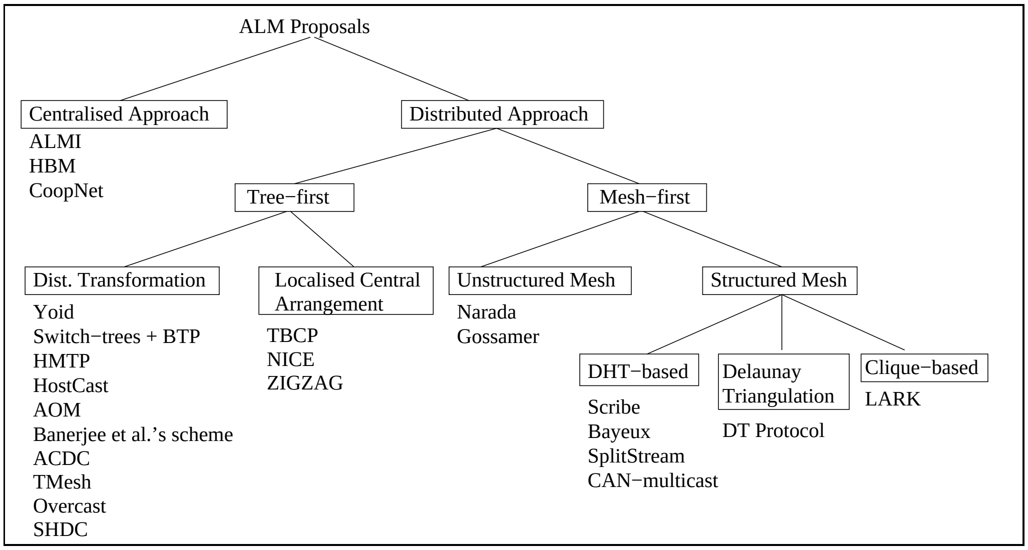


Figure 2.3: Classification of ALM proposals

2.5 Centralised ALM Approaches

ALMI, Host-based Multicast (HBM) and CoopNet are three distinctive examples of centralised ALM protocols.

ALMI [72] ALMI is designed by Pendarakis et al. as a middleware to support many-to-many multicast applications of relatively small size (several tens of members). In ALMI, a central controller arranges the members into a monitoring graph. Each member is responsible for monitoring the performance metrics (i.e. round trip delay) of the links between the member and its neighbours in the graph. Periodically, the collected information is reported to the controller, which will compute the multicast distribution tree from the updated monitoring graph. In [72], ALMI is used to create minimum cost spanning trees.

HBM [80] In HBM, the set of overlay nodes is divided into two groups: *core members* (CMs) and *non-core members* (non-CMs). CMs form the core distribution topology while non-CMs graft to the core topology as leaves. The notions of CM and non-CM are based on the estimated stability of the nodes. Stable nodes are categorised as CM while less stable nodes are non-CM. Unlike ALMI, HBM uses a complete graph as its monitoring graph. In other words, each node needs to monitor the links between itself and the rest of the nodes. HBM considers several overlay structures, which include tree, bus, star and hybrids of these basic structures. It also includes a redundant links addition algorithm to improves the robustness of the delivery structure.

CoopNet [70] CoopNet provides a resilient technique to deliver streaming contents from a single source. It makes use of multiple description coding (MDC) and multiple distribution trees to achieve robust data delivery. MDC is a method of encoding audio and/or video signal into $M > 1$ separate streams, or descriptions, such that any subset of these descriptions can be received and decoded into a signal with distortion (with respect to the original signal) depending on the number of descriptions received: the more descriptions received, the better the quality of the reconstructed signal. In CoopNet, each sub-stream is delivered using a different distribution tree (formed by the same set of members). This delivery mechanism provides robustness as the probability of all streams concurrently to fail to arrive is very low when M is sufficiently large.

In addition to protocol design efforts, several works have also investigated efficient overlay construction algorithms. In [87], Shi et al. consider the problem of creating degree-bounded minimum diameter overlay trees, which is NP-hard. They propose a greedy heuristic called the Compact Tree (CPT) algorithm. The CPT algorithm uses an incremental approach like Prim's algorithm to grow the tree. In particular, starting with the root node, CPT adds new nodes to the partial tree one at a time until all nodes are added. A node is selected for addition if it provides the smallest increment in the objective function (i.e. the tree diameter) to the partial tree without causing degree violation in the tree. In [60], Malouch et al. consider the delay-bounded version of the problem in a mixed system with both end hosts and proxies. They propose a heuristic which is similar in nature to CPT.

2.6 Decentralised ALM Approaches

While the centralised approach simplifies overlay construction and management, it may not scale well. In particular, the central controller needs to keep track of the information about all members, which is highly dynamic. In addition, it creates a single point of failure problem. Hence, a greater number of works investigate decentralised solutions.

In the decentralised approach, members actively participate in the creation and maintenance of the overlay. Typically, the existing decentralised proposals are classified into two groups: mesh-first and tree-first. The following two subsections review some of the proposals from these two groups. Unless specifically mentioned, the proposals to be discussed allow the overlay nodes to specify their own degree constraints so as to create degree-bounded overlays.

2.6.1 Tree-first Protocols

Tree-first protocols arrange the members directly into a tree structure. Examples of tree-first protocols are Yoid [36], HMTP [109], switch-trees [43], Banana Tree Protocol (BTP) [42], TBCP [62], Overcast [48], NICE [7], Zigzag [96], HostCast [57], ACDC [54], AOM [104], TMesh [100], SHDC [63] and Banerjee et al.'s scheme [9].

The next subsection (2.6.1.1) first provides a general discussion of issues related to the construction and optimisation of overlay trees. Subsections 2.6.1.2 and 2.6.1.3 then discuss in more detail the above-mentioned proposals.

2.6.1.1 Overlay Tree Construction and Optimisation

A tree is defined by a set of parent-child relationships between the nodes. All on-tree nodes (except the root) must have a parent node. The root node is normally the first member who initiates the multicast session, or the data source in single-source applications.

In a typical distributed tree building protocol, it is the responsibility of a node to locate its parent. Two techniques are normally used by a newcomer to locate an initial parent: (i) random selection; and (ii) select the tree root. If the initial join request to a potential parent fails, the newcomer has to retry with another candidate. In addition, an on-tree node may also wish to discover other nodes for overlay maintenance and improvement. We note that the initial member list obtained from the RP may contain only a partial view of existing members, and it may be out-of-date due to changes in the membership. We list several ways to discover additional on-tree nodes below.

- *Root path*. A root path for a node x is a list of nodes in the route via the overlay from x to the root. A root path can be formed in the following manner. At the tree root, the root path contains only the root node. This information is sent to each of the root's children which will append their own node identifier to the list they receive. These nodes in turn send their root path to their children. As this process continues down to the leaf nodes, all nodes will know their paths to the root. The root path is useful for loop detection. Specifically, when a node receives a root path that contains its own node identifier, it knows that there is a cycle in the tree and thus invokes a loop resolution solution. Most tree-based protocols maintain a root path.

- *Distributed depth-first searching (DFS)*. In this technique, a node searches down the tree by exploring one branch of the tree at a time as in the conventional depth-first traversal. The question of which branch will be chosen depends on the search criterion used.

- *Local region scoping*. In this technique, an on-tree node is constrained to know only the nodes that are within a predefined scope. The scope is normally small to reduce the communication overhead. For example, a local region for a node may consist of its parent, siblings and grandparent.

- *Tree random walk* [36]. This technique is used to find a random non-descendant node in a tree. In this technique, a node, say x, first transmits a discovery message to its parent. The parent will then randomly forward the message to one of its tree neighbours (other than x), which will continue to propagate the message in the similar way. As a tree is free of loops, the message will never reach a descendant of x. Useful information (e.g. an address) can be added to the message during its propagation. The message has a time-to-live field that determines its search scope. The last node that receives the message will reply to the originator.

- *RanSub* [53, 54]. RanSub utilises the structure of a tree to collect and disseminate information about the tree members. It consists of two phases: (i) *collect*; and (ii) *distribute*. In the collect phase, information (e.g. addresses, nodes state) is propagated from the leaf nodes to the root along the tree. At each node, the information is mixed, shuffled, and limited to a certain size before being sent to the parent node. Once the root has collected all the information from its children, it redistributes them down the tree in the distribute phase. Due to the reshuffling, RanSub makes sure that, over time, the information will be uniformly distributed to all nodes.

- *Gossipping*. While the above techniques are closely related to the tree structure, the gossip-style discovery mechanism is applicable to a general topology. In this technique, each node keeps a list of known members. Each member is associated with an integer as its *heartbeat counter*. Periodically, a node, say x, increments its own heartbeat counter and randomly picks another node, y, from the list, and sends to y its member list. Node y will merge the received list with its own list, and adopt the maximum heartbeat counter for each member. To avoid propagation of false information (i.e. "dead" members), a node periodically purges nodes whose heartbeat counters have not increased after an extended time from the member list. The gossip-based technique is applicable in several problem areas, e.g. ALM [18], failure detection system [97] and resource discovery [41].

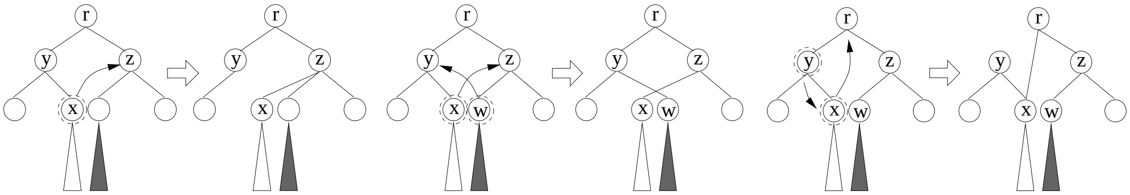
We note that some of these techniques may be used together. For example, HMTP [109] uses a node selected from the root path to begin a DFS for a potential parent.

Basically, the overlay is periodically reconfigured to improve and adapt to the changing environment. In particular, a member observes the performance of the links with its existing and potential neighbours. It then consults a set of optimisation rules to decide whether to add a new link and/or remove an existing link. For a tree, it is important to make sure that adding a new link does not create a loop, or that deleting a link does not partition the tree.

For our comparison study in Chapter 5, we classify the overlay reconfiguration techniques into the following two groups:

- *Distributed transformation*. In this technique, nodes perform independent transformation decisions to reconfigure the tree. There are two basic transformation operations: (i) switching and (ii) swapping. Switching refers to the process that a node switches from its existing parent to a new parent (see Figure 2.4 (a)), while swapping refers to the process where two nodes exchange their parent nodes simultaneously (see Figure 2.4 (b)). These operations can be combined to obtain more complex transformations such as the promotion operation in Figure 2.4 (c). We note that during the above operations, the subtree of a switching node is not changed, as shown in the figures.

- *Localised central arrangement*. This refers to the case where a representative node is responsible



(a) Switching: x switches from the original parent, y, to z

(b) Swapping: x and w swap parents

(c) Promotion of x to become parent of its original parent, y

Figure 2.4: Type of transformations in an overlay tree

for the overlay configuration for nodes within a local region. In other words, it acts like a scaled-down version of the centralised approach.

To simplify the following review of tree-first protocols, we group the protocols based on the optimisation techniques used, i.e. distributed transformation and localised central arrangement.

2.6.1.2 Distributed Transformation Protocols

Examples of tree-first distributed transformation protocols are Yoid, HMTP, Overcast, switch-trees, BTP, HostCast, ACDC, AOM, TMesh, SHDC and Banerjee et al.'s scheme. In the following discussion, we underline the proposals (or their variants) that are considered in the performance study in later chapters (i.e. Chapters 5, 6, 7 and 8).

Yoid [36] Yoid is one of the earliest ALM protocols. It focuses on the architectural aspect of general content distribution using a shared tree. Yoid maintains a tree which is augmented with random mesh links to improve robustness. While it uses the switch-parent operation to reconfigure the delivery tree, it does so to avoid some routing pathologies such as excessive delay and packet losses, rather than to optimise the tree.

Switch-trees [43] and BTP [42] Switch-trees defines a set of scope limited switch-parent algorithms for generic overlay tree optimisation. In the protocol, a newcomer is first attached to the tree root and then periodically tries to switch to a better position. Figure 2.5 illustrates the four switch-parent algorithms proposed — each with a different local scope definition (for simplicity, a binary tree is used for illustration purpose). In the figure, the grey node is the node wishing to perform a switch while the black nodes are the set of potential parents. A switching decision is based on the optimisation goal: for tree cost optimisation, a node will try to switch to a node that is closer than its existing parent; for root-diameter optimisation, a node will attempt to switch to a parent that provides a lower root delay (i.e. overlay delay from the node to the root). BTP is a specific example of the tree cost-optimised switch-1hop protocol.

HMTP [109] HMTP is also a switch parent-based protocol. It is specifically designed to build a low cost shared tree for many-to-many applications. Unlike switch-trees, HMTP tries to achieve a good

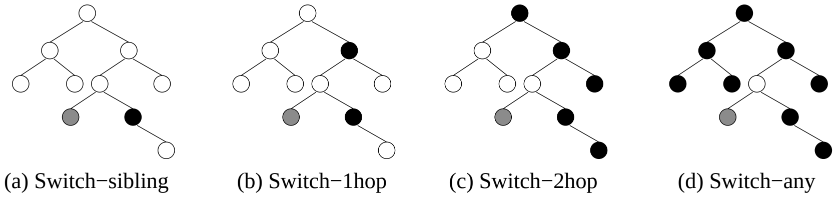


Figure 2.5: Switch-trees algorithms

solution quickly by placing newcomers at their (hopefully) optimal position at joining time. It employs a recursive greedy DFS to find the nearest potential parent on the tree. In the technique, a newcomer measures the distances from itself to the potential parent (initially this is the tree root) and to the potential parent's children. The search ends when the potential parent is the closest node to the newcomer, which will try to attach to the potential parent. Otherwise, the newcomer will continue the search with the closest node as its new potential parent. The newcomer also keeps in memory the most recently visited nodes so as to retrace to another branch when a join request has failed. An on-tree node periodically performs a re-join operation from a randomly selected node from its root path to find a better (i.e. closer) parent.

The basic joining procedures described above can be trapped into the triangle problem (see Section 2.2). Referring to Figure 2.1 (b), assume that node C is the newcomer and it has found B as the closest node during the searching process. Based on the above description, C will attach to B, which results in a triangle between the nodes. HMTP solves this by letting C knows the distance between B and B's parent, A. If C finds that the distance between it and A is smaller that the distance between B and A, it will try to attach to A instead. Of course, this is constrained by the number of children acceptable at A.

HostCast [57] HostCast is designed for one-to-many delay-sensitive applications such as media streaming. Hence, it tries to minimise the root-diameter. Similar to HMTP, it uses DFS to attach newcomers to the trees. However, it does not attempt to place newcomers in their potentially optimal position. Rather, a newcomer is attached to the tree as soon as it finds an unsaturated node. Once a node has found its parent (called a primary parent) in the tree, it establishes a set of peering relationships with its grandparent and parent's siblings to form a control mesh. These links are called secondary links and the corresponding peers are called secondary parents. The node monitors the quality of the primary and secondary links and periodically tries to switch to a secondary parent if it provides a better root delay. In addition, a node can execute a promotion transformation (see Figure 2.4 (c)) to alleviate the triangle problem. Figure 2.6 shows an example of HostCast delivery, and the corresponding control mesh.

AOM [104] AOM can be viewed as an extension to HMTP to improve its delay performance. In particular, AOM provides a set of switching conditions that combine effort to reduce the tree cost and

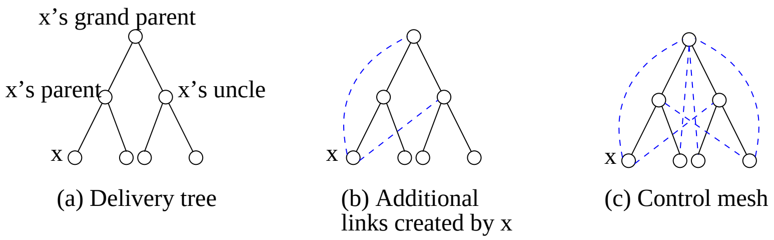


Figure 2.6: HostCast: delivery tree and control mesh

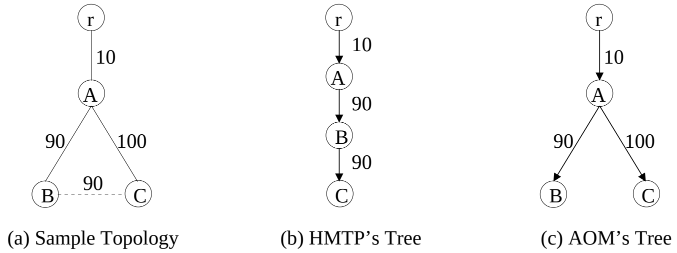


Figure 2.7: Example showing the delay improvement of AOM over HMTP

root-diameter. Assume that a node, C, tries to switch from its parent, A, to node B. C will switch to B under the following conditions.

$$d(C, B) \leq \alpha \times d(C, A) \tag{2.2}$$

$$D(B, root) \leq D(C, A, root) \tag{2.3}$$

$$D(C, B, root) \leq (1 + p) \times D(C, A, root) \tag{2.4}$$

In the above conditions, $root$ represents the tree root, $d(x, y)$ represents the unicast distance between x and y, $D(x, root)$ represents x's current root delay and $D(x, y, root)$ represents x's root delay via y, and α and p are two configurable parameters, where $0 < \alpha < 1$ and $p > 0$. The conditions essentially say that a better parent for C is the one that is closer to C, closer to the root than C, and through which C's new delay from the root is not penalised too much.

Consider the example in Figure 2.7 (a), where r is the tree root and C is deciding whether to switch from A to B. The value beside a link depicts its delay. With HMTP, C will switch to the closer node, B, and gives the tree as in Figure 2.7 (b). The resultant tree cost and delay are both 190 unit. With AOM, assume that $\alpha = 0.9$, thus Equation 2.2 and 2.3 are both satisfied by C. However, C can switch to B only if it is willing to accept poorer delay to r, i.e. from 110 unit to 190 unit. This will require[2] $p > 0.727$. The recommended value of p is 0.2 [104]. Hence, we will obtain the tree as in Figure 2.7 (c) which has a lower delay of 110 unit, but a larger tree cost of 200 unit, than the HMTP's tree. It is interesting to note that if the starting configuration is as Figure 2.7 (b), C will not be able to switch to A (to achieve the

[2] $D(C, B, root) \leq (1 + p) \times D(C, A, root) \Rightarrow 190 \leq (1 + p)110 \Rightarrow p \geq 8/11 \approx 0.727$.

lower delay tree, Figure 2.7 (c)), as Equation 2.2 will not be satisfied.

AOM uses a local region that is similar to HostCast. Its delay properties have been shown to outperform those of HMTP [104].

TMesh [100] TMesh is an overlay optimisation technique designed for many-to-many applications with a small set of active senders. TMesh begins with a shared tree (created by a tree-first protocol, such as HMTP); shortcut links are then added to the tree to form a mesh structure. The shortcut addition is initiated by the receivers so as to improve the average delay observed from the active senders in the session. TMesh runs a path-vector routing protocol to obtain trees rooted at each data source. TMesh will be discussed in a greater detail in Chapter 8, when we present our multiple shared trees many-to-many ALM proposal.

Scalable Hierarchical Distributed Clustering (SHDC) [63] SHDC creates an overlay tree by organising the members into multiple levels clusters: all members belong to a top-level cluster that is rooted at a well-known node; members are recursively grouped into smaller sub-clusters, until all clusters obtained are singleton-clusters containing only one member. Within each cluster, a leader is elected from the members of the cluster. A tree can be obtained by mapping the relationship of leader and members of a cluster to the relationship of parent and children of a tree. In SHDC, newcomers join in the overlay by recursively crossing the hierarchy to find the appropriate cluster. Periodically, a member rejoins the overlay to locate a better cluster. The grouping of members into a cluster is based on the notion of zone, which is defined according to the circular distance around a cluster head. SHDC does not restrict the size of the clusters. As a result, the tree built is not degree bounded. The dynamic behaviours of the protocol, e.g. changes of group membership and cluster leadership, were not discussed in [63].

Banerjee et al.'s Scheme [9] This scheme constructs low average latency trees for one-to-many real-time applications under a proxy-based system. In their system, every multicast member (end host) attaches to a nearby proxy. The proxies organise themselves into a delivery tree, rooted at the proxy where the data source resides. The delivery tree is created in two phases. First, all participating proxies are centrally organised into a degree-bounded tree with increasing distance from the source proxy. In the second phase, each proxy performs periodical distributed local transformations to improve the tree. The local transformations include parent switching, promotion and simultaneous swapping operations for nodes within two levels of each other. Of all the potential transformations, a node will select the one that provides the largest improvement to the objective function. In addition to local transformations, a node may, with a low probability, perform swapping with a node randomly selected using the tree random walk technique (Section 2.6.1.1). This is to avoid the solution from being trapped in a local minimum. The random swapping will be discussed in more detail in Section 5.1.1.5.

Besides the root delay (as in HostCast and delay-based switch-trees), each node also maintains the size of its subtree and the maximum subtree delay (i.e. the delay from the node to its farthest descendant) to aid the transformation decision.

ACDC [54] ACDC is another example of switch-parent protocol. It is targeted to build source-rooted trees that are degree- and root-diameter-bounded, and have low cost. Each ACDC node maintains the root delay and the maximum subtree delay. Together, these two values provide an estimation of the height of the tree branch in which the node resides. ACDC first creates a randomly connected tree. Periodically, a node performs distance measurements to a set of potential parents. Basically, if a node is currently in a branch that has delay larger than the delay target, the node will try a switch that reduces the root delay; if the branch is within the delay target, the node will try a switch that improves the tree cost. ACDC uses the RanSub technique described previously to distribute information about the tree (e.g. delay bound) and to discover switching targets.

Overcast [48] Overcast is designed for delay insensitive high-bandwidth single-sender applications. As discussed in Section 2.2, a bandwidth-optimised tree has high-bandwidth links near to the root with low-bandwidth links near to the leaves. Overcast attempts to achieve this structure by forcing new members to go down the tree as far as possible while not sacrificing the path bandwidth from the root. Periodically, a node x uses the switch-parent operation to try to move up the tree (switch to its grandparent) or move down the tree (switch to one of its siblings) to improve the tree.

2.6.1.3 Localised Central Arrangement Protocols

We focus on two distinctive protocols in this category: TBCP and NICE, and briefly mention Zigzag.

Tree Building Control Protocol (TBCP) [62] TBCP is proposed as a generic tree building protocol. The protocol defines a local region which includes a newcomer, its current potential parent, and the potential parent's children. During the joining phase, the newcomer performs distance measurements to all other nodes in the local region, and reports the results to the potential parent. With measurements obtained from previous rounds, the potential parent will have the complete distance matrix for the local region. Hence, it can evaluate the goodness of all possible local configurations (see Figure 2.8) based on a score function. The configuration with the best score value will be chosen to rearrange the members. In [62], the following function is used.

$$score\ function = \max_{m \in (C \cup \{N\})} D(p, m) \tag{2.5}$$

In the equation, $D(x, y)$ is the distance between node x and y along the overlay tree, p is the current

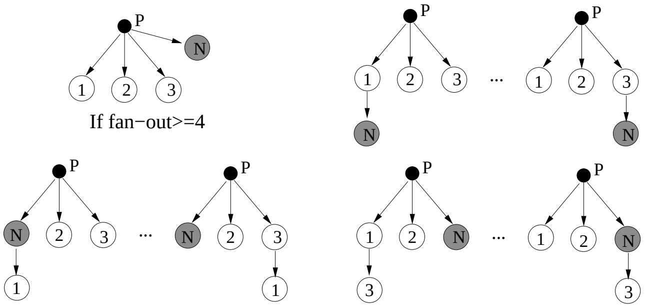


Figure 2.8: TBCP: local configurations

potential parent, C is the set of p's children and N the newcomer. Effectively, the score function calculates the maximum overlay distance provided by each configuration. The best configuration is thus the one with the smallest score. Similar to HMTP, all nodes (except the root) periodically rejoin one of its ancestors chosen at random to improve the tree quality.

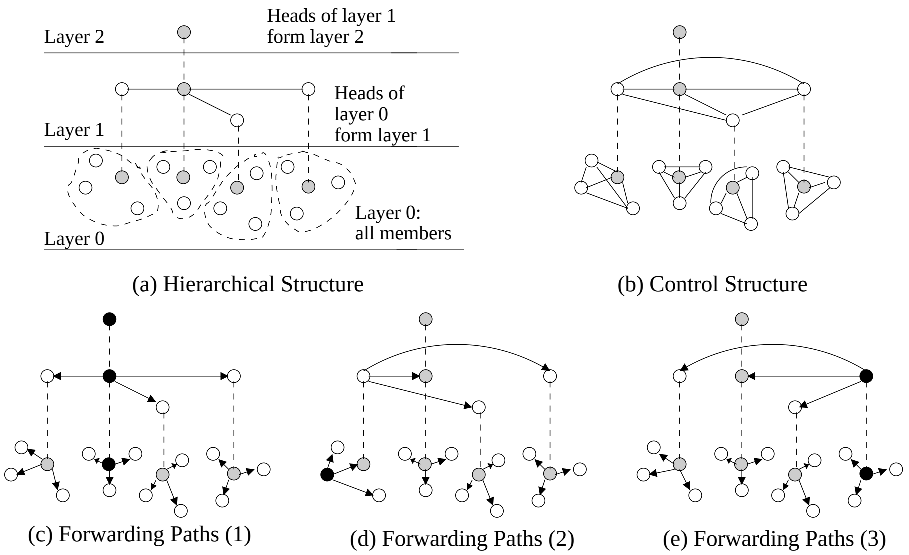
In [62], Mathy et al. introduce a domain-based concept to place nodes belonging to the same domain under the same subtree. In particular, each node is associated with a 32-bit *domainID* (e.g. the bitwise AND of a node's IP address and a netmask). Each domain has a domain root elected by the tree root. The first node joining from a given domain can be elected as the domain root of its domain. Domain roots find their place in the tree with the mechanism described above, starting from the tree root. When a non-domain-root node joins the tree, the root will redirect the node to its domain root, from which the node will begin its joining process. Along with the domain-based redirection, the following two constraints need to be enforced in the above joining mechanism.

1. A node P will discard any configuration in which a node from its own domain becomes a child of a node from a different domain.

2. To keep domain roots as high as possible in the tree (i.e. as close as possible to the tree root), configurations in which a node P keeps more than one node from its own domain as children, and sends a domain root of a different domain as child of one of its children, are discarded.

NICE [7] NICE is designed for low bandwidth large-scale many-to-many applications. It organises the overlay into a hierarchy of clusters, i.e. a mesh-based control topology. Unlike the previously mentioned proposals, it uses source-specific trees (obtained from the overlay) for data delivery. Each NICE cluster has a size between k and $3k - 1$ inclusively, where k is a configurable parameter. The cluster defines the local region in NICE, and is represented by a cluster leader. Beginning from the lowest level, cluster leaders at the same level form the next level clusters, until there is only a single cluster at the highest level. Figure 2.9 (a) and (b) show an example of a NICE hierarchy and the corresponding control mesh within each cluster.

In the NICE overlay construction process, all newcomers first join one of the lowest level clusters



(a) Hierarchical Structure (b) Control Structure

(c) Forwarding Paths (1) (d) Forwarding Paths (2) (e) Forwarding Paths (3)

Figure 2.9: NICE: hierarchy, control structure and data forwarding trees

using a DFS beginning from the highest-level cluster leader. NICE allows a cluster to temporarily violate the size bound. Periodically, a cluster leader checks the size of its cluster. If the cluster is too large ($> 3k - 1$), the leader rearranges it into two equal-sized sub-clusters and picks two new leaders such that they are the graph-theoretic centre of the respective cluster; if the cluster is undersized ($< k$), the leader will initiate a merge operation with a nearby cluster.

The NICE hierarchy implicitly defines the data forwarding paths, which are a set of source-specific trees. The forwarding mechanism works as follows: when a node h receives a packet from a node p, it will replicate and forward the packet to all clusters of which it is a member at each layer, except the clusters of which p is also a member. Due to this mechanism, for a n-node overlay, a NICE host can have as many as $O(k \log n)$ peers in its data path. Figures 2.9 (c), (d) and (e) show three distribution trees for three different sources (the nodes shown in black).

Zigzag [95] Zigzag is designed for delay-sensitive single-source streaming media applications. It adopts a similar hierarchical structure to NICE for overlay maintenance. However, its delivery structure is a source-rooted tree which has a maximum fan-out of $O(k^2)$. Unlike the other tree-first protocols discussed previously, NICE and Zigzag do not strictly limit the degree of the nodes in the resultant tree.

2.6.2 Mesh-first Protocols

Mesh-first protocols take a two-step procedure to build delivery trees. Initially, members organise themselves into a mesh structure. As a mesh normally includes redundant paths between the members, naive flooding will lead to unnecessary packet duplications. Hence, an additional step — a routing mechanism is needed to infer the delivery trees from the mesh. Existing mesh-first techniques can be classified based

```
EvaluateUtility ($y$) begin
utility = 0
for each member $m$ ($where\ m \neq x$) begin
    $CL$ = current latency between $x$ and $m$ along mesh
    $NL$ = new latency between $x$ and $m$ along mesh if edge $\langle x, y \rangle$ were added
    if ($NL < CL$) then begin
            $utility\ += \frac{CL-NL}{CL}$
    end
end
return $utility$
```

Figure 2.10: Narada: the algorithm x uses in determining the utility of adding a link to y

on the mesh structure they create: unstructured and structured meshes.

2.6.2.1 Unstructured Mesh-based Protocols

An unstructured mesh is a graph where no special structural information can be inferred from the graph to aid routing and management. For example, a random graph. Forming such a topology is simple: a newcomer simply attaches itself to some randomly selected members. To obtain loop-free routing trees from this type of topology, a conventional routing protocol such as distance-vector or link-state is needed. Narada and Gossamer are two representative ALM protocols in this category.

Narada [21] Narada, one of the earliest efforts on ALM, is designed for many-to-many applications. In Narada, members first organise themselves into a random mesh, which is then improved upon in an incremental manner. The improvement process involves two basic operations: addition of useful links to the mesh and deletion of less useful links to keep the mesh within a manageable size. To add a new link, x randomly selects a non-neighbour node, and requests a copy of its routing table. Assume that y is selected. Node x will compute the expected delay gain from x to other nodes if a link to y is added, using the utility function shown in Figure 2.10. The link will be added if the gain exceeds a threshold. To drop an existing link, x estimates the consensus cost of the links that it currently maintained. The consensus cost of a link is calculated as the maximum number of times that the link is used in data forwarding for both end nodes of the link. A link with consensus cost lower than a threshold will be dropped.

To obtain the distances between the members via the mesh, Narada runs a path-vector routing protocol, which extends the distance-vector protocol [93] by including the path information between the nodes in the routing updates. Including such information helps to avoid the well-known count-to-infinity problem [93]. As the routing protocol provides unique paths between the nodes, the multicast trees with any specific member as the source can then be computed using the reverse path forwarding technique [24] as in DVMRP [25].

One important issue in mesh management is the partition problem. Detecting a partition in a mesh is somewhat harder than in a tree. To solve the problem, Narada requires each node to maintain the membership of the overlay, where the dissemination of the membership is integrated with the routing protocol. This, however, leads to a relatively high control overhead ($O(n^2)$ for an n-node overlay). Thus, Narada is effective only for small-scale applications.

Gossamer [18] Gossamer's overlay creation and derivation of delivery trees bear several similarities to those of Narada. The main difference between the two is that Narada is based on the end host-only model while Gossamer is designed for a proxy-based system called Scattercast [18]. Gossamer reduces the routing overhead by limiting the routing advertisement to only proxies with active senders (so as to derive source-specific trees). A gossip-style node discovery protocol called Name Dropper++ is introduced to distribute membership information for overlay improvement. The utility function used in Gossamer is slightly different from Narada, but both of them strive to minimise the average delay between the members. Gossamer also simplifies mesh partition management by requiring every member to observe only the liveness of a small number of selected members.

2.6.2.2 Structured Mesh

While routing in an unstructured mesh is difficult, it is possible to relate the mesh members in a structural manner to simplify routing and management. Commonly used structural meshes are distributed hash table (DHT), Delaunay triangulation and clique.

DHT-based Protocols DHT overlays were initially designed for object routing and location in peer-to-peer networks. In a DHT overlay, each member is assigned a unique identifier (in general, we refer to it as *nodeId*) which can be a numerical value or a point in a coordinate system. Each of the nodeId represents a point in an abstract namespace. In the system, each node maintains a portion of the namespace in its routing table. The overlay routes a message to the node responsible for the portion of the namespace that contains the destination nodeId. In other words, all nodes only keep a small portion of member information in their respective routing tables, and a message is routed hop-by-hop from one node to another using the local routing table.

There are two main classes of routing algorithm in this type of topology: Chord [88], Pastry [81] and Tapestry [111] use a longest prefix matching technique to route in a ring; and CAN [77] routes in a Cartesian hyper-space by choosing a neighbouring node closer to the destination at each hop. Scribe [15], Bayeux [112] and CAN-multicast [78] are ALM protocols developed based on Pastry, Tapestry and CAN respectively. Next we describe Scribe, which will be used in our comparison study as the representative.

Scribe creates multicast trees on top of the overlays built by Pastry. In Pastry, each overlay node is assigned a random nodeId that is uniformly selected from a one-dimensional circular namespace of 128 bits. Given a message and a destination nodeId, Pastry routes the message to a node with the nodeId that is numerically closest to the key, among all live nodes. Figure 2.11 illustrates an example where a message targeted for `d47a1c` is routed from node `65a1fc` to node `d37492`.

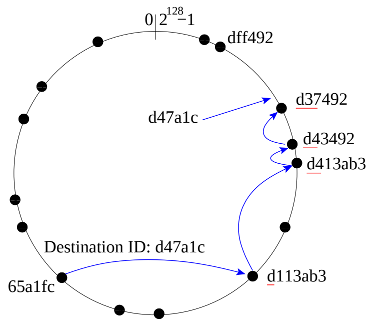


Figure 2.11: Pastry: routing in a circular namespace (each dot depicts a live node in the namespace).

The Pastry overlay is created in the following manner. First, a new member uses a deterministic hash function to create a unique nodeId (say `d47a1c` as in Figure 2.11) using inputs such as the IP address of the node. It then sends a join message addressed to its own nodeId via a topologically close-by overlay node, say node `65a1fc` as in Figure 2.11. Pastry assumes that the knowledge of such a node is learned from some out-of-band techniques, such as an expanded ring search. Node `d113ab3` then forwards the message hop-by-hop towards the destination nodeId. Finally, the message reaches node `d37492` which has the closest nodeId to `d47a1c`. The newcomer obtains the state information from nodes on the path from `65a1fc` to `d37492` to establish its routing table and neighbours list.

Scribe builds a unidirectional shared tree for each multicast session on top of the Pastry overlay. Each multicast session is assigned a *groupId* randomly selected from the namespace. Each groupId is maintained by a root node with a nodeId that is equal or close to the groupId. New members join to a session by sending join messages towards the root using Pastry routing primitive. When a join message hits an on-tree node, say y, the path from the newcomer to y forms a new branch of the tree.

In [16], Castro et al. propose SplitStream, which extends Scribe to deliver high-bandwidth contents using multiple trees. SplitStream addresses the inherent unbalanced forwarding load in a single tree delivery mechanism (see Section 2.2.1) by making sure that each node is the interior node in only one tree, and is the leaf node in all the remaining trees. Such trees can easily be built with the underlying Pastry and Scribe routing mechanism. The design is highly robust as failure of an interior node will affect only one tree.

Delaunay Triangulation-based Protocols Consider a set of vertices, A. A Delaunay triangulation is a triangulation graph for which each circumscribing circle of a triangle formed by three vertices in A, no vertex of A is in the interior of the circle (see Figure 2.12). In [58], Liebeherr et al. propose a DT protocol that constructs a Delaunay triangulation overlay in a decentralised manner and uses it for multicasting. In the protocol, each overlay node is assigned an (x, y) coordinate. The protocol distributedly derives Delaunay triangulation relationships for the overlay nodes based on their coordinates. The Delaunay triangulation graph constructed defines the neighbouring relationships among the overlay nodes: two

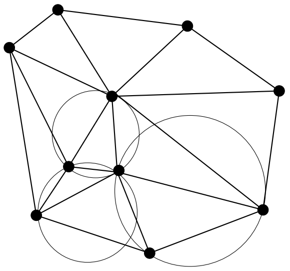


Figure 2.12: Delaunay triangulation

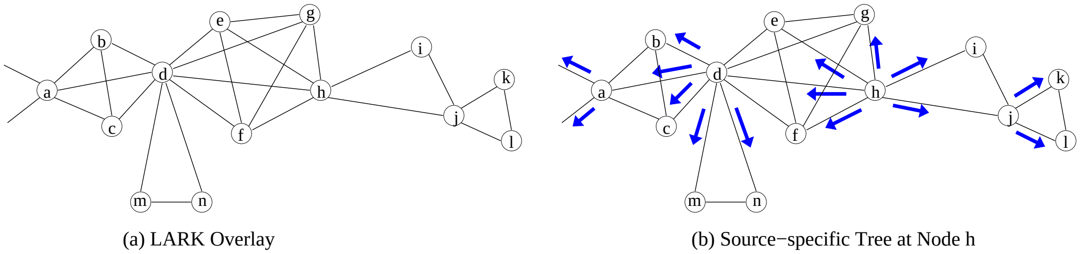


Figure 2.13: LARK: an example overlay and the corresponding data delivery tree

nodes are neighbours if the edge connecting them appears in the Delaunay triangulation graph. Once the Delaunay triangulation overlay is formed, compass routing [55] is used to derive source-specific trees for multicasting. A compass routing routes a message hop-by-hop from a node, s, to a destination, d, using only the coordinates of d, the position of the current node, and the directions of the edges incident with the current node. At each hop, the message is delivered over the edge with the closest slope to that of the line segment connecting the current node to d. The DT protocol is scalable in the sense that each overlay node only needs to maintain a small amount of neighbouring information. However, the data delivery performance depends on how well the logical address (i.e. the (x, y) coordinates) is mapped to the underlying network topology.

Clique-based Protocol LARK [49] is designed to be a light-weight and resilient protocol for many-to-many multicasting. In LARK, overlay members are organised into several interconnected cliques, where a clique is a fully connected graph. Figure 2.13 (a) shows an example of LARK's overlay structure. The nodes that interconnect different cliques are called bridge nodes. Routing in such an overlay is simple: when a node receives a message from a neighbour in a clique, it will forward the message to other neighbours in different cliques it belongs to. Figure 2.13 (b) illustrates the data tree originated at the source, node h. As a clique is a complete graph, LARK is resilient against node failures. However, its resilient property does not necessarily provide good delivery tree quality, as shown in [49].

2.6.3 Summary

Table 2.1 and 2.2 summarise the tree- and mesh-first distributed ALM protocols discussed previously. In the tables, we also provide additional information (e.g. maximum path length and per node state) about the protocols. We note that the maximum path length for several protocols is left undefined. For examples, Yoid, switch-trees, HMTP, etc. These protocols produce delivery trees that have varying degrees at each node (due to the nodes capacity constraint). The worst-case happens when all nodes can only accommodate one out-going stream. This results in a line topology which has a maximum number of overlay hops of $O(n)$, for an n-node overlay. However, if we assume that all nodes can contribute two out-going streams, the overlay can form a binary tree. For a full binary tree (i.e. every internal node has two children), the tree height or the maximum path length from the root to its furthest descendant is given by $\log_2 n$. In practice, we would expect that an overlay tree is unevenly populated, i.e. some nodes may not have any child while some nodes may have a number of children. On average, we believe the tree height (in terms of overlay hop) is in the order of $O(\log n)$.

2.7 Related Research Areas

This section briefly discusses some related research areas.

2.7.1 ALM in Mobile Adhoc Networks

Mobile Adhoc Networks (MANETs) are characterised by the highly dynamic, random multihop topologies that are likely composed of relatively bandwidth-constrained wireless links. This dynamic nature prevents the use of existing IP multicast protocols, e.g. DVMRP, CBT. AMRoute [105] is a protocol that uses ALM over mobile wireless networks. AMRoute continuously creates a mesh of bidirectional unicast tunnels between pairs of group members. A shared tree is created out of the mesh for data distribution. One member node is designated as the logical core, which is responsible for initiating the tree creation process periodically. The core can also migrate dynamically according to group membership and network connectivity. A key feature that distinguishes AMRoute and the works studied and proposed in this thesis is that AMRoute assumes a native wireless broadcast channel. Self-configuration in the absence of such a broadcast medium is a much harder problem. We believe that the wired and wireless ALM could be integrated with technique similar to Universal Multicast (see Section 2.1) in which a mobile member can be elected to represent its mobile peers, and provides a bridge to the outside world. This however is out of the scope of this thesis.

2.7.2 Peer-to-peer (P2P) Networks

P2P systems are typically file sharing networks between end hosts. Early P2P system such as Napster followed a semi client-server architecture. In Napster, a number of servers maintain databases of interesting resources (i.e. names and locations of music files). End hosts contact the servers to obtain peers that actually keep the desired music files, and make requests directly to the corresponding hosts. While the actual data connections happen between the end hosts, the popularity of the service may still result in overloading of the servers. Thus, modern P2P systems such as Gnutella [40] advocate a fully distributed approach. In particular, the members organise themselves into an overlay network as in ALM. Unlike ALM, however, the P2P overlay is mainly used to locate files efficiently. For example, a host x that wishes to find a file, F, floods the queries into the overlay. Once x identifies the host of F (say y), x will request the file directly from y. The flooding mechanism in Gnutella can potentially create a large amount of query traffic. This has prompted research of structural overlays which provide scalable overlay management and routing. Most notably, the distributed-hash tables proposals such as Pastry, Tapestry, Chord and CAN, which were discussed in Section 2.6.2.2.

2.7.3 Internet Distance Measurement Systems

A key issue in ALM is inferring the network metrics (e.g. bandwidth and delay) between the members. In this thesis, we consider only the delay metric. To obtain pair-wise delay information, the end systems could employ tools such as `ping`. However, having each host conducts a large number of measurements inevitably leads to a high overhead, both to the host and the network. Hence, several projects have emerged to provide scalable distance estimation services. We discuss two representatives below.

Internet Distance Maps Service (IDMaps) IDMaps [35] is an early attempt to provide an Internet scale distance service. IDMaps employs special hosts called *tracers* at various network locations. The tracers are organised into a logical topology and continuously monitor the distances among themselves in the logical topology. This serves as the infrastructure to estimate the distance between any two hosts in the Internet. For example, the distance between hosts A and B can be estimated as the distance between A and its nearest tracer T_1, plus the distance between B and is nearest tracer T_2, plus the shortest path distance from T_1 to T_2 over the tracer logical topology.

Global Network Positioning System (GNP) GNP [68] is a coordinate-based approach to network distance prediction. The key concept is to ask end hosts to maintain coordinates (i.e. a set of numbers) that characterise their locations in the Internet. The network distances can then be estimated by evaluating a distance function over the hosts' coordinates. GNP consists of two parts. In the first part, a small distributed set of hosts called Landmarks first measure the distances among themselves, and use the measured distances to compute their coordinates in a chosen geometric space. The coordinates are calculated by solving a function that minimises the error between the measured distances and the computed distances (i.e. using the coordinates). The Landmarks' coordinates serve as seeds to compute the coordinates of other end hosts in the second part. In [68], GNP has been shown to provide better distance estimation than IDMaps.

2.8 Chapter Summary

This chapter extends the understanding of ALM, as well as some related research areas. The main focus is on the construction of the multicast overlay topologies. We begin by discussing several important issues in an ALM overlay creation solution, i.e. system architectures, optimisation objectives and the service models. This is followed by a review of a number of representative centralised and decentralised overlay construction techniques. The proposals surveyed encompass a wide variety strategies in building, optimising and maintaining an overlay. In Chapter 5, we perform a performance comparison study of some of the proposals.

Protocol	System Architecture	Tree Type (per Session)	Design/ Optimisation Efforts	Optimisation Techniques	Nodes Discovery	Max. Fanout/ No. of Neighbours	State per Node	Max. Path Length	Applications
Distributed Transformation-based Protocols									
Yoid	End host only	BST	Avoid routing pathologies	Switch-parent	Random walk + DFS	$c+1$	$O(c)$	-	Generic content distribution
Switch-tree	End host only	SRT/ BST	Generic: Tree cost/ latency/ etc	Switch-parent	Local region scoping	$c+1$	Switch-1hop: $O(c)$ Switch-2hop: $O(c^2)$	-	Generic content distribution
HMTP	Hybrid	BST	Tree cost	Switch-parent	Root path + DFS	$c+1$	$O(c)$	-	Delay-insensitive bulk data transfer
ACDC	End host only	SRT	Latency + tree cost	Switch-parent	RanSub	c	$O(c+\log n)$	Delay constraint	Delay-sensitive applications
HostCast	End host only	SRT	Latency	Switch-parent + promotion	Local region scoping	c	$O(c)$	-	Delay-sensitive applications
Banerjee et al.'s scheme	Proxy-based	SRT	Latency	Switch-parent + various swapping	Local region scoping + random walk	c	$O(c+\log n)$	-	Delay-sensitive applications
Overcast	Proxy-based	SRT	Bandwidth	Switch-parent	Local region scoping	c	Each node maintains info. about all nodes under its subtree: $O(n)$	-	Bandwidth-intensive applications
TMesh	End host only	MSRT	Latency	Tree + shortcut links	Root path + DFS	$c+1$	$O($ # senders $)$	-	Many-to-many delay-sensitive applications with small senders set
SHDC	End host only	SRT	Scalability	Switch-parent	Root path + DFS	c	$O(c)$	-	Creating control structure for large-scale applications
Localised Central Arrangement-based Protocols									
TBCP	End host only	SRT/ BST	Generic: tree cost/ latency/ etc	Select the best config. from all possible local config.	Local region scoping	c	$O(c+$ root path size$)$	-	Generic content distribution
NICE	End host only	MSRT	Scalability	Cluster members based on their proximity	Local region scoping (cluster)	$O(K \log_K n)$	Max: $O(\log_K n)$ Avg: $O(K)$	$O(\log_K n)$ overlay hops	Low-bandwidth large-scale content distribution
Zigzag	End host only	SRT	Latency	Cluster members to reduce source to receivers latency	Local region scoping (cluster)	$O(K^2)$	Max: $O(\log_K n)$ Avg: $O(K)$	$O(\log_K n)$ overlay hops	Delay-sensitive applications

Legends

SRT: Source rooted tree	c: max. # children of a tree node
BST: Bi-directional shared tree	K: maximum cluster size in NICE and Zigzag
UST: unidirectional shared tree	n: # of overlay nodes
MSRT: Multiple source rooted tree	

Table 2.1: Summary of tree-first protocols

Protocol	System Architecture	Tree Type (per Session)	Design/ Optimisation Efforts	Optimisation Techniques	Nodes Discovery	Max. Fanout/ No. of Neighbours	State per Node	Max. Path Length	Applications
Unstructured Overlay-based Protocols									
Narada	End host only	MSRT	Latency	Add/ delete links with a utility function	Path-vector routing protocol	m	$O(n)$	-	Small-scale delay-sensitive many-to-many applications
Gossamer	Proxy-based	MSRT	Latency	Add/ delete links with a utility function	Path-vector routing protocol + gossiping	m	$O(n)$	-	Delay-sensitive many-to-many applications
Structured Overlay-based Protocols									
Scribe	End host only	URT	Scalability	Pastry routing	Pastry routing	# of Pastry neighbours: $\lceil \log_{(2^b)} n \rceil \times (2^b - 1)$; Tree fanout: c	$\lceil \log_{(2^b)} n \rceil \times (2^b - 1)$	$\lceil \log_{(2^b)} n \rceil$ overlay hops	Large-scale Publish-subscribe applications
Bayeux	End host only	URT	Scalability	Tapestry routing	Tapestry routing	# of Tapestry neighbours: $b \log_b n$; Tree fanout: c	$b \log_b n$	$\log_b n$ overlay hops	Large-scale content distribution
CAN-multicast	End host only	CAN directed flooding	Scalability	CAN routing	CAN routing	$2d$	$2d$	$(d/4) \times z^{1/4}$ overlay hops	Large-scale content distribution
Delaunay Triangulation (DT) Protocol	End host only	MSRT	Scalability	Delaunay triangulations graph	Delaunay triangulations graph	Max: $n - 1$; Avg: 6	Max: $n - 1$; Avg: 6	-	Large-scale content distribution
LARK	End host only	MSRT	Resilient and scalability	Clique structure	Clique structure	m	# number of clique members	-	Large-scale content distribution

Legends

SRT: Source rooted tree	c: max. # children of a tree node
BST: Bi-directional shared tree	m: max. # neighbours of a mesh node
UST: unidirectional shared tree	n: # of overlay nodes
MSRT: Multiple source rooted tree	b: design parameter in Scribe and Bayeux
	z: # of zones in CAN-multicast
	d: # of dimensions in CAN-multicast

Table 2.2: Summary of mesh-first protocols

Chapter 3

System Model and Evaluation Environment

This chapter presents the system model and the simulation design used in the rest of this thesis. In the next section, we first describe the system and overlay network model considered. Section 3.2 then describes the simulation design, which includes the setting of the network topologies, multicast group and some general parameters, as well as the performance metrics used to quantify the ALM proposals. The section also explains the working of the simulator, called `ALMSim`, that we have developed for the experiments. Finally, Section 3.3 concludes this chapter.

3.1 System Model

3.1.1 System Architecture

In Chapter 2, we discuss two versions of ALM system architecture, i.e. the end systems only and the hybrid (mixed end system and network layer multicast) approaches.

This work considers the end system only model. However, recall that the end system only model can be further divided into the end host only and proxy-based systems. This thesis considers the first model, where the multicast members are the end hosts which directly participate in the overlay construction and data delivery. This allows us to concentrate mainly on techniques used to build the overlays, while avoiding any complex interaction between the end hosts and the proxies or network layer multicast infrastructure.

It is worth pointing out that the techniques studied are directly applicable to build overlays for the proxy-based system. However, a protocol specifically designs for such a system can further exploit the more powerful and potentially better available bandwidth of the proxies. In addition, the overlays built can also be used by a hybrid approach (e.g. Zhang et al.'s Universal Multicast [110] (see Section 2.1)) to interconnect islands of IP multicast networks.

3.1.2 Network Model

The physical network is represented by a set of routers which are interconnected by links, as shown by panel 1, Figure 3.1. The end systems, i.e. members of the multicast session, are connected to the routers at different points through access links. Each router can accommodate at most one end system. With this, we assume that each router represents a multicast capable local network. If there is more than one multicast member in the network, they will communicate among themselves using the underlying network layer multicast capability, and one of them will be chosen as the representative that bridges this network with other members in other networks.

The overlay network is formed by the end systems on top of the physical network. Theoretically, it can be modelled as a complete graph, $G = (V, E)$, where V is the set of vertices and $E = V \times V$ is the set of edges. This is illustrated as panel 2 in Figure 3.1. Each vertex in V represents an end system. An edge, $\langle i, j \rangle$ in E corresponds to the unicast path from i to j in the physical topology. The delay of edge $\langle i, j \rangle$ is the end-to-end delay from i to j via the physical topology.

For practical reason, an ALM overlay is often a subgraph of the complete overlay graph, which takes the form either of a tree or of a connected mesh. An example of an ALM tree is shown in panel 3, Figure 3.1. For conciseness, the rest of this thesis will use *overlay* to refer to an ALM overlay, unless specified otherwise.

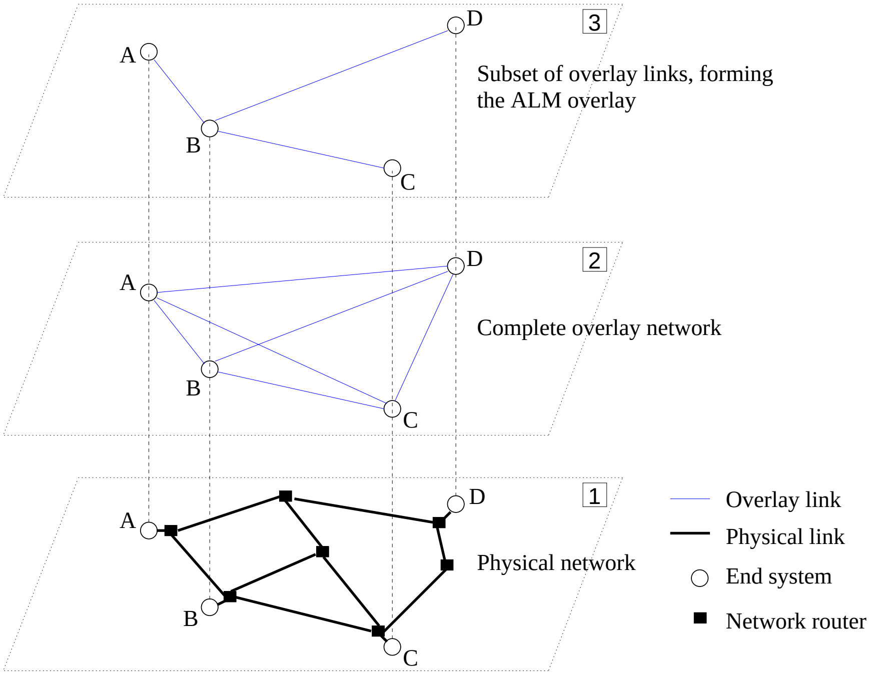


Figure 3.1: Representation of the network model

Bandwidth Model We study techniques that constructs efficient ALM overlay structures. The main requirement for the resultant overlay is that the maximum number of out-going data streams (i.e. fan-out or out-degree) that a node can contribute is limited. In other words, the delivery structure needs to be degree bounded. In a measurement study by Bhattacharyya et al. [13], they show that the backbone network is often over provisioned, and thus the bottleneck is mostly at the access links of the end systems. In other words, we can abstract the available bandwidth of an end system as the bandwidth of its access link. Hence, if we know the bandwidth of the access link and the data rate of a multicast session, we can calculate the number of out-degree that a member can contribute to the session. Modelling the bandwidth limitation as the degree bound also enables us to focus on other optimisation metrics, in particular, tree cost and delay. Most of the ALM proposals discussed in Chapter 2 allow the members to specify a degree bound in building overlays.

Consider a multicast application in which the source transmits data at a rate of B units per second. We will assume that the capacity of any incoming or out-going access link is no less than B. Let the out-going access bandwidth of a node i be r_i, we can then calculate i's out-degree bound as $\lfloor r_i/B \rfloor$. The similar assumption is also being made by Banerjee et al. [9]. The access bandwidth of a node can be approximated based on the access technology used by the node, e.g. dial-up, Cable or DSL. This estimation may be inaccurate as a node may have other applications that connect to the Internet, and thus need to share the bandwidth. However, we believe that cooperation between the end systems can be used to dynamically adapt the degree constraint. For example, let node x be an upstream node serving several other nodes. Initially, x determines the maximum number of connections it can contribute based on its access technology. Then, during the course of the session, the downstream nodes observe and report the received data rate to x. Based on the reports, x may increase or decrease its degree bound. A threshold value and the history of the data transfer can be used to improve the accuracy of the prediction process. It is the responsibility of the overlay construction protocol to ensure that an overlay stays connected in case of changes to node degrees. For simplicity, we assume that the degrees assigned to the end systems are constant throughout a session.

Even if the source injects traffic at very low rate, to make good use of its memory and processing power, the end user may also wish to limit the resources used by a single application. This is especially true if the ALM is applied on top of the end host only model. On the other hand, while the proxy-based systems normally have larger bandwidth capability, they normally support more than one session. Hence, each of the sessions running on such systems may only get a share of the available capacity.

Overlay Link Delay and Cost As described previously, the delay of an overlay is the end-to-end delay between the two end points of the link, which is the sum of the delays of all physical links that the overlay link traverses. We also use the overlay link delay as the cost of the overlay link.

For simplicity, we will assume that there will be no changes to the link delay throughout the course of a simulation. In practice, this assumption is certainly not true. In an overlay, a node typically maintains the delays from itself to several other nodes. A change in the delay value may require the node to re-evaluate its on-tree position, and may result in a overlay reconfiguration. However, we believe that there are several ways that can be used to limit the impacts of the changes to the delay metric.

- *Cache and threshold*. In this case, the delay value is cached, and is only updated if the difference between the current and the new values exceeds a predefined threshold. This technique has been discussed in HMTP [109].
- *Quantise the delay value*. In this case, we can quantise the delay value to several discrete levels. For example, delay values that is smaller than 10 ms is assigned as level 1, delays that are between 10 ms and 20 ms is assigned as level 2, and so on. The delay ranges can be configured so that smaller delay values have a finer representation, while larger delays have a coarser representation. Note that a coarser delay representation will result in more ties in decision making, which potentially affects the quality of the overlay built.

In fact, the above ideas have been investigated in previous study of quality of service routing. For example, Apostolopoulos et al. [4] consider several variants of the above two strategies to limit the update of bandwidth metric. Integrating the above techniques with an overlay construction proposal and studying their performance implications in a real-world environment is on-going work.

3.2 Simulation Design

3.2.1 Use of Simulation

In this work, we make extensive use of simulations for our performance studies. The proposals considered include both centralised algorithms and distributed protocols (more than 10 different proposals). The size and the diversity of the proposals studied prevented us from considering a close-form analytical evaluation.

In recent years, the ALM community has began to use PlanetLab [74], a wide-area overlay testbed, to run real-world experiments. PlanetLab offers an opportunity to run experiments subject to realistic network characteristics. One of our objectives is to evaluate and compare several existing proposals. To do so requires a unified and controlled platform for the experiments. PlanetLab is thus unsuitable for our purpose as it introduces a number of uncontrollable and unpredictable variables such as the ever changing network conditions and the system load of the host machines. These additional factors prevent repeatable experiments, and thus prevent a fair comparison between different proposals. We also wish to examine the proposals under some reasonably large network configurations, e.g. up to thousands of nodes. However, at the time of writing (May 2005), PlanetLab only consists of over 500 nodes, hosted at over 270 sites. Therefore simulation is used because it provides a more controllable and configurable environment for running the experiments. It also allows us to monitor closely the mechanic of a proposal more easily.

3.2.2 Topologies

A key challenge in studying protocol behaviour lies in the representation of the underlying topology. There is evidence that the performance of protocols has a strong relationship with the topology used. For example, a clustering protocol may perform well in topologies that already exhibit good clustering properties, e.g. hierarchical networks. In order to avoid any bias in our evaluations, we use topologies created from three distinct models:

1. *Random Waxman graph*. This is the popular random graph model presented by Waxman [102]. In this model, nodes are randomly distributed over a Cartesian coordinate system. The probability that an edge exists between any two nodes, u and v, is given by the following probability function

$$P(u,v) = \beta \exp \frac{-d(u,v)}{L\alpha} \tag{3.1}$$

   where $d(u,v)$ is the distance between the two nodes, L is the maximum possible distance, and α and β are parameters in the range $0 < \alpha, \beta \leq 1$. Larger values of α increase the proportion of longer edges to shorter edges, while larger values of β increase the average node degree. While there are extensions to this basic model to better represent a realistic network, e.g. [28], we choose to use the basic model as it has been used in previous work (e.g. Narada [21] and HMTP [109]), and is included in the popular GT-ITM topology generator [39].

2. *Transit-stub graph*. This model represents a network as a two-level hierarchical graph consisting of stub domains interconnected by transit domains. The stub domains represent campus networks or other collections of interconnected LANs, while transit domains represent wide-area networks. The connectivities within the stub domains and the transit domains are generated using the Waxman model. Both transit-stub and Waxman topologies are created with the GT-ITM topology generator.

3. *Power-law graph*. This is a model based on the observations that node degree in the AS-level topology of the Internet is closely related to a set of power laws [32]. In particular, the probability, $P(k)$, that a node in the network is connected to k other nodes is bounded, decaying as a power-law, $P(k) \sim k^{-\gamma}$ [11]. We use the power-law generation model due to Barabási and Albert [11] in the BRITE topology generator [38] to generate such topologies. The model suggests two possible causes for the emergence of power-law behaviour in the frequency of out-degrees in network topologies: (i) incremental growth; and (ii) preferential connectivity. Incremental growth

> refers to the growing of network size due to the continual addition of new nodes. Preferential connectivity refers to the tendency of a new node to connect to existing nodes that are highly connected or popular. This assumes that a network topology is generated by a set of local events, such as the addition of new nodes and links, or rewiring of links from one node to another.

We consider topologies of reasonably large sizes, i.e. from 1000 to 10000. Running experiments on smaller networks is considerably faster than on larger networks. Hence, for the comparison study in Chapter 4 and 5, we use the smaller (1000-node) networks due to the size of the techniques under consideration. In Chapter 6, 7 and 8, we use networks of 5000 (power-law) and 10000 (transit-stub) nodes to compare our proposals with the best techniques observed in the comparison study. In these chapters, we also use the smaller networks to investigate the effects of various parameter settings on our proposals.

Table 3.1 lists the topologies used in our simulations along with their characteristics. We choose topologies that exhibit a rich diversity of configurations, both within the same model and between the different models. For Waxman and power-law models, the topologies mainly differ in terms of node degree. For transit-stub model, the topologies are created such that they have different sizes of transit and stub domains. For examples, TS1k-0 has 1 transit domains with 4 routers each, each transit router is attached with 5 stub domains, each with 50 routers; TS1k-1 has 2 transit domain with 10 routers, each transit router is attached with 5 stub domains, each with 10 nodes (see Table 3.2). These configurations contribute to the differences in size[1] (e.g. 1004, 1010 and 1020 compared to 1000) with other models. Note that for larger networks (5000 and 10000 nodes), we only consider the transit-stub and power-law topologies as they better represent the realistic networks. For the power-law model, we only use topologies up to 5000 nodes as computing the all-pair paths (used by `ALMSim` for routing purposes) takes a considerable amount of time. On the other hand, we could take advantage of the hierarchical structure in the transit-stub topologies to divide the computation, so that a larger size is possible.

3.2.3 Multicast Members Selection

We study the construction of efficient ALM overlays for a given set of members. The members are randomly attached to the network routers. Note that for transit-stub topologies, the members are attached only to routers in the stub domains. As mentioned previously, each router can only accommodate at most one member. We assume that the latency from a router to its attached member is negligible. For distributed proposals, the set of members will join the multicast session one at a time randomly, within a predefined time. In most cases, we let all members join the session within the first 50 seconds of the simulation, and let the protocols organise the members into a stable overlay. Typically, for smaller

[1] The number of nodes in a transit-stub network can be calculated as: (# transit domains)×(# nodes per transit domain)×[1 + (# stub domains per transit node)×(# nodes per stub domain)].

Model	Name	Nodes	Links	Mean	Diameter		Mean Path Length	
				Degree	Hop	Delay (ms)	Hop	Delay (ms)
Random Waxman	WM1k-0	1000	4000	4.00	9	97	4.96	37.05
	WM1k-1	1000	6000	6.00	7	73	3.97	26.46
	WM1k-2	1000	8000	8.00	6	62	3.51	24.97
Transit-stub	TS1k-0	1004	14522	14.46	10	367	6.38	181.97
	TS1k-1	1020	3242	3.18	22	666	8.42	255.55
	TS1k-2	1010	4074	4.03	15	622	8.04	274.62
	TS10k-0	10100	39200	3.88	30	2784	12.84	1245.36
	TS10k-1	10020	195852	19.55	15	1284	8.60	563.90
Power-law	PL1k-0	1000	11598	11.60	5	99	3.04	32.20
	PL1k-1	1000	16216	16.22	5	77	2.82	25.15
	PL1k-2	1000	19614	19.61	5	65	2.65	19.37
	PL5k-0	5000	46698	9.34	7	95	3.80	33.61
	PL5k-1	5000	93048	18.61	5	63	2.76	21.25

Table 3.1: Characteristics of the topologies used in the simulations

Name	no. of transit domains	no. of transit nodes per domain	no. of stub domain per transit node	no. of stub nodes per domain
TS1k-0	1	4	5	50
TS1k-1	2	10	5	10
TS1k-2	1	10	5	20
TS10k-0	10	10	4	25
TS10k-1	5	4	5	100

Table 3.2: Configurations of the transit-stub topologies

networks (with 1000 nodes), we use group sizes ranging from 32 to 256, while for larger networks (with 5000 to 10000 nodes), the group size ranges from 32 to 1024. The reason for this is to avoid the network being too densely populated by the members.

We divide the experiments into two groups, according to the multicast service models under consideration:

1. One-to-many: one of the members is elected as data source.

2. many-to-many: more than one of the members are data sources.

For both cases, the first node to join a session will become the tree root for tree-based protocols (e.g. HMTP, TBCP, etc). For the one-to-many case, the root node is also the data source for the group. Unless specified otherwise, all members are potential data sources in the many-to-many scenario. This is because the data delivery quality depends on the location of the senders, especially for protocols that route packets over a single shared tree. In a shared tree, a sender that is a remote descendent of the tree root will result in a higher end-to-end delay compared to a sender that is close to the root.

To account for the bandwidth limitations and its heterogeneity in end system multicast, each member is assign a degree bound as discussed in Section 3.1. Typically, the node degree ranges from 2 to 10. To account for any impact of the degree distribution to the proposals investigated, we consider two degree assignment distributions: (i) uniform; and (ii) truncated binomial. With the uniform distribution, the given degree range is assigned to the overlay nodes roughly with the same probability. With the binomial distribution, degrees around a given mean value have more chances of being assigned to the nodes. We experiment with several values of the mean, which represent the lower end, middle and upper end of the given range. The truncated binomial distribution is also used by Shi in [86].

3.2.4 Performance Metrics and Representation of Results

This section discusses the performance metrics used and how we present our results in the rest of this thesis.

3.2.4.1 Performance Metrics

Our investigations focus on the quality of the delivery structure built and the properties of the technique under consideration.

The quality of the data distribution is judged by the following metrics:

- **Relative Delay Penalty (RDP)** [21]. RDP represents the additional delay incurred by an ALM solution. We consider two versions of RDP: (i) *RMP*, the ratio between the maximum delay using the overlay and the maximum delay using direct unicast connections; and (ii) *RAP*, the ratio between the average delay using the overlay and the average delay using direct unicast connections. These metrics were introduced by Castro et al. in their comparison of distributed-hash table based ALM proposals [17]. For the one-to-many scenario, the delay is measured from the data source to all other nodes; for many-to-many scenario, the delay is measured between all node pairs. We note that both RAP and RMP are normalised by two different denominators. Consequently, in some cases, RAP may be larger than RMP. To avoid confusion, they should be interpreted independently.

  The original RDP [21] is a per-pair metric, specifically, it is the ratio between the overlay delay and the unicast delay between two nodes. The reason for not using RDP is that it sometimes provides a false indication of the delay performance, where a large RDP does not necessary mean a large delay. For example, consider two nodes which has a very short unicast distance between them, say 10 ms. If their overlay delay is 50 ms (which is a reasonably small delay value), their RDP will be as high as 5. RAP and RMP avoid this problem by considering the delays of all pair of nodes.

- **Tree Cost Ratio** [109]. Tree cost is defined as the sum of delays on the tree's links (see Section 2.2). It provides a simplified view of the total network resource consumption of a tree. We calculate the ratio of the cost between an overlay tree and the corresponding network layer multicast tree. The network layer multicast tree is a shortest path tree rooted at the router where the first joining member attaches itself. This is to simulate the tree calculated due to IP multicast protocols such as DVMRP [25]. For the many-to-many experiments, when a technique uses more than one tree for data forwarding (e.g. NICE and Narada), we calculate the tree cost as the average of all trees used.

- **Link Stress** [21]. Link stress represents the number of copies of an identical packet sent over a single physical link. It represents the redundant traffic imposed by an ALM solution, as the stress for a network layer multicast is always one. We measure the worst-case (or maximum) and average stress due to an overlay solution. The maximum link stress gives the maximum number of duplicates seen by any single physical link, while the average link stress is calculated as the sum of all link stress divided by the total number of physical links involved.

Other properties of a technique that may influence protocol selection include the communication overhead, convergence and failure recovery speed. The measurement of convergence and failure recovery will be explained in more detail in later chapters.

For communication overhead, we measure the number of bytes of *control* messages sent and received by the overlay members. We assume that each message is carried using TCP over IPv4, which incurs a basic penalty of 40 bytes per packet. We further assume that all distributed proposals studied use a common packet header (20 bytes) similar to ALMI [72], which includes information such as protocol version, session identifier, packet source identifier, sequence number, control flags and packet length. This roughly represents the basic information requires for two protocol hosts to communicate properly. Any additional information, in particular, node identifier (4 bytes) and distance metric (4 bytes), is added on top of the basic size. We do not consider the overhead due to network measurement. This is because, typically, each overlay node only needs to know the distances between itself and a small number of members. The distances can be cached to reduce measurement overhead. In addition, it may be possible to obtain the distance information from an Internet distance service such as IDMaps [35] (see Section 2.7.3), as they become more widely deployed.

3.2.4.2 Representation of Results

In this thesis, we typically present the results as the average of 50 independent runs of a simulation scenario. Each scenario uses a similar ALM proposal, proposal-specific settings and general settings such as topology and group size, but with different set of group members. We report only the average values so as to avoid cluttering up the figures showing results, due to the number of proposals being considered. We have found that the average values can represent the data rather well. We illustrate this by showing the results of tree cost ratio and RMP with 95 percent confidence interval for several representative proposals that we study in Chapter 5, in Figures 3.2 (a) and (b). We can see that the error ranges are reasonably small, and the difference between the proposals can be clearly observed from the figures, so we do not include the error bars in later graphs. Typically, the cost ratio has smaller error bars than RMP. We believe that this is because the selection of members has more influence on the delay

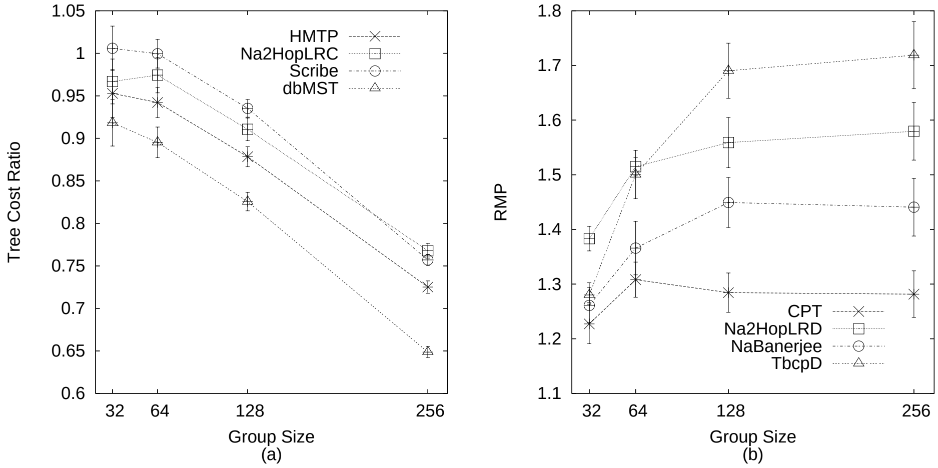


Figure 3.2: Sample comparison results taken from Chapter 5

(RMP) than the tree cost.

3.2.5 The Simulator: ALMSim

We have developed ALMSim, a packet-level, discrete-event simulator written in Java. This section sketches the high-level design of ALMSim and describes some of its features.

3.2.5.1 ALMSim Design

ALMSim is designed to investigate the efficiency of different overlay construction proposals. It takes inputs from a configuration file and from the command line which specify the simulation scenario. Each scenario consists of the configurations for the network topology, multicast members and the proposal under study. ALMSim runs the chosen proposal to build a multicast overlay, and evaluates the overlay with the performance metrics described previously.

In order to capture metrics such as the communication overhead and convergence speed, a packet-level simulation is needed. While there exist many general purposes packet-level network simulator (e.g. ns [69] and j-sim [46]), we chose to write our own. Existing simulators like ns often represent the network in a finer detail than we required. For example, for memory and processing overhead consideration, we do not simulate packet queuing in ALMSim (see Section 3.2.5.2 for details); however, this is an integrated part at these simulators. Such detailed simulation consumes substantial memory, processing power and time, which prevents large-scale evaluation.

In order to use ALMSim as a comparison platform for different techniques, it is important that new techniques can be added easily. This is done by separating the overlay construction protocols from the underlying network operations. The underlying network performs operations such as packet forwarding and statistic tracking, which are independent of the upper layer overlay construction technique. This will be discussed in detail in the next section. Together with the functional separation, the use of object-oriented design principle in the programming improves the extendibility of `ALMSim`.

An important feature of a comparison platform is the ability to configure various parameters easily. `ALMSim` provides flexible configuration by reading simulation settings from an input file and from the command line options. The inputs from the command line will override the inputs from the configuration file. In this manner, a simple batch script can be used to automate lengthy simulations.

3.2.5.2 Functional Components

`ALMSim` consists of the following core functional components: parsers, network representation, events and scheduler, statistic collection and centralised algorithms.

Parsers `ALMSim` consists of two main parsers. One reads inputs from a configuration file and from the command line to determine the simulation scenario (e.g. protocol parameters, group size, simulation time, etc). The other parser reads a topology file from the topology generator (i.e. GT-ITM or BRITE) to construct the network structure.

Network representation The physical network is represented by nodes as routers, links which connect two adjacent routers, protocol agents which implement the end-system multicast protocol and packets which carry data and control information between the agents. The physical link is characterised by its propagation delay (we do not model bandwidth as we do not model packet queuing, see below). The protocol agent is attached to one of the physical nodes.

This architecture is essentially adopted from `ns`, with substantial simplification. In order to reduce the complexity of a simulation, we do not model the queuing delay and packet loss in the network. This is a common assumption in the study of ALM overlay construction proposals (e.g. evaluation of Narada [21]). Without packet queuing, when a router receives a packet, it immediately sends the packet out via the appropriate interface. Packet queuing is only relevant if the network carries a large amount of traffic. However, as described previously, each of our simulation experiment involves only a single overlay construction proposal. The only traffic in the network are relatively small control messages exchanged between the protocol agents, and occasional data packets injected into the network for statistic purpose (i.e. link stress measurement). We also assume that the physical links are reliable such that there are no losses of protocol messages between the members. Note that the reliability of the applications data is out of the scope of this thesis. In addition, we assume that the physical network is fixed throughout a simulation, i.e. we do not simulate failure of physical nodes or links, and the physical links' delays are fixed. Finally, routing in the network layer is based on the shortest path first policy.

Events and scheduler These are the core and "engine" of an event-driven simulator. An event is associated with a time stamp, an action and an identity of the corresponding network element (e.g. a router or a protocol agent) where the action is to be performed. Examples of the events are member join, leave, fail, send data, packet arrival and evaluation[2]. The scheduler orders the list of events according to increasing time stamps. It is implemented as a priority queue using the heap data structure [23]. The event list is initialised with certain known events and their associated times. The scheduler takes the top event (with the smallest time stamp) and executes the action associated with the event. The virtual clock of the simulation "jumps" to the time stamp of this event. The execution of an event might add more future events to the event list. The scheduler repeats this process until the event list exhausted or the virtual clock exceeds a predefined stopping time.

Statistic collection `ALMSim` collects a number of statistics for computing the desired performance metrics. The statistic collector is integrated with the network structure. For examples, a stress collector is associated with each physical link to count the number of duplicated data packets that flow over the link; and each protocol agent keeps track of the number of control messages sent and received to measure the protocol overhead.

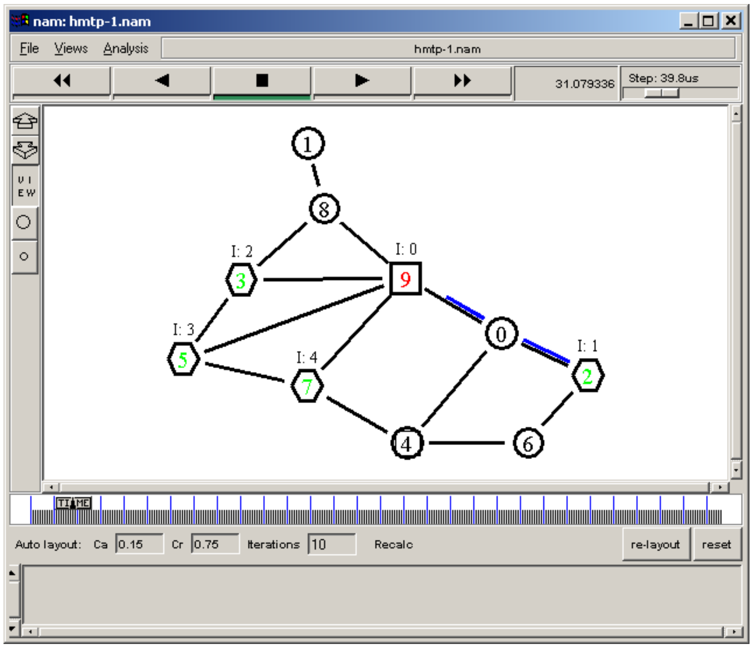
Centralised algorithms In addition to the distributed protocols, we also investigate several centralised algorithms in overlay construction. Examples are various heuristics for the degree-constrained minimum cost and minimum delay tree problems (see Chapter 4). Each of these algorithms take inputs of the network topology and the multicast members, and outputs an ALM overlay.

3.2.5.3 Validation Efforts

One major concern with simulation experimentation is the correctness of the simulation tool. We have taken substantial efforts to validate `ALMSim`.

`ALMSim` provides a detailed events and packets logging facility. This enables the user to inspect each simulation step, and thus validate the running of the simulation engine as well as the overlay construction techniques. For example, by comparing the step-by-step execution of an implemented proposal with its original specification, we could verify the correctness of our implementation. For the various techniques implemented, we also try to reproduce the results based on the published work and compare the results. In some cases where faithful replication of the experiments is impossible (e.g. due to the unavailability of the topology and selection of the multicast members), we use reasoning to compare the observed results with the expected behaviour of the technique. `ALMSim` is able to produce a `NAM` [3] formatted packet events file. `NAM` is a popular network animator closely associated with `ns`. It can be used to visualise the communication patterns between the nodes. We typically use this technique to identify quickly misbehaving communication patterns (e.g. unnecessary packet looping) in a protocol, and resource to the detailed events and packets logs to actually locate the error. Figure 3.3 shows an instance of NAM

[2] The evaluation event triggers the calculation of the performance metrics.



Auto layout: Ca 0.15 Cr 0.75 Iterations 10

Figure 3.3: An example showing an instance of NAM

running an animation for HMTP, on top of a 10 nodes network. In the network shown, circles represent routers, box[3] and hexagons represent the nodes with end systems.

3.3 Chapter Summary

In this chapter, we describe the system model used in this thesis. Our model assumes that multicast members are end systems which directly take part in overlay construction and maintenance. We abstract the bandwidth limitation at these members as the degree constraints. This requires the resultant ALM delivery structure to be degree bounded. We also discuss the design of our simulation environment, which includes the topologies, multicast group, performance metrics and the working of `ALMSim`, our ALM simulator. `ALMSim` is used for all evaluations reported in this thesis.

[3] The boxed node is the first member of the multicast group.

Chapter 4

GreedyMesh

This chapter presents a centralised greedy heuristic, called GreedyMesh, for the NP-complete minimum diameter degree-bounded subgraph problem. The subgraph is in the form of a mesh, which serves as a shared structure from which degree-bounded source-specific trees can be obtained.

The need for such an algorithm arises from our evaluation of distributed overlay building proposals. We first recall that distributed proposals build ALM trees with only limited knowledge of the underlying topology. Typically, the trees are degree constrained. One of the key performance metrics is the data delivery delay from the source to the recipients. It is interesting to compare the delay performance of these distributed proposals with an optimum or near optimum solution using a centralised algorithm. In Section 3.2.4.1, we have seen that the delay of an ALM overlay is judged based on two versions of relative delay penalty (RAP and RMP), compared to the corresponding IP multicast tree. However, IP multicast trees are formed directly by the underlying physical links, and are not concerned with the degree limitation at the members. In other words, this does not provide a comparison with trees created using overlay links, and subjected to members' degree constraints. GreedyMesh provides an approximate solution to create low delay degree-bounded source-specific trees. Hence, it is suitable for comparison with distributed proposals for many-to-many multicasting.

The rest of this chapter is organised as follows. The next section provides a background on the mesh creation problem, along with discussion on some related works. In Section 4.2, we present and analyse GreedyMesh. In Section 4.3, we examine the performance of GreedyMesh by comparing it with another centralised algorithm. Finally, Section 4.4 concludes this chapter.

4.1 Background

This section first formally describes the subgraph problem and discusses its NP-completeness, which is followed by discussion on some related research.

4.1.1 Minimum Diameter Degree-bounded Subgraph Problem (MDDSP)

The overlay network is modelled as an undirected complete graph $G = (V, E)$, where V is a set of vertices representing the multicast members, and $E = V \times V$ is the set of edges. Each overlay edge, $e \in E$, has an associated communication delay, $c(e)$, and each vertex, $v \in V$ of graph G, has an associated degree constraint, $d_{max}(v)$. The diameter of a graph is the maximum shortest path distance between any two vertices via the graph. The minimum diameter degree-bounded subgraph problem (MDDSP) can then be stated as follows.

> Given an undirected complete graph $G = (V, E)$, a degree bound $d_{max}(v) \in N$ for each vertex $v \in V$ and a cost $c(e) \in Z^+$ for each edge $e \in E$; find a subgraph G' of G of minimum diameter, subject to the degree constraints, $d_{max}(v)$ for all $v \in G'$.

The NP-completeness of MDDSP can be proven by showing that the decision version of the problem — finding a diameter- and degree-bounded subgraph — is NP-complete. The diameter-bounded subgraph problem can be viewed as the well-studied k-spanner problem. Consider a connected graph, $G = (V, E)$. A subgraph $G' = (V, E')$ is a k-spanner if for every u, $v \in V$,

$$\frac{dist(u, v, G')}{dist(u, v, G)} \leq k, \tag{4.1}$$

where $dist(u, v, G')$ denotes the distance from u to v in G' [52]. In other words, the k-spanner bounds the diameter of the subgraph to k times the diameter of the original graph. The degree-bounded k-spanner problem has been proven to be NP-complete by Kortsarz and Peleg in [52]. Hence, MDDSP is also NP-complete.

4.1.2 Related Work

As discussed above, MDDSP is closely related to the k-spanner problem. In [52], Kortsarz and Peleg consider a special case of a 2-spanner with minimum maximum degree. They propose a probabilistic algorithm that computes a 2-spanner that has a maximum degree that is no more than $\Delta^{1/4}$ times the optimum solution, where Δ is the maximum vertex degree in the resultant spanner. The algorithm involves solving an Integer Linear Program, which could be very time consuming. More importantly, their algorithm does not try to honour the degree bound of each individual vertex, and thus is not suitable for our purpose. Most studies on k-spanner consider the *sparsity* of the resultant spanner, where the spanner has as few edges as possible. As explained by Kortsarz and Peleg, doing so may result in a vertex with very high degree, which is undesirable for our purpose.

In [87], Shi et al. consider the problem of generating minimum diameter degree-bounded trees, which is also NP-complete. The obvious difference between our problem and theirs is that they consider a tree that must be loop-free, while our subgraph (mesh) can have loops. A tree is directly applicable for multicasting, while we need to first derive trees out of a mesh. Their algorithm is thus more suitable for comparing ALM proposals that use a single tree, such as HMTP [109], TBCP [62], etc; while our algorithm can be compared with proposals such as Narada [21] that create source-specific trees out of an overlay mesh.

Shi et al. devise a greedy heuristic called Compact Tree (CPT) for the problem. CPT maintains a partial tree, T, which initially contains only the tree root, $r \in V$. CPT then incrementally grows the partial tree, T, until it includes all vertices. At each iteration, the new feasible edge (with one of its endpoints in T while the other one has yet to be included in T) which provides the smallest increase in the objective function (i.e. tree diameter) is added to T. An edge is feasible if adding it will not cause degree violation of its two end vertices. The complexity of CPT is $O(|V|^3)$. Our GreedyMesh algorithm works in a similar fashion to CPT, in the sense that it incrementally adds in new edges to a partial subgraph until no more edges can be included.

In [60], Malouch et al. look at the problem of constructing a degree-bounded tree with given delay bound, under a mixed end-systems and proxies network model. Their algorithm is quite similar in nature to CPT, except that they try to minimise the delay from the root to its farthest descendant (root diameter), rather than the delay between any two nodes on the tree (tree diameter). (Note that it is straightforward to modify CPT's optimisation objective so as to generate trees with small root diameters.) In [51], Konemann et al. study the minimum root diameter degree-bounded tree problem. However, their solution only attempts to bound the overall maximum degree, rather the degree for each individual vertex.

4.2 GreedyMesh Algorithm

This section presents GreedyMesh, our simple greedy mesh construction algorithm. The inputs to GreedyMesh are the complete graph, $G = (V, E)$ (which provides the complete distance matrix between the vertices), and the degree constraints for the vertices, $d_{max}(v)\ \forall\ v \in V$. GreedyMesh outputs a subgraph, $G' = (V, E' \subseteq E)$ which fulfils the degree constraint at each individual vertex. Figure 4.1 illustrates the steps of the algorithm.

GreedyMesh starts with a minimally connected subgraph, i.e. a tree that spans all vertices. The initial tree can be generated by any degree-bounded spanning tree algorithm. This will be explained shortly in Section 4.2.1. After obtaining the initial subgraph, G', GreedyMesh computes the spare degrees for all vertices, as in lines 2 and 3. Next, GreedyMesh inserts the vertices that still have some spare degree into an initially empty set, F. From line 7 onwards, GreedyMesh enters the main loop where new edges are added into G' one at a time, until no more edges can be added. Within the loop, GreedyMesh first obtains a vertex, u, from F. Our implementation simply selects a vertex from F in a round robin manner. Next, we calculate the set of delays from u to all other vertices via G', using Dijkstra's shortest path algorithm [23]. From lines 10 through 16, we sum up the delay gain observed from u to all other

```
Algorithm : GreedyMesh Algorithm
Input: Complete graph $G(V, E)$, Degree constraints $d_{max}(v)$
Output: Connected degree-bounded mesh, $G'(V, E'), E' \subseteq E$, s.t. $d_{max}(v)\ \forall\ v \in V$
GreedyMesh($G, d_{max}$)
        genDBST($G, d_{max}$) := Generate a degree-bounded spanning tree from $G$ subjects to $d_{max}$
        getNode($S$) := Get a node from the given set, $S$
        sptAlg($u, G$) := Compute the shortest path distances from $u$ to all other nodes
        $d_{used}(v)$ := Current used degree for vertex $v$
        $d_{spare}(v)$ := Current spare degree for vertex $v$
        $F$ := Set of vertices with spare degree, $d_{spare}(v) > 0$
        $D_u$ := Set of shortest path delays from $u$ to other nodes
        $U_{u,v}$ := Delay gain for $u$ if a link to $v$ is to be added
(1)     $G'(V, E') \leftarrow$ genDBST($G, d_{max}$)
(2)     foreach $v \in V$
(3)         $d_{spare}(v) \leftarrow d_{max}(v) - d_{used}(v)$
(4)     $F \leftarrow \emptyset$
(5)     foreach $v \in V \wedge d_{spare}(v) > 0$
(6)         $F \leftarrow F \cup \{v\}$
(7)     while $|F| > 1$
(8)         $u \leftarrow$ getNode($F$)
(9)         $D_u \leftarrow$ sptAlg($u, G'$)
(10)        foreach $v \neq u \wedge \langle u, v\rangle \notin E' \wedge v \in F$
(11)            $G'' \leftarrow G'(V, E' \cup \langle u, v\rangle)$
(12)            $D'_u \leftarrow$ sptAlg($u, G''$)
(13)            $g \leftarrow 0$ /* $g$ := gain */
(14)            foreach $w \in V \setminus \{u\}$
(15)                $g \leftarrow g + \frac{D_u(w) - D'_u(w)}{D_u(w)}$
(16)            $U_{u,v} \leftarrow g$
(17)        $b \leftarrow \arg\max\{\, U_{u,v} : \forall v \in F\}$ /* $\langle u, b\rangle$ := edge with the largest gain */
(18)        $G' \leftarrow (V, E' \cup \langle u, b\rangle)$
(19)        $d_{spare}(u) \leftarrow d_{spare}(u) - 1$
(20)        $d_{spare}(b) \leftarrow d_{spare}(b) - 1$
(21)        if $d_{spare}(u) \equiv 0$
(22)            $F \leftarrow F \setminus \{u\}$
(23)        if $d_{spare}(b) \equiv 0$
(24)            $F \leftarrow F \setminus \{b\}$
```

Figure 4.1: The GreedyMesh algorithm

vertices, for each feasible edge incident from u, when added to G'. The delay gain function is adopted from Narada's [21] utility function (see Figure 2.10). In line 17, we locate the edge, $\langle u, b\rangle$ that gives the largest delay gain. Line 18 inserts the edge into G'. From line 19 through 24, we update the spare degrees for b and u, as well as removing them from F if necessary. Note that if u fails to find a feasible edge, it will also be removed from F. Eventually, the algorithm terminates when there is only one vertex in F, indicating that no more edges can be added.

The computed subgraph will be connected as long as the initial tree structure is connected. There are two possible cases where a connected tree cannot be found [87].

1. The total available spare degree of the vertices is less than the minimum required degree, which for a tree, is $2 \times (|V| - 1)$. This requirement can be easily calculated as follows: a tree has $|V| - 1$ edges and each edge uses one incident edge at each of its two end-vertices. We could run a check on the available degree of the vertices before running GreedyMesh.

2. During the creation of the tree, a vertex with a degree constraint of one (i.e. a leaf vertex) is added to the tree when the total spare degree of all vertices in the partial tree is equal to one. This reduces

the total spare degree of the partial tree to zero, hence no new vertices can be added to it. The tree creation algorithm can alleviate this problem by keeping a count of the spare degree of the partial tree, and defer the addition of a leaf vertex if it reduces the count to zero.

For simplicity, our evaluation only considers vertices with degree constraint greater than or equal to 2.

4.2.1 Variants of GreedyMesh

In the GreedyMesh algorithm, there are a number possible ways to generate the initial degree-bounded spanning tree, as represented by the `getDBST` function in line 1. In Section 4.3, we examine the performance implications of using three different initial tree structures: (i) random degree-bounded tree; (ii) degree-bounded minimum spanning tree; and (iii) degree-bounded minimum diameter spanning tree.

- *Random degree-bounded tree*. We generate a random tree by growing a partial tree in an incremental manner. Starting with an arbitrary vertex, we randomly pick a new feasible edge into the partial tree until all vertices are in the tree. The random tree is used as the worst-case scenario to compare the following more informed tree construction algorithms.
- *Degree-bounded minimum spanning tree*. This tree creation problem is also an NP-complete problem [50]. There have been many heuristic solutions for the problem. We consider a simple heuristic [67], which is based on Prim's algorithm for the (unconstrained) minimum spanning tree problem [23]. The algorithm starts with a partial tree containing an arbitrary vertex. At each iteration, it adds the shortest new eligible edge to the partial tree (recall that we treat link delay as link cost). The algorithm continues until all vertices are connected. The time complexity of the algorithm is $O(|E| \log |V|)$. In the rest of this thesis, we will refer to the heuristic as dbMST.
- *Degree-bounded minimum diameter tree*. We use the Compact Tree algorithm (CPT) by Shi et al. [87] to generate such a tree. We use the most central vertex, i.e. the vertex that has the smallest distance to all other vertices, as the tree root. As mentioned in Section 4.1.2, the complexity of this algorithm is $O(|V|^3)$.

4.2.2 Analysis of the Algorithm

This section analyses the time complexity of GreedyMesh. First of all, GreedyMesh calls for a degree-bounded spanning tree algorithm to compute a connected subgraph. There are a number of possible choices of degree-bounded spanning tree algorithm as explained in the previous section. For now, let us assume that the running time of such an algorithm is $O(\lambda)$.

Let Δ be the maximum spare degree for all vertices in the initial subgraph. It is easy to see that the main `while` loop (line 7) runs for at most $\Delta\,|V\,|$ times. Each iteration executes a Dijkstra's shortest path algorithm calculation. By using a heap implementation, the Dijkstra algorithm has a run time of

$O(|E| \log |V|)$ [23]. The first `foreach` loop (line 10) runs at most $|V|$ times, with each iteration involving a Dijkstra shortest path calculation. Hence, the run time of the main loop can be calculated as $\Delta\ |V|\ (|E| \log |V| + |V||E| \log |V|) = O(\Delta\ |V|^2|E| \log |V|)$.

Consequently, the overall running time of the algorithm is $O(\max\{\lambda, \Delta\ |V|^2|E| \log |V|\})$. The worst-case with Δ close to $|V|$, results in a run time of $O(\max\{\lambda, |V|^3|E| \log |V|\})$. However, this is unlikely to happen in a practical environment, especially if the multicast members are end users' machines.

4.3 Performance Evaluation

We first investigate the impacts of initial tree layout for GreedyMesh in Section 4.3.1. Section 4.3.2 then compares GreedyMesh with source-specific trees generated with CPT.

We use the algorithms to build overlays out of a set of 1000-node topologies (see Section 3.2.2). We have found that the performance trend of the algorithms stay quite similar across the topologies. Hence, we only show results from a representative topology, i.e. TS1k-0 (see Table 3.1). A more detailed investigation on the impacts of the underlying topologies can be found in Section 5.2.4. In the experiments, the multicast group members are randomly chosen, and the group size ranges from 32 to 256. The degree constraint of each member is randomly assigned from a value between 2 to 10, using a uniform distribution. As GreedyMesh is used to create source-specific trees, we only consider the many-to-many data delivery model. With this, we assume that every member is a data source. For all the results presented, each data point in the graph represents an average over 50 independent runs.

We judge the algorithms based on the quality of the overlays created, using the metrics introduced in Section 3.2.4.1. To recap, RAP (RMP) represents the ratio between the average (maximum) delay using the overlay and the average (maximum) delay obtained with direct unicast transmission; tree cost ratio represents the ratio between the overlay tree cost and the IP multicast tree cost; and finally, link stress denotes the number of identical copies of a packet flows over a single physical link. For all these metrics, the smaller the value, the better the performance. RMP also gives an indication of how well the algorithms minimise the overlay diameter.

4.3.1 Impacts of Initial Tree Layout

We first consider the quality in terms of RMP for trees generated by the three tree generation algorithms. Each of the trees is used as a bidirectional shared tree. Unsurprisingly, the result in Figure 4.2 shows that their performance is in the following order: CPT gives the best RMP, which is followed by dbMST and finally, the random tree. Note that the random tree's RMP at group size 256, which is around 8.0, is omitted to provide a clearer view of dbMST and CPT curves.

Figure 4.3 (a) to (d) compare the quality of GreedyMesh's overlays based on the above three initial

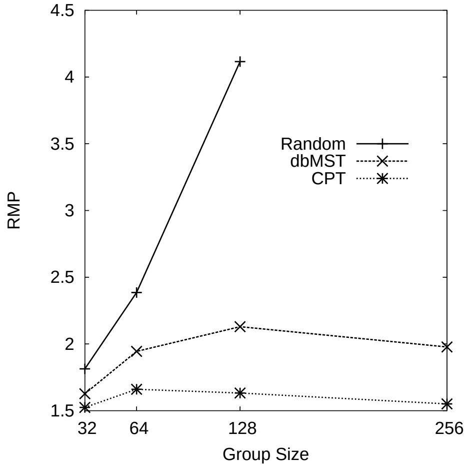


Figure 4.2: Delay performance of the tree generation algorithms

tree structures. From the figures, it is clear that the initial tree structure has a significant effect on the resultant mesh. We can observe that meshes created from dbMST performs the best, which is followed by CPT and finally the random tree, in all the performance metrics considered. Comparing the RMP performance with Figure 4.2, we can see the extra links added by GreedyMesh indeed improves the original structures' delay. In particular, averaged over all group sizes, GreedyMesh improves upon CPT by 14.7%, dbMST by 38.1% and random tree by 59.4%, respectively.

To explain the observation, we first consider CPT. CPT is designed to create low diameter trees. To do so, it incrementally grows a partial tree by adding new vertices that result in the smallest increase in tree diameter. Let us assume that all edges have unit weight, i.e. the distance between each node-pair is one. We can see that CPT will grow the partial tree level-by-level: starting from the root, r, it first adds vertices to become r's children; once r has used up all of its spare degree, the algorithm will try to add new vertices to r's children, i.e. the next level of the tree. The process continues from one level to the next level, until all vertices are in the tree. Hence, the resultant tree will have a compact structure where vertices at higher levels (close to r) are full of children. In our experiments, the edges have widely different weights. But generally, trees generated by CPT will still have a compact structure. The compact structure, when used in GreedyMesh, allows few edges to be added to vertices at higher tree levels. In other words, new edges are mostly introduced to connect a leaf or vertices near the leaf, which results in little delay gain. This can be confirmed from the smallest RMP improvement (14.7%) observed above.

The dbMST on the other hand tries to connect the vertices using all the shortest edges. In this way, the tree clusters together vertices that are topologically close. This results in a small delay between vertices that are in a "cluster". The extra edges introduced by GreedyMesh then help to reduce the distance between vertices in different clusters, and this results in an overall low delay mesh. The inclusion of many short links also explains the better tree cost ratio and link stress performance.

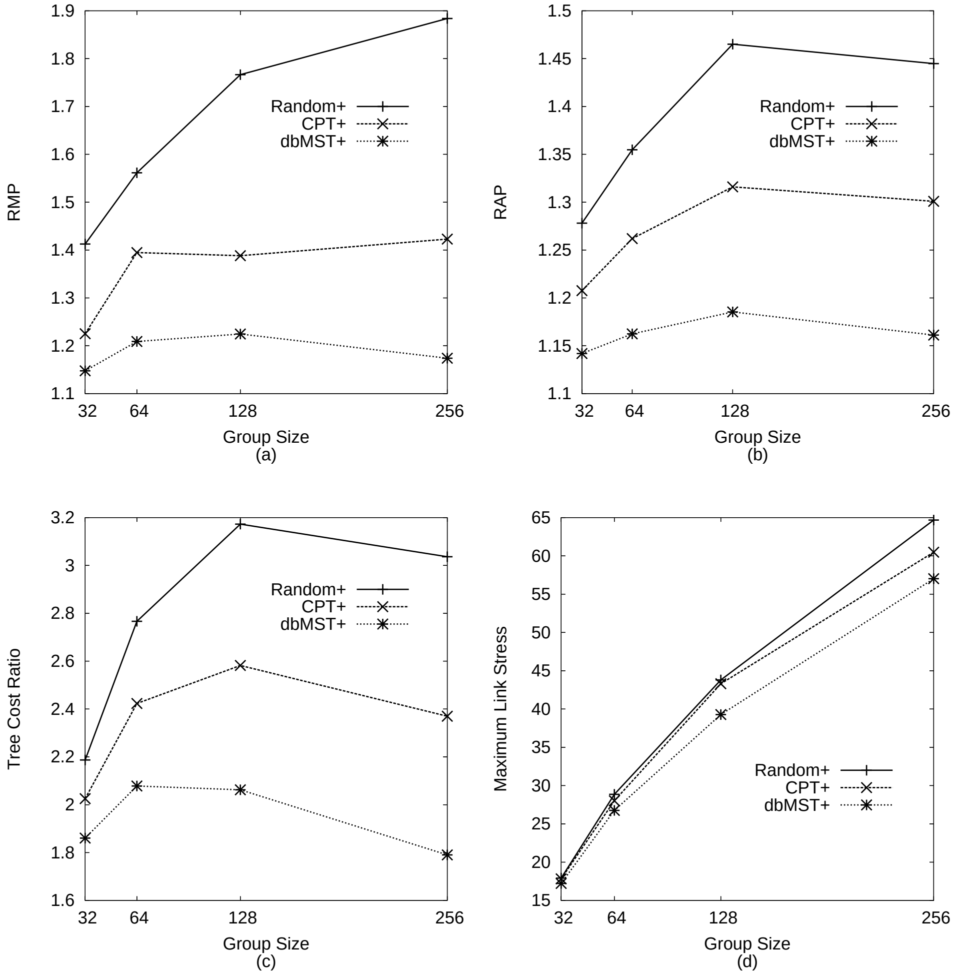


Figure 4.3: Comparing variants of GreedyMesh algorithm

Finally, in the random tree structure, edges are added arbitrarily. It is possible that some of the edges connect vertices that are far apart, which results in many ineffective triangles (see Section 2.2.3) in the tree. Hence, it gives the worst performance.

4.3.2 Comparison Study

We are not aware of any other algorithm that attempts to create low diameter degree-bounded meshes like GreedyMesh. The closest candidate is an algorithm for the low diameter degree-bounded tree, such as CPT. In the previous section, we have shown that GreedyMesh could improve upon the shared trees generated by CPT. To provide a fairer comparison, we use CPT to generate a set of source-specific trees for each member. For example, for a group of 32 members, we compute 32 compact trees rooted at each member. With a source-rooted tree, we would like to minimise the root diameter, instead of the tree diameter which is more appropriate for shared tree data delivery. We thus modify the original CPT

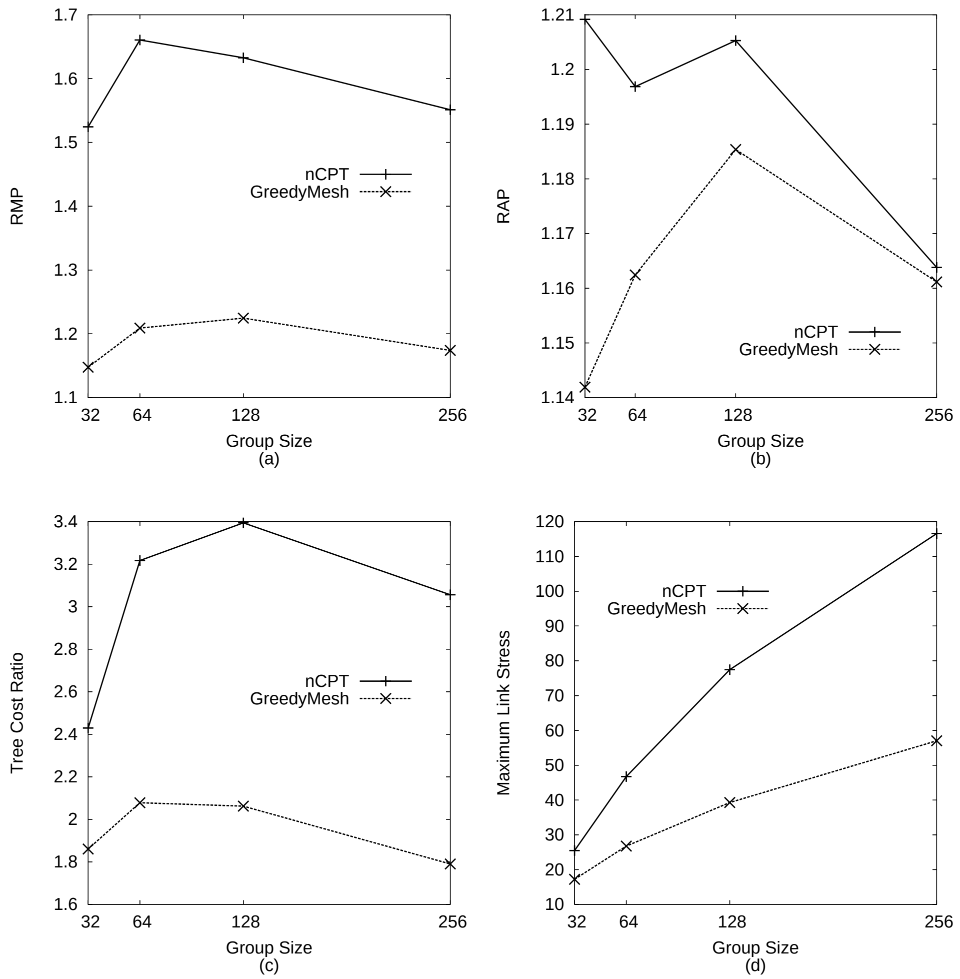


Figure 4.4: Comparing GreedyMesh with nCPT

such that when adding a new vertex to the partial tree, the vertex that results in the smallest increase in the root diameter is chosen. We will refer to this version of CPT as nCPT. For GreedyMesh, we use the dbMST to create the starting tree structure.

Figure 4.4 (a) to (d) depict the comparison results. For RMP, it is clear that GreedyMesh significantly outperforms nCPT. In terms of RAP, the performance advantage of GreedyMesh diminishes with the group size. It is worth noting that GreedyMesh creates trees that are limited by a shared mesh. On the other hand, trees generated by CPT are independent of each other. Potentially, for a multicast group, the total number of links used by CPT's trees will be more than those of GreedyMesh's mesh. To prove this, we plot the total number of unique overlay links used by GreedyMesh and nCPT in Figure 4.5. For GreedyMesh, this is the number of links contain in its mesh overlay; for nCPT, it is the total number of unique overlay links used by all the source-specific trees. The figure also shows the result for a shared tree, denoted as SharedTree. Obviously, SharedTree uses the smallest number of links, i.e. one less than

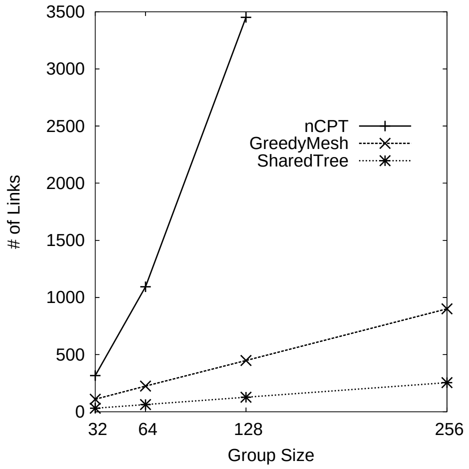


Figure 4.5: Number of links used by a shared tree, GreedyMesh's mesh and nCPT's source-specific trees

the group size. The results clearly confirm that nCPT uses significantly more links than GreedyMesh. We omit the data point for nCPT at group size 256 (which is around 10000) for the sake of GreedyMesh and SharedTree results. The advantage of having fewer links is that it is more manageable, if the overlay is to be used in a real-world environment. Traditionally, having more links is often considered as a way for load sharing. However, this benefit is unclear in the case of overlay networks. This is because each overlay link potentially traverses multiple physically links, and it is possible that the traffic flowing over different overlay links will be mapped onto the same physical links.

In terms of the tree cost ratio (TCR) and link stress, it is clear that nCPT results in poorer resource usage and larger link stress. This is because GreedyMesh is based on the minimum spanning tree, hence consists of a large number of short overlay links between the members. We recall that we defined tree cost as the summation of the delays of the overlay tree links. Thus, short links reduce the tree cost. In addition, the shorter an overlay link, the smaller the likelihood that it traverses multiple physical links; hence this reduces the chances of packets duplication.

Note that RMP, RAP and TCR are relative values. While their curves show an inconsistent growth trend (i.e. up-and-down), the absolute tree cost and overlay delays observed (not shown here) actually increase with the group sizes.

4.4 Chapter Summary

This chapter presents GreedyMesh, a greedy heuristic for the minimum diameter degree-bounded subgraph problem. From the resultant subgraph, we could derive a source-rooted tree for each vertex. Thus, it is suitable as a benchmark for distributed ALM proposals that use source-specific trees for many-to-many multicasting, e.g. Narada. GreedyMesh will be used in our comparison study in Chapters 5 and 8.

GreedyMesh could also be used in a centralised ALM proposals such as ALMI [72] to create small-scale low delay overlays. In such a case, the overlay members may need to implement a routing protocol to obtain the delivery trees.

GreedyMesh grows a subgraph incrementally, beginning from a degree-bounded tree. The tree can be generated by any existing degree-bounded tree creation algorithm. The initial tree structure has significant impacts on the quality of the resultant subgraph. We have found that the degree-bounded minimum spanning tree provides the best starting structure. We also show that GreedyMesh performs well compared to source-rooted trees created by the Compact Tree algorithm.

Chapter 5

A Performance Comparison of Existing ALM Protocols

In Chapter 2, we reviewed a number of distributed ALM proposals which lead to a variety of choices in creating and optimising multicast overlays. It is essential to understand the strengths and weaknesses of these existing proposals. Previous studies of the performance of these techniques either consider only a small number of approaches (e.g. [7]), or techniques that exhibit a similar nature (e.g. [47, 17]), or merely provide high-level descriptive comparison (e.g. [29, 6]). More importantly, as most of the proposals were evaluated under different assumptions and simulation settings, it becomes difficult to relate the effectiveness of the techniques.

As described in Chapter 3, we represent the bandwidth limitation of the nodes by creating degree bounded overlays. This narrows down our focus to two widely considered metrics: tree cost and delay. As described in Section 2.2, tree cost indicates the network resource consumed by an overlay tree. Hence, a low cost tree is suitable for bulk data transfer. On the other hand, delay is important for applications that require timely delivery. In this chapter, we conduct a detailed performance comparison of several tree cost and delay optimised protocols, under a uniform simulation environment (Chapter 3). The chosen proposals represent the different classes discussed in Chapter 2.

This work provides the first step towards improving and/or designing distributed proposals that create low delay and low cost ALM trees. The experiments were designed for two main purposes: to understand the properties of different proposals and to compare the quality of the overlays built by them. The results and observations found in this chapter lead to the development of our own proposals in later chapters. Some general findings are also applicable to other work on self-organising overlay creation techniques.

The rest of this chapter is structured as follows. In the next section, we first discuss the chosen proposals. We then report the observed results along with the analysis in Section 5.2. Section 5.3 positions this work with some other studies. Finally, Section 5.4 concludes this chapter.

5.1 Proposals Considered

The selected proposals are HMTP [109], AOM [104], variants of switch-trees [43] (which includes a variant of HostCast [57]), Banerjee et al.'s scheme [9], NICE [7], TBCP [62], Narada [21] and Scribe [15], and this selection is shown in Figure 5.1. (Note that BTP is a specific version of switch-trees that uses one-hop switching.) These proposals capture the diversity in overlay construction, optimisation and maintenance. Specifically, HMTP, AOM, switch-trees variants, Banerjee et al.'s scheme, NICE and TBCP all follow the tree-first approach. Within this group, we have both distributed transformation and the localised central arrangement techniques (Section 2.6.1). In addition, NICE's cluster-based hierarchical structure opens another avenue for comparison. For the mesh-first approach, Scribe and Narada represent the structured and unstructured mesh-based proposals, respectively. There are several reasons why we choose Scribe from the many proposals in the same class. First, it can impose strict degree constraints on the nodes, unlike the Delaunay triangulation [58] and LARK [49] protocols. There is also little significant differences between Scribe and Bayeux [112]. Finally, it has been shown to out-perform CAN-multicast [17].

In the next subsection, we discuss issues related to parameter settings and the implementation of the chosen proposals. We refer the reader to Chapter 2 for detailed description of the proposals. In Section 5.1.2, we provide a naming system for the proposals to ease the discussion.

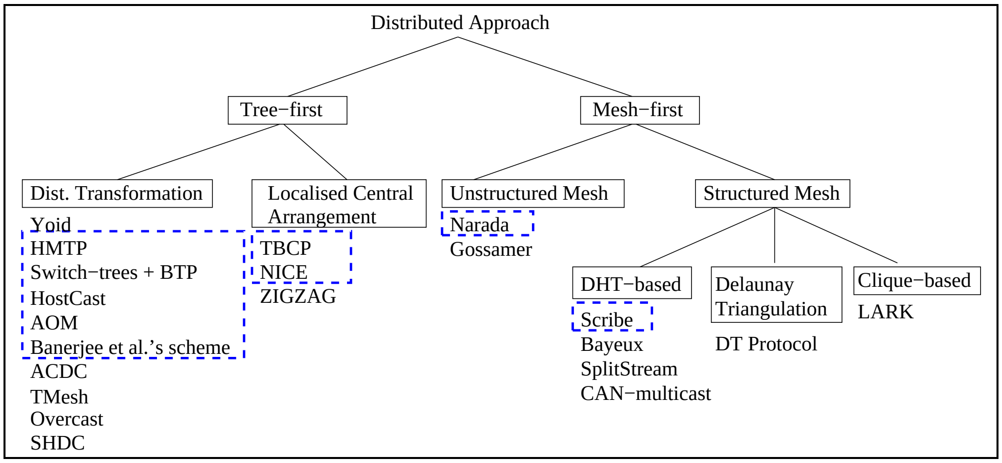


Figure 5.1: Selection of proposals in our comparison study

5.1.1 Parameter Settings, Implementation and Enhancements of the Chosen Proposals

For the aforementioned proposals, there are a number of configurable parameters that may affect their performance. Two common parameters that can greatly affect the overlays built are the frequency of the overlay improvement process and the members' degree-bounds. To provide a fair comparison, similar parameter is used when possible. For instance, all distributed proposals use an overlay improvement period of 30 seconds. Unfortunately, not all proposals produce overlays that fulfil the given degree-bound. In particular, Narada and NICE. This, and some other proposal-specific settings will be discussed in the following subsections.

We have tried to implement the chosen proposals based on their original specifications. However, for some proposals, we have made some modifications that simplify the evaluation, while preserving the original characteristics of the proposals. The following discussion also include enhancements that we have made to TBCP.

5.1.1.1 NICE and Narada

The implementation of NICE [7, 8] and Narada [21] is adapted from the `myns` simulator [66], which was used by Banerjee et al. in [7, 8]. We have ported the original `C++` code to our `Java`-based `ALMSim`.

NICE NICE builds an overlay in the form of multiple level clusters. The size of each cluster is constrained in the range of $k \leq size \leq 3k-1$, where k is an user-defined parameter. Unlike most proposals, the maximum fan-out of a NICE node can be as high as $(3k-1)\log_{3k-1} n$, for an n-node overlay. In the experiments, we typically set k to 3 (as in [7, 8]), which results in a maximum cluster size of 8.

Narada Narada builds an overlay mesh, and creates a separate tree for each of the members using the path-vector routing protocol and reverse path forwarding technique. As described in Section 2.6.2, Narada improves upon the mesh with an aid of a utility function (Figure 2.10). A node will add a new mesh link if the utility of the link exceeds a given threshold, while an existing mesh link is dropped if its consensus cost is less than a drop threshold. In [21], Chu et al. recommend that the add threshold is calculated as a function of the group size, and of the available and maximum degree of the nodes involved. On the other hand, the drop threshold must be less than or equal to the add threshold to avoid dropping and adding a link immediately. We experimented with several possible functions, such as $\max\{f_x, f_y\}$, $\max\{f_x^*, f_y^*\}$, $\frac{n}{\max\{f_x,f_y\}}$, $\frac{n}{\max\{f_x^*,f_y^*\}}$, $\frac{n}{(f_x+f_y)/2}$, $\frac{n}{(f_x^*+f_y^*)/2}$, and some other permutations of these functions, where n is the group size, x and y are the nodes involved, and f_x, f_x^*, f_y and f_y^* are the available and maximum degree of x and y respectively. We found that, on average, as long as the functions yield thresholds that are of the order of the degree of the nodes, they result in similar performance. In our experiments, we calculate the add threshold as $\frac{n}{\max\{f_x,f_y\}}$, while the drop threshold is half of the add threshold.

Narada allows each individual node to decide its own maximum degree bound. However, the actual degree of a node (i.e. the number of overlay links keeps by a node) is regulated based only on the configuration of the add and drop thresholds, as explained above. Obviously, a node could strictly enforce the degree bound by accepting new links only while it still has sparse degree. Strictly enforcing the degree bound has two drawbacks. First, it limits the degree of freedom in overlay reconfiguration, and could lead to a less efficient overlay structure. Secondly and more importantly, it could result in mesh partition, as pointed out by Banerjee et al. [7]. We have implemented two versions of Narada:

- *Narada*: This version regulates the degrees of the nodes solely based on the configuration of the thresholds.
- *Narada-SD*: This version tries to strictly enforce the degree bound — a node will reject addition of new link if it has reached its degree bound.

We have indeed found that Narada-SD occasionally causes partitioning in the mesh, while Narada causes some nodes to have excessive links. This will be discussed in more detail in Section 5.2.2

5.1.1.2 HMTP Variants

We have implemented two versions of HMTP [109] which differ only in the joining strategy. The first version simulates the original HMTP as described in Section 2.3.1.2. Specifically, newcomers first contact the tree root, and then use its greedy depth-first search (DFS) technique to find the best attachment point. In the second version, the newcomers begin the DFS from a randomly selected on-tree node. The reason for doing so is to understand the efficiency of HMTP's DFS in locating nearby nodes for tree cost minimisation (see Section 5.2.5).

5.1.1.3 AOM

AOM [104] provides three conditions (Equation 2.2 to 2.4) for nodes to decide if a switch-parent operation is beneficial. These conditions are weighted by two parameters: α and p (where $0 < \alpha < 1$ and $p > 0$), and are reproduced as follows,

- Eq. 2.2: $d(C, B) \leq \alpha \times d(C, A)$
- Eq. 2.3: $D(B, root) \leq D(C, A, root)$
- Eq. 2.4: $D(C, B, root) \leq (1 + p) \times D(C, A, root)$

where C is the node performing a switch, while A and B are C's current parent and potential parent, respectively. The value of α determines how much closer C is to B compared to the distance between C and A. The smaller α is, the closer the distance between C and B needs to be. On the other hand, the value of p determines the degree of degradation in overlay delay measured from the root that C is willing to sacrifice if C switch from A to B. The larger the value of p, the larger the delay penalty. In our experiments, we set $\alpha = 0.9$ and $p = 0.2$, following [104].

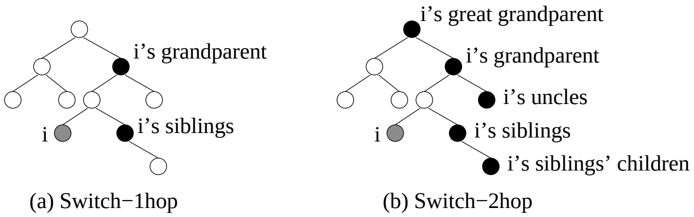


Figure 5.2: Switch-1hop and switch-2hop

5.1.1.4 Switch-trees Variants

In switch-trees [43], each node, i (except if i is the tree root) performs distance measurement to nodes within a predefined (local) search scope, and may switch to one of the node (say j) if j provides better performance than i's current parent (in terms of tree cost or delay). In [43], Helder et al. propose four variants of switch-trees algorithms: switch-sibling, switch-1hop, switch-2hop and switch-any (see Section 2.6.1), each with a different search scope. Of these algorithms, we consider only switch-1hop and switch-2hop. First, the switch-sibling's local scope differs from the switch-1hop's scope only by one node (see Figure 2.5), and it has been shown to have poorer performance [43]. On the other hand, switch-any considers all non-descendant nodes as targets, and thus is not practical.

Assume that the maximum number of neighbours (parent and children) of any nodes in the tree is given by Δ. Consider Figure 5.2, we can estimate the size of the search scope for switch-1hop and switch-2hop as follows.

- Switch-1hop: Δ, calculated as 1 (for i's grandparent) + $(\Delta - 1)$ (for i's siblings)
- Switch-2hop: $\Delta^2 + \Delta$, calculated as 1 (for i's great grandparent) + 1 (for i's grandparent) + $(\Delta - 1)$ (for i's uncles) + $(\Delta - 1)$ (for i's siblings) + $(\Delta - 1)\Delta$ (for i's siblings' children)

From the above, we can see that the size of switch-2hop's search scope could grow rather large. For instance, a Δ of 10 will result in a scope size of 110. First, for an end-host only ALM system, we would expect the members have differing degree constraints, which are typically small. More importantly, an implementation can limit the number of measurements based on the changes of nodes within the scope.

In addition to the two local scopes considered, we have implemented a version where a switching node considers a randomly selected non-descendent node as target. This can be achieved by using the tree random walk technique proposed by Francis et al. (see Section 2.6.1.1). We call this version *switch-random*. Hence, we can compare this with the efficiency of using localised switching.

An Extension An obvious extension to the above is to combine both localised and random approaches. To achieve this, we interleave both approaches during the course of the multicast session. Specifically, we interleave them probabilistically with an exponential distribution such that more local switching is done in the early stage, while more random switching is involved as the time proceeds. The intuition behind

```
$i :=$ The number of improvement rounds. Initially, $i = 0$
$p \leftarrow b^{-i/b}$
if uniform$(0, 1) < p$
  Select switching targets from local region
else
  Randomly selects a non-descendeant node as target
$i$ ++
```

Figure 5.3: Selection of local region or random node selection technique

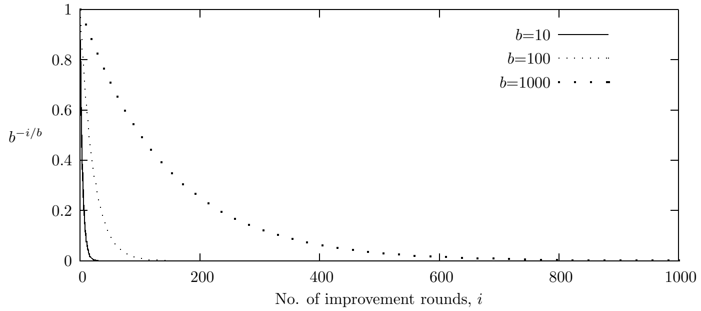


Figure 5.4: The distribution of function, $f(b) = b^{-i/b}$, for different values of b

doing so is to improve the overlay quickly using local transformation, and let the random searching to further explore the search space to yield better improvement.

When a node is carrying out the improvement process, it selects between the approaches (local-scoped selection and random selection) with a probability of $b^{-i/b}$, where b is the exponential base and i is the number of improvement rounds that the node has performed thus far (see Figure 5.3). Figure 5.4 depicts the distribution of the function for three values of b: 10, 100 and 1000. It is easy to see that the function favours local-scoped selection during the early stage, however, as the time proceeds, more random selection will be used. The curve for smaller b (e.g. 10) also indicates that the chance of using local-scoped selection diminishes very quickly. To avoid this, i is reset to its initial value when it reaches b.

Henceforth, we will refer to this mixed local and random version as LR. Our evaluation includes the 1hop and 2hop versions of switch-trees for this extension. Unless specified otherwise, we present results obtained with $b = 20$. The impact of the values of b will be studied in Section 5.2.1.1.

Joining Strategies As the switch-trees algorithms are only concerned with the overlay improvement process, we consider the following three simple joining strategies.

1. *Root-first.* This is similar to the approach taken in [43]. In this strategy, all newcomers first attach themselves to the tree root. Due to this, the root may quickly exceed its fan-out capability. To enforce the degree bound for the root, the root will force some of the nodes to switch to new parents in their first periodic improvement round. The offloading decision is based on the distances between the root and the nodes. In particular, the root will try to keep nodes that are close to it.

2. *Next-available*. In this variant, the DFS is used by newcomers to locate an unsaturated on-tree node as quickly as possible. Like HMTP, a newcomer begins the search from the tree root, and at each level of the search the nearest branch will be considered. Unlike HMTP, the newcomer will attach to the first feasible parent that it has found, rather than carry on the search for an optimal position. Hence, this approach somehow simulates the joining strategy in HostCast.

3. *Random*. In this variant, a newcomer attaches itself to a randomly selected on-tree node. This serves as the worst-case scenario in which the distance information about the existing overlay and members is not available.

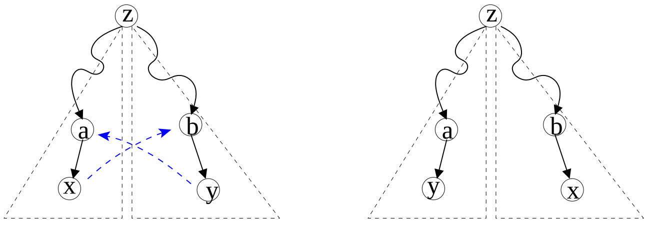
Each switch-trees algorithm implemented can be used to minimise the tree cost or root-diameter as described in Section 2.6.1. To overcome the triangle problem, we include a promotion operation (see Figure 2.4 (c)) in the protocol. In other words, our version of delay-based switch-2hop that uses the next-available joining strategy can be viewed as a variant of HostCast. In fact, as switch-2hop considers more switching targets than HostCast, we expect it to perform better than the original HostCast.

5.1.1.5 Banerjee et al.'s Scheme

The original scheme proposed by Banerjee et al. [9] is intended for a proxy-based system. In such a system, the end hosts attach to their respective nearest proxies, which self-organise into a multicast overlay. Their scheme is applied to build the proxies overlay, aiming at minimising the average latency observed by the end hosts. In our study, we focus on the technique uses to construct the multicast overlay, where all members actively participate in the overlay creation process. Due to this, a more suitable optimisation metric is the maximum latency from the root to all other members, i.e. root-diameter. This is also in line with the objective of other tree-based delay-optimised protocols, e.g. switch-trees and HostCast.

In the original Banerjee et al.'s scheme, newcomers are first centrally arranged into an overlay tree by the root proxy. While this centralised approach is suitable for more powerful proxy machines, it may not be feasible in an end-host only environment. For this reason, we consider two of the distributed joining strategies discussed above, i.e. next-available and random. We note that the root-first strategy is not used as it is in conflict with the transformation strategies used in their scheme. Specifically, most transformations require the knowledge of a grandparent, which is not available when all nodes are directly attached to the root.

In our implementation, each node maintains the delay from the root to itself as well as the maximum subtree delay (i.e. the delay from the node to its farthest descendant), as in the original scheme. Periodically, a node tries to perform a local transformation or random swapping that improves the delay from the root and does not increase the maximum subtree delay (see Section 2.6.1). A node performs the random swapping with a small probability, p. For local transformation, the node will choose of all the



(a) Before Swapping (b) After Swapping

Figure 5.5: Banerjee et al.'s scheme: an example of random swapping

potential transformations, the one that provides the largest delay gain. For random swapping, the node may still perform the swap even if it does not bring any delay gain. Figure 5.5 (a) shows an example where node x is trying to swap position with a randomly selected node, y, so as to achieve the configuration in Figure 5.5 (b). Node z is the least common ancestor of x and y, while a and b are the parent of x and y respectively. Let the increase of the maximum subtree delay of z, Δ, be as follows[1]:

$$\Delta = (D'_{z,x} - D_{z,x}) + (D'_{z,y} - D_{z,y}) \tag{5.1}$$

where $D'_{z,x}$ and $D'_{z,y}$ denote the delays from z to x and y respective along the overlay if the swapping is performed, and $D_{z,x}$ and $D_{z,y}$ denote the same prior to the swapping. Banerjee et al. use a simulated annealing based technique to decide probabilistically when to perform the swap operation. Specifically, the swap operation is performed: (i) with a probability of 1 if $\Delta < 0$; and (ii) with a probability of $e^{-\Delta/T}$ if $\Delta \geq 0$, where T is the "temperature" parameter of the simulated annealing technique. It is easy to see that the probability of the swap gets exponentially smaller with increase in Δ. On the other hand, increase in T increases the probability of the swap.

In [9], Banerjee et al. show that random swapping can offer an improvement to their solution (for the problem of minimising the average latency). We have conducted some experiments to study the impacts of random swapping for our current objective function — minimising the root-diameter. We ran Banerjee et al.'s scheme (using the next-available joining strategy) with different values of the probability of using random swapping, p (0.00 for no random swapping, 0.02, 0.05 and 0.10), and the temperature parameter, T (5, 10, 20, 100, 500, 1000, 2000, ..., 5000)[2]. The detailed simulation settings will be explained in Section 5.2.

We consider the delay performance in terms of RMP, which gives an indication of how well the proposal provides low root-diameter trees (see Section 3.2.4.1). In Figure 5.6, we show a representative result obtained with $p = 0.10$ and different values of T, as well as a version that does not use random swapping (i.e. $p = 0.0$). Results with other combinations of p and T are quite similar, and thus are omitted. We can see that for small group size (32 nodes), all versions with random swapping performs

[1] Equation 5.1 is adapted from the corresponding equation used in the original Banerjee et al.'s scheme [9].
[2] In [9], Banerjee et al. used p =0.02, 0.05 and 0.10; T = 5, 10 and 20.

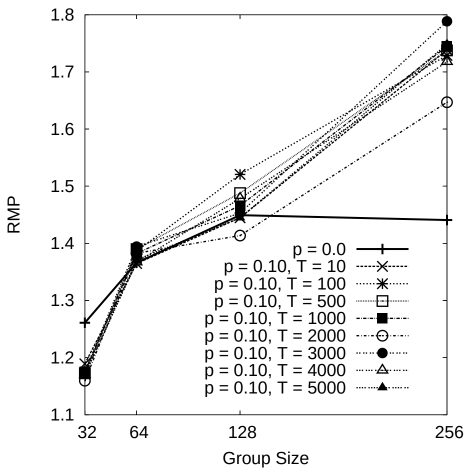


Figure 5.6: Banerjee et al.'s scheme: performance with and without random swapping

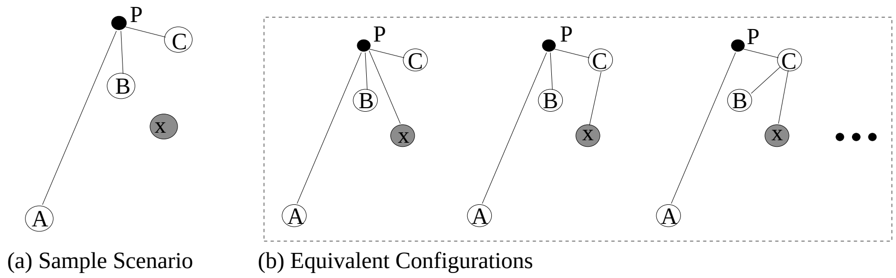
better than the version without it. The advantage of using random swapping diminishes as the group size increases. For the largest group size considered (256 nodes), the version without random swapping actually outperforms those with random swapping. Overall, the results show the probabilistic swapping gives poorer average performance in the experiments. Following this, we consider variants of Banerjee et al.'s scheme that use only local transformation.

Aside: Implementation of Switch-tree and Banerjee et al.'s scheme For both switch-trees and Banerjee et al.'s scheme, in order to capture the main properties of the schemes while avoiding complication in distributed tree maintenance (e.g. looping and partitioning problems), we use a flow-level approach for the transformation process. Specifically, when a node (say x) has successfully chosen a new parent (say y), the simulator directly reconfigures the connections between the nodes involved: x is detached from its current parent, and a link is established between x and y. We note that the detailed tree maintenance procedures are not given in [43] and [9].

5.1.1.6 TBCP

We recall that TBCP [62] uses a changeable score function to identify the best overlay configuration for nodes within a small region. In addition, a domain-based concept is used to organise nodes from the same domain under the same subtree.

In our implementation of TBCP, we have excluded the domain-based technique from TBCP's tree building procedure. This is because the main interest of this work is to understand and compare the efficiency of different overlay construction strategies employed by various proposals. TBCP's domain-based technique can easily be adapted for other proposals. In addition, the technique requires the tree root to keep track of all the domains that participate in the overlay. This may cause a scalability concern



(a) Sample Scenario (b) Equivalent Configurations

Figure 5.7: TBCP: dominant link, $\langle P, A\rangle$, in local configuration

about the protocol.

In [89], we propose two enhancements to the TBCP basic tree building procedure: (i) a tie-breaking rule; and (ii) a new score function.

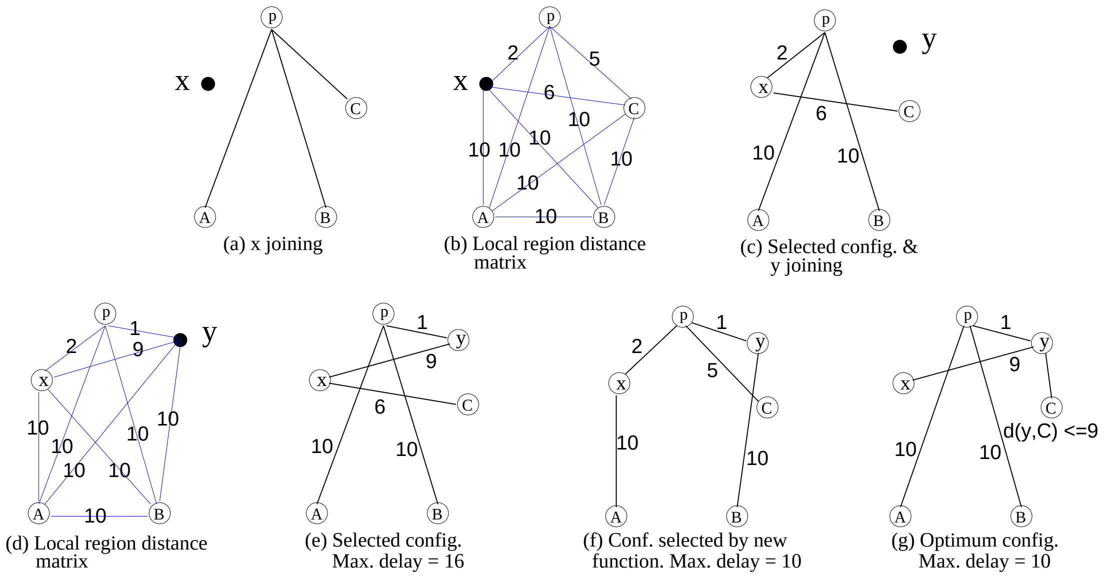
Tie-breaking Rule TBCP's original score function, Equation 2.5, is designed to achieve a low delay tree by organising nodes within the local region into a configuration that yields the smallest maximum overlay delay. It is easy to see that there may be more than one configuration that will provide the same minimum score value. This can happen when the overlay distance between two nodes dominates the other distances, as illustrated by the example in Figure 5.7. In the figure, the local region consists of node P, its children, nodes A, B and C, and node x, the newcomer. Let us assume that the minimum overlay distance of all potential configurations is equivalent to the distance between P and A. For example, the configurations in Figure 5.7 (b) are equivalent in terms of score value, and thus are all potential solutions. In such a case, TBCP will favour configurations resulting in the newcomer "moving", to provide stability for already joined nodes. Then, if there is more than one possible configuration, the tie is broken arbitrarily.

We propose a new tie-breaking rule based on the configuration cost, which for a configuration i is given by

$$\text{configuration cost}, C(i) = \sum_{\forall e \in E_i} d(e) \tag{5.2}$$

where E_i is the set of overlay links in configuration i and $d(e)$ is the delay value of overlay link e. This function can be easily computed by P. The configuration with the smallest cost will be chosen. Referring to Equation 2.1 (overlay tree cost $= \sum_{\forall e \in E_T} d(e)$ where E_T is the set of overlay links in the tree), we can see that Equation 5.2 is essentially a scaled-down version of the tree cost metric. In other words, our tie-breaking rule favours the configuration that consumes the least network resource.

However, there are other possible ways to break the tie. For example, one may choose to break the tie by comparing the distances between two configurations in a lexicographic order. In this case, we first sort the overlay distances from P to other nodes in a configuration in a non-increasing order. For two configurations with equal score value, we compare each of their next largest overlay distances.



(a) x joining

(b) Local region distance matrix

(c) Selected config. & y joining

(d) Local region distance matrix

(e) Selected config. Max. delay = 16

(f) Conf. selected by new function. Max. delay = 10

(g) Optimum config. Max. delay = 10

Figure 5.8: TBCP: sequence of tree construction steps

The configuration that gives the first smaller distance will be chosen. Even if this or other choices may performs better than our new tie-breaking rule, we argue below that, TBCP's original score function (Equation 2.5) is not suitable for tree-wide delay optimisation.

New Score Function The following discussion is based on the example given in Figure 5.8. In the figure, we show a sequence of hypothetical TBCP tree construction steps. First, panel (a) shows that a newcomer, x, is joining to the tree rooted at P. Let us assume that each node has a maximum fan-out of 3 and the local region distance matrix is as shown in panel (b). According to Equation 2.5, P will select the configuration with the smallest maximum overlay delay. It is easy to see that the best configuration is the one with a score of 10 as in panel (c). Panel (c) also shows that another node, y is joining the tree. Panel (d) depicts the new local region distance matrix. Note that node C is not included in the current local region as it is not a direct descendent of P. Now, the best score of 10 can be achieved with the configuration as depicted in panel (e). However, if we look at the tree as a whole, we can see that the maximum overlay delay is 16 (provided by the branch forms by nodes P, x, y and C), which is obviously not the best possible solution. The problem gets worse as the number of members increases.

The above observation suggests that a score function that greedily try to minimise the delay in a local region is not suitable for tree-wide delay optimisation. Obviously, the limitations of the score function could be overcome by using a larger scope. However, this increases the complexity and reduces the scalability of the original approach.

With the above observation in mind, we propose an alternative function which minimises the occurrence of *triangles* in the overlay, rather than focusing directly on the delay metric. As described in Section 2.2, triangles play a vital role in delay and cost optimisation. Our function prevents the formation

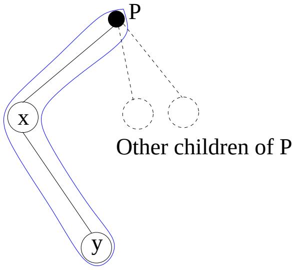


Figure 5.9: TBCP: the segment that forms a triangle in local configuration

of an ineffective triangle in a local region when a new node is introduced into the region. To explain the function, we first refer back to Figure 2.8. From the figure, we can observe that each of the potential configurations has at most one overlay branch that can form a triangle. Our function evaluates only the affected branch. Now, consider the configuration in Figure 5.9 where node P is the potential parent which forms a triangle with x and y. The score function for a configuration i can now be given as,

$$score(i) = \frac{d(P,x) + d(x,y)}{d(P,y)} \tag{5.3}$$

With this, the best configuration is the one that provides the smallest score. If there is a tie, the configuration cost will again be used.

Let return to the example in Figure 5.8. Now, consider the new function given in Equation 5.3. Initially, when node x joins the overlay, two configurations give the best score: configuration that consists of branch involves nodes P, x and A and configuration with branch involves nodes P, x and B. As both configurations provides similar score and cost (i.e. score $= \frac{10+2}{10} = 1.2$ and cost $= 12$), P will randomly choose one of them. Assume that the branch forms by P, x and A is chosen. Applying the same procedure for y results in configuration (f), which gives a maximum delay of 12. By taking the overlay as a whole, one can determine that an optimum solution has a maximum delay of 10 (e.g. the configuration in panel (g), assuming that $d(y,C) \leq 9$). However, this requires the knowledge of all members, which is not feasible as nodes may join at different times. Moreover, comparing the configuration in panel (f) and (g), we can see that our solution results in lower tree cost (i.e. 28 versus 30 + $d(y,C)$).

Figure 5.10 plots the comparison of delay in terms of RMP and tree cost ratio (see Section 3.2.4.1) for various group sizes, for the following three variants of TBCP (the detailed simulation settings will be explained in Section 5.2):

- TBCP (Original). This version uses the original score function and breaks the tie arbitrarily.
- TBCP (Original + Tie-break). In this version, on top of the original score function, we use the configuration cost to break the tie.
- TBCP (New Function + Tie-break). This version uses the proposed new score function (Equation 5.3), and uses the configuration cost to break the tie.

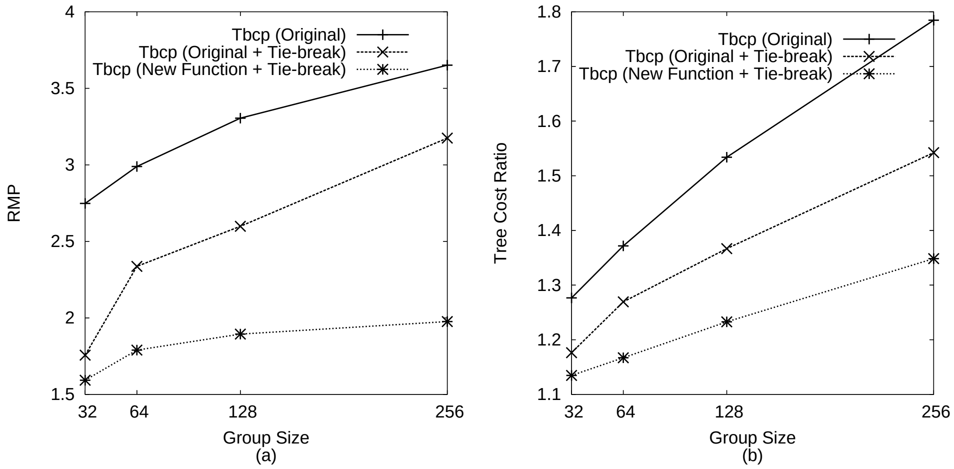


Figure 5.10: TBCP: comparing the variants

From the result, it is clear that our new score function provides a substantial improvement in both metrics. Interestingly, a noticeable gain can be observed by a simple modification to the tie-breaking rule. More analysis regarding the performance trends under various group sizes will be given in Section 5.2.

Henceforth, we will use TbcpD to refer to the above-mentioned TBCP (New Function + Tie-break). We also use TbcpC to represent a version of TBCP that takes the configuration cost (Equation 5.3) as the score function. This is to investigate if the function is suitable for tree cost minimisation. The suffixes "D" and "C" indicate the optimisation objective (delay or tree cost) of the protocol.

5.1.1.7 Scribe

Scribe constructs multicast trees on top of overlays built with Pastry [81], an efficient peer-to-peer routing and object location protocol (see Section 2.6.2). Pastry's overlays have been shown to exhibit good locality properties, i.e. a message can be routed from one node to another node with small relative delay penalty, and Scribe has utilised these properties to connect nodes that are topologically close together into a tree.

Instead of implementing the whole Pastry-Scribe protocol suite, our Scribe implementation only tries to capture the properties of the resultant overlay. We build Scribe trees in the following manner. When a newcomer, say x, joins a session, we attach x to the nearest on-tree node, say y. Essentially, this simulates an ideal Pastry routing. Then, if y finds that adding x violates its degree bound, y will execute the "bottleneck remover algorithm" as suggested in Scribe. Specifically, y drops its farthest child, which will be redirected to one of y's remaining children. Once attached to the tree, a node periodically rejoins the tree with the above procedures to improve its on-tree position. This is to simulate the self-organising capability of the protocol.

5.1.2 Naming of the Protocols

For ease of exposition, we classify the protocols into two classes: cost-optimised protocols (CoPs) and delay-optimised protocols (DoPs), as shown in Table 5.1.

In the table, we name the variants of switch-trees and Banerjee et al.'s scheme using the following convention: <join-strategy><transformation-type><optimisation-metric> where join-strategy refers to one of the *Root*, *Na* (next-available) and *Random*; the transformation-type refers to one of the *1Hop*, *2Hop*, *Random*, *LR* (mixed local and random node selection) and *Banerjee*; finally, the optimisation-metric is either *D* (delay) or *C* (cost). As Banerjee et al.'s scheme only has the delay version, the suffix "D" is omitted. For example, Root1HopC refers to the cost-optimised version of switch-trees that uses the root-first joining strategy and one-hop switching, while NaBanerjee refers to a version of Banerjee et al.'s scheme that uses the next-available joining strategy. Scribe is regarded as a CoP as its overlay trees are formed by placing close-by nodes together. We note that NICE is included in both categories, as its overlays provide a compromise between these two metrics, as will be explained in the next section.

The table also includes several centralised algorithms: dbMST (degree-bounded Minimum Spanning Tree), CPT (Compact Tree) and GreedyMesh (see Chapter 4). These algorithms help us to understand how well the distributed approaches perform in comparison with techniques that utilise global knowledge. We note that two versions of CPT were used: (i) a version that minimise the tree diameter for many-to-many multicasting; and (ii) a version that minimise the root-diameter for one-to-many multicasting.

Cost-optimised Protocols (CoPs)	**Delay-optimised Protocols (DoPs)**
1. Variants of switch-trees:	1. Variants of switch-trees:
{Root,Na,Random}{1Hop,2Hop,Random,LR}C	{Root,Na,Random}{1Hop,2Hop,Random,LR}D
2. HMTP	2. Banerjee et al.'s scheme: {Na,Random}{Banerjee}
3. TbcpC	3. TbcpD
4. AOM	4. Narada
5. Scribe	5. CPT
6. dbMST	6. GreedyMesh
7. NICE	

Table 5.1: Protocols considered and their naming

5.2 Results and Analysis

This section reports the results of our performance evaluations, which serve two main purposes: (i) to compare the quality of the overlays built; and (ii) to understand the properties of the different techniques.

The experiments were run with nine 1000-node networks, generated by the Transit-stub, Waxman and power-law models as described in Section 3.2.2. (A subset of these experiments was also run on some 2000- and 10000-node topologies [90, 92].) The multicast members are selected randomly from the network nodes, as discussed in Section 3.2.3. Unless specified otherwise, we use the following

Parameter	**Settings**
Topology	Transit-stub, Waxman and Power-law graphs with 1000 nodes.
Group size	32, 64, 128 and 256.
Simulation duration	Multicast members join the session within the first 50 seconds, then the simulation stops when the overlay has stabilised.
Max. fan-out (per node)	2 – 10, drawn from either a uniform or a truncated binomial distribution. For NICE, $k = 3$.
Overlay improvement period (per node)	30 seconds.
Number of runs per scenario	50.
AOM specific parameters	$\alpha = 0.9$, $p = 0.2$.
LR version of switch-trees	Exponent base, $b = 20$

Table 5.2: Settings used in the performance comparison

configurations. First, the group sizes range from 32 to 256. In most cases, the members randomly join in the session one by one within the first 50 seconds of the simulation, and we run the simulation for 2400 seconds, sufficient for the resultant overlay to stabilise. Each member is assigned a maximum fan-out of 2 to 10, drawn from a uniform or truncated binomial distribution. For NICE, we set $k = 3$ as described previously. Typically, for each simulation configuration, we run 50 independent experiments and report the average. Table 5.2 summarises the settings used.

Our experiments can be divided into two, according to the multicast service models considered:

1. *One-to-many*. In this case, one member is selected as the data source while the other members act as receivers. In each simulation, the first joining member serves as the data source as well as the root of the data delivery tree. Narada is omitted from the experiments as it is designed specifically to create multiple trees for multi-source model.

2. *Many-to-many*. In this case, we treat all members as both sender and receiver. For protocols that create a single tree for data delivery (i.e HMTP, switch-trees, AOM, TBCP, Banerjee et al.'s scheme, Scribe), the first member will become the tree root. To be fair to these protocols, CPT uses the same node as tree root. The resultant tree will be used as a bidirectional shared tree. On the other hand, NICE and Narada use source-specific trees for multicasting.

We report results obtained from a transit-stub network (i.e. TS1k-0 as in Table 3.1), and state the main differences observed across the different topologies. A more detailed investigation on the impacts of the underlying topologies is given in Section 5.2.4.

The rest of this section is organised as follows. The next two sections discuss the results for the one-to-many and many-to-many model respectively. In Section 5.2.3 and 5.2.4, we examine the impacts of the overlay nodes' degrees and the underlying topologies. Section 5.2.5 and 5.2.6 study the convergence speed and protocol overhead of some selected distributed proposals. We then offer a summary and discussion in Section 5.2.7.

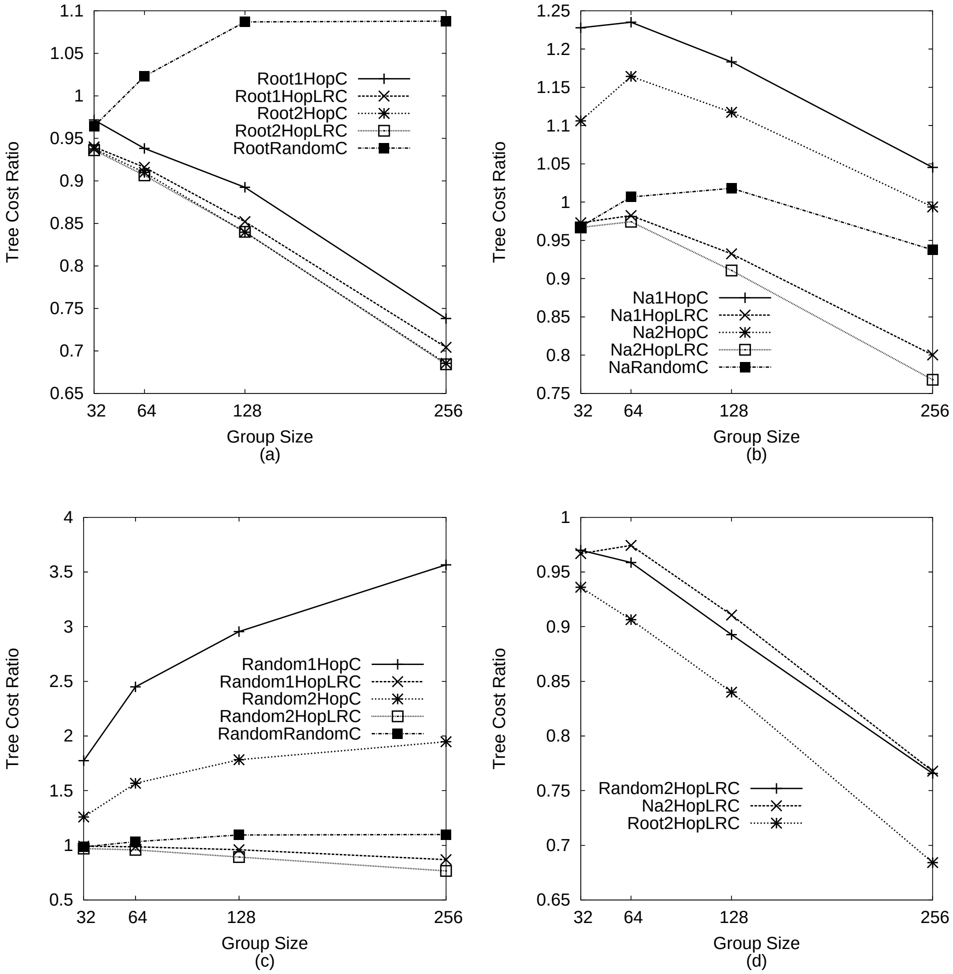


Figure 5.11: Cost-optimised switch-trees: TCR for various joining strategies. (a) Root-first, (b) Next-available, (c) Random and (d) Best proposals

5.2.1 One-to-many Data Delivery

We divide the discussion into cost- and delay-optimised protocols, according to the optimisation metrics under consideration.

5.2.1.1 Cost-optimised Protocols (CoPs)

We begin by looking at the performance of the optimisation metric, i.e. the tree cost, followed by link stress and delay (represented by RAP and RMP).

Tree Cost Ratio (TCR) for Switch-trees Variants As switch-trees consists of a number of alternatives, we discuss its variants first, followed by other proposals. Figures 5.11 (a) to (c) plot TCR results for variants of switch-trees grouped by the three joining strategies considered.

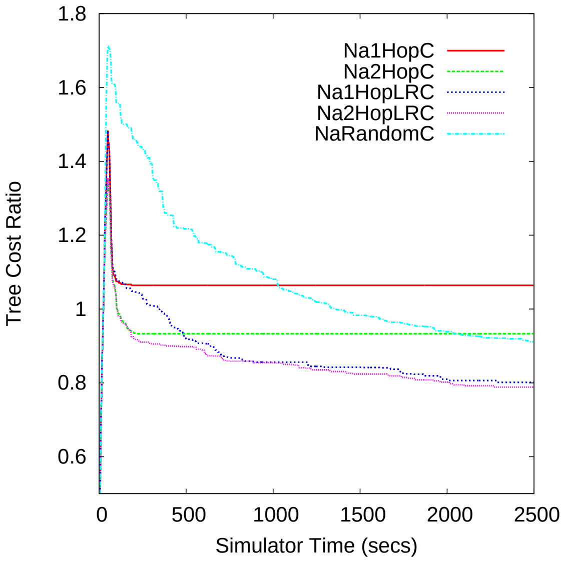


Figure 5.12: Cost-optimised switch-trees: evolution of TCR with next-available joining strategy

For the root-first strategy, we can see that the TCR values for all variants, except RootRandom, decrease with the group size. Within this group, the 2Hop and 2HopLR variants perform the best, followed closely by 1HopLR. The random version performs the worst. This confirms the observation by Helder et al. [43] that informed parent choice is required to achieve low cost trees.

Next, we turn to the next-available and the random strategies, which show similar relative performance. We can see that the LR variants perform considerably better than other schemes. More interestingly, the switch-random version now out-performs the two local scope-only versions, which is in contrast with the observation above. To explain this, we plot the evolution of TCR for the next-available variants in an experiment with 256 members in Figure 5.12. From the figure, we can see that the two local scope-only approaches (i.e. 1Hop and 2Hop) converge very quickly, while the random approach takes a much longer time to settle down. This is because the random version allows a node to slowly explore all the potential switching targets, and thus improve the tree gradually. This also highlights the trade-off in the local scope approach: fast convergence time at the expense of limited search space exploration. If it is used together with the random joining strategy, its lack of exploration power will lead to large TCR values as shown by Random1HopC and Random2HopC, in Figure 5.11 (c). Figure 5.12 also shows that the LR versions — mixed local and random scopes — indeed take advantage of the two extremes.

Observing the trends from Figures 5.11 (a) to (c), it is clear that the larger the switching scope, the better the performance, as expected. Figure 5.11 (d) depicts the best schemes (all 2HopLR-based) selected from each joining strategy considered. From the figure, it is clear that the root-first strategy

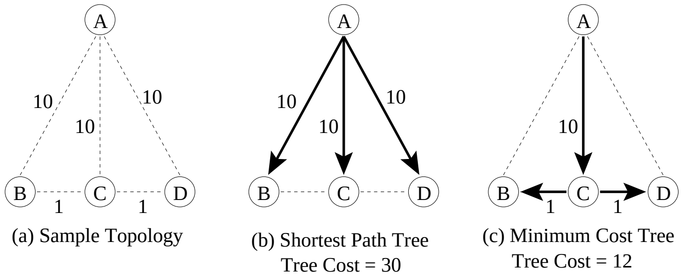


Figure 5.13: Comparing shortest path tree with minimum cost tree

performs the best, followed by next-available and the random approaches. We recall that in the root-first strategy, all nodes are first attached to the root. Hence, initially, a switching node will have many siblings (and other local region nodes), and has more chances of finding the optimum placement. The disadvantage of this approach is that a node may need to sustain a higher measurement overhead at the initial stage. This reduces the practicality of the solution. On the other hand, the next-available strategy always enforces the degree constraints. Hence, it results in a smaller number of local region nodes. However, as it uses DFS to attach a newcomer to the nearest unsaturated on-tree node that the newcomer first encounters, the resultant tree is partially clustered. This provides a reasonably good starting point for further optimisation, in comparison with the random strategy.

Our results generally agree with the observations made by Helder et al. [43]. Specifically, simple switch-trees algorithms such as 1hop and 2hop switching can produce reasonably low cost trees. These two localised techniques provide informed parent choice, hence performs better than the random approach in most cases. We extend their work by considering several different joining strategies — their evaluation only considers the root-first joining strategy. More importantly, we show that a mixed local and random approach (i.e. LR) can offer improvement to pure local switching.

In general, it is interesting to note that the TCR values for some schemes fall below one[3]. We first recall that TCR is the ratio between the cost of the overlay and the network layer multicast trees, where the network layer multicast tree is the router-level shortest path tree. Figure 5.13 illustrates a simple scenario where a shortest path tree can have a much higher tree cost than a minimum cost tree (both rooted at A). Thus, we believe the case that TCR < 1 is due to the structure of the underlying topology. In particular, we have found that it happens in experiments using topologies with large average node degrees. For example, the results shown here are obtained from TS1k-0 which has an average node degree of 14.46 (Table 3.1). A similar trend was also observed under the power-law topologies. With more large degree nodes, the shortest path tree has the tendency to use them as a hub to connect other nodes, as depicted by Figure 5.13 (b). This results in high tree cost. On the other hand, as the overlay tree is built to minimise the cost, mostly short links will be included in the tree. Therefore, even with

[3] Note that the case for TCR > 1 is expected as an ALM tree generally uses more physical links than a network layer multicast tree.

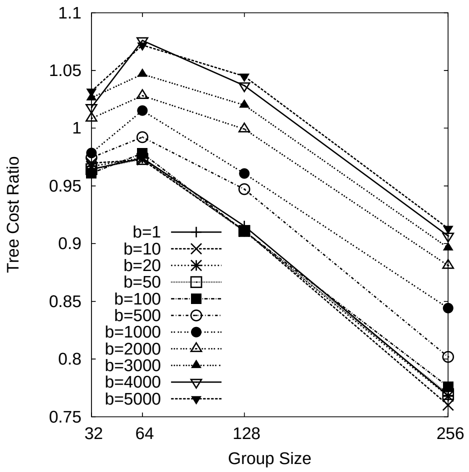


Figure 5.14: Na2HopLRC: impacts of the value of b

some redundancy, the overlay tree may achieve a lower tree cost.

Aside: Impacts of Exponential Base, b, for the LR Variants Here we take an aside to examine the impacts of exponential base, b, for switch-trees variants that use the LR (mixed local and random) node selection strategy. We show the representative results obtained with Na2HopLRC. Figure 5.14 depicts the TCR performance for Na2HopLRC run with values of b that range from 1 to 5000. From the figure, we can see that small values of b (i.e. 1, 10, 20, 100) give quite similar performance. After that, the larger the b value (i.e. from 500 to 5000), the worse the TCR gets. As discussed in Section 5.1.1.4, large b favours the selection of local region nodes. With larger b values, there is a high probability of using local search. Thus, the strategy behaves like a pure local search technique. On the other hand, using a small value of b with a periodic reset results in a good balance of local and random node selection, and thus benefits from both approaches. We have observed a similar trend for the delay-optimised version of switch-trees' LR variants. In later chapters (6, 7 and 8), we use the LR technique in all our own proposals for choosing potential overlay neighbours.

TCR for Switch-trees and other CoPs We are now in a position to discuss switch-trees along with the other CoPs shown in Table 5.1. As discussed above, the root-first strategy may not be feasible, and we thus use the Na2HopLRC which has the second best result, to represent cost-optimised switch-trees. Figures 5.15 (a) to (d) depicts the comparison results in terms of TCR, maximum link stress, RMP and RAP.

From the TCR performance, we can roughly group the proposals into two classes: dbMST, Scribe, HMTP and Na2HopLRC which produce relatively low cost trees, and NICE, AOM and TbcpC which give higher cost trees. In the former class, the centralised dbMST performs the best, as expected. It is followed closely by HMTP, Na2HopLRC and Scribe. This proves that low cost trees can be achieved

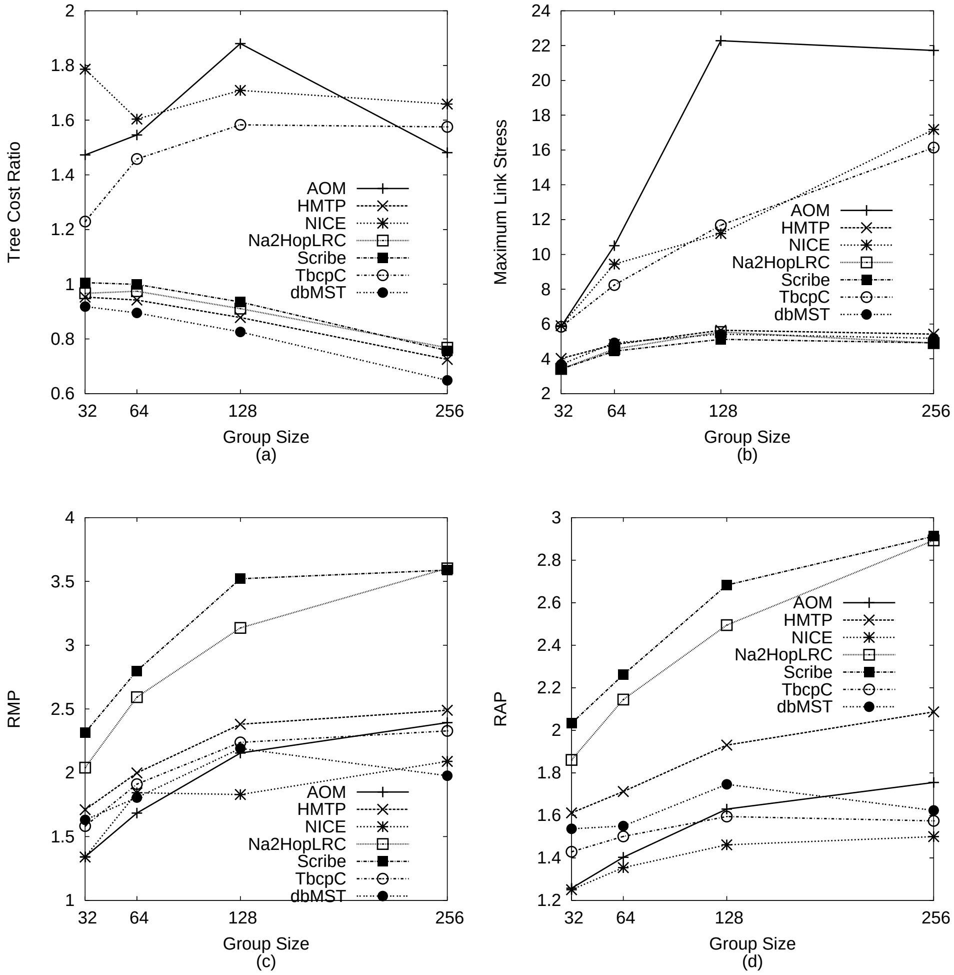


Figure 5.15: CoPs comparison results: (a) Tree cost ratio, (b) Maximum link stress, (c) RMP and (d) RAP

with relatively simple distributed solutions — the switch-parent operation of switch-trees and HMTP with a suitable node selection strategy. The low TCR values observed for this class suggests that the techniques can exploit the locality effect introduced when more members are added into a network. We have found that under some topologies, Scribe performs slightly better than HMTP. We believe that this is because our version of Scribe has better knowledge of the underlying topology. One potential drawback of HMTP is that its search scope is unconstrained. Specifically, as the search is greedy, a node will continue to look for a better parent as long as there are potential targets. In the experiments with 256 nodes, we have found that the maximum number of contacts that a node can make before settling down is about 30. Other proposals typically incur smaller number of contacts. This also partly explains why HMTP performs better than other protocols.

While not shown in the figure, we also examine the impacts of the triangle problem in tree cost optimisation, using HMTP. Our investigation shows that if the triangle optimisation (Section 2.6.1) is disabled for HMTP, the average TCR over all group sizes increases from 0.87 to 0.94. More importantly, the average RMP and RAP values increase from 2.15 and 1.84 to 3.01 and 2.40, respectively.

We now consider the other class. For TbcpC, the TCR values increase with the group size. This suggests that the chosen score function, i.e. Equation 5.2, is not suitable for cost optimisation. For AOM, we first recall that an AOM node can perform a switch only if the conditions given in Equation 2.2 to 2.4, weighted by α and p are fulfilled. The results were obtained using the best setting suggested in [104], where $\alpha = 0.9$ and $p = 0.2$. We also experimented with several other combinations of the parameters. We have found that the optimal setting of α and p is very sensitive to the topology used, i.e. a particular setting may permit more switching in some topologies, and thus performs better; however, the same setting may suffer in other topologies. This is an undesirable property as the best model for the Internet is still an open question. NICE also exhibits similar performance to these two protocols. While the TCR for these three protocols may look higher than those of the former class, they are however smaller than the delay-optimised techniques as shown in Figure 5.17.

Link Stress Link stress represents the redundant traffic injected into the network. Figure 5.15 (b) illustrates the result for the maximum link stress, i.e. the worst-case load on a single link. We can observe that the performance trend roughly follows those of the TCR. In other words, techniques that produce low cost trees also have lower traffic redundancy, as observed previously in Section 4.3.2. The average link stress, calculated as the ratio between the total stress and the number of physical links involved, exhibits a similar trend to TCR and the maximum stress, is therefore omitted.

RMP and RAP Figures 5.15 (c) and (d) depict the delay performance in terms of RMP and RAP, respectively. Overall, we make the following observations. In most cases, NICE has the best delay performance, while Scribe and Na2HopLRC perform the worst. AOM always produces lower delay trees than HMTP; this is as expected as AOM is designed to improve the delay performance of HMTP [104], as explained in Section 2.6.1.2. The relative performance among other proposals is not consistent across the topologies tested. This, we believe is due to the fact that these proposals are designed to build low cost trees — the delay of the trees is a by-product that depends on the structure of trees built, which in turn depends on the placement of the members and the structure of the underlying topology.

To be fair to Scribe, the poor delay property shown here is because, in our version, new nodes are attached to the nearest on-tree nodes (by exploiting the underlying topology knowledge). While this results in a low cost structure, depending on the underlying topology, some nodes may be connected in a long series, which results in high delay. However, we believe that our version approximates the properties of Scribe under an ideal Pastry routing.

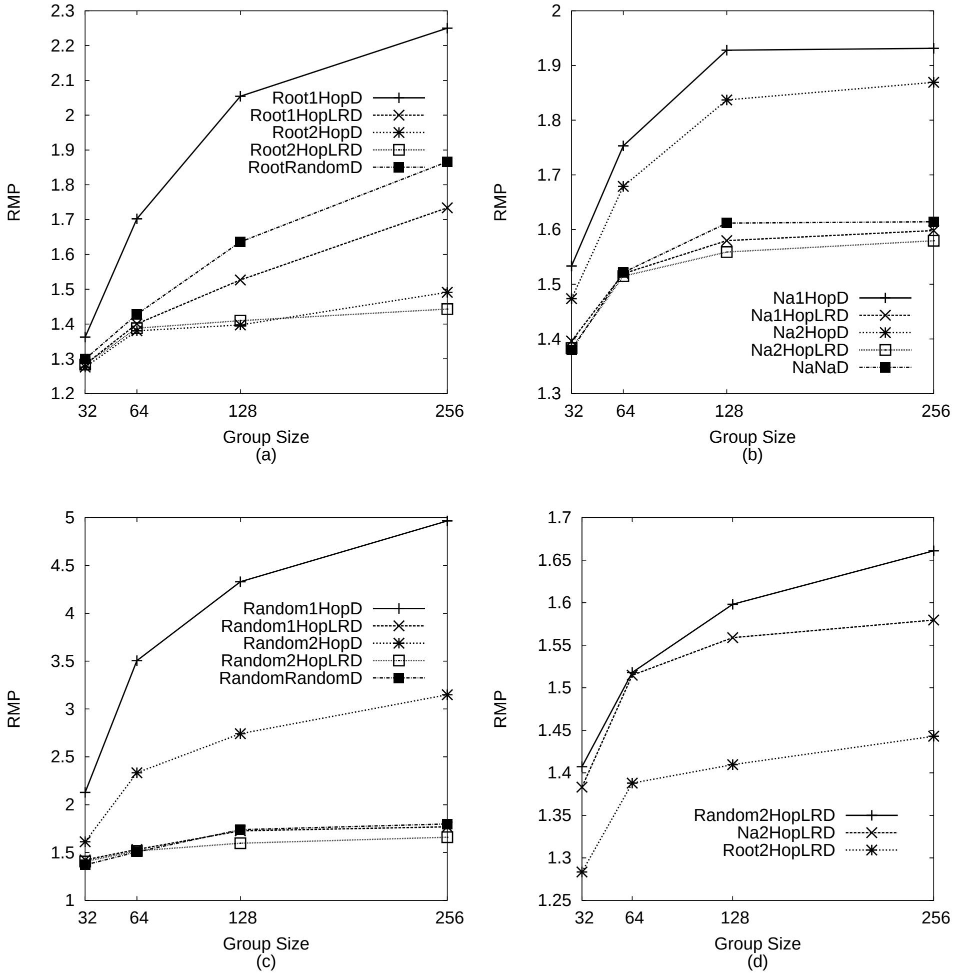


Figure 5.16: Delay-optimised switch-trees: RMP for various joining strategies. (a) Root-first, (b) Next-available, (c) Random, (d) Best proposals

5.2.1.2 Delay-optimised Protocols (DoPs)

We begin by looking at the performance of the optimisation metric, i.e. delay in terms of RMP and RAP, follows by tree cost ratio and link stress.

RMP and RAP We first consider the delay-optimised variants of switch-trees. Figure 5.16 (a) to (d) depict their RMP performance. (The RAP shows a similar trend, and thus is omitted.) If we compare the results presented here with Figures 5.11 (a) to (d) (i.e. TCR of the cost-optimised switch-trees variants), we can observe that the relative comparison of the variants follows a similar trend. This shows the observations that we made on the impact of the joining strategies and switching scopes still apply for the delay-optimised variants. Following this, we use Na2HopLRD as the delay-optimised switch-trees representative.

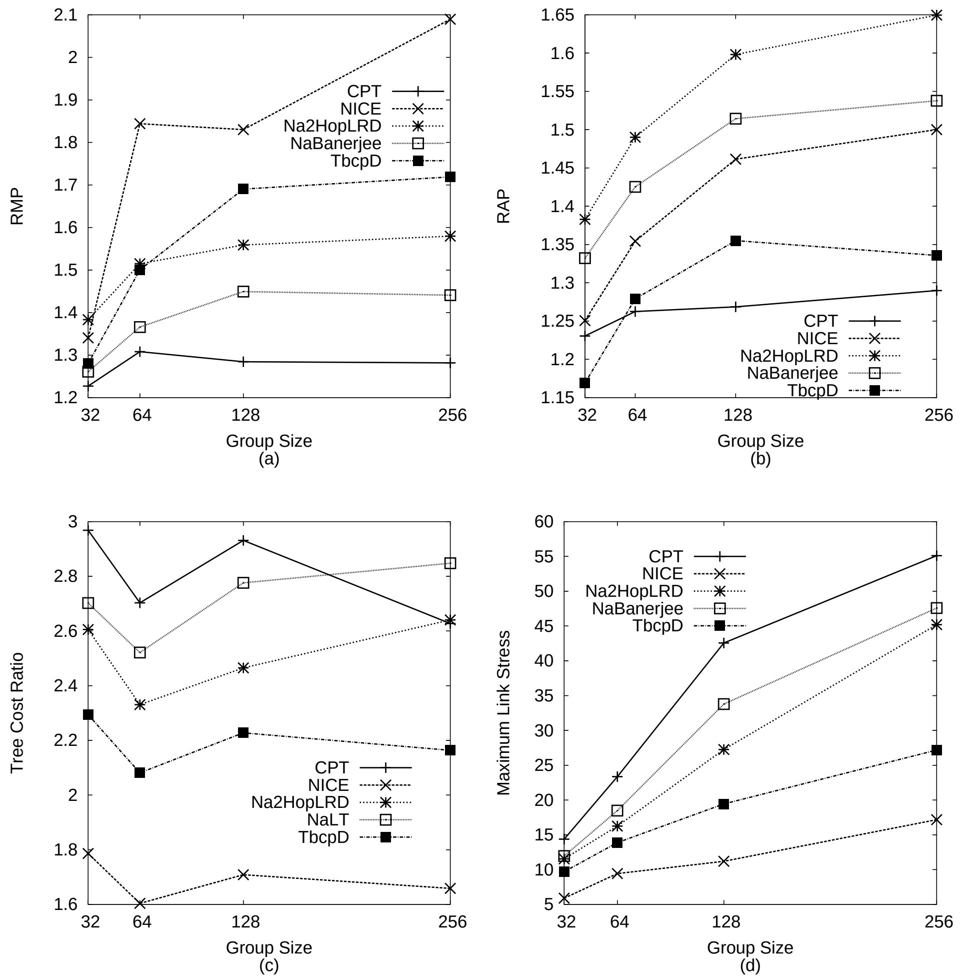


Figure 5.17: DoPs comparison results: (a) RMP, (b) RAP, (c) Tree cost ratio and (d) Maximum link stress

Figures 5.17 (a) and (b) illustrate the RMP and RAP results for Na2HopLRD along with other proposals. For Banerjee et al.'s scheme, we have found that RandomBanerjee performs worse than NaBanerjee, as expected. This is also consistent with the results for the impacts of joining strategy observed for switch-trees variants. We thus exclude it from the results.

In terms of RMP, unsurprisingly, the centralised CPT out-performs the distributed techniques. Within the distributed protocols, NaBanerjee performs the best, followed by Na2HopLRD, TbcpD and finally, NICE. Comparing the performance of NaBanerjee with Na2HopLRD (which uses a larger local scope), we can conclude that a more flexible transformation scheme and additional information (i.e. subtree delay) help in delay optimisation. For TbcpD, we can see that it always provides a better RAP than other distributed proposals. For a small group size (i.e. 32), it actually out-performs CPT (note that CPT is used to create low root-diameter trees). We attribute this to our triangle-based score function (Section 5.1.1.6) which minimises the ineffective triangles in the tree. It is also worth pointing out that

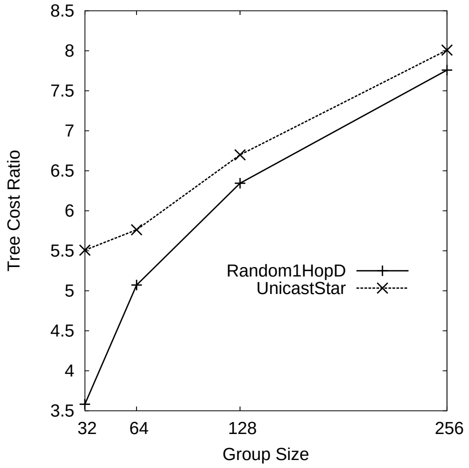


Figure 5.18: TCR for Random1HopD and unicast star overlay

it uses a smaller transformation scope than other distributed proposals. On the other hand, the triangle-based score function could not provide good RMP. We believe it is a challenging task to provide a score function for TBCP that achieves good RAP and RMP, using only the small scope size. While NICE shows rather poor RMP performance, its RAP (average delay) is better than NaBanerjee and Na2HopLRD. This will be discussed shortly in Section 5.2.1.3.

Tree Cost Ratio and Link Stress Figures 5.17 (c) and (d) illustrate the TCR and maximum link stress performance of the DoPs, respectively. Compared with the performance of RMP, we can observe an inverse relationship between the delay and both cost and stress. This can be explained by the tree structure built. The objective function of DoPs is to minimise the delay from the root to the receivers. To achieve this, all receivers need to be placed as close as possible to the root. If there is no degree constraint, all receivers can be directly attached to the root, which results in a star topology. Under a degree-constrained environment, we can envisage that the nodes will try to fill in each level of the tree, which results in a highly compact structure. Due to this, some long links may be introduced into the tree, and this results in higher tree cost and link stress. A more detailed discussion on the tendency to create a compact tree structure with delay-optimised proposals is given in Section 4.3.1, in the context of CPT.

Figure 5.18 shows a property of the TCR for Random1HopD. The figure compares Random1HopD's TCR values with those obtained from a star overlay (UnicastStar). The UnicastStar delivers the data directly from the sender to each receiver. Thus, it provides the best delay performance at the expense of straining the sending node. In addition, its worst-case stress can be as high as the group size. This indeed justifies the case for more intelligent ALM protocols. Now, it is interesting to see that the degree-constrained trees created by Random1HopD have comparable TCR values to those of UnicastStar. We believe there are two main reasons for this. First, delay-based switching is greedy, i.e. when a node has found a parent that provides low delay, it will greedily stick to it. Secondly, the random joining strategy results in poor initial tree layout. For example, nodes that are farther away from the root may directly attach to the root. As this provides the best delay performance, these nodes will occupy the free slots at the root, which prevents a better configuration from occurring. This suggests that the greedy problem needs special attention in designing a DoP.

We point out that the stress values observed in experiments with the transit-stub topologies are much higher compared with the corresponding cases in the power-law and random topologies. In a transit-stub network, each stub domain is attached to a transit domain via a limited number of stub-transit links, and traffic from one stub domain to another stub domain must go through these links. As a delay-based overlay often has a compact structure, many direct overlay links between members in different stub domain are likely to be used. This results in high traffic concentration (i.e. stress) at the stub-transit links. On the other hand, flat topologies (e.g. random and power-law) allow traffic to be distributed more evenly to all links. In our simulation, the transit-stub topologies connect each stub domain to a transit domain with only one physical link. This is to simplify the calculation of routing table used by `ALMSim` for packets routing. The implication of this is that interdomain traffic can only use those limited stub-transit links, hence producing a high stress value.

5.2.1.3 Discussion

This section discusses some of the main points from the above findings. The discussion is also applicable to the analysis in the following sections, particularly evaluation of the many-to-many model.

For tree cost optimisation, we have seen that a simple switch-parent operation and a suitable parent selection technique can yield reasonably low cost trees. For example HMTP, that uses DFS to locate a potential parent, consistently give results that are close to those for centralised dbMST. Our version of a mixed local region and random node selection technique (i.e. LR) provides another promising strategy for a tree structure. Our investigations also show that by solving the triangle problem, we not only improve the tree cost, but also reduce the delay.

In terms of delay optimisation, the results show that Banerjee et al.'s scheme can produce trees with the smallest maximum delay from the tree root to the other nodes. However, as low delay trees tend to be more compact in structure, this results in high tree cost (i.e. resource usage) and link stress. As the compact structure is built in a distributed way using limited topology knowledge, nodes that are topologically close may be placed farther apart. Thus, the average delay performance may suffer. For example, this can be seen from the better RAP (i.e. average delay) performance of our version of TBCP, which tries to minimise the ineffective triangles between the nodes.

If we compare the best CoPs (HMTP, Na2HopLRC and Scribe) with the best DoPs (NaBanerjee), we can see a trade-off between delay and tree cost (as well as link stress). In other words, minimising tree cost results in high end-to-end delay; minimising delay results in high tree cost and link stress. Between

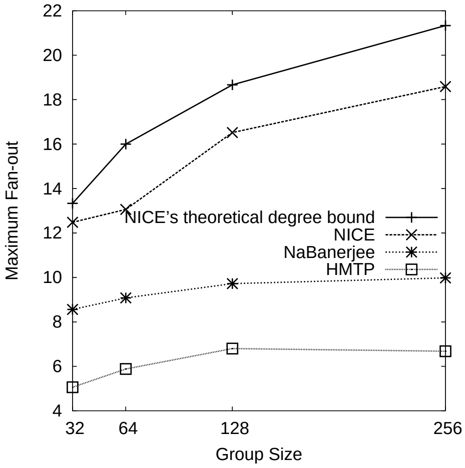


Figure 5.19: Maximum fan-out for HMTP, NaBanerjee and NICE (implementation and theory)

these two extremes, there is a wide spectrum of other trees that provide varying cost and delay values.

NICE, on the other hand, shows a compromise performance between the two extremes. Specifically, it produces trees that have moderate tree cost, stress and delay, compared to the best CoPs and DoPs. This can be explained by the structure of its overlays. In NICE, the members are organised into a multi-level clustered overlay. Each cluster is bounded in size and is represented by a leader, which is the graph-theoretic centre of the cluster. In other words, the leader has the smallest maximum distance to all the members in the cluster. The lowest level clusters consist of all members. The leaders of lowest level clusters form the second level clusters. This clustering process continues until the top-most level, which consists of only one node. In our simulation, the top-most node acts as the data source. Due to the hierarchical arrangement, nearby nodes will be contained within a cluster. This reduces the tree cost and link stress. However, the inter-cluster communication is done via longer links between the cluster leaders, which increases the tree cost and stress. The use of short and long links also results in a moderate delay performance.

A potential drawback of the NICE structure is that the top-most node may have a fan-out of $K \log_K(n))$, where K is the maximum cluster size and n is the group size. Figure 5.19 depicts the fan-out variation for NICE along with HMTP, NaBanerjee and the function $K \log_K(n)$, where $K = 8$ ($K = 3k - 1$, $k = 3$). The plot shows that NICE's theoretical worst-case fan-out roughly follows the function $K \log_K(n)$ while HMTP and NaBanerjee fulfil the fan-out constraint used, i.e. 10. The fact that a good delay-optimised proposal such as NaBanerjee shows larger fan-out values than HMTP (cost-optimised) also proves that the delay-optimised solution generates more compact trees.

The above observation suggests that NICE is unsuitable for high-bandwidth applications. In [8], Banerjee et al. suggest a delegation-based data forwarding scheme to reduce the worst-case fan-out of NICE. The scheme works as follows: a cluster leader sends data to its lowest layer cluster members,

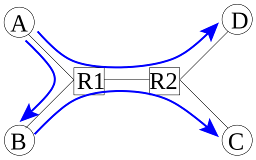


Figure 5.20: Link stress due to uncorrelated overlay links

which will in turn forward the data to the upper layers. This approach limits the maximum fan-out of any node to $3k$, i.e. roughly the cluster size. However, this brings additional complexity to the protocol, as the delegations need to be carefully selected to ensure efficiency. More importantly, it does not solve the problem when each member may have a different fan-out capability. It remains unclear how to create trees that honour the degree bound of each individual node while preserving the original properties and integrity of NICE.

The results in previous sections show that the maximum link stress due to the proposals can be rather high, especially for the delay-optimised proposals. We have pointed out in the previous section that this is partly because the results were obtained from a transit-stub topology, where the connectivity between the stub and transit domains is limited by a handful of stub-transit links. In addition, as discussed by Chu et al. [21], the maximum stress on the Internet may be lower than seen in the simulations. This is due to the fact that the ratio of the group size to topology size (i.e. density) is much higher in simulations than in actual practice. For example, the results were obtained with topologies of 1000 nodes, which are orders of magnitude smaller than the Internet. An increase in group density increases the probability that a physical link could be shared by multiple uncorrelated overlay links. Consider the example in Figure 5.20, uncorrelated overlay links $\langle A, D\rangle$ and $\langle B, C\rangle$ share the physical link $\langle R1, R2\rangle$. This could increase the maximum stress with the ALM proposals in our study, as they are only able to regulate fan-out of the members and not stress of the physical links.

5.2.2 Many-to-many Data Delivery

In the experiments with many-to-many model, we include Narada (see Section 2.6.2) and the GreedyMesh algorithm (see Chapter 4). Most observations are consistent with the one-to-many case. Thus, we only discuss the following representatives: HMTP, Na2HopLRD, TbcpD, NaBanerjee, NICE, Narada and CPT. Note that, except for NICE and Narada which use source-specific trees, all the other proposals use a single shared tree for data delivery.

For shared-tree based DoPs, the appropriate objective is to minimise the tree diameter, i.e. the maximum delay between any two members via the tree. The centralised CPT uses this objective function. However, it is a non-trivial task to measure and maintain this information in an environment where members join and leave freely. Hence, our implementations of Na2HopLRD and NaBanerjee try to minimise the root-diameter, as in the one-to-many case.

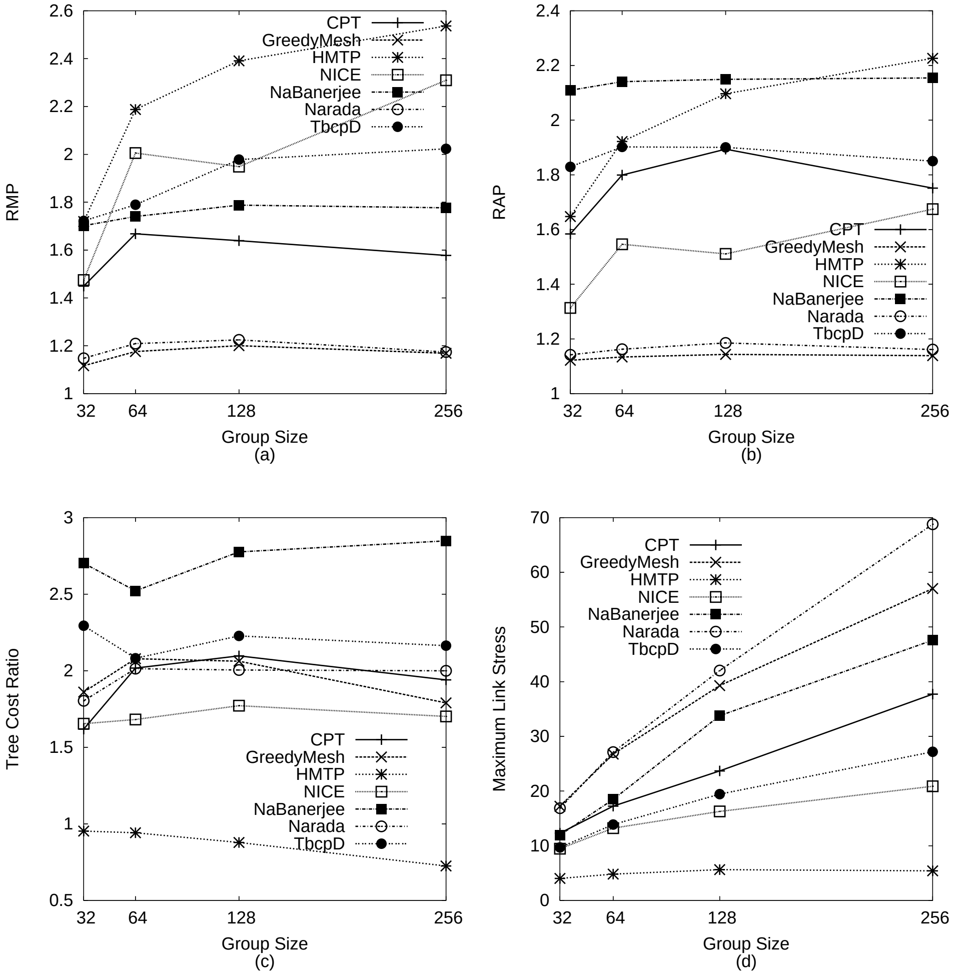


Figure 5.21: Many-to-many performance: (a) RMP, (b) RAP, (c) TCR and (d) Maximum link stress

Figures 5.21 (a) to (d) depict the comparison results. The discussion will focus on the differences observed from the one-to-many case. First, we can see that the centralised GreedyMesh gives the best RMP and RAP performance, as expected. Interestingly, Narada, the distributed mesh construction protocol follows it closely. This is mainly due to the fact that some of the Narada nodes may violate their degree constraints, as discussed in Section 5.1.1.1. Figure 5.22 confirms this by showing the maximum fan-out observed for Narada and GreedyMesh.

As discussed in Section 5.1.1.1, we also implemented a version of Narada, called Narada-SD, which tries to strictly enforce the degree bound. We have found that Narada-SD occasionally causes the overlay to partition, especially in cases where the nodes have a small degree bound. In Figures 5.23 (a) and (b), we plot the RMP and TCR of Narada-SD (averaging over cases where the overlays are connected) along with CPT and Narada — the version that does not strictly limit the nodes degree. We can see that Narada-SD's RMP is initially close to Narada, and quickly increases with the group size, and eventually

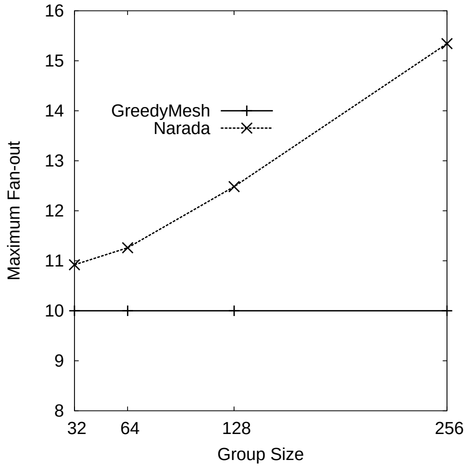


Figure 5.22: Maximum fan-out for Narada and GreedyMesh

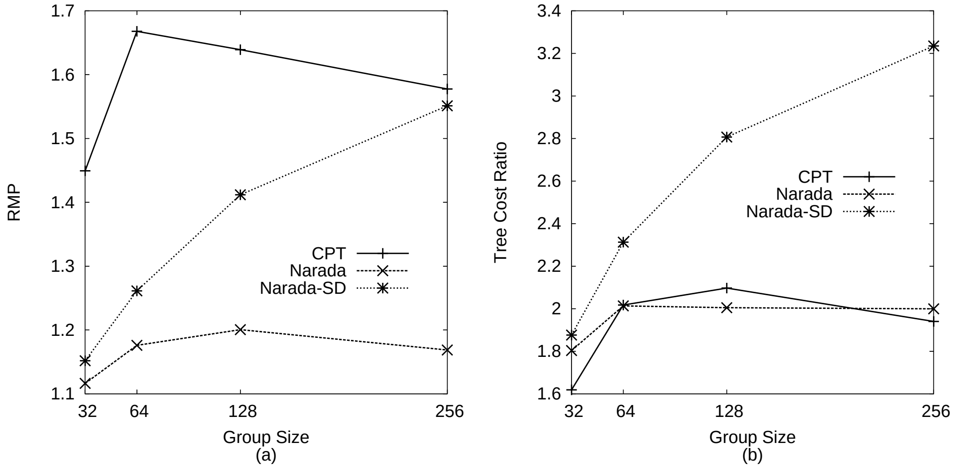


Figure 5.23: Comparing Narada, Narada-SD and CPT: (a) RMP; and (b) TCR

is close to the RMP of CPT. Obviously, limiting the nodes degree reduces the flexibility in overlay reconfiguration, and hence the poorer performance. The inefficiency manifests itself as the number of members increases. However, it is worth noting that Narada-SD still outperforms all the shared-tree proposals, for the group sizes considered. This shows that the source-specific trees approach can offer better delay performance than the shared tree proposals. The significant difference between TCR for Narada and Narada-SD indicates that Narada contains more short links in the overlays. We recall that in Section 4.3.1, we show for GreedyMesh that including short links into a mesh helps to improve the delay performance. We believe the fact that Narada has many short links provides another explanation of why its delay is on par with GreedyMesh, as shown in Figure 5.21.

In terms of RAP (Figure 5.21 (b)), NICE now performs better than all shared tree proposals, partly due to its source-specific trees approach (we recall that some of the NICE nodes violate their degree constraints). HMTP, the CoP, gives RAP that is initially better than NaBanerjee, but poorer as the group size increases. We believe that this is because HMTP achieves its low cost tree by placing nodes that are topologically close together, and hence reduces the average latency between most of the members (unlike the experiments in the one-to-many case, all members are now data sources). This provides low average delay (RAP) when the number of members is small. On the other hand, NaBanerjee creates trees that are more compact which contain many long links; this reduces the maximum delay rather than average delay. Finally, for TCR and link stress, we can see that NaBanerjee performs the worst, while HMTP is the best.

To summarise, the relative comparison between the proposals are quite similar for the one-to-many and many-to-many models. One important observation is that protocols that use source-specific trees (i.e. Narada and NICE) generally provide good RMP and/or RAP performance.

5.2.3 Effects of the Fan-out

This section investigates the impacts of fan-out (i.e. out degree) of the overlay nodes on the overlay quality. The results to be shown are obtained using the one-to-many model (similar performance trend were observed with many-to-many model, unless specified otherwise). We consider two set of experiments:

1. *Varying Fan-out*. In this set of experiments, all overlay nodes are assigned a similar maximum fan-out, with each experiment running with 256 members. We consider fan-out values that range from 2 to 10. By fixing the fan-out bound, it is easier to examine the relationship between the fan-out and the quality of the overlay built. Since Narada does not strictly enforce the fan-out bound, it is omitted from the experiments. We also study the implications of cluster size on NICE. In particular, we vary the lower bound cluster size, k, from 2 to 10. The size of NICE clusters is bounded by: $k <$ size $\leq 3k - 1$.

2. *Degree Distribution*. This set of experiments is used to determine the impacts of the distribution used for degree assignment. Earlier experiments in previous sections used a uniform distribution. Here, we consider a truncated binomial distribution to randomly assign a maximum fan-out of between 2 and 10 to each member. Three mean values are used: 4, 6 and 8, which correspond to the lower, middle and upper ends of the given range. As the maximum fan-out for each NICE node is independent of the above degree assignment process, it is omitted from the experiments.

5.2.3.1 Varying Fan-out

Figures 5.24 (a) to (d) show the variation of different metrics with the fan-out (or k), for four representative protocols: HMTP, NaBanerjee, NICE and TbcpD. In the figures, the x-axis represents the maximum fan-out assigned to the members, except NICE where its x-axis represents k, the lower bound on cluster

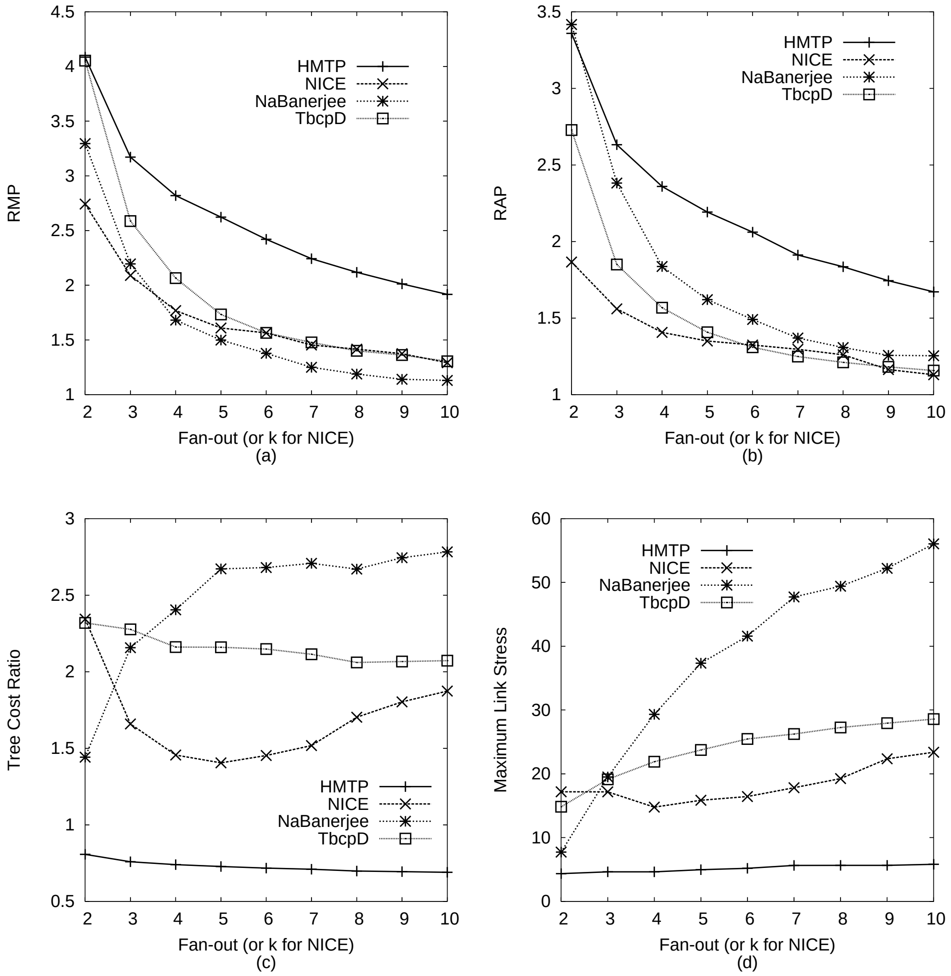


Figure 5.24: Impacts of the degree: varying fan-out

size. Due to this, the maximum fan-out for NICE nodes could be larger than other proposals. We include NICE in the same figure as the other proposals for convenience. The results for NICE should be interpreted independently.

First, it is clear that the delay property in terms of RMP for the proposals improves with larger fan-out values. This is as expected as a larger fan-out results in a wider, and thus shorter tree. The results also show that the delay gain is more significant for smaller fan-out values (e.g. from 2 to 5). The RAP results are similar to RMP. Larger fan-out also causes an increase in the worst-case link stress (Figure 5.24 (d)), especially for NaBanerjee that builds trees that have compact structure (see discussion in Section 5.2.1.3).

Figure 5.24 (c) shows the TCR performance. The result show a slowly decreasing trend for HMTP. For switch-parent based protocols such as HMTP and switch-trees, a small fan-out means that a node can easily become full with children. This restricts the movement of the nodes, and therefore results in poor performance. Increasing fan-out increases the freedom of movement, and thus gives a better performance. A larger fan-out also increases the size of the DFS search space, which increases the probability that a node locates its optimal position. Increase of fan-out also shows little impact on the HMTP worst-case stress, as shown in Figure 5.24 (d). This is because it produces low cost trees that consist of many short overlay links, which reduces the chances of packet duplication in the physical links (see Section 5.2.1.1).

The results also show that TbcpD and NaBanerjee exhibit quite different properties in terms of TCR and stress with increasing fan-out. For TbcpD, the TCR decreases slightly rather than increases, as it does in NaBanerjee. In addition, its worst-case stress also increases much slower than in NaBanerjee. This, we believe, is because TbcpD tries to minimise the triangles in the tree, which thus helps to reduce the tree cost. As a result, while the link stress increases, it occurs at a slower pace.

NICE shows a distinct TCR trend where the value decreases initially, and increases after a turning point. The k value at the turning point differs from one topology to another topology. Typically, it is around the vicinity of 5. This can be explained as follows. Small values of k result in many small clusters, which requires many inter-cluster links to connect the clusters together. As the inter-cluster links are normally longer than the intra-cluster links, this gives the higher tree cost. As k increases, the cluster size increases while the number of clusters decreases. Thus, fewer inter-cluster links are needed. This results in the reduction of TCR. However, when the cluster size increases further, even nodes within the same cluster may be quite a distance apart. Hence, TCR increases again.

In [90], we have found that for the many-to-many case, NICE's TCR and stress values increase linearly with k. We believe that this is due to NICE's data forwarding strategy. In the one-to-many case, the data is forwarded from the top-most cluster leader (the data source) to its cluster members (leaders of the lower level clusters), which continue to forward the data in the similar manner until the bottom layer. As the leaders are the graph-theoretic centre of their respective clusters, they use the smallest distance to reach each of their members. Thus, the lowest cost tree is used. On the other hand, we consider all members to be data sources in the many-to-many case. Here, data is forwarded using source-specific trees rooted at each member (see Section 2.6.1). When a non-leader member acts as a data source, its data paths are often longer than those of a leader member. For example, see Figures 2.9 (c), (d) and (e). Therefore, not all data paths will follow the cost-effective leader to members routes. Collectively, this results in the linear increases of TCR and stress.

5.2.3.2 Binomial Degree Distribution

In general, we have found that the relative comparison among the techniques under the binomial distribution is similar to those of uniform distribution. Thus, we only mention the result for HMTP and NaBanerjee, as the representatives for CoPs and DoPs respectively. Their TCR and RMP performance are shown in Figures 5.25 (a) and (b), respectively. In the figure, each curve represents the result for a

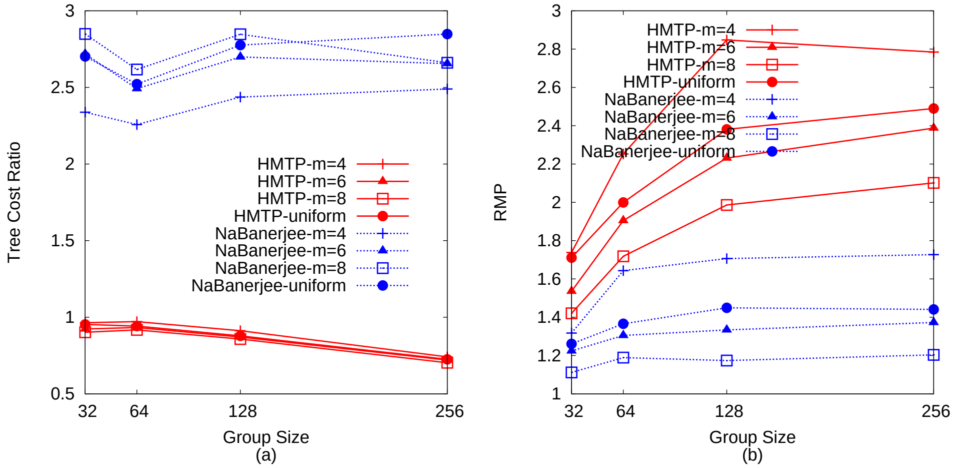


Figure 5.25: Impacts of the degree: different degree distributions

protocol obtained under a particular distribution. For example, HMTP-uniform refers to HMTP run with the uniform distribution; while HMTP-m=4, HMTP-m=6 and HMTP-m=8 refer to HMTP run with the binomial distribution, using different values of the mean.

First, the results reassert our previous observation on the trade-off between cost and delay. In addition, the relative performance of the techniques due to different mean values show similar trends to those in the investigation of the impacts of degree bound. Consider the performance of TCR for HMTP and RMP for NaBanerjee respectively. We can observe that small mean (i.e. 4) results in more nodes with a small degree bound, and thus give poor TCR and RMP properties, while a large mean (i.e. 8) results in more nodes with large degree bounds, and thus performs better.

5.2.4 Effects of the Underlying Topologies

Next, we consider the effects of the underlying topologies to the overlay constructed. Note that it is not our aim to analyse the detailed relationship between a particular protocol and a particular topology model. Rather, we are more concerned about the relationship between the optimisation metrics and the topology structure. For this reason, we consider HMTP and NaBanerjee which represent the CoPs and DoPs, respectively. We examine their TCR and RMP properties under different topologies.

Figures 5.26, 5.27 and 5.28 depict the performance of the protocols under topologies created based on the transit-stub, power-law and random Waxman models. For each model, we consider three different topologies of 1000 nodes (see Table 3.1), as represented by the three curves for each protocol (HMTP/NaBanerjee-1, 2 and 3) in each plot.

In terms of TCR, HMTP, the CoP, always produces trees with the lowest cost. For power-law and the random Waxman graphs, we can find that the TCR values are relatively smaller than in the transit-stub graphs. This is due to the degree distribution of the network nodes, as explained in Section 5.2.1.1.

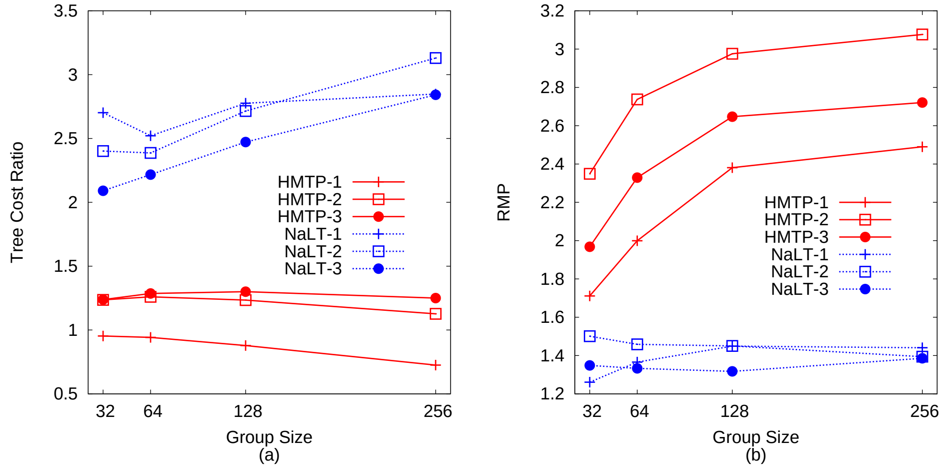


Figure 5.26: Impacts of the topologies: transit-stub topologies

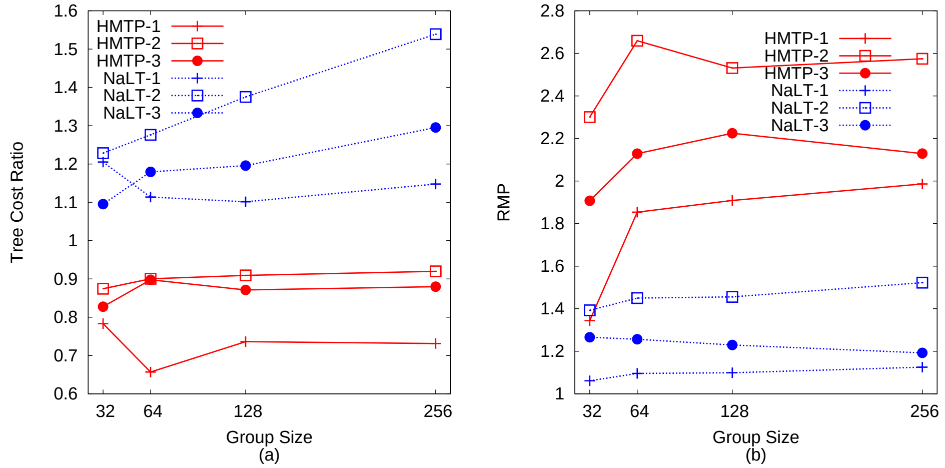


Figure 5.27: Impacts of the topologies: power-law topologies

Specifically, power law and random topologies have more large degree nodes. The cost ratio for HMTP normally stays at about the same value, or shows a decreasing trend with growing group sizes. This again proves that it can exploit the locality introduced when the group size is close to the network size. On the other hand, NaBanerjee always provides trees with lower delay, as expected.

In summary, the results show that a well-designed protocol can achieve its desired optimisation objective (e.g. cost or delay) under different topologies. For example, HMTP can produce low cost trees while NaBanerjee can produce relatively low delay trees.

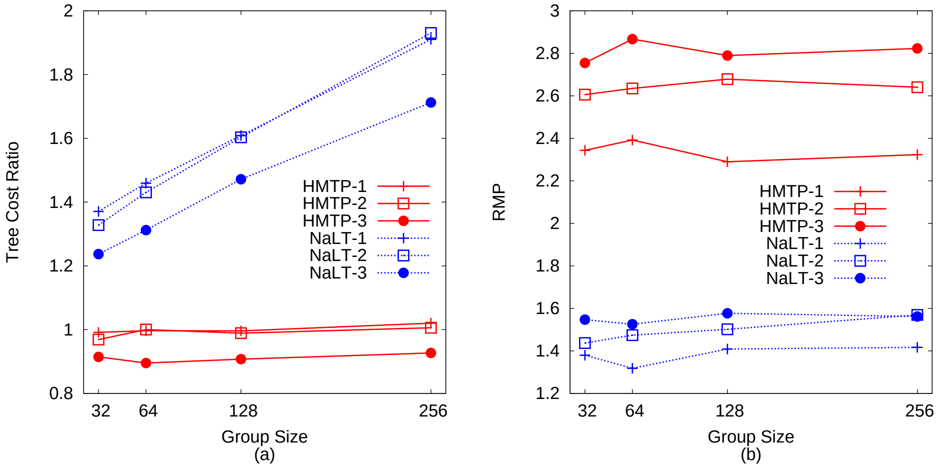


Figure 5.28: Impacts of the topologies: random Waxman topologies

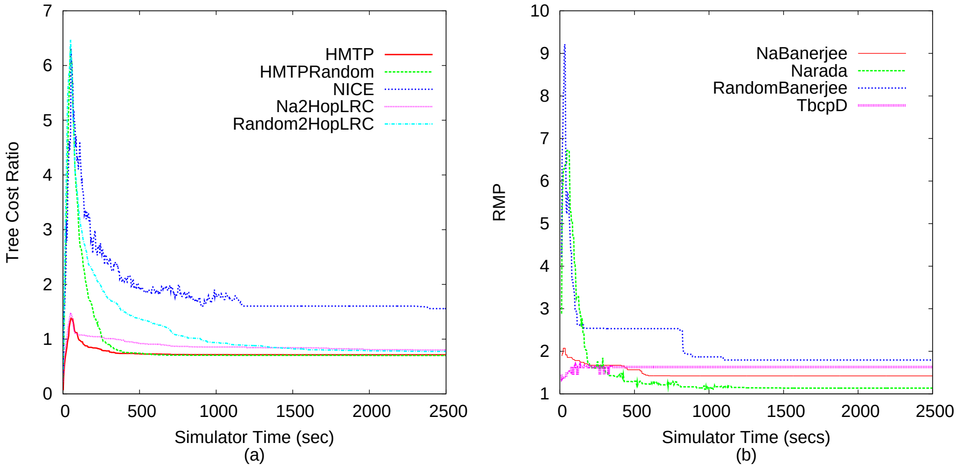


Figure 5.29: Convergence properties: (a) CoPs' TCR and (b) DoPs' RMP

5.2.5 Convergence Properties of the Protocols

This section examines the convergence speed of various distributed protocols for experiments with 256 members. In the experiments, all members join the multicast group within the first 50 seconds. Once a node has successfully joined the overlay, it initiates a periodic overlay improvement every 30 seconds.

Figure 5.29 (a) shows the convergence of TCR for the CoPs representatives: NICE, Na2HopLRC, Random2HopLRC and variants of HMTP. As described in Section 5.1.1.2, we include a modified version of HMTP which is called HMTPRandom. This version differs from the original version only in its joining strategy. Specifically, a newcomer begins its joining process from a randomly chosen on-tree member, rather than from the root.

In general, we can see that TCR increases as new members are added to the system. Once all members have joined, it begins to decrease until a stable value is reached. Comparing the two versions of HMTP, it is clear that the random version results in a high initial TCR due to the randomly connected tree structure. Similar observations can be drawn from Na2HopLRC and Random2HopLRC. The fact that HMTPRandom can quickly converge to a TCR value that is similar to HMTP demonstrates the effectiveness of its DFS technique.

The result also shows that NICE has a rather high initial TCR — similar to the randomly connected tree. This is because NICE allows clusters to grow over the size bound, and splits the clusters only at each periodic round. It is also partly attributable to NICE's longer convergence time compared to other protocols. Another reason for this is that NICE needs to maintain the invariant that each cluster leader must be the graph theoretic centre of its cluster. As a change in a lower layer cluster may result in changes in higher layer clusters, it takes a longer time to settle down — about 1200 seconds with a TCR of 1.56, which is higher than other CoPs. In [7], Banerjee et al. show that NICE's overlay converges in less than 400 seconds, for a group of 128 members. In their evaluation, they use a much smaller improvement period, i.e. 5 seconds, while we use 30 seconds, so as to consistent with other proposals. We have found that using a smaller improvement period only increases NICE's message overhead marginally, as shown in [7]. We examine the overhead of the proposals in the next section.

Figure 5.29 (b) shows the RMP convergence properties of delay-optimised representatives: Narada, NaBanerjee, RandomBanerjee and TbcpD. First, Narada and RandomBanerjee show high RMP at the initial stage due to the random layout. After all members have joined, the RMP values quickly reduce to a much smaller value for both protocols. Comparing RandomBanerjee with NaBanerjee (as well as Random2HopLRC with Na2HopLRC for TCR), we again see that the final result is related to the initial tree layout — a more structural layout often provides a better performance. For Narada, after the quick improvement stage, the RMP continues to decrease at a much slower pace until it finally stabilises at about 1400 seconds. This indicates that most changes happen in the early stage of the multicast session, as reported in [21].

Unlike other protocols, TbcpD's RMP values increase as new members join in the overlay, and stay about the same after all members joined. We have also found a similar trend in other versions of TBCP (i.e. the original version and TbcpC). This shows that TBCP joining mechanism can quickly place the nodes into their best position (depends on the score function), thus requiring fewer changes at the later stages.

5.2.6 Overhead Evaluation

The overhead of an ALM protocol largely depends on the overlay structure used and how it is maintained. In general, the protocols investigated use three different control structures: (i) tree (i.e. switch-trees,

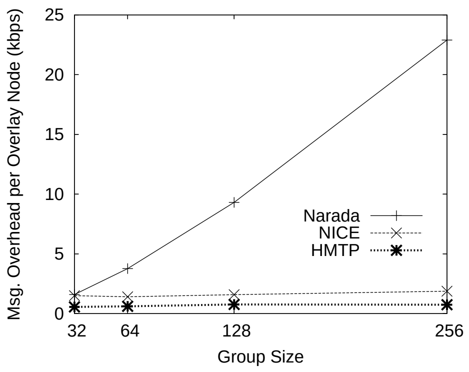


Figure 5.30: Protocol overhead: HMTP, Narada and NICE

HMTP, TBCP and Scribe[4]); (ii) hierarchical clusters (i.e. NICE); and (iii) flat mesh (i.e. Narada).

Figure 5.30 shows the message overhead measured as the control messages sent and received per overlay node, in kbps, for three representative protocols: HMTP, NICE and Narada. We consider messages used in the construction and improvement of the overlay, as well as refresh messages exchanged between the neighbouring nodes for maintaining the overlay. We assume that each message is carried using TCP over IPv4, which incurs a basic cost of 40 bytes per packet (see Section 3.2.4.1).

From the figure, it is clear that Narada, which uses a flat mesh topology and the path-vector protocol, imposes the largest control overhead. In addition, its control overhead grows quickly with the group size, highlighting the scalability problem of the protocol. NICE, on the other hand, shows a reasonably small overhead which stays almost the same across the group sizes. This is because the messages are confined within clusters that are bounded in size. Finally, we can see that HMTP, the tree-based technique has the lowest control overhead. This is because a tree uses significantly fewer links than hierarchical clusters or a flat mesh topology.

5.2.7 Summary and Discussion

Based on the previous observations, we can summarise our main findings as follows.

- There exists a trade-off between delay and tree cost optimisations: minimising tree cost results in high end-to-end delay; minimising delay results in high link stress and resource usage.
- For tree cost optimisation, the depth-first search (DFS) technique used by HMTP can effectively construct low cost trees. By using a modified version of the protocol (HMTPRandom), we also shows that this technique can converge rather quickly and is quite independent of the initial tree layout. In addition, we show that by solving the triangle problem, we not only improve the tree

[4] While Scribe is built from the Pastry mesh, the mesh links are loosely maintained (see Pastry [81].). Therefore, the majority of the maintenance overhead is on the multicast tree structure

cost property, but also reduce the overlay delay. However, as this technique greedily searches for an optimal placement, it could carry on the search while there are potential targets. This may result in the exploration of all of the group members, although this is unlikely to happen in a realistic network. A real world implementation of the technique should limit the search scope.

- In terms of delay optimisation, for the one-to-many service model, we found that Banerjee et al.'s scheme performs the best for the maximum end-to-end delay metric (i.e. RMP), while our version of TBCP performs the best for the average delay (i.e. RAP). For the many-to-many case, Narada (which uses source-specific trees) always performs the best. NICE, which also adopts the source-specific trees approach, has good average delay performance compared to protocols that use shared tree routing. Hence, we conclude that the source-specific trees approach has better support for delay-sensitive multi-sender applications. In fact, the superiority of the delay performance of source-specific trees approach over shared tree approach has long been acknowledged in the study of network layer multicast [103]. However, existing source-specific ALM protocols still have some weaknesses. For examples, Narada incurs a large protocol overhead, while NICE, the Delaunay triangulation protocol and LARK do not provide a degree constrained overlay. How to achieve low delay degree-bounded overlay trees with low overhead for many-to-many multicasting is an interesting research topic. In Chapter 8, we address this problem with a multiple shared trees proposal.

- In general, an ALM overlay always yields a smaller resource usage and link stress than the unicast star overlay. However, it is important to point out that a badly designed delay-optimised protocol can result in poor delay performance as well as high resource usage.

- NICE, a hierarchical cluster-based protocol, can strike a balance between tree cost and delay optimisation due to its overlay structure and choice of cluster leader. Unfortunately, the resultant overlay is not constrained in nodes degree, which makes it unsuitable for environments where members have heterogeneous capacities.

- For variants of switch-trees, we found that the initial tree layout can affect the final performance. In particular, a simple DFS (next-available) can create a reasonably good initial tree structure for both cost- and delay-optimised switching functions. In terms of switching scope, we show that local scope switching can provide fast convergence due to informed parent choice. But its localised properties also limit its exploration power. Fortunately, we can interleave the local scope with random node selection to yield good convergence and better exploration.

- We demonstrate the greedy nature of delay-based switching, as in the case of Random1HopD (see Section 5.2.1.2). The greedy problem can result in poor delay as well as poor resource usage.

- We also examine the impacts of fan-out on the overlays built. In general, we observe that larger fan-out values allow more flexible overlay reconfiguration to happen, and hence provide better performance. Conversely, smaller fan-out limits the movement of the nodes, and thus gives poorer performance.
- By experimenting with different topology models, we believe that a well-designed protocol can achieve its desired optimisation objective under different topologies. In addition, the performance of certain metrics (i.e. link stress and tree cost) is related to the underlying topology structures.
- In terms of protocol overhead, we show that protocols that use the spanning tree structure consume the least control traffic, while protocols that use a flat mesh structure and a conventional routing protocol have the worst overhead. The hierarchical cluster structure has a reasonably small protocol overhead.

5.3 Related Work

In this work, we limit our comparison to representative tree cost- and delay-optimised protocols. As discussed in Chapter 2, there are other interesting metrics that can be considered. We believe that some of our findings are applicable to these cases. For instance, the observations on the switching scope and transformation techniques can be useful to other tree-based proposals. A good example is Overcast [48], a tree-first protocol that tries to create high-bandwidth trees using local-scoped switching. It could be improved by using the mixed local and random switching scope.

Since the initial proposals on ALM (e.g. [36, 21]), there has been some other comparison work. Typically, the investigation considers only a small number of proposals. For examples, in [7], Banerjee et al.'s propose NICE and compared it with Narada; in [104], Wu et al. introduce AOM and contrasted it with HMTP.

Another class of comparison work considers proposals that exhibit a similar nature. For example, Castro et al. [17] evaluate ALM overlays built using DHT-based overlays, in particular CAN and Pastry. They investigate two data delivery mechanisms for these overlays: tree building and flooding. Their results show that the tree-based approach consistently outperforms the flooding approach, and that Pastry-based overlays out-performs CAN's.

In [47], Jain et al. evaluate the potential of DHT-based overlays. They compare CAN-multicast [78] and Chord [88] with NICE and Narada. They consider two ways of creating the DHT-based overlays: (i) *topology-agnostic* where the overlays are built without using any topology knowledge; and (ii) *topology-aware* where the complete topology information is used, i.e. all members are assumed to have ideal topology knowledge. Their results show that the topology-agnostic versions of CAN-multicast and Chord can have a relative delay penalty that is more than NICE and Narada by at least a factor of two. On the other hand, the topology-aware versions can achieve a comparable performance with NICE and Narada.

These comparisons indeed provide some insights into the different techniques. However, as they mostly make different assumptions and use different simulation settings, it becomes difficult to make an overall comparison. In this chapter, we have considered representatives from a larger class of overlay creation and maintenance techniques, and evaluate them under a unified environment. We also looked in detail at the various components that contribute to the good/poor performance of a particular strategy.

There are several excellent surveys on existing strategies related to overlay multicast. In [29], El-Sayed et al. review and discuss several multicast proposals that offer an alternative due to the lack of deployment of network layer multicast. They classify the proposals into several categories (see Section 1.1.1 for details): based on a reflector approach, creating an automatic overlay topology (i.e. ALM), or relying on a specific routing service. ALM protocols studied in this chapter belong to the class of automatic (i.e. self-organised) overlays. In [6], Banerjee and Bhattacharjee compare several ALM protocols: Narada, HMTP, Yoid, Scribe, CAN-multicast and NICE. Both surveys focus on the high-level protocol description of the various techniques. On the other hand, we have focused on detailed quantitative evaluations.

5.4 Chapter Summary

In this chapter, we investigated the efficiency of several self-organising techniques for building low cost and low delay ALM trees. The techniques studied encompass representatives from the two main overlay construction techniques, i.e. tree-first and mesh-first. The tree-first protocols considered include HMTP, TBCP, NICE, and variants of switch-trees (including a version of HostCast) and Banerjee et al.'s scheme. We consider Scribe and Narada as the mesh-first representatives. The various aspects of the protocols were examined under a unified simulation environment, using `ALMSim`.

This work is the first step towards designing protocols to building low delay and low cost ALM trees. From the results, it is clear that these two metrics conflict with each other, and it is better to consider them separately. The results show that HMTP, a simple distributed cost-optimised protocol, can produce trees that have comparable costs to those created by a centralised algorithm.

The delay-optimised protocols, on the other hand, still exhibit several noticeable weaknesses. Specifically, in terms of one-to-many delivery model, we show that Banerjee et al.'s scheme can yield trees with small root-diameter, but at the expense of the average delay to all the members. Our improved version of TBCP provides good average but poorer worst-case delay. We thus believe, the main challenge for a good delay-optimised protocol is that: it should provide low maximum and average delay properties, as well as result in reasonably small traffic redundancy and network resource usage. In Chapter 7, we introduce a mesh-based approach to create trees that exhibit the desired properties.

In terms of many-to-many multicasting, the results also reveal that the source-specific trees approach (i.e. Narada) can yield better delay properties than the shared-tree approach. However, it imposes a much higher protocol overhead, which limits its usability for larger group sizes. In Chapter 8, we investigate a multiple shared trees strategy as a compromise between the shared-tree and the source-specific trees.

This chapter emphasises techniques used to build efficient degree-bounded overlay trees. In next chapter, we consider the problem of managing degree-bounded overlay trees. In particular, we propose a generic framework for creating and maintaining a degree-bounded tree using a mesh structure.

Chapter 6

Mesh-based Overlay Tree Construction and Maintenance Framework

This chapter considers the problem of managing a degree-bounded overlay tree. We introduce a framework for creating and maintaining a loop-free degree-bounded tree by using an overlay mesh. The tree can be used either as a source-rooted tree for single-source applications, or as a shared tree for multi-source applications. The main contribution of this work is a fast tree recovery scheme, which explicitly harnesses the multiple paths property of the mesh. We also take advantage of the tree structure to reduce the maintenance overhead of the overlay. The framework is generic, and can thus be used by existing protocols to maintain their tree structure.

To illustrate the working of the framework, we apply it to a case study: a root-diameter and degree-bounded, minimum cost tree creation problem. We compare it with ACDC [54], an existing proposal for the problem, in terms of the quality of the trees built, and two tree recovery schemes in terms of recovery speed. Simulation results show that our proposal provides better tree quality and recovery speed.

The rest of this chapter is organised as follows. The next section first positions our work with some related research. In Section 6.2, we present the proposed framework. Section 6.3 evaluates various aspects of the framework using the case study. Finally, Section 6.4 concludes this chapter.

6.1 Related Work

One of the key issues in multicast overlay management is keeping the overlay connected after node departure (fail or simply leaving the multicast session voluntarily).

A tree is the natural structure for multicasting. It simplifies the data forwarding as it is inherently loop-free. In a tree, when a non-leaf node departs, its immediate children (and nodes under their subtree) will be partitioned from the main structure. They need to be able to reconnect quickly to the main tree (i.e. obtain a new parent) so as to resume the data flow. This is the tree restoration problem. In a degree-bounded tree problem, it is important that the restored tree does not violate the fan-out capacity of the nodes. Due to the limited capacity at the nodes, a degree violation may disrupt the data service [107].

We can classify the ALM proposals based on the structure of the control topology used to manage the overlay:

- *Tree-based*. In this case, a tree is used as the control or management structure. Typically, the tree also serves as the data delivery tree. Examples of protocols that fall into this class are HMTP and TBCP described in Chapter 2.

- *Mesh-based*. In this case, a connected overlay mesh is used to connect all members. The delivery tree is embedded in the mesh. A routing mechanism is needed to obtain the delivery tree from the mesh. As a mesh provides redundant paths between the nodes, it is considered to be more resilient than a tree. Thus, tree-first protocols like Yoid and HostCast include extra links in addition to the delivery tree to form a control mesh. NICE and Zigzag which use hierarchical clusters in the overlay also fall into this group. Obviously, all mesh-first protocols considered in Chapter 2 are mesh-based protocols.

It is worth noting that these two classes are difference from the tree-first and mesh-first (see Section 2.6), which classify the proposals based on the way that the proposals construct the overlay.

Reconstructing a degree-bounded tree is a harder task compared with the unconstrained case. In the unconstrained case, when a non-leaf node departs from the overlay, its immediate children can quickly reconnect to their grandparent, and the recovery process is done. On the other hand, in the constrained case, the grandparent may not be able to accept all of its grandchildren, due to its degree limitation. Hence, the rejected grandchildren need to locate a feasible parent quickly.

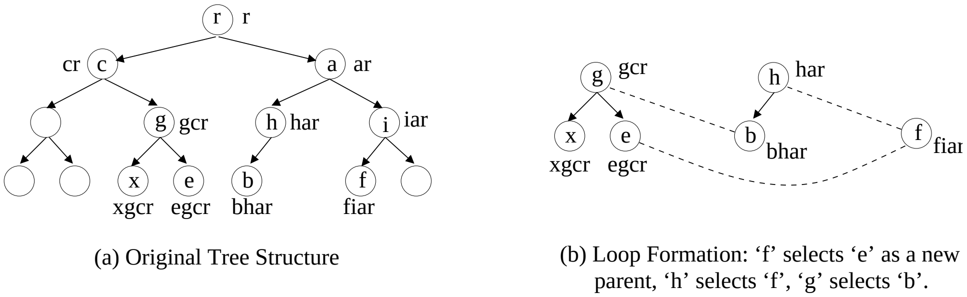
Existing tree-based protocols often follow a reactive approach to repair a tree partition. A reactive approach performs the tree restoration process *after* detecting a node's departure. In [107], Yang and Fei investigate several reactive schemes — grandparent, grandparent-all, root, root-all — proposed by Deshpande et al. [26]. Simulation results show that the grandparent scheme yields the best recovery time — the time from when a node loses its parent until it finds a new parent. In the grandparent scheme, when a node departs, its children will request to attach to their grandparent so as to reconnect to the tree. As all requests go to the same node, some of them may be rejected. If the grandparent cannot accept a request, it will redirect the request to one of its children. The rejoin process continues recursively down the tree until the recovering nodes finally attach to the tree. As the redirection target is arbitrarily chosen, a considerable amount of time may elapse before these nodes finally find a feasible parent. The root scheme is similar to the grandparent scheme, except that the children of the departed node contact the root rather than their grandparent. Grandparent- and root-all differ to the corresponding schemes above in that all the descendants of the departed node, instead of just its children, try to rejoin the tree.

A similar strategy is used in other tree-based protocols. They mainly differ in choosing the target parent for nodes performing the recovery process. For example, in HMTP, each recovery node randomly selects the target parent from one of the ancestors (i.e. nodes along the path from the recovery node to the tree root). This reduces the chances that all nodes contact the same node at one time, and may improve the recovery time. However, this approach may not be suitable for all conditions. For example, for the delay-optimised problem, the tree is likely to have a compact structure, where nodes at tree levels close to the root will mostly be occupied. This reduces the chances of the request being granted by the ancestor nodes.

In their paper [107], Yang and Fei propose a proactive tree recovery scheme. As opposed to the reactive approach, the proactive approach plans for the departures *before* they happen. The basic idea of the Yang and Fei's scheme is that each non-leaf node in the overlay tree precomputes a parent-to-be for each of its children before it departs. Thus, when they actually depart, their children can quickly reattach to the tree. Their proposal was found to outperform all the reactive proposals investigated.

Our mesh-based framework can be viewed as a middle ground between the reactive and proactive approaches. Like the reactive approach, the tree restoration process starts after the node's departure. While not explicitly precomputing the target parent for each of the nodes, such information is implicitly contained in the mesh overlay. In Section 6.3.2, we will show that our approach is comparable with, and sometimes better than, Yang and Fei's proactive scheme.

Another concern in a tree structure is the formation of loops in the tree. Loops typically form during the overlay reconfiguration process. A loop will result in endless packet circulation and, potentially, partition the tree. Thus, it is important to have a quick loop detection and termination procedure. A root path for a node, say x, is the list of nodes in the route from x to the root, via the overlay. The root path is propagated to all tree nodes in the following manner: a node appends itself to the root path it receives from its parent, and forwards its to all of its children. If every on-tree node maintains a root path, there exists a simple loop avoidance technique: a node, x, accepts a new child only if the new child is not in the root path of x. However, as pointed out by Francis et al. [36], this simple technique does not guarantee there are no loops at all. A loop could still happen if two or more nodes that are the roots of different subtrees select new parents at approximately the same time. Consider the following scenario presented by Francis et al.. Figure 6.1 (a) shows a tree rooted at r. The root paths for the nodes are given beside the nodes. Now, assume that for some reason, node f joins e as its new parent, h joins f, and g joins b, then a loop involving nodes $efhbg$ is formed (see Figure 6.1 (b)). This is because at the instant that the nodes join their new parents, the parents' root paths have yet to indicate a loop. For loop detection, when a node switches to a new parent, its new root path is quickly propagated over the subtree of that node.



(a) Original Tree Structure

(b) Loop Formation: ‘f’ selects ‘e’ as a new parent, ‘h’ selects ‘f’, ‘g’ selects ‘b’.

Figure 6.1: Example of loop formation [36]

If a node receives a root path that already contains its own information, it can break the loop by trying to switch to another node. However, it is possible that more than one node will detect the loop, and try to rejoin the tree. More importantly, there is no guarantee that the new configuration does not contain a loop. All these could prolong the tree convergence time [36]. To counter the above problem, Francis et al. propose to associate an integer value, called *switch-stamp*, to every node in a root path. When a node receives the first root path from its new parent, the switch-stamp for the node is set to be greater than any of the switch-stamps of the nodes in the received root path. With this, when a loop is formed, a node with the largest switch-stamp can be deterministically chosen to break the loop. In addition, no new join request will be accepted by nodes in the loop, until the loop is resolved.

Our framework maintains a mesh overlay. Unlike the tree, loops are an inherent feature of the mesh. In other words, there is no need to prevent loop formation in the mesh. However, we do need to make sure that the delivery tree is loop-free. Our approach consists of two parts. First, the simple loop avoidance technique mentioned above is used when a node tries to add a tree link, so as to prevent most potential loops. Secondly, we run the path-vector routing protocol over the mesh to resolve any loops in the tree.

The path-vector protocol is derived from the well-studied distance-vector (sometimes called distributed Bellman-Ford) routing protocol [93]. In distance-vector routing, each node periodically exchanges its own routing table with its neighbouring nodes. The routing table consists of $<$destination, distance$>$ tuples. On receiving a routing table from a neighbour, a node updates its own routing entries for destinations that the neighbour believes to have a better route for. As the update decision is based solely on the distance value, distance-vector routing can be trapped by the well-known count-to-infinity problem, in which the routing update is bouncing back and forth between several nodes for an extended time. The path-vector protocol solves the count-to-infinity problem by including the whole path list for each destination in the routing message. This allows nodes to quickly detect a loop, and thus improves the route convergence time. It is easy to see that the idea of using root path in the tree-based proposals is a variant of path-vector routing. What differentiates the tree-based proposals from ours is that they must always maintain a tree structure, hence a loop must be resolved by altering the overlay structure. In our case, we resolve a loop by reestablishing the on-tree relationship for nodes involved in the loop.

A common belief is that a mesh is more resilient to the partitioning problem than a tree. It is true that a mesh may still be connected even after some nodes disappear simultaneously. Note, however, that the data distribution topology used is still a tree. We believe that existing mesh-based proposals have yet to fully exploit the advantage of a mesh structure for tree restoration, specifically, in terms of the degree-bounded tree restoration problem. We discuss some of these limitations as follows.

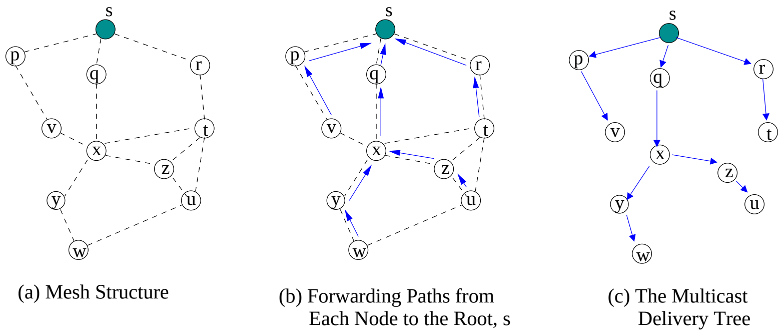
- *Delivery tree derived from the mesh is not degree-bounded*. NICE and Zigzag use a multi-level clusters overlay for maintenance. Both protocols constraint the size of each cluster to a value between k and $3k - 1$ inclusively, where k is a configurable parameter. Due to this, the degree for each individual node can be as high as $(3k - 1) \log_{3k-1} n$, for an n nodes overlay. Other examples that fall into this group are Delaunay triangulation protocol and LARK. It remains unclear how these proposals could build delivery trees that honour the degree bound for each individual node, while still preserving the original properties and integrity of the proposals.

- *Delivery tree is degree-bounded but not the mesh*. Tree-first protocols like Yoid and HostCast try to improve tree robustness by adding extra links into the tree structure (thus, result in a control mesh topology). In Yoid, the mesh links are randomly added, while in HostCast, extra links are added only between nodes within a predefined local region (see Figure 2.6). However, these links are added without considering the degree constraints of the nodes. As a result, they may not be useful when the degree constraints need to be enforced during the recovery process. A similar problem is also faced by the DHT (distributed hash table) based protocols (e.g. Scribe, Bayeux and CAN-multicast).

Our framework maintains a degree-bounded mesh, and the delivery tree is embedded in the mesh. As the degree constraints are decided by each individual node based on their bandwidth limitation, the mesh links are directly useful for tree restoration.

Our framework can be viewed as a restricted version of Narada [21] and Gossamer [18]. Both proposals maintain a mesh overlay, and create source-specific trees for many-to-many multicasting. The trees are obtained from the mesh using the path-vector routing protocol. Our framework follows the mesh-based approach, and uses the same routing protocol for tree derivation. However, we only consider a single tree in the overlay. This allows us to make some simple modifications to the routing procedure so that it is tightly integrated with the tree structure. This reduces the number of communication messages required. We also include procedures that take advantage of the redundant links information to help the tree restoration process.

All previously mentioned schemes, as well as ours, works in the control plane. That is, they try to provide uninterrupted data flow by repairing the delivery tree using the control topology. The reliability of the data is managed by the upper-level applications. Complementary to this is the data plane approach, which also tries to provide reliable data transmission. An example is PRM (Probabilistic Resilient Multicast) [10] proposed by Banerjee et al.. In PRM, besides sending data over a delivery tree, a randomised



(a) Mesh Structure

(b) Forwarding Paths from Each Node to the Root, s

(c) The Multicast Delivery Tree

Figure 6.2: Example of mesh overlay: (a) The mesh structure; (b) Forwarding paths from each node to the root, s; and (c) The multicast delivery tree

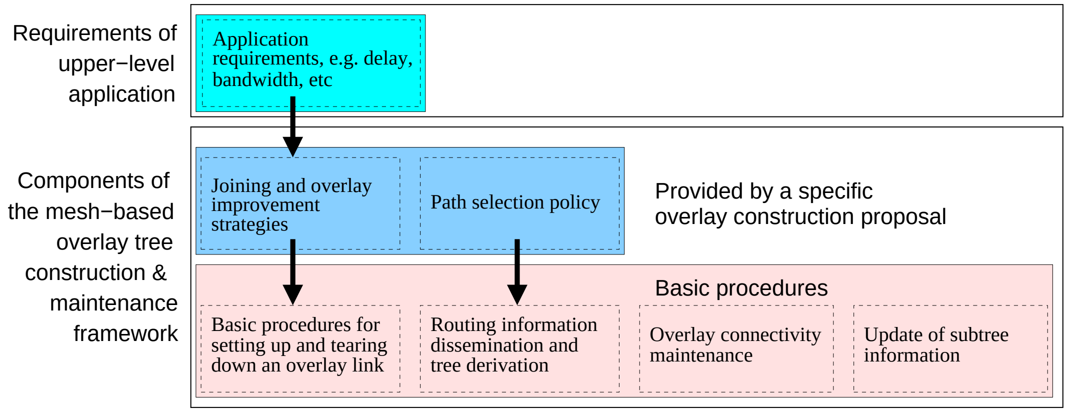

Requirements of upper−level application
Application requirements, e.g. delay, bandwidth, etc
Components of the mesh−based overlay tree construction & maintenance framework
Joining and overlay improvement strategies
Path selection policy
Provided by a specific overlay construction proposal
Basic procedures
Basic procedures for setting up and tearing down an overlay link
Routing information dissemination and tree derivation
Overlay connectivity maintenance
Update of subtree information

Figure 6.3: The components of the framework and its relationship with upper-level application

forwarding method is used to deliver extra copies of the data to the nodes. In particular, each overlay node randomly chooses a constant number of other overlay nodes and forwards data to each of them with a low probability. This can provide high delivery ratios in case of node failure, at the expense of higher data volume [107]. This scheme can be used to improve the transient behaviour of the control plane solutions.

6.2 Framework Description

Our mesh-based framework provides basic operations for the construction and maintenance of a degree-bounded overlay delivery tree using an overlay mesh. Figure 6.2 (a) depicts an example of the mesh overlay. Figure 6.2 (b) shows the intermediate step to obtain the delivery tree (to be explained shortly), which is shown in Figure 6.2 (c).

Figure 6.3 illustrates the components of the framework and its relationship with upper-level applications. The framework itself comprises two levels. The lower level consists of four components which provide basic functionality for creating and managing an overlay. They include (i) basic procedures for setting up and tearing down the overlay links; (ii) routing information dissemination and derivation of the delivery tree (consists of a path-vector routing protocol); (iii) rules that ensure the connectivity of the tree; and (iv) a procedure for updating subtree information. These four components work together to ensure the connectivity of the overlay, as well as the loop-free feature of the delivery tree. The upper-level of the framework consists of two components, which include (i) the joining (i.e. how a newcomer will join the overlay) and optimisation (i.e. rules for reconfiguring the overlay) strategies; and (ii) the path selection policy to obtain the tree. The joining and optimisation strategies decide when and where to add or drop an overlay link; while the path selection policy is used to guide the derivation of the delivery tree. These two components are provided by a specific overlay construction proposal which applies the framework. The joining and optimisation strategies and path selection policy are in turn driven by the upper-level application needs (e.g. delay, bandwidth, etc).

The rest of this section describes the framework, and is structured as follows. In the next two subsections (6.2.1, 6.2.2), we discuss the overlay structure and state information use by the framework. The basic procedures provided by the framework: (i) setting up and tearing down overlay links; (ii) routing process and delivery tree derivation; (iii) overlay maintenance; and (iv) subtree information update, will be given in Sections 6.2.3, 6.2.4, 6.2.5 and 6.2.6, respectively. Section 6.2.7 summarises the control messages used in the framework, which is followed by an analysis of the framework in Section 6.2.8. In Section 6.3, we illustrate how a specific overlay construction protocol can make use of the basic procedures using a case study.

6.2.1 Overlay Structure

The framework maintains overlays in the form of a connected degree-bounded mesh. The mesh connects the tree root, s with all other members. The degree bound for a node, i is represented by $d_{max}(i)$. It is calculated based on the maximum fan-out of i, which in turn depends on i's bandwidth limitation (see Section 3.1). The value of $d_{max}(i)$ is determined based on the type of the tree to be created. For a source-rooted tree, $d_{max}(i)$ is equal to one plus the maximum fan-out of i, where the additional one accounts for i's incoming link from its parent. If i is the tree root, which is also the source, the one is omitted. For a shared tree, $d_{max}(i)$ is set equal to i's maximum fan-out. This is because any node in the shared tree can be a data source. In the rest of this chapter, we refer to the tree maintained by the framework as the *delivery tree*.

Two nodes in a session are said to have a neighbouring (or peering) relationship when the overlay link between them exists in the constructed mesh. For node i, the set of neighbours in the overlay is represented by N_i^m. The links between i and its neighbours are called mesh links (thus, all links in the overlay are mesh links). A mesh link may or may not appear in the delivery tree as shown in Figure 6.2. In other words, the set of mesh links can be further classified into two types (which define the neighbouring relationship between the two end points of a link):

1. *Tree links*. These are links that exist in the delivery tree. One end-point of a tree link defines the parent node, while the other end defines the child node. All nodes (except s) must have a parent node. For a node i, we use N_i^t to represent its set of *tree neighbours*. The parent and children of i are represented as p_i and C_i respectively. Thus, $N_i^t = \{p_i\} \cup C_i$.

2. *Non-tree links*. These are links that are not included in the delivery tree. For node i, the set of *non-tree neighbours* are represented by $N_i^{t'}$, i.e. $N_i^{t'} = N_i^m \setminus N_i^t$.

We further define N_i^w as the pending neighbours of i, i.e. the set of nodes that i has agreed to accept as neighbours, while waiting for the neighbour setup process to complete (see Section 6.2.3). Consider node x in Figure 6.2, we can see that $N_x^m = \{q, t, v, y, z\}$, $N_x^t = \{q, y, z\}$ where $p_x = q$ and $C_x = \{y, z\}$, and $N_x^{t'} = \{t, v\}$.

The degree constraint for a node i, $d_{max}(i)$ can be enforced by making sure that

$$|N_i^m| + |N_i^w| \leq d_{max}(i). \tag{6.1}$$

Since the delivery tree is derived from the degree-bounded mesh, the degree bounds for the nodes in the tree are guaranteed. This simplifies the tree restoration process as the non-tree links available in the mesh are immediately eligible for repairing a tree partition. To aid the overlay recovery process, we also define $d_{res}(i)$, the residual degree for i, as

$$d_{res}(i) = d_{max}(i) - |N_i^t| \;. \tag{6.2}$$

It represents the number of nodes that i can still accept as its tree neighbours. We further define $d_{res}(T_i)$ as the total residual degree of the subtree rooted at i. Take node x in Figure 6.2 (c) as an example. Assume that $d_{max}(x) = 5$, $d_{max}(y) = 3$, $d_{max}(z) = 3$, $d_{max}(w) = 2$ and $d_{max}(u) = 2$. For x, $| N_x^t |$ is 3, thus $d_{res}(x)$ is 2. This indicates that x can still accommodate two more tree neighbours (other than t and v). However, to do so, it needs to drop the non-tree links with t and v, so as to fulfil the degree constraint. For t and v, they just need to negotiate a change of status with x to become x's tree neighbours. We can also calculate the total residual degree for subtree rooted at x, $d_{res}(T_x)$ as 6, where x has 2 spare degree, and each of its descendants (y, z, w, u) each has one spare degree. Table 6.1 summarises the notations used in the framework.

The delivery tree is obtained based on the reverse path tree concept used in DVMRP [25]. Every node participates in a path-vector routing protocol to learn the paths from itself to s. Given multiple routes to s, a node selects the “best” path as its routing path, where the goodness of a path is judged by the desired properties, e.g. low delay, low cost, etc[1]. The delivery tree is obtained as the union of all the reverse of these paths, i.e. a reverse routing tree. The next hop that a node uses in its path to s is therefore

[1] It is important to note that the path selection policy must result in loop-free paths.

Notation	Description
$d_{max}(i)$	Node i's degree bound
$d_{res}(i)$	Residual degree at i
$d_{res}(T_i)$	Residual degree for subtree rooted at i
N_i^m	Set of i's mesh neighbours
N_i^t	Set of i's tree neighbours
$N_i^{t'}$	Set of i's non-tree neighbours
N_i^w	Set of i's pending neighbours
p_i	Node i's parent
C_i	Set of i's children

Table 6.1: Notations used in the framework

the node's delivery tree parent. For example, Figure 6.2 (b) illustrates the routing paths from each of the nodes to the root, and the reverse delivery tree is shown in Figure 6.2 (c). The details procedures will be explained in Section 6.2.4.

6.2.2 Overlay Node State

This section describes the basic information used by the framework. Extra information may be added by a specific overlay construction protocol.

A node, i, maintains the following information.

- The information (IP address and communication port number) of its mesh neighbours and the root node, s.

- Residual degree for each of its children, and the total residual degree of the nodes in the subtree rooted at each child: $d_{res}(c)$ and $d_{res}(T_c) \forall c \in C_i$. This information is provided using the subtree information update procedure, Section 6.2.6. This information will be used to redirect a node which actively looking for a parent node (see Section 6.2.3). The idea of using the residual degree information is borrowed from Yang and Fei's tree recovery proposal [107] (see Section 6.3.2.2).

- Root path. A root path is the list of nodes in the overlay route from a node to the root. Take node w in Figure 6.2 (c) as an example, its root path is $\{w, y, x, q, s\}$. For ease of exposition, we classify a root path into three types:

    1. Routing path. This is the forwarding path from a node to the root, obtained from the routing process (see Figure 6.2 (b)). As explained before, the reverse of it forms part of the delivery tree. A node uses the next hop of the routing path as its tree parent. The routing protocol requires neighbouring nodes to periodically exchange their routing path. Using the whole path provides a simple way for loop avoidance. In particular, if the path from a neighbour of node i includes i, i will treat it as an invalid path. An invalid path will not be considered in

deriving the delivery tree (see Section 6.2.4). One or more performance metrics (e.g. delay, bandwidth, etc) may be associated with a path. This gives the cost of using a path.

2. Non-tree neighbours' root paths (non-TNRPs). For node i, a non-TNRP is a valid path via one of i's non-tree neighbours, $N_i^{t'}$. It provides a first tier alternative route to the root. A node applies the path selection algorithm on all of its non-TNRPs to find the best alternative path to the root. The selected path will be advertised to its parent, which becomes the parent's *tree children's root path*, as described below.

3. Tree children's root paths (TCRPs). These are the "best" alternative paths provided by i's children. They serve as the second tier alternative routes to the root. Both non-TNRPs and TCRPs are used in the overlay recovery process (see Section 6.2.5).

Referring to Figure 6.2 (b), x's routing path is $\{x, q, s\}$, where q is x's parent. In addition, x has two non-TNRPs, which are $\{x, t, r, s\}$ and $\{x, v, p, s\}$, obtained from t and v respectively. Node x also has one TCRP: $\{x, z, t, r, s\}$ from its child, z. Note that x's child y does not provide any valid TCRP for x since it does not have any non-tree neighbours.

6.2.3 Setting Up and Tearing Down Overlay Links

An overlay consists of a set of overlay links connecting the members. As mentioned previously, each link is represented by the neighbouring relationship between two nodes. In other words, an overlay is constructed by forming the relationships between the nodes, which involves setting up and tearing down the links.

The establishment of a new link consists of a sequence of request, reply and acknowledgement procedures, occurring between the two end points of the link. For conciseness, we will refer to the node that is currently performing a requesting process as i, and its potential neighbour as j. The procedures ensure that both nodes reach a common consensus on their relationship, i.e. parent-child or merely mesh neighbours. During the process, i and/or j may need to drop an existing neighbour so as to enforce the degree bound.

Briefly, the three procedures perform the following functions.

- *Request procedure*. Node i initiates a link establishment request to a potential neighbour. The identity of the target neighbour is determined by the overlay construction protocol that uses the framework.

- *Reply procedure*. Node j processes i's request, and decides if i can be accepted as a neighbour, and if yes, what type of relationship will be established. Again, the rules used in the decision making is provided by the overlay construction protocol. The admission reply will be sent back to i.

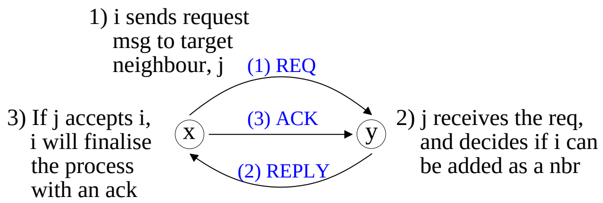


Figure 6.4: The request, reply and acknowledge sequence for setting up an overlay link

- *Acknowledgement procedure*. If j accepts i as a neighbour, i will determine if a common consensus has been reached between the nodes. A positive or negative acknowledgement will be sent to j to finalise the link establishment process. If j rejects i's request, no further action is needed.

Figure 6.4 offers a simple summary of the procedures. From the figure, we can see that when a node (say i) initiates a message to another node (say j), i may expect a reply message from j. To avoid deadlock in the waiting process, i will start a timer when it transmits the message. If the timer expires before the reply from j reaches i, i will consider j as unreachable and clean up the intermediate information formed during the process.

We divide the link establishment procedures based on the following two circumstances:

1. *Join or rejoin the overlay*. In this case, a child to parent tree link will be created. As mentioned previously, all nodes (except the root) must have a parent node. The procedures are needed when: (i) a newcomer tries to join in the overlay; and (ii) an existing member loses its parent, and needs to reattach to the tree. It is worth recalling that the delivery tree is obtained from the mesh. Hence, adding a tree link also means adding a mesh link.

2. *Overlay reconfiguration*. An overlay needs to be reconfigured from time to time for several reasons. First, the initial structure does not necessary provide the desired robustness and/or quality in data delivery. Secondly, the overlay needs to adapt to changes in the overlay memberships (when members join/leave/fail) as well as changes in the underlying network conditions, which may happen throughout the session. This may result in either a tree link or a pure mesh link. The overlay construction protocol determines when a node should perform an improvement operation, and how to reconfigure the overlay.

The following two subsections (6.2.3.1 and 6.2.3.2) describe in details these two cases. Section 6.2.3.3 describes the procedure of tearing down an overlay link.

6.2.3.1 Join or Rejoin the Overlay

Request Procedure The request procedure marks the start of a link establishment process. To create a new tree link, i will first select a number of potential parents, and send to each of them a JOIN_REQ message. The number of potential parents should be limited to the residual degree of i. By allowing multiple requests, we hope to attach i quickly to the overlay. If more than one node accepts i's request, only one of them will be used as i's parent (see the acknowledgement procedure below). How the potential parents are chosen depends on the overlay construction protocol. For example, a newcomer may use the tree root or some randomly selected nodes as potential parents. For a node trying to reattach to the overlay tree, its potential parent is provided by the overlay recovery procedure (see Section 6.2.5).

Reply Procedure On receiving a JOIN_REQ message, j performs a simple admission control procedure. The main criterion is whether j still has spare capacity to accept a new child. (Note that a specific protocol can provide additional admission control rules.) If j still has spare degree (see Equation 6.1), i will be accepted as its delivery tree child; or if j has a non-tree neighbour (say k), j will accept i by dropping k. Node k will be sent a LINK_DROP message (see Section 6.2.3.3). Otherwise, j will reject i's request. In other words, a mesh link will be dropped in favour of a tree link.

If j can accept i, it adds i into its pending neighbours set, N_j^w. The JOIN_REPLY message from j to i will contain an acceptance flag and j's routing path information. When i receives the reply, the acknowledgement procedure will be used.

On the other hand, if j rejects i, the JOIN_REPLY message will contain a rejection flag and a list of j's tree children. The residual degree information of the children will be included. When i receives this information, it first sorts the list in ascending order based on the residual degree of the nodes, using their total subtree degree as a tie-breaker. It then pushes the sorted list into a stack such that the node with the largest spare degree is at the top. If i needs to perform a rejoin, it will pop the join targets from the stack. This simulates depth-first searching down the delivery tree, which prevents i from randomly selecting the join target. The use of residual degree as a tie-breaker allows nodes to quickly locate a feasible parent. This idea is borrowed from Yang and Fei's proactive recovery scheme [107]. In addition, i records the recent history of nodes that have rejected its requests, so as to prevent redundant requests. The idea of using a stack and request history is borrowed from HMTP [109].

Acknowledgement Procedure Node i will take the sender of the first acceptance reply that it receives as the parent node in the delivery tree. A JOIN_ACK message will be sent to the parent to update its neighbour lists. For other nodes that are also able to accept i as a child, i includes them as the neighbour nodes in the mesh and replies to them with a JOIN_ACK about this intention — these nodes will change their neighbour type accordingly.

6.2.3.2 Overlay Reconfiguration

Request Procedure In this case, i sends a PEERING_REQ message to j indicating its desire to establish a neighbouring relationship. How j is chosen depends on the desired improvement under consideration. For example, to achieve good robustness, i may wish to add a link to a node which provides a path that is disjoint from i's existing routing path. Again, this is determined by the overlay construction protocol.

Reply Procedure When j receives the PEERING_REQ from i, it performs an admission control process to decide if i can be accepted and the type of the neighbouring relationship to be established with i. The main criterion of the admission control algorithm is the degree bound of node j. In addition, the algorithm must also consider the connectivity of the overlay for accepting a new neighbour. Specifically, if by accepting i, j needs to drop an existing neighbour — it is important that the overlay stays connected after the changes.

If i is accepted, it will be added into N_j^w, and j will trigger some changes such as path recomputation and distribution (see Section 6.2.4), if necessary. The admission result will be conveyed back to i using a PEERING_REPLY message.

Acknowledgement Procedure When i receives an acceptance reply from j, it will perform the necessary changes (i.e. update neighbour list and path recomputation) if it is accepted. In addition, it will reply to j with a PEERING_ACK message to confirm the neighbouring relationship to finalise the link establishment. If the two nodes cannot reach a common consensus at this point (i.e. an agreement about the neighbouring relationship to be established), the link will not be added.

6.2.3.3 Tearing Down an Overlay Link

To drop an existing link, a node simply issues a LINK_DROP message to the corresponding neighbour, and purges the neighbour from the corresponding neighbour lists. When the neighbour receives the message, it updates its neighbour lists, and performs a check to see if there is any changes to its path to the root. If the node finds that it is disconnected from the tree, it consults the overlay level maintenance procedure, Section 6.2.5. For changes in the routing path (e.g. change of parent), the node will trigger the routing procedure to distribute its new routing information. Otherwise, nothing has to be done.

6.2.4 Routing Process and Delivery Tree Derivation

This section describes the routing process uses to achieve the loop-free routing tree. The framework uses a path-vector protocol, similar in nature to the Border Gateway Routing Protocol (BGP) [79]. Note that BGP is a much more complicated protocol which includes complex policies that manage routes between different routing domains. In our case, the overlay can be viewed as a single routing domain, thus every node uses the same routing policy.

Figure 6.5 illustrates a simple model of the routing process in an overlay node. Each node maintains, for each of its mesh neighbours, the routing path and cost metric they used to reach the tree root, s in an incoming routing base. Examples of cost metric can be the overlay path's hop count, delay, bandwidth

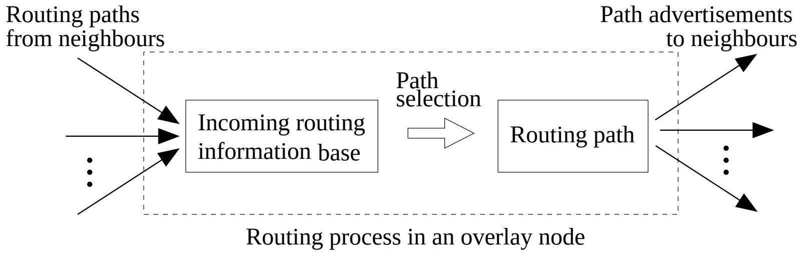


Figure 6.5: Model of routing process in an overlay node

or a combination of them. This is determined based on the requirements of the upper-level applications. A node will place newly received routing information from a neighbour into the incoming routing base. It then executes a path selection algorithm which picks the best path to s using the policy provided by the overlay construction protocol. The next hop of the selected path will become its tree parent. If the computation results in a change in the parent node, the node will confirm its child status with the new parent, and withdraw its child status from its existing parent. The changes will then be advertised to other neighbours. The routing update continues to propagate until the routing path to s has converged. The union of all the paths from every node to s form a reverse tree rooted at s, which becomes the delivery tree (see Figure 6.2).

Routing Update: Simple Case The routing update process takes advantage of the fact that there is only one destination (i.e. the root) in the topology, and makes use of the tree structure to reduce the number of messages exchanged. For simplicity, we first explain how the messages are propagated in the case when there are no changes in the routing paths. The changed case will be described in detail afterwards. Under normal conditions, the root periodically triggers routing information dissemination across the overlay. It sends to each of its neighbours a copy of its routing information using a PATH_ADVERT message. If a routing message is received by a tree child, the child will propagate the message to all its neighbours, except the sender. On the other hand, if the message is received by a mesh neighbour, the message will not be forwarded. In this way, the routing updates travelling across the overlay roughly follow the delivery tree structure. This is illustrated by the sample overlay in Figure 6.6. In the figure, five nodes are connected in a ring topology. The delivery tree is rooted at s, and is shown as the dark arrowed lines. Our tree-based message propagation scheme is as shown in Figure 6.6 (b), where the small arrows depict the direction of the routing messages. Figure 6.6 (c) depicts the working of the conventional routing protocol which requires each neighbouring pair to exchange routing messages. Considering an n-node network, a spanning tree will have $n-1$ links. With our technique, for each tree link (i.e. parent-child link), the message will only flow from the parent to child. This saves $n-1$ copies of routing message.

Routing Update: Detailed Description Now we describe the detailed routing operation. Periodically, the root sends to each of its neighbours its routing information, i.e. a path consists of itself, and routing

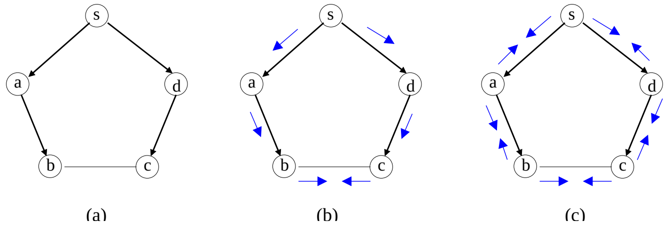


Figure 6.6: Example showing the dissemination of routing messages

cost of zero. The routing information will then be propagated throughout the overlay in the following manner.

When a node, say i, receives a routing message from a neighbour, j, it first tries to validate the received path. That is, if the path already contains i, i will mark the path as invalid. Otherwise, i will update the path and the corresponding cost as follows.

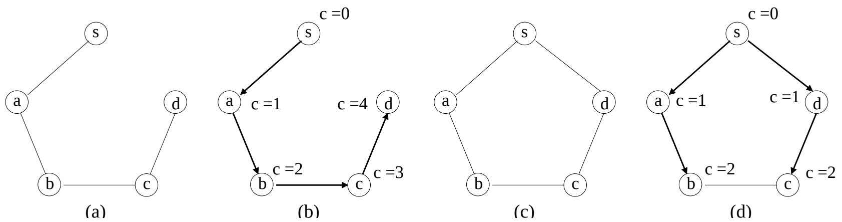
1. Routing path: append its own information to the path, and

2. Routing cost: set the routing cost as the cost it uses to reach the root. For example, if the shortest path policy is used, i's cost will be the summation of the cost between i and j, and the cost of the path from j to the root.

The path will be added into i's incoming routing base. In normal conditions, besides i's parent, only its non-tree neighbours will provide valid paths. Given the set of valid paths in its routing base, i consults a path selection policy to select the best path to s. The path selection policy is based on the optimisation objective considered. For example, a node can use the shortest path first policy to select the least cost path to the root. The path selection may result in the following cases.

1. There is no change to i's existing routing path. In this case, i will propagate its own routing information to its other neighbours, if and only if j is i's current delivery tree parent. This reduces the message overhead as described above.

2. Node i finds a new path via a neighbour, say k. In this case, i will initiate a parent request to k with a PARENT_REQ message. On receiving a PARENT_REQ from a neighbour (in this case, i), k converts i's status to a child node and replies to it with a CHILD_ACK message. (Since i and k are already neighbours, this will not result in degree violation in either nodes.) Once i receives the CHILD_ACK from k, it will replace the existing parent with k and update the existing parent with a PARENT_WITHDRAWAL message. The link between i and its old parent will not be torn down; rather, it is changed into a mesh link. It is easy to see that this process is essentially a parent-switch operation. However, unlike the parent-switch in tree-based proposals, no link will be deleted. After the parent-switch operation, i will replace its routing path with the one via its new parent, k. It then distributes the new path information to all of its neighbours, except k.



(a) (b) (c) (d)

Figure 6.7: Example of routing operation

3. The routing cost and/or the nodes in the path has changed, but not the next hop of the path. This means that while there have been changes to the routing information, the best path is still via the same neighbour. In this case, i will update its routing information, and distribute the information to other neighbours.

4. Node i has lost the path to the root. This may happen when a routing loop is formed due to misinformed overlay reconfiguration operations. Node i will trigger the overlay level maintenance (Section 6.2.5.2) to find a new parent.

The routing information is then disseminated in a similar manner to all nodes. The routing update will also be triggered by any node if the path information changes due to overlay recovery, an overlay reconfiguration process or changes in the path cost.

We explain some operations of the routing process with the sample topology in Figure 6.7. In the figure, the overlay consists of five nodes, with s acting as the root. Assume that all links have unit cost, and the shortest path first policy is used. Panel (a) shows the case that the overlay is connected in a line topology. Obviously, the delivery tree for the overlay also follows the line structure. Panel (b) depicts the delivery tree and the routing cost at each node. Now, assume that d adds a link to s, thus forming the ring topology in panel (c). Node d now learns that it can reach s via a shorter route, i.e. via the direct link to s with a cost of 1. It will request s to be its new parent, and then withdraw its child status from c. It then updates c with its new routing information, i.e. the path to s with a cost of 1. Note that in an implementation, one could piggyback the routing update with the withdrawal message to the old parent, so as to reduce the communication overhead. When c receives the update from d, it will find that the path via d is shorter than its existing path. Thus, it will perform the parent switching operation, and update b with its new information. When b receives c's message, there will not be any changes to its routing path as the path via a is shorter. Thus, the routing update is done. Now, the delivery tree will be as in panel (d).

6.2.5 Overlay Maintenance

Overlay maintenance takes care of the connectivity of the overlay. We devise procedures that exploit the mesh multipath properties to achieve quick recovery. We separate the maintenance tasks into two levels:

1. *Link level*. This level monitors the liveness of an overlay link between two nodes. If a link appears to be disconnected, the overlay level maintenance will be notified to perform any necessary recovery process. Hence, each node only needs to monitor its own overlay links. This reduces the communication overhead.

2. *Overlay level*. This level takes care of the connectivity of the whole overlay, i.e. it repairs partitions in the overlay.

The following two subsections detail their operations.

6.2.5.1 Link Level Maintenance

At this level, two neighbouring nodes exchanges periodic REFRESH messages (heartbeats) to monitor the liveness of the overlay link between them. Each REFRESH message is tagged with a sequence number to detect out-of-order delivery. A node assumes its neighbour has failed if it has not receive a REFRESH message from its neighbour after a predefined time period. The refresh period depends on the estimated distance between the two nodes, as well as the criticality of the link. For example, a tree link should be monitored at a higher frequency than a mesh link. In addition, any data flows between two nodes can be viewed as heartbeats to reduce unnecessary control messages.

In some cases, a node may wish to leave the session prematurely. We require the node to inform each of its neighbours using a LEAVE message. On receiving such a message or on detecting neighbour failure, a node will trigger the overlay level maintenance process to perform any necessary recovery operation. We note that even if a leaving node fails to send out a LEAVE message, the heartbeat mechanism will still detect the departure of the node, although it will take a longer time. Hereafter, we will use the term *depart* to refer a node either failing or leaving, unless specified otherwise.

6.2.5.2 Overlay Level Maintenance

This level maintains the connectivity of the entire overlay. As the delivery tree is a spanning tree interconnects all the members in the overlay, it is sufficient to guarantee connectivity by making sure that the delivery tree is not partitioned. In order to do so, each node closely monitors the status of its parent using link level maintenance.

If the departed neighbour is a child node or a non-tree neighbour, a node only needs to update its neighbour list and the information associated with the neighbour. Otherwise (a node loses its parent), the following restoration procedures will be used. The process may also be triggered by the routing process on detecting a loop (see Section 6.2.4). We note that the restoration process enforces the overlay nodes degree constraints.

The recovery process consists of three stages, which a node will try one by one, until it is successful. Briefly, in stage 1, a node will attempt to attach to one of its non-tree neighbours that provides an alternative path to the root. If this fails, in stage 2 the node will next try the neighbours of its children. If stage 2 fails, the node will need to rejoin the overlay (stage 3). The detailed operation is as follows.

Consider that a non-leaf node, p, departs from the session. We explain the recovery operation that will be performed by one of its children, i. On detecting the departure of the parent node, i first performs the following preparation steps:

1. Remove p from its current routing path, and push the nodes in the path onto a stack in the order such that its grandparent will be at the top while the root will be at the bottom of the stack. The stack (which is called the *rejoin* stack) will be used in Stage 3, in case the first two stages fail.

2. Validate all the paths obtained from its neighbours (including the TCRPs obtained from the children). Specifically, if a path contains p, the path is considered invalid and will not be used in the path selection algorithm.

Stage 1 (using non-TNRPs) At this stage, i will try to reconnect to the tree via one of its non-tree neighbours, if there are any.

From the updated path information, i uses the path selection algorithm to find a neighbour that provides the best alternative path to the root. Say that the node is k, which will become i's potential parent. At this stage, the path selection only considers the non-TNRPs. (The TCRPs will be considered in Stage 2.) Node i then sends a PARENT_REQ message to k, and sets a waiting timer for the reply from k. The message contains the information about p. If a potential parent does not exist, i will proceed to Stage 2. If the waiting timer expires, i will invalidate k's path, and try to recompute and request to another alternative parent.

On receiving a PARENT_REQ message which contains the information of the departed node, p from i, k simply performs a path selection where paths via i and p will be excluded from the computation. If a valid path is available, k returns a CHILD_ACK to i with an acceptance flag as well as its routing path. It also converts i's status from a mesh neighbour to a child. Otherwise, the message will contain a rejection flag. It is possible that k cannot find a path after excluding i and p from the path selection algorithm. For example, k just switches to a new parent that uses one of these node in the path to reach the root, and k's new routing information has yet to propagate to i. In other words, k is in the same subtree as i that has been detached from the main tree. However, k will not trigger the recovery process as long as it believes its parent is still alive. The reason for this is to constrain the recovery process to nodes that actually detect the partition, thus minimising the changes in the overlay.

On receiving the CHILD_ACK message, i first cancels the timer associated with k. If the message indicates that k can accept i as a child node, the recovery process is done. In this case, i updates its routing path and triggers a distribution of PATH_ADVERT messages to all its neighbours, excluding k. The PATH_ADVERT message triggers the recomputation of paths for its neighbours, if necessary (see Section 6.2.4 for details.). If the parent request is rejected, i will try to recompute another alternative parent and retry the above procedures. The process continues until no more alternative parents are available, in which the Stage 2 recovery process will be used.

Consider z and y in Figure 6.2 (b), and assume that they have both detected that their parent, x, has left the overlay. As z has a non-TNRP via t, it can quickly attach to t to repair the tree. On the other hand, y will consider the procedures to be described below.

Stage 2 (using TCRPs) If Stage 1 recovery fails, it means that i does not have a direct neighbour that has an alternative path to the root. Here, i will consider the TCRPs provided by its children.

Specifically, i performs the path selection algorithm using the TCRPs. If there is a valid path via a child, c, this means that the next hop node, say h, that c uses to reach the root has spare capacity to accept a new child. This is because c is a mesh neighbour of h, which can be dropped by h in order to accept a new child if it has reached its degree bound. Hence, i will set h as potential parent and initiate a link establishment procedures to h. If State 2 recovery fails, i will proceed with the procedures in Stage 3.

At this point, we can see an alternative recovery technique. Since i knows that c has an alternative route via h, it can notify c so as to convert h into parent. Then, i can convert c into its parent. The obvious advantage of this approach is that there is no need to setup a new link — the nodes only need to reestablish their relationships. However, it requires two changes to the tree structure (i becomes c's child and c becomes h'c child), compared to only one in our approach (i becomes h's child). More importantly, the alternative approach becomes more complicated if the request from c to h has failed — then, should c carry on the recovery process, or should it notify i so as to continue the recovery? With our approach, the recovery decision is always local to i. Thus, we decided not to use this alternative approach.

Refer to node y in Figure 6.2. We can see that y has a TCRP via its child, w: $\{y, w, u, z, x, q, s\}$. However, as the path passes through x, y regards the path as invalid. Thus, y will have to consider Stage 3 recovery.

Stage 3 (rejoin recovery) Reaching this stage indicates that i could not find an alternative route via its neighbours. Here, i will performs a rejoin process similar to the initial joining procedures. However, instead of joining from an arbitrary node, i will rejoin using the rejoin stack mentioned above. Specifically, i pops a node from the stack and begins the joining process from there. As described previously, the first node is i's old parent's parent, i.e. the grandparent.

Figure 6.8 illustrates three sample cases that trigger the different stages of the recovery process.

6.2.6 Subtree Information Update

The framework provides a SUBTREE_UPDATE message to carry information from a node along the path to the root. Each piece of information is associated with a type and value. Examples of information

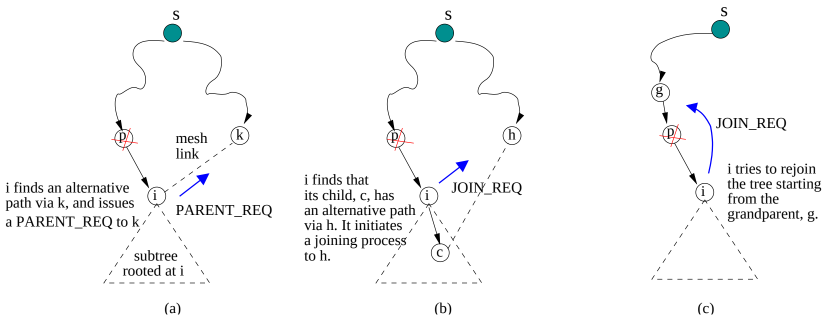


Figure 6.8: Three stages of the recovery process

are the total residual degree in a subtree (Section 6.2.2), and the maximum subtree delay (i.e. delay from a node to its farthest descendant), to be discussed in our case study (Section 6.3).

Take the total subtree residual degree for example. Recall that the residual degree for a node represents the number of new tree links that the node can still accommodate. For a leaf node, i, its total subtree residual degree ($d_{res}(T_i)$)is equal to its residual degree ($d_{res}(i)$). Each non-leaf node maintains the total subtree residual degree for each of its children. Say j is i's parent. Node i reports $d_{res}(T_i)$ to j, which in turn calculates its own $d_{res}(T_j)$ as

$$d_{res}(T_j) = \sum_{\forall c \in C_j} d_{res}(T_c) \tag{6.3}$$

Node j then reports its new $d_{res}(T_j)$ to its parent. The process continues along the path up to the root.

6.2.7 Summary of Control Messages

This section summarises the control messages described previously. Note that we only provide the functionality of the messages, and left the exact format of the messages opens to the actual implementation.

- JOIN_REQ, JOIN_REPLY and JOIN_ACK. These messages are used to establish a new overlay link between two nodes.

- PEERING_REQ, PEERING_REPLY and PEERING_ACK. These messages are used for adding new links into the overlay, during the overlay improvement process.

- LINK_DROP. This message is used by a node to tear down an existing link with its neighbour.

- REFRESH. This is the periodic heartbeat message exchanges between a pair of neighbours to monitor the liveness of the link between them.

- PATH_ADVERT. This message carries the routing path information of a node to its neighbour.

- PARENT_REQ, CHILD_ACK and PARENT_WITHDRAWAL. These messages are used between two neighbours to change their neighbouring relationship. For example, a node, i, sends a PARENT_REQ to a mesh neighbour, say j, to convert j to its delivery tree parent. Node j replies with CHILD_ACK message if it agrees with the change. Node i then sends a PARENT_WITHDRAWAL message to its old parent to convert it to a mesh neighbour.

- LEAVE. This message is sent by a node to its neighbours to inform the intention to leave the session voluntarily.
- SUBTREE_UPDATE. This message carries information (e.g. residual degree) from a node to its upstream ancestors, up to the root.

6.2.8 Discussion

One major concern with a mesh-based approach that uses a conventional routing protocol is its scalability. For example, as seen in Chapter 5, Narada has high protocol overhead. As pointed out in Section 6.1, our framework has several similarities with Narada. In particular, both schemes run the path-vector routing protocol to obtain the delivery trees. However, we note that Narada is designed for many-to-many multicasting. For each node, a source-specific tree will be created from the mesh. Thus, all nodes need to advertise their respective routing table. In an n-node overlay, the size of each routing message will be in the order of $O(n)$. This results in an aggregated overhead of $O(n^2)$ for the whole population. Furthermore, Narada requires each node to maintain the liveness of other nodes for partition detection. On the other hand, in our framework, each node needs only maintains the route to the tree root. By exploiting the tree structure, the routing update overhead can also be reduced. In terms of overlay management, each node only needs to know their respective neighbours.

We recall that some tree-based overlay protocols (e.g. HMTP and TBCP) maintain a root path. The root path is delivered from the root to the members for loop detection and/or prevention. This can be viewed as a variant of the path-vector protocol. As a tree has relatively fewer links compared to a mesh, they will impose less control overhead. In Section 6.3.2.3, we show that our framework incurs a reasonably low overhead, while providing several advantages over the tree-based approach.

6.3 Case Study: Root-diameter and Degree-bounded, Low Cost Tree Problem

In this section, we demonstrate how the mesh-based framework can be used with a case study: a root-diameter- and degree-bounded, low cost overlay tree creation problem.

We first present the problem statement. The overlay network is modelled as an undirected complete graph, $G = (V, E)$, as defined in Section 4.1.1. We consider the edge delay also represents the edge cost. Thus, the *tree cost* is defined as the summation of the delays on all the overlay links in the tree.

The root-diameter of a tree is the maximum shortest path distance from the tree root to any vertex via the tree. An overlay tree is to be formed using nodes in V. A special node, $s \in V$, is designated as the data source as well as the tree root. Let Δ represents the delay bound. Now, the root-diameter and degree-bounded, low cost tree problem can be defined as follows.

> Given an undirected complete graph $G = (V, E)$, a degree bound $d_{max}(v) \in N$ for each vertex $v \in V$ and a delay $c(e) \in Z^+$ for each edge $e \in E$; find a tree, T rooted at s spanning nodes in V of minimum tree cost, subject to the delay constraint (root-diameter $\leq \Delta$) and the degree constraints, $d_{max}(v)$ for all $v \in V$.

Solving this problem using global knowledge is NP-complete [82]. For scalability reason, the problem needs to be solve in a decentralised manner. Thus, the challenge is to approximate the global solution in a decentralised manner using partial information.

The main reason for choosing the problem is because a tree-based solution, called ACDC [54], is available. A short overview of ACDC can be found in Section 2.6.1.2, a more detailed description will be given shortly in Section 6.3.2.1. We adapt several concepts used in ACDC in our solution, and call the resultant protocol *dbMeshTree*. With this, we can perform a comparison between the mesh-based approach and the tree-based approach. In the next subsection, we present the dbMeshTree protocol. It is followed by an evaluation of the trees built by dbMeshTree and ACDC. We also evaluate the robustness of the mesh-based framework compared with two tree restoration schemes.

6.3.1 dbMeshTree Description

We first provide an overview of dbMeshTree. Basically, it extends the framework in the following manner.

- Provide a joining strategy for newcomers.
- Include an overlay improvement strategy, which is adopted from ACDC.
- Define the routing cost as the overlay delay from a node to the destination (i.e. the root), and include a path selection policy to help to obtain the delivery tree.

In the resultant protocol, the overlay is first randomly structured, similar to ACDC. Overlay members then use periodic improvement to try to achieve the delay bound and minimise the tree cost. The protocol is designed to be scalable. In particular, each reconfiguration process only involves the nodes that are engaged in the operation. Only local information at the nodes is used for decision making. In addition, the measurement overhead per node is fixed, i.e. each node is allowed to probe only a small fixed number of other members per improvement round.

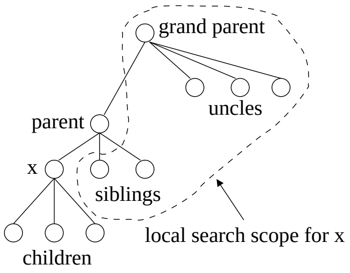


Figure 6.9: Local search scope for node x

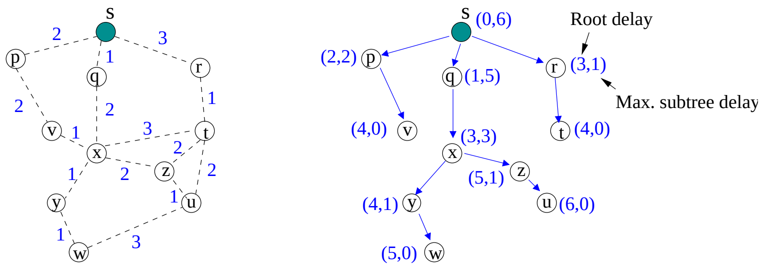
The remainder of this section describes each component of dbMeshTree. The next subsection (6.3.1.1) presents the additional information required by dbMeshTree. Section 6.3.1.2 discusses the overlay construction process. Section 6.3.1.3 presents the core of the protocol, i.e. improving the overlay towards the desired structure. Then, section 6.3.1.4 explains how the data delivery tree is formed.

6.3.1.1 Notation and Node State

In addition to the basic information needed by the framework (Section 6.2.2), a node, i, also maintains the following information.

- Delay bound, Δ. The target delay bound. We assume that it is provided by the application that uses the protocol.
- The unicast delay between i and each of its neighbours. Henceforth, we will use $d(i, j)$ to represent the unicast delay between i and j.
- Routing cost for each valid path, i.e. the delay from i to the root using an overlay path (we also refer to this as the root delay). It is defined as the summation of the delay of the overlay links in the path. We use $\Upsilon_i(j)$ to represent the overlay delay from i to the root via its neighbour j. This information is carried in the PATH_ADVERT message.
- Maximum subtree delay, Λ_i, which represents the maximum delay from i to its farthest descendant. Combining Λ_i with the root delay above enables i to estimate the current tree height for the tree branch that i is in. This information is propagated from a node to its upstream ancestors using the SUBTREE_UPDATE message.
- List of members (other than i's neighbours) in the overlay. The list contains nodes within i's local scope and some randomly selected nodes. This information is used for overlay improvement.

  We define the local scope for a node, x as in Figure 6.9. It includes x's grandparent, siblings and uncles on the delivery tree. Node x can obtain the information in the following manner. Let x's parent be p. Node p will inform x about its parent, siblings (which p learned from its parent) and children (excluding x). On receiving such information, x can update its local region information



(a) Mesh Structure

(b) Delivery Tree Structure

Figure 6.10: Illustration of the notations on: (a) Mesh structure and (b) Delivery tree structure

Notation	Description
Δ	Target delay bound
$\Upsilon_i(j)$	Overlay delay from i to the root via a neighbour, j
Λ_i	Maximum delay for subtree rooted at i
H_i	Tree height contributed by tree branch consists of i

Table 6.2: Additional notations introduced by dbMeshTree

as follows: parent of p is x's grandparent; p's siblings are x's uncles; and p's other children are x's siblings. This information can be piggybacked on the REFRESH messages exchanged between a child-parent pair.

To learn about other overlay nodes, we use the gossip-style node discovery technique as described in Section 2.6.1.1. Basically, a node, say i, maintains a list of known members. Periodically, i randomly picks a node, say j, from the list and sends to j a randomly constructed fixed-size member list (8 nodes in our implementation). When j receives the list, it updates its own member list, and replies to i with a list of members that it knows about. With this, each node will gradually learn about other members of the overlay. Each node is associated with a heartbeat counter to handle changes in the membership (see Section 2.6.1.1).

It is worth pointing out that a node does not keep the distance information to these nodes. The node also does not try to keep an accurate view of the overlay membership.

In Figure 6.10, we show an example overlay mesh structure and the corresponding delivery tree. In panel (a), the value beside each link represents the delay of the link. In panel (b), we show the values of root delay and maximum subtree delay for each node in the form of (root delay, maximum subtree delay). We can see that the tree has a height of 6, i.e. from the root delay of u or the maximum subtree delay of s.

Based on its root delay and the subtree delay, node i can calculate the tree height, H_i, contributed by its tree branch. Specifically,

$$H_i = \Upsilon_i(p_i) + \Lambda_i, \tag{6.4}$$

where p_i is i's parent node. Hence, i will know if it is on a tree branch that is within the delay bound. Assume that the delay bound for the example in Figure 6.10 is 5. There are three possible cases for i:

- $H_i \leq \Delta$: This indicates that all nodes on the path from the root to i, and i's subtree descendants fulfil the delay target. Examples of nodes that fulfil this condition are p, r, t, v, w and y.
- $\Upsilon_i(p_i) \leq \Delta$ and $H_i > \Delta$: This indicates that nodes in the branch from the root up to i is delay bounded, but nodes under i's subtree are not. Nodes q, x and z are in this category.
- $\Upsilon_i(p_i) > \Delta$: The delay bound is violated by i and its subtree descendants. For example, node u.

6.3.1.2 Initial Overlay Construction

As mentioned earlier, dbMeshTree first creates a randomly connected overlay and relies on periodic reconfiguration to achieve the desired structure. This section discusses the growing of the overlay as newcomers join in.

We assume that a well-known Rendezvous Point (RP) is available to bootstrap new members into an existing session (see Section 2.4). A newcomer, i, first obtains the information (the IP address) of the root node, and a small list of overlay members from the RP. Node i then selects a number of members (limited by its degree bound) from the list as joining targets, and initiates the request, reply and acknowledgement sequence to each of the nodes. The reason for sending multiple requests is to quickly locate a feasible parent for i. In addition, the member list may be out-dated due to changes in the membership. The number of initial joining targets is a configurable parameter.

The main reason for using the random joining strategy is to provide a fair comparison with ACDC, which begins with a random tree. However, the random strategy also helps to distribute the joining overhead among the overlay members. This avoids overloading a single node, e.g. the root, especially during the early stage of a session where many nodes are likely to join at about the same time. This also serves as the worst-case scenario where distance information about other nodes is initially unavailable.

6.3.1.3 Overlay Reconfiguration

Once joined to the overlay, each node (except the root) performs a periodic improvement process to try to achieve the target tree structure. The process is adapted from the switch parent operation used in ACDC. In particular, if a node i finds that its descendants all lie within the delay bound (based on H_i), it will try to minimise the tree cost by finding a closer parent; otherwise, it will try to find a parent that provides shorter route to the root so as to minimise the tree height.

Node i periodically selects a non-neighbour node (say j) as potential neighbour, and initiates a peering request sequence to j. To select a potential neighbour, i first forms a fixed-size candidates set. The candidates are chosen from the overlay members that i maintains (see Section 6.3.1.1). The mixed local and random node selection strategy (LR) described in Section 5.1.1.4 is used. In particular, candidates are selected from either the local region or randomly from other known members. The local/random choice is made depending on a probability that favours local selection during the early stages (to improve the overlay quickly) and random selection in later improvement rounds (to explore the search space thoroughly).

Once the candidates are selected, i estimates the distance between itself and these nodes. Node i also obtains the routing information (path and cost) of these nodes during the probing process. By inspecting the obtained paths, i can avoid choosing nodes that have i in their paths (indicating these nodes are descendants of i) as potential parents. Node i selects a potential parent, j, from all the valid candidates using one of the following conditions, based on the given order:

1. $H_i \leq \Delta$: i will select j if $d(i,j) < d(i,p_i)$ and $\Upsilon_i(j) + \Lambda_i \leq \Delta$, i.e. j is closer to i than i's existing parent while the new tree height for i is still fulfilling the delay bound.

2. $H_i > \Delta$ and $\Upsilon_x(p_i) \leq \Delta$: i will select j if $\Upsilon_i(j) \leq \Upsilon_i(p_i)$ and $d(i,j) < d(i,p_i)$, i.e. j provides a shorter or equal distance to the root and it is closer to i than p_i.

3. $\Upsilon_i(p_i) > \Delta$: i will select j with the smallest $\Upsilon_i(j)$.

4. Otherwise, j is randomly chosen.

In the first three cases, i is trying to replace its existing parent with a better one (in terms of cost or delay). Thus i will send the PEERING_REQ message to j indicating a parent request. Node j will accept i as long as it still has spare degree, or if it has a non-tree neighbour which can be dropped to accept i. If j accepts i's request, i will set j as its parent. For its old parent, say k, there are two possibilities: first, if the new link with j does not result in degree violation, i sends a PARENT_WITHDRAWAL message to k to change the link between them to a non-tree link; otherwise, i sends a LINK_DROP message to k to drop the link.

In the fourth case above, since there is no node that is better than i's existing parent, i will try to setup a non-tree link to improve the robustness. However, the request is only sent if i still has spare degree for a new neighbour. In this case, j only accepts i if it still has spare degree.

6.3.1.4 Delivery Tree Derivation

As discussed in Section 6.2.4, the routing process disseminates the path information to all nodes. Each node validates and stores the paths from each of its neighbours in the incoming routing base. Given multiple valid paths to the root, each node selects the best path with a route selection policy. To explain the policy, we first define a *feasible* path as the path that gives a resultant tree height within the delay target, Δ. Thus, for a node i, a feasible path via a neighbour j will fulfil the relation $\Upsilon_i(j) + \Lambda_i \leq \Delta$. The following policy will be used by i to choose the best routing path.

1. First, choose a feasible path. If more than one such path exists, select the one provided by the nearest neighbour.

2. Otherwise (no feasible path), select the path that provides the smallest root distance.

The next hop (i.e. i's neighbour) of the chosen path will becomes i's delivery tree parent. In case of a tie, the IP address of the next hop nodes will be considered: the node with the smallest IP address will be chosen.

The policy essentially prioritises the delay over the tree cost, in an effort to achieve a delay-bounded tree that has low cost. The first condition makes sure that i and its subtree nodes are within the delay bound. If there is more than one feasible path, i selects the one that reduces the tree cost — our second optimisation objective. However, if no feasible path exists (i.e. the second condition), the path that yields the smallest tree height will be used.

6.3.2 Performance Evaluation

We evaluate dbMeshTree from two perspectives: (i) quality of the tree constructed; and (ii) robustness of the protocol. For the overlay tree quality, we compare dbMeshTree against ACDC. In terms of robustness, we investigate how fast the overlay tree can be restored after some nodes depart from the overlay. We compare dbMeshTree against the grandparent and proactive tree recovery techniques studied by Yang and Fei [107].

6.3.2.1 Comparison of Tree Quality

We first describe ACDC. Like dbMeshTree, ACDC initially constructs a randomly connected tree. It then relies on a periodic switch parent operation to improve the tree. The switching conditions in dbMeshTree are borrowed from ACDC. Unlike dbMeshTree, ACDC maintains only a tree structure throughout the session. In addition, the way that it selects the switching candidates is different from dbMeshTree. In dbMeshTree, the candidates are selected from a predefined local region and nodes learned via the gossip-style discovery protocol. On the other hand, ACDC uses a technique called *RanSub* (Section 2.6.1.1) to distribute a set of switching targets (called a probe set) to the nodes on the tree. All the probe sets within each epoch are formed to follow an ordering which makes it impossible for two nodes to simultaneously pick new parents that will introduce a loop in the tree. As a result, ACDC does not keep a root path for loop prevention.

As ACDC and dbMeshTree use different techniques to select the switching candidates, we also considered a tree-only version of dbMeshTree to prevent bias against ACDC. Specifically, this version of dbMeshTree maintains only the delivery tree structure, instead of a mesh. To achieve this, when a node performs a switching operation, it drops the link to its old parent. Hence, the tree structure is preserved. We have found that there is no significant difference between this variant of dbMeshTree and ACDC.

Group Size	dbMeshTree	ACDC
64	95	95
128	97	94
256	91	83
512	71	50

Table 6.3: Success rate (%) for ACDC and dbMeshTree

This shows that the performance differences observed in the following results are not due the difference in the way that the switching candidates are selected. The tree-only version of dbMeshTree is therefore omitted from the following discussion.

In the experiments, all members randomly join the overlay within the first 50 seconds. We consider four group sizes: 64, 128, 256 and 512. The first member is designated as the tree root. Each run last for 3600 seconds, sufficient for the overlay tree to stabilise. For both ACDC and dbMeshTree, the number of switching candidates is set to 5 per improvement round. For dbMeshTree, all newcomers use only one joining target (see Section 6.3.1.2), so the overlay is initially a tree (until extra links are added when nodes begin their improvement process), as in ACDC. For dbMeshTree, the improvement period is 30 seconds. For ACDC, we use a smaller period (i.e. 15 seconds) which is needed to achieve similar performance compared to dbMeshTree. We report results from the 10100-node transit-stub topology (TS10k-0) described in Section 3.2.2.

We first look at how well the protocols can achieve a delay-bounded tree. In the experiments, all members are assigned a maximum out-degree of 10 as in the evaluation of ACDC [53]. To provide a tight delay bound, we use the root-diameter from trees calculated by the Compact Tree (CPT) algorithm (see Chapter 4). Specifically, for a given set of members, we first ran CPT to calculate a low root-diameter tree. The root-diameter of the tree is used as target delay bound for both ACDC and dbMeshTree, running with the same set of members. For each group size, we conduct 100 independent runs.

Table 6.3 depicts the percentage of trials in which the protocols successfully achieve the delay targets. It is clear that the success rate using dbMeshTree is consistently higher than ACDC. We believe this is due to the multiple paths property of the mesh. We explain this with the following example. Consider a node, i which has not achieved the delay target. In a tree, each node maintains a single path (via its parent) to the root. Thus, i's delay performance can be improved if: (i) it switches to a better parent; or (ii) the delay performance of its current path is improved. On the other hand, in a mesh, a node maintains multiple root paths. Hence, besides the above two cases, i may improve its delay when one of its mesh neighbours obtains better delay performance. In other words, maintaining multiple paths at a time gives better chances of improving the overlay.

The result also shows that the distributed solutions cannot always achieve the delay targets. We note that the centralised algorithm (CPT) computes trees using the full knowledge of the network topology, the membership and the degree bound of each member. On the other hand, ACDC and dbMeshTree only

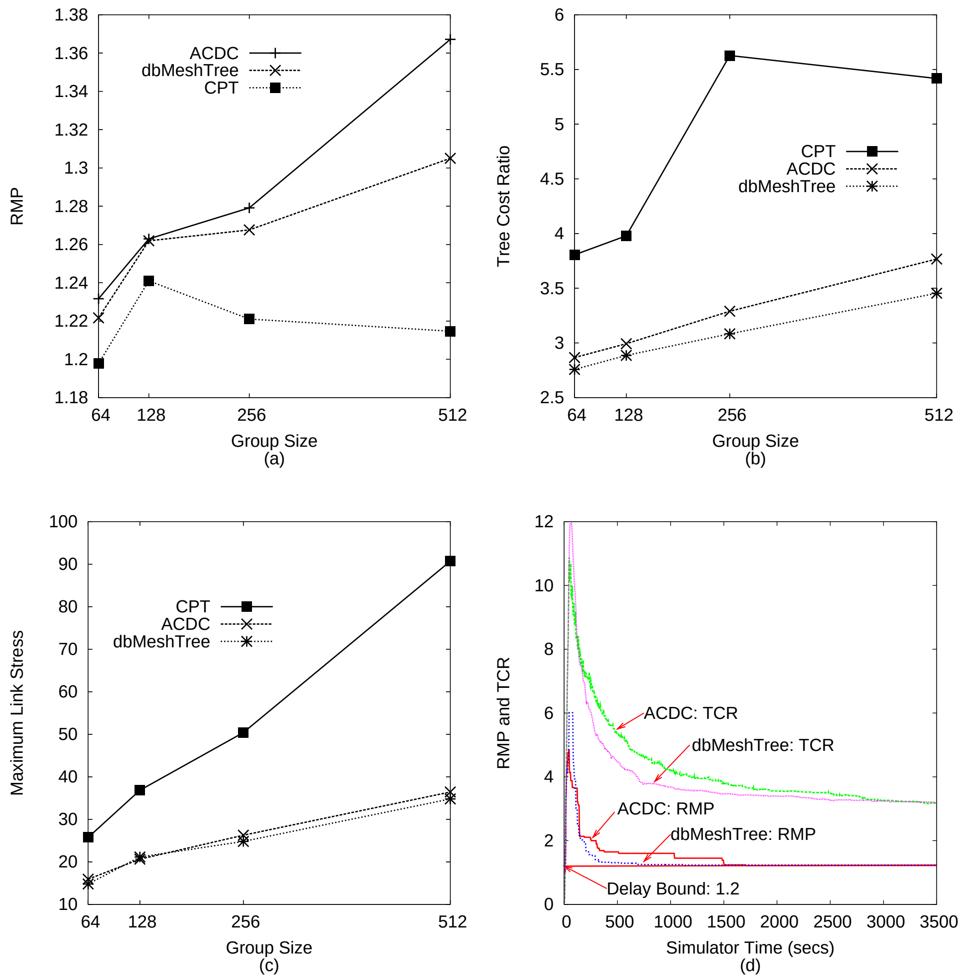


Figure 6.11: Quality of the overlay tree: (a) RMP, (b) Tree cost ratio, (c) Maximum link stress, and (d) Convergence properties

have partial knowledge of this information. In addition, the degree bounds of the nodes also limit the way that the overlay can be reconfigured. These limitations become more prominent as the number of members increases. This is confirmed by the drop in the success rate with the group size, as shown in Table 6.3.

We also examine the actual delay performance, in terms of RMP (see Chapter 3). In Figure 6.11 (a), we plot the RMP averages over experiments (for each group size) in which the protocols fail to achieve the delay targets. From the figure, we can see that the delay bound (depicted by CPT's RMP) is rather tight, i.e. just over 1.2 times of the maximum delay using the direct unicast connections. The RMP for ACDC and dbMeshTree are always less than 1.4, showing that they perform reasonably well.

We examine our second optimisation objective: the tree cost ratio of the trees build (see Chapter 3). Figure 6.11 (b) shows the result. From the figure, we can observe that dbMeshTree and ACDC always yield trees with lower cost than CPT, and dbMeshTree has the best performance. The cost ratio for both distributed protocols increases with the group size. In order to see how well the protocols optimise the tree cost, we also consider trees computed by the centralised heuristic for the degree-bounded, minimum cost tree problem [67] (see Chapter 4). The centralised algorithm indeed gives overlay trees with very low cost, compared to the two protocols. In particular, the cost ratios range from 1.1 to 1.2 for the group sizes. However, the corresponding RMP values range from 2.2 to 3.4.

We also conduct some limited experiments using smaller network topologies (e.g. 600 and 2000 nodes). For these topologies, we found that both ACDC and dbMeshTree can produce trees with low cost while keeping the delay within bound. This suggests that the delay target is easier to realise in certain network conditions, and thus allows the nodes to carry out cost minimisation. This is in line with the observations made by Kostic et al. [54].

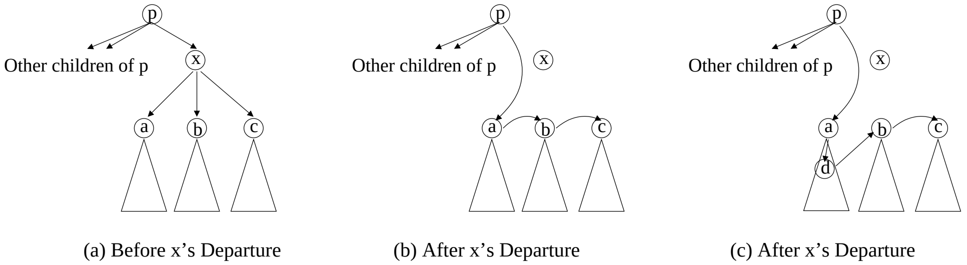
Figure 6.11 (c) depicts the worst-case stress of CPT and the protocols. We can see that dbMeshTree is marginally better than ACDC in most cases, and their stress performance is considerably lower than CPT. The performance advantage of the protocols increases with the group size. The observation is in line with our previous observation that trees with lower cost also have lower stress (Chapter 5).

In Figure 6.11 (d), we show the evolution of RMP and tree cost ratio of dbMeshTree and ACDC for an experiment with 512 members. The delay target for the experiment is set to 1.2 times of the maximum delay achieved using a unicast star overlay. From the figure, we can see that the RMP and cost ratio increase quickly as members are joining the overlay. This is because the initial overlay is randomly connected. In the first 200s, the RMP values of both protocols decrease rapidly to about a value of 2. After that, dbMeshTree continues to improve its delay and achieve the delay target at about 750s. On the other hand, ACDC achieves the delay target after about 1700s. The cost ratio curves show a similar trend. We note that ACDC uses a smaller improvement period (15 seconds compared with 30 seconds for dbMeshTree). This indicates that the dbMeshTree converges much faster than ACDC.

6.3.2.2 Robustness of the Overlay

In this section, we investigate the problem of restoring the degree-bounded overlay tree upon node departures. As mentioned in Section 6.1, it is important that the recovery process does not result in a degree violation in any node. This problem has been considered by Yang and Fei [107], in the context of tree-based protocols. Here we compare our mesh-based failure recovery scheme (using dbMeshTree) with the two approaches that have been shown to perform the best in their paper:

- *Grandparent scheme*. This is a reactive approach where the tree restoration process starts *after* node departures. The grandparent scheme was initially proposed by Deshpande et al. [26], along with several other variants (grandparent-all, root, root-all, as described in Section 6.1). Each of these schemes differ slightly in the way that a recovery node locates its first rejoin target. In [107], Yang and Fei show the grandparent scheme outperforms other variants. In this scheme, the children



(a) Before x's Departure (b) After x's Departure (c) After x's Departure

Figure 6.12: The proactive recovery scheme

of the departed node first try to attach to their grandparent. The grandparent will try to accommodate them as long as it has spare capacity. Otherwise, it will redirect them to its descendants.

The ACDC recovery process is another version of a reactive approach. Instead of using the grandparent as rejoin target, the target is arbitrarily chosen from the rejoining node's probe set. Since both approaches differ only in their rejoin targets, we believe that there will be no significant difference in the recovery speed. Hence, we only consider the grandparent scheme in the experiments.

- *Proactive scheme* [107]. Unlike the reactive approach, the proactive approach plans for the departures *before* they happen. The basic idea of Yang and Fei's scheme is that each non-leaf node precalculates a parent-to-be for each of its children, during the course of the session. Thus, when a non-leaf node actually departs, its children can immediately request to their respective parent-to-be. If a node does not have a parent-to-be[2], it will try to reattach to its grandparent. Unlike the grandparent scheme above, the grandparent node will use the residual degree information of its subtree nodes to redirect any request that it fails to accept.

  In [107], Yang and Fei use the heuristic for the degree-bounded minimum cost tree problem [67] to compute the parent-to-be information. Consider the case in Figure 6.12 (a) where x is performing the computation. Node x will try to find a degree-bounded minimum spanning tree rooted at its parent, p that connects all its children (a, b and c). For example, see Figure 6.12 (b). If a feasible tree cannot be found using only these nodes (due to degree constraints), x's grandchildren will be used (see Figure 6.12 (c)). The distance information used by x is provided by the nodes involved. This scheme has been shown to outperform the grandparent and other schemes proposed in [26].

In the experiments, we model the departure event as a leave event. This is because we are interested in the recovery time — the duration from when a node loses its parent until it finally attaches to a new parent. When a node leaves the group, it will inform all its neighbours about this intention; once it has left the group, the node will not reply to any message send to it.

[2] This can happen under some cases [107]. First, when a node has just joined the tree, and its parent leaves before it finishes the computation. Secondly, it is possible that the parent was too busy with the delivery task and scheduled the computation for a later time.

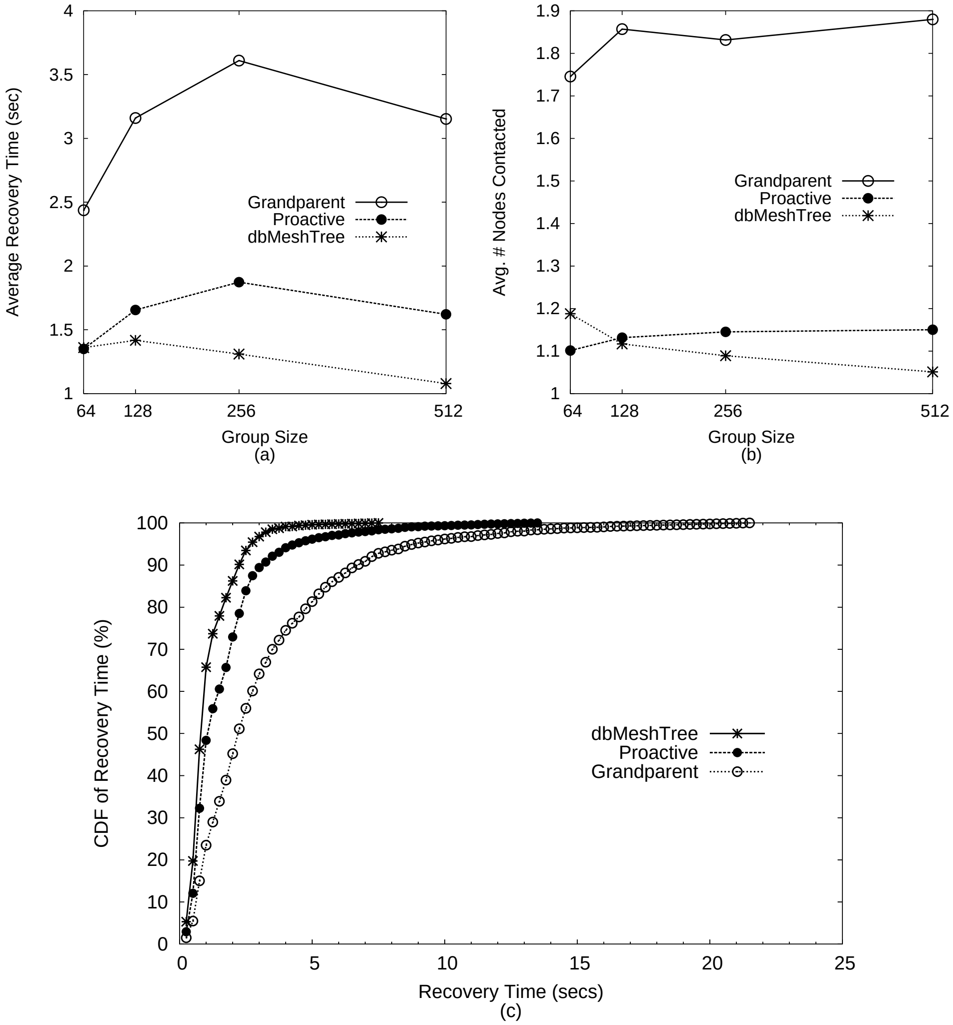


Figure 6.13: Robustness properties of the protocols: (a) Average recovery time; (b) Average number of nodes contacted per affected node, and (c) Cumulative distribution of the recovery time

In each experiment, we first run dbMeshTree to create an overlay tree before generating any departure events. The tree will be used by the grandparent and proactive schemes, so that all schemes begin with the same structure for any departure event. After that, a node is randomly chosen to leave or rejoin (from the nodes that have left) the overlay. The rate of joining and leaving is modelled as a Poisson process as in [107]. We use a rate of $6/minute$, which means that on average, there is a node joining or leaving the overlay tree every 10s. The out-degrees of the nodes are uniformly distributed between 2 and 6. Since a smaller degree results in a taller tree with fewer leaf nodes, the chances that a departing node is a non-leaf node is higher. Each simulation lasts for two simulated hours. We present average results obtained from 20 independent runs.

Figure 6.13 (a) plots the result for the average recovery time, which is the average time for an affected node to find a new parent. It is clear that the grandparent scheme always has the worst performance. This

	dbMeshTree	Proactive Scheme	Grandparent scheme
Stage 1 recovery/attach to parent-to-be	80.36%	89.04%	N/A
Stage 2 recovery	15.93%	N/A	N/A
Stage 3 recovery (rejoin recovery)	3.71%	10.96%	100%

Table 6.4: Breakdown of the types of recovery perform by the different schemes

is expected since when a non-leaf node departs, all of its children will try to attach to the grandparent. Due to the degree constraint, the grandparent can only accommodate some of these nodes. Consequently, many nodes have to search through a number of candidate nodes before attaching to a new parent. The result also shows that in most cases, our mesh-based scheme can actually outperform the proactive scheme. We believe this is because our mesh-based scheme offers more alternatives for a recovery node, i.e. either via the node's direct neighbours (stage 1), or via the node's children' neighbours (stage 2). To confirm this, we examine the breakdown of the types of recovery perform by these three schemes.

We first recall the types of recovery perform by each of the schemes. Our mesh-based recovery scheme consists of three recovery stages: stage 1 where a recovery node tries to attach to a direct neighbour; stage 2 where the node tries to attach to a neighbour of its child; and stage 3 where the node performs a rejoin recovery. Correspondingly, the proactive scheme consists of two stages: the node first tries to attach to its parent-to-be; and then tries a rejoin recovery. The grandparent scheme only performs rejoin recovery. Table 6.4 depicts the breakdown of the recovery types, obtained from experiments with groups of 512 nodes. It is clear that all the recoveries performed by the grandparent scheme are rejoin recoveries. For the proactive scheme, of all the recoveries made, 10.96% are rejoin recoveries, while the rest use parent-to-be recovery. For our mesh-based scheme, only 3.71% of all recoveries are rejoin recoveries, while stage 1 and stage 2 recoveries make up the remainder 96.29%. As explained above, rejoin recovery typically requires a longer recovery time. On the other hand, the proactive scheme's parent-to-be recovery and our stage 1 and stage 2 recovery allow a node to attach quickly to an eligible node. The fact that our scheme uses the fewest rejoin recoveries explains the better recovery time achieved.

Figure 6.13 (b) plots the corresponding average number of nodes contacted by an affected node during the recovery process. From the figures, we can observe that the grandparent recovery scheme always has the worst performance. The fact that both dbMeshTree and the proactive scheme require a much smaller number of contacts for tree restoration compared to the grandparent scheme also partly explains the longer time taken by the grandparent scheme.

Figure 6.13 (c) depicts the cumulative distribution of the recovery time for experiments with 512 nodes. Results for other group sizes show a similar trend. We can see that for dbMeshTree, about 85% of the recoveries are done within 2s and about 99% of recoveries are done within 5s; for the proactive scheme, the percentages are 70% and 95% respectively; and for the grandparent scheme, the percentages are 45% and 80% only. While this result indicates that our mesh-based approach can provide faster

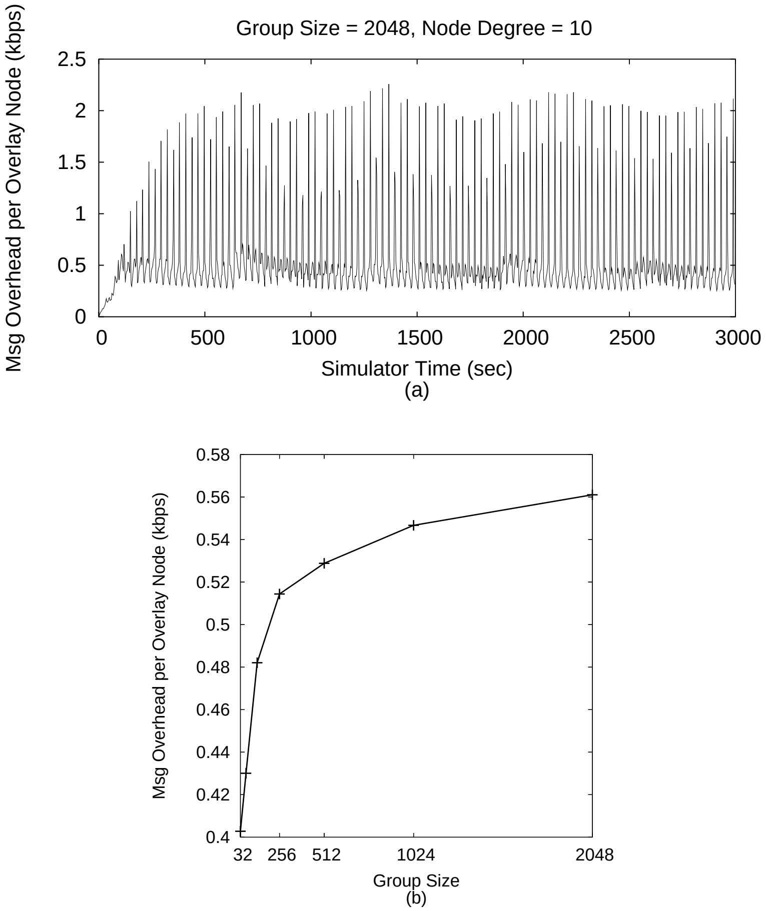


Figure 6.14: Protocol overhead performance

recovery time, it also shows that the worst-case result (e.g. a recovery time more than 5 seconds) may not be acceptable. Examining the traces of the simulated event reveals that the worst results are due to multiple simultaneous departures. In particular, this happens when a node, x, initiates a rejoin operation to a recently departed member. The departed member will not respond to x's request, which will eventually timeout and trigger x's rejoin mechanism. Multiple such cases result in a long recovery time. This suggests that a control plane only solution is not suitable for working environments with high churn rate. However, it can be coupled with a data plane solution such as PRM [10] (see Section 6.1) to improve the transient performance.

6.3.2.3 Protocol Overhead

Figure 6.14 (a) illustrates the protocol overhead property for dbMeshTree, obtained from an experiment with 2048 members. Each member can have up to 10 mesh neighbours. In the figure, we show the control messages sent and received per overlay node (in kbps) during a multicast session. The following settings are used by dbMeshTree.

- Periodic refresh between a neighbouring pair: 5 seconds.
- Periodic routing update: 30 seconds.
- Periodic overlay improvement: 30 seconds. The gossip-style nodes discovery runs every 30 seconds.

We can observe that the normal operating point is around 0.5 kbps, which is reasonably low. The overhead is largely contributed by the refresh messages between the neighbouring nodes. Periodically, the overhead shoots up to values from 1.5 to 2.0 kbps. This represents the control messages used during the routing updates, overlay refinement and gossip-style node discovery. To see how the overhead scales with the group size, we plot the average overhead for group size ranges from 32 to 2048 in Figure 6.14 (b). We can see that the overhead increases very slowly from about 0.4 kbps for 32-node overlay to less than 0.6 kbps for a group size of 2048. This is because, in our proposal, a node only communicates with its neighbours, and a small fixed number of other nodes during its improvement process. Increasing the group size only marginally increases the overlay path length between the nodes. This slightly increases the size of messages that carry path information, and thus increases the overhead slowly across the group sizes. Note that simulations run with 2048 members typically take a considerable amount of time, thus our later experiments only consider group sizes of up to 1024 nodes.

6.4 Chapter Summary

This chapter has described a framework for creating and maintaining a degree-bounded overlay trees. The tree is embedded in a degree-bounded mesh. Our mesh-based approach provides several advantages over a tree-based solution. First, it improves the robustness of the tree. In addition, the mesh is more flexible in achieving a better configuration.

As a case study, we devised a protocol, called dbMeshTree that uses the framework to build a low cost, delay- and degree-bounded overlay tree. The tree creation problem is NP-complete, even if computed centrally with full network and membership information. Our simulation results show that dbMeshTree can provide a higher success rate in achieving the delay bound compared with a tree-based solution called ACDC. In addition, it provides trees with lower cost and stress. We also compare dbMeshTree with two tree-based recovery schemes in terms of recovery speed: dbMeshTree outperforms both schemes. More importantly, the control overhead of the protocol is reasonably small, which allows it to be considered for large-scale applications.

The proposed framework is generic and can be used for other tree creation problems. In the next chapter, it is used in the degree-bounded minimum delay tree problem. In Chapter 8, we adapt the framework for a multiple tree creation problem.

Chapter 7

MeshTree

This chapter considers the problem of constructing a minimum root-diameter degree-bounded overlay tree in a distributed manner. In contrast to the root-diameter- and degree-bounded, low cost tree problem studied in previous chapter, our current problem aims to obtain a degree-bounded tree with the lowest (rather than bounded) root-diameter. This new problem also does not explicitly try to minimise the tree cost. This problem is NP-complete [60]. A low root-diameter tree is useful for single-source applications that require fast data delivery, for example, critical event notification.

This chapter is organised as follows. The next section discusses some issues related to the tree creation problem. The discussion includes two issues — the greedy problem and delay-cost trade-off, which can happen in some distributed degree-bounded, delay-optimised trees solutions. In Section 7.2, we introduce a concept called a MeshTree that addresses the above two issues. The section also analyses the potential of MeshTree by using a simple centralised implementation. Section 7.3 then presents and evaluates a distributed protocol for MeshTree. Finally, Section 7.4 concludes this chapter.

7.1 Building Minimum Root-diameter Degree-bounded Trees

This section discusses the minimum root-diameter degree-bounded tree creation problem. We begin with the problem statement. Then, we look into the differences between this problem and the root-diameter-bounded problem. This is followed by a discussion on several existing efforts. We end this section by analysing two limitations, called the greedy problem and delay-cost trade-off, that can affect some distributed tree building proposals. Our proposed solution, MeshTree, is explicitly devised to solve these two problems.

7.1.1 Problem Formulation

Using the definitions given in Section 6.3, the minimum root-diameter degree-bounded tree problem can be stated as follows.

> Given an undirected complete graph $G = (V, E)$, a degree bound $d_{max}(v) \in N$ for each vertex $v \in V$ and a delay $c(e) \in Z^+$ for each edge $e \in E$; find a tree, T rooted at s spanning nodes in V of minimum root-diameter, subject to the degree constraints, $d_{max}(v)$ for all $v \in V$.

In [60], Malouch et al. proved that the above problem is NP-complete even under a centralised computation model where full topology information is available. For scalability reasons, we are interested in a distributed solution.

7.1.2 Why not the Root-diameter Bounded Solution?

In Chapter 6, we show that it is possible to approximate degree-bounded trees that have bounded root-diameter using the distributed ACDC [54] or our dbMeshTree proposal. In the chapter, we use the root-diameter obtained from the centralised Compact Tree (CPT) algorithm [87] as the target delay bound for both ACDC and dbMeshTree, and show that they could build trees that have root-diameters that are close to those of the CPT. This prompts an interesting question — can a delay-bounded solution (i.e. ACDC or dbMeshTree) be used to minimise the root-diameter, i.e. our current problem?

To use dbMeshTree or ACDC, one would need to provide an appropriate target delay bound. Assume that there is an optimum solution which provides minimum root-diameter degree-bounded trees. The delay target needs to be small enough to avoid over-estimation while big enough to avoid under-estimation of the delay given by the optimum solution. It is clear that over-estimation is undesirable, as it will result in trees with large delay. To understand the impacts of under-estimation, we conduct the following simulation experiments.

We used dbMeshTree to create 100 overlays for a group size of 512, on top of the 10100-node transit-stub topology (TS10k-0 as in Table 3.1). Figures 7.1 (a) and (b) depict the delay performance in terms of RMP, and tree cost ratio respectively. In the experiments, we ran dbMeshTree with delay bounds of 0.5, 0.75 and 1.0 times of the maximum root-diameter obtained using a unicast star overlay (these are represented by the 0.50 $\times$, 0.75 $\times$ and 1.00 $\times$ curves in the figures). This ensures that the delay target is always under-estimated. We also include the results obtained using the centralised CPT. From Figure 7.1, we can observe that dbMeshTree's RMP and tree cost ratio performs much worse than those of CPT. More importantly, dbMeshTree's RMP values vary between 1.4 to 3.0, compared to the much smaller range of 1.2 to 1.4 given by CPT. This indicates that dbMeshTree provides an unpredictable delay performance. The poor performance of dbMeshTree can be explained in terms of how it works (which also applies to ACDC). In the protocol, a node keeps track of the current tree height. If the tree height

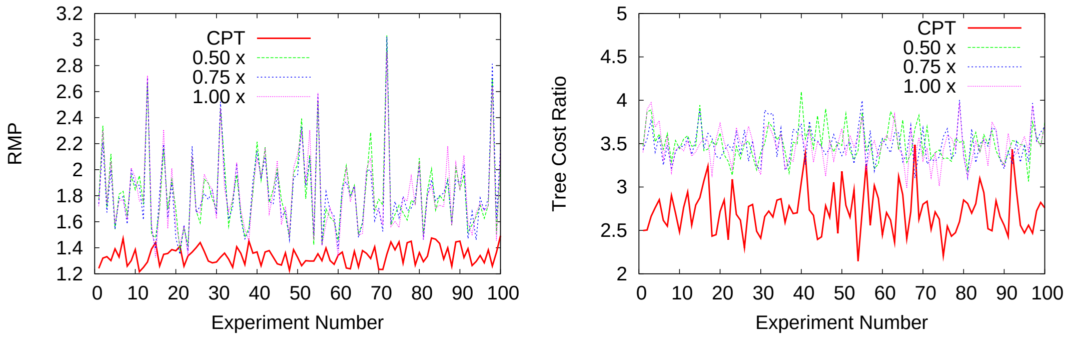


Figure 7.1: Performance of dbMeshTree with under-estimated target delay bound: (a) RMP; and (b) Tree cost ratio

is smaller than the delay target, the node will have more flexible transformation options. On the other hand, if the tree height exceeds the delay target, the node can only switch to a lower delay parent. The smaller the delay target, the fewer the possibilities for changes to the overlay, and thus the performance is poor. Overall, the results show that the root-diameter bounded solution is not suitable for cases with an unknown delay bound.

7.1.3 Prior Work

The problems of creating degree-bounded trees with small or bounded root-diameter have been studied under both centralised and decentralised environments. In [87], Shi et al. propose the centralised CPT algorithm that we use extensively in our performance evaluations. A detailed description of CPT can be found in Chapter 4. In [60], Malouch et al. consider the delay-bounded version of the problem in a mixed end hosts and proxies system. They designed a heuristic solution that is similar in nature to CPT.

A centralised algorithm can be used in conjunction with a centralised tree building protocol such as ALMI [72] or HBM [80] (see Chapter 2). However, a centralised protocol is only suitable if the number of members are very small, e.g. within a few tens of members. This is partly because the centralised algorithm requires the complete distance matrix of the members. The distance information can be obtained using active end-to-end measurement technique (e.g. the `ping` program). Each measurement typically requires two nodes to exchange some probe messages, which consume a certain amount of network bandwidth. For an n-node overlay, $O(n^2)$ measurements will be needed to infer all the distances. This limits the scalability of the solution. In recent years, there have been efforts to develop scalable distance estimation system, such as IDMaps [35] and the coordinate-based global network positioning [68] (see Section 2.7). Indeed, with the maturity of these systems, the distance information can be obtained more easily. However, the overlay membership and underlying network conditions can change over time. Each change may require a recomputation and redistribution of the overlay structure. This can still make the centralised solution impractical.

As discussed in Chapter 2, several decentralised proposals exist for the minimum root-diameter degree-bounded tree problem. For examples, TBCP, HostCast, switch-trees and Banerjee et al.'s scheme. For practical reasons, these protocols limit the knowledge of each overlay node. In particular, each node knows only the distances to a small number of other members. Due to the limited topology knowledge, the initial overlays typically have poor performance. Thus, these protocols improve upon the initial structure with localised incremental overlay reconfiguration. They offer different improvement strategies: switch-trees and HostCast use simple parent switching; TBCP adopts a localised central reconfiguration strategy; while Banerjee et al. opt for a more flexible transformation scheme. These strategies fulfil another requirement for practicality: each operation is local to the nodes involved, i.e. there is no global coordination between the nodes. Our comparison study in Chapter 5 reveals that Banerjee et al.'s scheme outperforms the other proposals in creating low root-diameter trees. However, the tree may have poor average delay from the root to its members. We have seen that our enhanced version of TBCP performs better in terms of average delay. For both measures, switch-trees and HostCast perform the worst.

In this chapter, we propose a distributed mesh-based protocol to create low root-diameter trees. The protocol has several similarities to earlier mesh-based protocols, such as Narada [21] and Gossamer [18]. In particular, the protocol improves the overlay from a randomly connected structure through periodic improvement, and uses the path-vector routing protocol to help obtain the delivery tree. However, the overlay improvement process is significantly different from these protocols. We note that both Narada and Gossamer are designed to construct multiple source-specific trees for many-to-many multicasting. Thus, their optimisation process is tailored to improve all the trees involved. Our protocol, on the other hand, is designed for single tree optimisation. Indeed, it is easy to adapt Narada and Gossamer for single tree data delivery. However, doing so essentially translates the technique to the parent switching strategy. We explain this in the case of Narada. As discussed in Section 2.6.2, a Narada node, say i, periodically estimates the benefit of adding a new overlay neighbour, say j. Link $\langle i, j \rangle$ will be added if doing so offers a substantial performance gain. Node i calculates the gain as the delay improvement to all other members in the overlay that can be observed after link $\langle i, j \rangle$ is added. To optimise for a single tree, i needs only consider the delay gain to the tree root. Thus, a new neighbour that offers a shorter route to the root will be added, and will become i's delivery tree parent. This is similar to the parent switching used in switch-trees and HostCast (except that in switch-trees or HostCast, node i will have to drop its old parent immediately after switching to the new parent, while Narada will drop any unused links periodically).

The parent switching approach is arguably the most basic distributed improvement solution. In the next section, we analyse its behaviour in an effort to understand the limitations in distributed tree building. Our own proposal, MeshTree, is driven by the observed limitations.

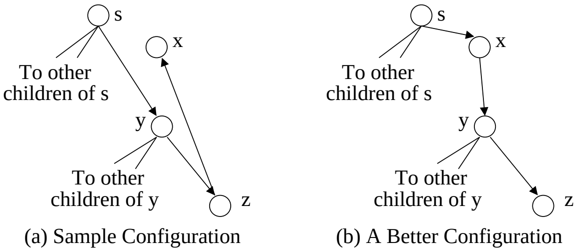


Figure 7.2: The greedy problem due to degree constraint in delay optimisation

7.1.4 The Greedy Problem and Delay-Cost Trade-off

Greedy Problem The optimisation problem considered attempts to minimise the delay from the tree root to its farthest descendant. With parent switching (as in the delay-based switch-trees and HostCast), this can be achieved as every node tries to get as close as possible to the root. This delay-based switching is greedy in nature as a switch will only be made if a better parent is available. In a distributed environment, nodes need to make decisions based on limited topology knowledge. Besides, there can be little coordination between the nodes during the process. Combining the degree constraints and these limitations, the greedy nature of delay-based switching can result in poor overlay structures. We call this the *greedy problem*.

The greedy problem can be explained using the example in Figure 7.2 (a). The figure shows a tree rooted at s. Assume that the distance between any two nodes is proportional to its distance in the drawing. We can see that x, which is topologically close to s, is positioned under z. This results in a long path from s to x, which gives poor delay performance. This can happen as y joins in the overlay before x, and has attached to the tree as s's child. Obviously, s provides the best delay performance. Thus, y will greedily stick to s. The same case also apply to other children of s. When s has reached its degree bound, other nodes such as x will be prevented from attaching to s. This excludes the possibility for a better configuration, such as the example in Figure 7.2 (b).

A simpler version of this problem is the triangle problem discussed in Section 2.2. To recall, it arises when the inefficient structure involves three consecutive nodes along the delivery path. While this problem can easily be solved by using the promotion operation (a child swaps position with its parent, as in Figure 2.4 (c)), there is no such simple solution for the greedy problem discussed above.

Delay-cost Trade-off The configuration in Figure 7.2 (b) suggests that the greedy problem can be avoided if the nodes are connected based on their relative position on the underlying topology, i.e. if nodes are clustered using their proximity measures. Since the aim is to construct a tree, we can view this as the minimum spanning tree problem with the delay between two nodes as the cost function. In Chapter 5, our evaluation results show that distributed solutions such as HMTP can yield trees with reasonably low cost. However, the results also show that a low cost tree often has a higher end-to-end delay, i.e. *delay-cost trade-off*.

To summarise, the discussion points out that delay-based switching can be easily trapped in the greedy problem. This results in poor overlay performance. One potential solution for the greedy problem is to create a low cost tree. Unfortunately, a low cost tree does not necessarily provide low latency, which is the objective of our tree creation problem.

7.2 The MeshTree Concept

This section presents the concept behind MeshTree. In the next subsection (7.2.1), we develop an overlay structure that addresses the above two conflicting problems. As a proof of concept, we devise a simple centralised algorithm to create the structure, and compare it against CPT, in Section 7.2.2. A distributed version of the solution will be given in Section 7.3.

7.2.1 MeshTree Overlay Structure

The propose overlay structure is based on two simple ideas.

1. To solve the greedy problem, the structure must contain a low cost tree which connects nodes that are topologically close together. The tree is called *backbone* tree, and is rooted at the source, s.

2. To improve the delay property of the backbone tree, shortcut links are added on top of the tree.

Essentially, this results in a mesh overlay. To fulfil the degree constraints, the mesh is degree-bounded based on each individual node's capacity limitation. The low delay tree can then be obtained from the mesh as the shortest path tree rooted at s. We will refer to this idea as a MeshTree.

Figure 7.3 illustrates the concept of a MeshTree. In panel (a), we show a low cost tree rooted at s, connecting nodes from a up to f. An overlay link $\langle s, e\rangle$ is then added to form the mesh in panel (b). The figure also depicts the length of the overlay links. Now, it is easy to see that the tree in panel (a) has a root-diameter of 6 units, i.e. the path from s to f. On the other hand, the shortest path tree calculated from the mesh in panel (b) will reduce the root-diameter to just 3 units (see panel (c)).

We will examine the delay property of MeshTree overlay with a centralised implementation in the next section. Here, we look at a more important question: can one build the MeshTree overlay in a distributed manner? Our answer consists of three parts.

1. *Creating the low cost tree*. As shown in Section 5.2.1.1, simple parent switching coupled with a suitable node selection strategy can build trees that have reasonably low cost. For example, HMTP and variants of switch-trees that use the mixed local and random node selection strategy.

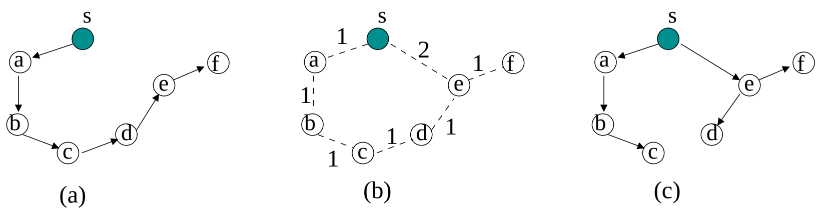


Figure 7.3: Illustration of the MeshTree concept: (a) A low cost tree; (b) Adding a link $\langle s, e\rangle$ to become a Mesh; and (c) The low delay tree

2. *Adding the shortcut links*. First, each overlay node can easily maintain its overlay delay from the root, s (as the sum of the overlay links' delays from s to the node). With this, a simple technique can be developed to add links that improve the delay performance.

3. *Obtaining the low delay tree*. With the mesh structure, the framework presented in Chapter 6 can be used to maintain and obtain the data delivery tree. The shortest path first path selection policy will provide the low delay tree.

At this point, we consider if the compact tree structure can be obtained in a distributed manner. We first review the CPT algorithm. The algorithm maintains a partial tree which grows at each iteration until all nodes are included in the tree. Thus, the nodes can be grouped into two sets: (i) on-tree nodes; and (ii) non-tree nodes. To begin with, the partial tree contains only the root node. At each iteration, the algorithm adds a non-tree node to the partial tree. The selected node is such that adding it will result in the least increase in the partial tree's delay, while preserving the degree constraints. It is clear that the process requires a priori information of all the members, and their distance matrix.

Among the existing distributed efforts, we believe that Banerjee et al.'s scheme [9] can best achieve the compact tree structure. In the scheme, every node maintains the overlay delay to the root and its subtree delay. Based on this information, a node will try to perform a transformation (e.g. a switching, swapping or promotion operation as explained in Section 2.6.1) that improves its delay to the root while not increasing the overall tree height. Each transformation improves the tree, until the tree finally converges. In Section 7.3.6, we will compare our alternative approach to their solution.

7.2.2 Centralised Implementation

As a proof of concept, we modify the centralised mesh generation algorithm (GreedyMesh) introduced in Chapter 4 to create the MeshTree structure. Figure 7.4 shows the algorithm.

The algorithm works as follows. To start with, it generates a degree-bounded minimum spanning tree (line 1). This serves as the low cost backbone tree. The algorithm then calculates the available degree at each node before entering the main loop (line 7 to 24). Within the loop, a new link (i.e. the shortcut link) is added to the current overlay at each iteration. The chosen link is the one that gives the largest delay improvement (in terms of the sums of weighted gains of delay to all other nodes) with respect to

```
Algorithm : Centralised MeshTree Construction
Input: Complete graph $G(V,E)$, Degree constraints $d_{max}(v)$, The root node, $s$
Output: Connected degree-bounded mesh $G'(V,E'), E' \subseteq E$, s.t. $d_{max}(v) \forall v \in V$
CMeshTree($G, d_{max}$)
        genDBMST($G, d_{max}$) := Generate a degree-bounded minimum spanning tree from $G$ subject to $d_{max}$
        getNode($S$) := Get a node from the given set, $S$
        sptAlg($u, G$) := Compute the shortest path distances from $u$ to all other nodes
        $d_{used}(v)$ := Current used degree for vertex $v$
        $d_{spare}(v)$ := Current spare degree for vertex $v$
        $F$ := Set of vertices with spare degree, $d_{spare}(v) > 0$
        $D_u$ := Set of shortest path delays from $u$ to other nodes
        $U_{u,v}$ := Delay gain for $u$ if a link to $v$ is to be added
(1)     $G'(V,E') \leftarrow$ genDBMST($G, d_{max}$)
(2)     foreach $v \in V$
(3)         $d_{spare}(v) \leftarrow d_{max}(v) - d_{used}(v)$
(4)     $F \leftarrow \emptyset$
(5)     foreach $v \in V \wedge d_{spare}(v) > 0$
(6)         $F \leftarrow F \cup \{v\}$
(7)     while $|F| > 1$
(8)         $u \leftarrow$ getNode($F$)
(9)         $D_s \leftarrow$ sptAlg($s, G'$)
(10)        foreach $v \neq u \wedge \langle u,v \rangle \notin E' \wedge v \in F$
(11)            $G'' \leftarrow G'(V, E' \cup \langle u,v \rangle)$
(12)            $D'_u \leftarrow$ sptAlg($s, G''$)
(13)            $g \leftarrow 0$ /* $g$ := gain */
(14)            foreach $w \in V \setminus \{s\}$
(15)                $g \leftarrow g + \frac{D_s(w) - D'_s(w)}{D_s(w)}$
(16)            $U_{u,v} \leftarrow g$
(17)        $b \leftarrow \arg\max \{ U_{u,v} : \forall v \in F \}$ /* $b$ := best selected node */
(18)        $G' \leftarrow (V, E' \cup \langle b,u \rangle)$
(19)        $d_{spare}(u) \leftarrow d_{spare}(u) - 1$
(20)        $d_{spare}(b) \leftarrow d_{spare}(b) - 1$
(21)        if $d_{spare}(u) \equiv 0$
(22)            $F \leftarrow F \setminus \{u\}$
(23)        if $d_{spare}(b) \equiv 0$
(24)            $F \leftarrow F \setminus \{b\}$
```

Figure 7.4: The centralised MeshTree algorithm

the source, s. Note that a link will only be considered if adding it will not result in degree violation. The algorithm terminates when there are no more feasible links. Overall, the algorithm differs from the one given in Chapter 4 only in two aspects. First, it explicitly uses the low cost tree as the initial structure (line 1). Then, when calculating the tree delay, it considers only the delay from the source, s to all other nodes (line 9 and 12). Thus, we refer the readers to Chapter 4 for a detailed analysis of the algorithm.

Given the degree-bounded mesh, a shortest path algorithm, such as Dijkstra's [23], can be used to obtain the low delay tree rooted at s.

We compare the delay property of MeshTree overlays with the low delay trees built by CPT. We ran simulations on nine 1000-node topologies (see Section 3.2.2). The group sizes range from 32 to 256. Figure 7.5 depicts two representative results (from two different topologies) in terms of RMP. We recall that RMP represents the ratio between the maximum overlay delay and the maximum delay using unicast from s to all other nodes. Each data point to be shown is the average of 50 independent runs. We found that both algorithms often yield comparable performance. There are cases where the centralised MeshTree outperforms CPT (e.g. Figure 7.5 (a)); while on other occasions, CPT performs better (e.g.

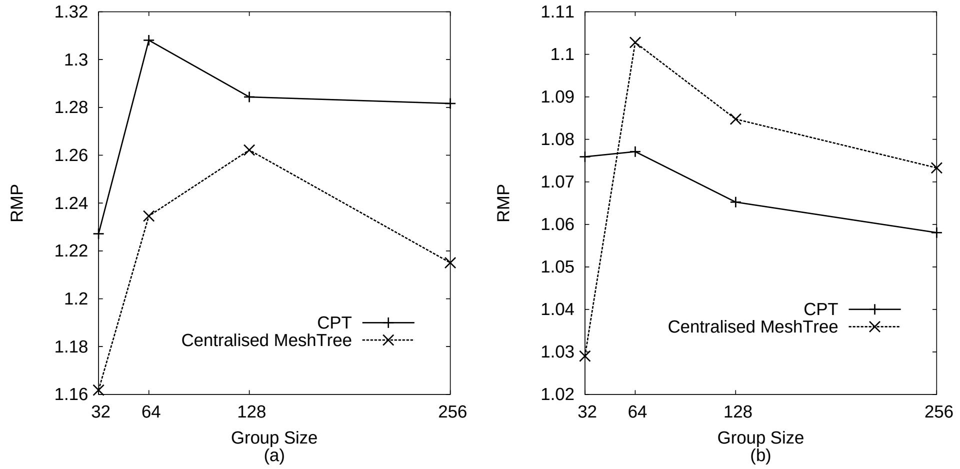


Figure 7.5: Representative delay properties of the centralised algorithms (from two different topologies: (a) TS1k-0; and (b) PL1k-0)

Figure 7.5 (b)). These trends were observed across the various topology models considered (i.e. transit-stub, power-law and random Waxman, see Appendix C). In other words, the results are independent of the underlying topology model used. Thus, we conclude that the MeshTree concept offers an alternative to the centralised CPT algorithm. More importantly, unlike CPT, it can be implemented in a distributed manner.

7.3 Distributed MeshTree Protocol

This section presents a distributed solution for MeshTree. For conciseness, we will refer to the distributed protocol as MeshTree, unless specified otherwise.

The main objective of the MeshTree protocol is to construct the desired overlay structure: a degree-bounded mesh that contains a low cost backbone tree with shortcut links. Then, to obtain the low delay tree out of the mesh for data delivery.

To achieve the overlay structure in a scalable manner, MeshTree uses the incremental improvement strategy typically used by distributed tree building protocols. First, the overlay grows when newcomers join in the session. Newcomers are randomly attached to the overlay. Thus, the initial overlay is unoptimised. Then, every MeshTree node (except the source, s) periodically tries to improve its own local overlay structure. Each improvement process involves adding/deleting links to/from the overlay using only the topology knowledge of the nodes involved.

MeshTree makes use of the mesh-based framework introduced in Chapter 6 to construct and maintain the mesh overlay, and to derive the delivery tree. MeshTree has several similarities with dbMeshTree introduced in the previous chapter (Section 6.3). Thus, some notations and procedures are necessarily common to MeshTree and dbMeshTree. For clarity, the following discussions will reiterate the shared ideas and information. However, to avoid unnecessary repetition, we refer the reader to previous chapter for some detailed description.

The rest of this section describes the protocol. The next subsection introduces the notations and state information used. The four components of the protocol: (i) initial overlay construction; (ii) overlay reconfiguration; (iii) delivery tree derivation; and (iv) overlay maintenance, will be presented in Section 7.3.2, 7.3.3, 7.3.4 and 7.3.5 respectively. Section 7.3.6 evaluates the performance of MeshTree, and Section 7.3.7 discusses an alternative application of MeshTree overlay.

7.3.1 Notation and Node State

MeshTree constructs a degree-constrained overlay mesh. The mesh includes a backbone tree and a delivery tree, both rooted at the data source, s. Since the delivery tree is used for one-to-many delivery, every node (except s) must have a parent node from which it receives the data stream. The degree bound for a node, i is represented by $d_{max}(i)$ which includes the incoming link from the parent (except s) and the out-going links to the set of downstream children.

Two overlay nodes are said to have a neighbouring (or peering) relationship when there is an overlay link between them in the constructed mesh. In general, the set of neighbours for a node, i is represented by N_i^m. The link between i and its neighbours are called mesh links. A mesh link may or may not appear in the backbone and/or delivery tree. For ease of exposition, we group the set of mesh links into three subsets

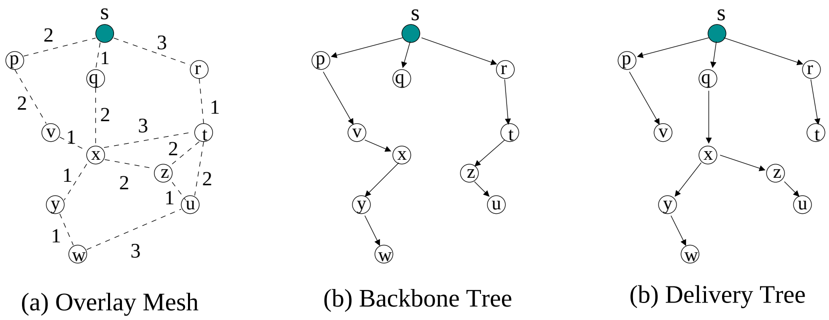
1. *Backbone tree links*. These are links included in the backbone tree. For a node i, the set of backbone tree neighbours is represented by N_i^b. The backbone tree parent and children of i are represented as p_i^b and C_i^b, respectively. Thus, $N_i^b = \{p_i^b\} \cup C_i^b$.

2. *Delivery tree links*. There are links that exist in the delivery tree. We use N_i^d to represent i's delivery tree neighbours. As above, p_i^d and C_i^d refer to i's delivery tree parent and children respectively, and $N_i^d = \{p_i^d\} \cup C_i^d$. Note that a backbone link can also be a delivery tree link, and vice versa.

3. *Non-tree links*. These are links that are neither the backbone nor the delivery tree link. In other words, these are purely mesh links. For i, the set of pure mesh neighbours is represented by N_i^o. Thus, $N_i^o = N_i^m \setminus (N_i^b \cup N_i^d)$.

As we are using the mesh-based framework in Chapter 6, we also use N_i^w, the pending neighbours for i; and $d_{res}(i)$, the residual degree at i.

We use Figure 7.6 to help to explain the notations. Figure 7.6 (a) depicts an example of a MeshTree overlay. In the figure, s is the data source and the rest of the nodes are receivers. The value beside a link represents its delay value. Both backbone tree (Figure 7.6 (b)) and delivery tree (Figure 7.6 (c)) can be



(a) Overlay Mesh (b) Backbone Tree (b) Delivery Tree

Figure 7.6: Example of MeshTree overlay: (a) The mesh; (b) The backbone tree; and (c) The delivery tree

obtained from the mesh. Take node x as an example, we can see that $N_x^m = \{q, t, v, y, z\}$, $N_x^b = \{v, y\}$ with $p_x^b = v$ and $C_x^b = \{y\}$, $N_x^d = \{q, y, z\}$ with $p_x^d = q$ and $C_x^d = \{y, z\}$, and finally $N_t^o = \{t\}$.

In addition to the basic information needed by the framework (e.g. addresses of the tree root and mesh neighbours, residual degree and root path information; see Section 6.2.2 for details), a node, i, also maintains the following information, as in dbMeshTree (see Section 6.3.1.1).

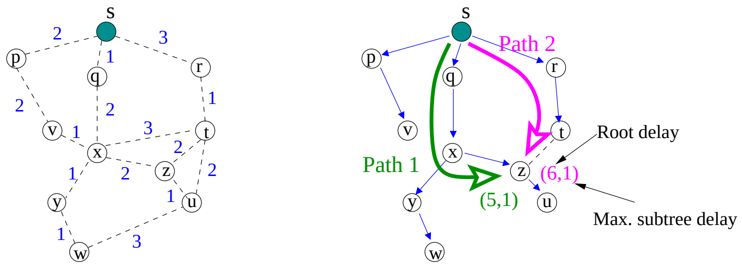
- The unicast delay between i and each of its neighbours. Henceforth, we will use $d(i, j)$ to represent the unicast delay between i and j.
- Routing cost for each valid path, i.e. the delay from i to the root using an overlay path (we also refer to this as the root delay). It is defined as the summation of the delay of the overlay links in the path. We use $\Upsilon_i(j)$ to represent the overlay delay from i to the root using its neighbour j, via the delivery tree.
- Maximum subtree delay, Λ_i, which represents the maximum delay from i to its furthest descendants via the delivery tree. By combining the root delay and subtree delay, i can estimate the tree height, H_i, contributed by its tree branch. Specifically,

$$H_i = \Upsilon_i(p_i^d) + \Lambda_i. \tag{7.1}$$

- Information about other members in the overlay. Basically, every node loosely maintains a list of other members currently in the overlay. The list includes nodes that are within a small overlay distance of i, and other non-neighbour members acquired with a gossip-style node discovery technique (see Section 6.3.1.1 for details). This information is used in the overlay improvement process.

MeshTree introduces the following three pieces of information.

- Backbone tree root path. This is used as a simple loop avoidance (Section 7.3.3) and detection (Section 7.3.5) mechanism for the backbone tree.



(a) Mesh Structure

(b) Two Paths to the Root, s at z

Figure 7.7: An example showing how x can estimate the tree height for its child, z

- The routing cost (or root delay) via i's delivery tree's children' root paths (TCRPs as in Section 6.2.2). We recall that i's TCRPs are the "best" alternative paths provided by i's delivery tree children.

- The maximum subtree delay for each of its delivery tree children.

The root delay for i's TCRPs and delivery children' subtree delay will be used in the overlay reconfiguration process (Section 7.3.3). Briefly, during the process, a node may need to drop a delivery tree child so as to accommodate a new neighbour. The node may select a child that, when dropped, will result in the least increase to the tree height. This requires an estimation of the tree height when the child switches to an alternative parent. An example is shown in Figure 7.7: panel (a) repeats the previously shown overlay mesh, and panel (b) shows that z can reach the root, s via two paths. With the shortest path routing, z will choose path 1 which is 5 units away from s. Thus, x becomes z's delivery tree parent. Node z will inform x about its maximum subtree delay (1 unit) and the alternative path cost via t (6 units) to s. With this information, x can calculate the alternative tree height for z as $H_z = \Upsilon_z(t) + \Lambda_z = 7$. Note that the TCRP is also used in the tree recovery by the mesh-based framework, as described in Section 6.2.5.2.

7.3.2 Initial Overlay Construction

Similar to the dbMeshTree protocol, MeshTree bootstraps newcomers into the overlay randomly. Here we point out the similarities and differences between the two protocols.

In both protocols, a newcomer, say x, first obtains information (the IP address) about the root node, and a small list of overlay members from the well-known Rendezvous Point (RP). Node x then selects a fixed number of members (limited by $d_{max}(x)$) from the list as joining targets and initiates the request, reply and acknowledgement sequence (see Section 6.2.3.1) to each of the nodes.

The difference arises when x's request is rejected by a node, say y. In dbMeshTree, y will provide x with a list of its delivery tree children so as to continue the search. However, in MeshTree, the redirection is based on the backbone tree. By using the backbone tree, it is hoped that x can find a closer node to attach to. When x finally attaches to a node, the node will become x's parent in both backbone and delivery trees.

7.3.3 Overlay Reconfiguration

Once joined in the overlay, all nodes (except the root) perform periodic reconfiguration to improve their own local overlay structure. In the process, a node will try to locate and add a new overlay link that will improve the overlay performance. In order to achieve the desired MeshTree structure, i.e. a low cost backbone augmented with shortcut links, the process will favour a configuration that improves the backbone cost over a configuration that reduces the delivery tree delay. If adding the link will result in degree violation at either of the link's end-points, an existing link will be dropped.

We first provide an overview of the overlay improvement process. Consider that the improvement process is initiated by a node, x. The process consists of the following steps.

- *Part I: Identify a potential neighbour*. Node x first needs to identify a potential neighbour. This can be further divided into the following two steps.

    1. Form a candidates set from the set of (non-neighbour) members that x maintains.
    2. Pick a potential neighbour (say y) from the candidates set.

- *Part II: Establish the overlay link between x and y*. Node x then tries to setup the overlay link to y. This consists of the following request, reply and acknowledgement sequence.

    1. Request procedure: x initiates a peering request to y.
    2. Reply procedure: y processes x's request, and decides if it will accept x as a neighbour. This consists of the following two steps.

        (a) Determine the neighbour type. Node y needs to first determine the neighbouring relationship to be formed with x, e.g. x as y's backbone parent or child, or a mesh neighbour.

        (b) Accept / reject. Now, based on the neighbour type determined above, y will decide if it can accept x as a neighbour.

        The decision made by y is then sent back to x.

    3. Acknowledgement procedure: On receiving y's reply, x finalises the link establishment process and either confirms the link between x and y, or rejects the creation of the link.

The detailed operations are as follows.

Part I: Identify a Potential Neighbour

I.1 Form a Candidates Set To select a potential neighbour, x first forms a fixed-size set of candidates. The size of candidates set is a configurable parameter. The candidates are chosen from the set of overlay members that x maintains (see Section 7.3.1), using the mixed local and random node selection strategy described in Section 5.1.1.4. Once the candidates are selected, x estimates the distance between itself and these nodes. Node x also obtains the routing cost of these nodes during the probing process. These nodes will also indicate if they are descendants of x in the backbone and delivery trees. This can be done by checking if x is included in their respective root paths.

I.2 Pick a Potential Neighbour Using the information gathered, x will pick the node that gives the most reduction in the backbone tree cost (to be explained shortly). If no such node exists, x will pick the node that most improves the delivery tree delay. Otherwise, a node is randomly picked from the list. The last case is to increase the connectivity of the mesh, and thus improves the robustness. In the last case, x will only perform a link request if it still has spare degree. Assume y is picked as the potential neighbour.

The reduction in the backbone cost is estimated by comparing the distance between x and its current backbone tree parent, p_x^b, and the distance between x and y. To consider y as the potential neighbour, the distance between x and y must be smaller than the distance between x and p_x^b, as well as the distances between x and other candidates. As a simple loop avoidance step, x excludes candidates that are descendant of its backbone tree from the estimation. In a similar fashion, x can estimate the improvement in the delivery tree delay using the routing cost provided by the candidates.

Part II: Establish the Overlay Link

Node x needs to initiate a request sequence to y. An overlay link will be created if x and y can reach a common consensus about their neighbouring relationship, i.e. whether the new link is a backbone tree or a mesh link. The role of the link in the delivery tree will be established by the routing process.

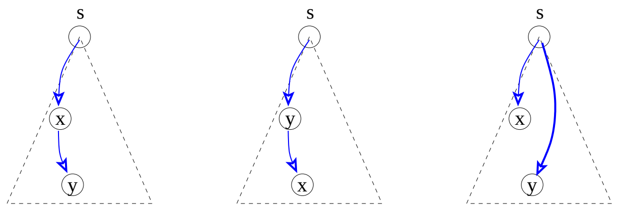
Next we detail the request, reply and acknowledgement sequence that occurs during the link negotiation process.

II.1 Request Procedure First, x sends a peering request message to y indicating its desire to establish a neighbouring relationship. The message contains the following information.

- The measured distance between x and y, $d(x, y)$.
- The overlay paths and costs of x's backbone and delivery trees. The delivery tree cost refers to x's routing cost, i.e. the overlay distance from x to the root, while the backbone tree cost refers to the distance between x and its backbone tree parent.

This information is needed for y's admission control algorithm.

II.2 Reply Procedure When y receives the request message from x, it needs to perform admission control to decide if x can be accepted and the type of neighbouring relationship that can be established with x. The admission control results is one of the following:



(a) x as y's ancestor (b) y as x's ancestor (c) x as y are unrelated

Figure 7.8: Relationship between x and y in the backbone tree

- Reject x's request,
- Accept x as a backbone tree child,
- Accept x as the backbone tree parent, or
- Accept x as a pure mesh neighbour.

The result will be returned to x with a peering reply message. If y accepts x as a neighbour, the reply message will contains y's backbone and delivery trees' information as in the request message. In addition, y will add x into the pending neighbour list, N_y^w, and wait for the acknowledgement from x. N_y^w is used to enforce the degree constraint as in Equation 6.1, Section 6.2.1.

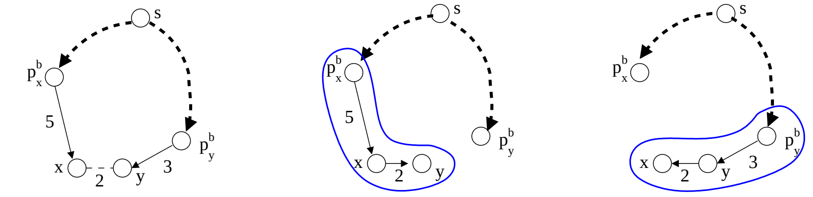
The admission control algorithm consists of two main parts: (i) determine the neighbouring relationship with x; and (ii) decide whether to accept or reject x's request.

II.2.a Determine the Neighbour Type This process determines whether x is treated by y as a backbone tree parent or child, or a mesh neighbour.

The first step of the process is to determine the relationship between x and y on the backbone tree. Figure 7.8 illustrates three possible cases: (a) x is an ancestor of y; (b) y is an ancestor of x; and (c) x and y are unrelated. This provides a clue so that an ancestor node will not try to consider its descendant as a parent. The relationship can be easily inferred from the backbone root paths for x (included in the request message) and for y. The operations for the three cases are as follows.

- *x is y's ancestor (or vice versa).* If x is y's ancestor, it can only become y's parent or mesh neighbour. In this case, the descendant, y will treat the ancestor (x) as a potential backbone parent if the distance between x and y is smaller than the distance between y and p_y^b. (A similar consideration is needed in the case that y is x's ancestor.) Otherwise, y will regard x as a mesh neighbour. This is to reduce the cost of the backbone tree.

- *x and y are unrelated.* If x and y are unrelated, one can freely become parent or child of the other node. In this case, y will try to use the configuration that provides the lowest cost. To do so, y compares the following distances: $d(x, y)$, $d(x, p_x^b)$ (given in the request message), and $d(y, p_y^b)$. If $d(x, y)$ is smaller than one or both of the $d(x, p_x^b)$ and $d(y, p_y^b)$, the node (x or y) that has



(a) Sample Configuration (b) Cost = 7 (c) Cost = 5

Figure 7.9: Example of determining a lower cost configuration

made a larger distance from its parent will be regarded as the child node. This helps to obtain a configuration with a lower overall cost. This can be seen in the example in Figure 7.9. Panel (a) shows the sample configuration and the corresponding distances between the nodes. In the figure, $d(x, y)$ is smaller than $d(x, p_x^b)$ and $d(y, p_y^b)$, and $d(y, p_y^b)$ is smaller than $d(x, p_x^b)$. Thus, the configuration in panel (c) will give a lower backbone tree cost than the one in panel (b). Finally, if $d(x, y)$ is the largest of these distances, y will regard x as a mesh neighbour.

II.2.b Accept / Reject The main decision making is based on the available degree at y. There are three main cases.

1. If $|N_y^m| + |N_y^w|< d_{max}(y)$, then y can immediately accept x as a new neighbour.
2. Else if $|N_y^m| + |N_y^w| - |N_y^o|< d_{max}(y)$, then y can still accept x at the expense of a node chosen from N_y^o.
3. Otherwise, y will execute a pruning procedure to decide if there is any neighbour that can be dropped in flavour of x.

The first condition indicates that y still has spare degree, and thus can accept x as a new neighbour. The second condition indicates that y has some pure mesh neighbours in its pure mesh neighbour list, N_y^o. Hence, x can still be accepted, but a randomly selected node from N_y^o (if there is more than one such node) will be dropped. The reason for including the pending neighbours (N_y^w) in the conditions above is to prevent y from exceeding its degree bound. The third condition indicates that y does not have any spare degree for a new neighbour. It needs to execute a pruning decision to determine if it is beneficial to drop an existing neighbour in order to accept x. This will be discussed in the rest of this section.

Node y considers a neighbour, v, is *prunable* if v fulfils the following criteria:

- v is y's backbone tree parent, and x is replacing v, i.e. y is trying to switch to a closer backbone parent.
- v is y's delivery tree parent, and x can provide a shorter route to the root, i.e. $\Upsilon_y(x) < \Upsilon_y(p_y^d)$. Hence, y is trying to improve its root delay.

```
if $p_y^b \notin N_y^d$
  return $p_y^b$
if $p_y^b \equiv p_y^d \wedge \exists$ alt. path to $s$
  return $p_y^b$
if $p_y^b \in C_y^d \wedge H_{p_y^b} < H_y$
  return $p_y^b$
if $\Upsilon_y(x) < \Upsilon_y(p_y^d)$
  if $p_y^d \notin C_y^b$
    return $p_y^d$
  elsif $p_y^d \in C_y^b \wedge d(y,x) < d(y,p_y^d)$
    return $p_y^d$
if $H_c < H_y : c \in C_y^d \wedge c \notin C_y^b$
    return $c$
if $d(y,x) < d(y,c) : c \in C_y^b \wedge c \notin C_y^d$
    return $c$
if $H_c < H_y \wedge d(y,x) < d(y,c) : c \in C_y^b \wedge c \in C_y^d$
    return $c$
return $nil$
```

Figure 7.10: Conditions used by y to determine a prunable neighbour so as to accept x as the backbone tree parent

- v is one of y's delivery tree children, and v has an alternate path to s. In this case, y needs to make sure that the alternative tree height for v does not exceed that of y (see Section 7.3.1). This is to prevent an increase in the delivery tree height.

- v is one of y's backbone tree children, and the distance between x and y is smaller than the distance between y and v. Here, y is trying to reduce the backbone tree cost.

Based on the above criteria and the new neighbouring relationship to be established, we devise a set of conditions to determine if an existing neighbour can be pruned. Figure 7.10 depicts the pruning conditions used by y in order to accept x as its backbone tree parent. The conditions return a neighbour that can be dropped, if one exists.

- *Accept x as a backbone parent*. To begin with, y examines its current backbone tree parent, p_y^b. Node p_y^b will be returned if it is not also y's delivery tree neighbour. This is because p_y^b is to be replaced by x, and dropping p_y^b will preserve the backbone's tree structure. In the case that p_y^b is also y's delivery tree parent, p_y^d, it will be selected as long as y has an alternative path (via x or other existing neighbours) to the root, s. If p_y^b is also y's delivery child, y needs to make sure that p_y^b has an alternative path which will not increase the delivery tree height. In the fourth case, y begins by examining its current delivery tree parent, p_y^d. First, y makes sure that x provides a shorter route to the root ($\Upsilon_y(x) < \Upsilon_y(p_y^d)$). This is to replace p_y^d with x to improve y's root delay. Now, if p_y^d is not also a backbone tree neighbour, it will be returned. Otherwise, y can only drop p_y^d if $d(x,y)$ is smaller than $d(y,p_y^d)$. This is because including link ($\langle x,y \rangle$) will improve the backbone tree cost. Next, y will examine its delivery tree and backbone tree children. Starting with the delivery tree children, y can drop one which has an alternative path that does not increase the tree height. For the backbone tree children, a child node can be dropped if it is further from y than x. Finally, the last condition in Figure 7.10 considers both cases together.

- *Accept x as a backbone child or a mesh neighbour*. These two cases actually are subset of the above conditions. If x is to be accepted as a backbone child, y will exclude the test for the backbone parent. Otherwise, if x is to be added as a mesh neighbour, y will consider conditions that involves only its delivery tree parent.

In summary, the above procedures help to achieve a low cost backbone tree augmented with shortcut links in the following manner:

- y will drop its current backbone tree parent if and only if x is closer to y, and x will become y new backbone parent. This makes sure that the backbone is in the form of a tree. In addition, if the backbone parent is also a delivery tree parent, it can still be dropped if doing so will not detach y from the delivery tree.
- To improve the tree delay, y will drop its current delivery tree parent if x offers a shorter route. However, this can be done only if it does not increase the backbone tree cost.
- The pruning conditions prioritise the backbone neighbours over the delivery tree neighbours. In particular, the delivery tree parent and children are considered before the backbone children. The backbone children are also not considered when x is to be added as a mesh neighbour.

Referring back to the greedy example in Figure 7.2, the configuration in panel (b) can be achieved if node s can accept x by pruning an existing child. Otherwise, when one of the children finds a closer node as backbone parent, it may detach itself from s and so allow another configuration to happen.

We note that the prunable neighbour is not dropped in this instance, rather, its information is recorded and will only be dropped on receiving a positive acknowledgement from x. To prevent transient disruption to the data delivery, a parent node continues to transmit data to the pruned child for a short time.

II.3 Acknowledgement Procedure When x receives the acceptance reply from y, it will use the same admission control procedure as described in the previous section (using its current information and the neighbour type determined by y) to admit y as a neighbour. This is to avoid any discrepancy due to stale information. If y is accepted, x will update the neighbours list, and trigger the route recomputation. (The routing process will reconfigure the delivery tree links, if necessary.) It also returns an acceptance acknowledgement to y so as to finalise the link addition process. Otherwise, if y is rejected, a rejection acknowledge is returned to y so as to update y's pending neighbour list. On receiving the acknowledgement message from x (accept or reject), y will update the corresponding information accordingly.

7.3.4 Data Delivery

MeshTree uses the routing process in the mesh-based framework to disseminate the (delivery tree) path information to all nodes. Each node validates and stores the paths from each of its neighbours in the incoming routing base. As our aim is to find a low delay tree, we use the shortest path first policy to select the best path. That is, given multiple valid paths to the root, a node, say x, will select the one that provides the smallest overlay delay. The next hop (i.e. x's neighbour) of the chosen path will become x's delivery tree parent, p_x^d. In case of a tie, the IP address of the next hop nodes will be considered; the node with the smallest IP address will be chosen.

7.3.5 Overlay Maintenance

The mesh-based framework is used to manage the connectivity of the delivery tree (see Section 6.2.5 for details). This section discusses the maintenance of the backbone tree links.

Unlike the delivery tree, the backbone structure is only loosely managed. In other words, while we try to maintain the connectivity of the backbone tree, it can occasionally become partitioned. When a partition happens, the protocol will not attempt to repair it immediately.

Under normal conditions, the joining and optimisation procedures will result in a loop-free backbone tree. However, occasionally, a loop may be formed due to multiple simultaneous transformations or a transformation which is done based on stale information (see Section 6.1).

The backbone root path is used for loop detection. In particular, a backbone tree parent node periodically refreshes its children with its backbone root path. The information can be piggybacked in the refresh messages exchanged between the neighbours. If a node, say x, receives a root path that contains its address, this indicates a loop has been formed, and x will break the loop by withdrawing its child status from the parent node (while keeping the overlay link). Thus, x and its backbone tree's descendants are now partitioned from the backbone tree. Node x then replaces the root path with a new list which contains only itself, and quickly updates its descendants with the new path. This can be achieved by including a "push" flag in the update messages to its children, so that the messages will be propagated to its subtree descendants in the backbone tree immediately. It is clear that the process will partition the backbone tree, with the subtree rooted at x being isolated from the main tree. Node x will attempt to repair the partition (by finding a new backbone tree parent) during its periodic improvement process.

7.3.6 Performance Evaluation

This section evaluates the performance of the distributed MeshTree protocol. We focus on the quality of the overlays built. As MeshTree is layered on top of the mesh-based framework discussed in the previous chapter, its other properties (e.g. protocol overhead, and failure recovery) are quite similar to those of dbMeshTree. We thus refer readers to Section 6.3.2.2 and Section 6.3.2.3 for details. In short, MeshTree has reasonably small protocol overhead and is quick to respond to nodes departure.

We compare MeshTree with two other distributed protocols: Banerjee et al.'s scheme (NaBanerjee) and TBCP (TbcpD), which were shown to provide the best RMP and RAP performance respectively, in Chapter 5. We also include the centralised CPT in the experiments. (CPT is used instead of the centralised MeshTree as they produce comparable results, and CPT is more computationally efficient.)

We conduct extensive simulations on a set of transit-stub ($\approx$ 10000 nodes) and power-law (5000 nodes) topologies (see Section 3.2.2). The results presented here are obtained from TS10k-0, a 10100-node transit-stub network. We point out the differences observed from other topologies. The properties of the topologies can be found in Section 3.2.2. For all the results to be presented (except Figures 7.11 (f), 7.13 and 7.14), each data point in the graph represents averages over 50 independent runs. In the experiments, the number of members ranges from 32 to 1024. The first member is selected as the data source. The out-degrees of the overlay nodes are uniformly distributed between 2 and 10. All distributed proposals use a tree improvement period of 30 seconds, and the results are collected after 3600 seconds, which is sufficient for the trees to stabilise.

MeshTree has a number of configurable parameters. First, it allows a newcomer to initiate a number of multiple joining requests when trying to attach to the overlay. Typically, we set this value to one. The impact of this parameter will be discussed later in this section. Secondly, MeshTree uses the mixed local and random node selection technique (see Section 5.1.1.4), which requires an exponent base value, b. We have observed no significant differences with values of b ranging from 1 to 100 (the same trends were also observed in Section 5.2.1.1). The following results were obtained with b set to 20. Finally, at each periodic improvement process, a node selects a potential neighbour from a set of candidates (Section 7.3.3). We use a maximum of 5 candidates in all our experiments.

We first examine the quality of the overlays built. Figure 7.11 (a) and (b) depict the delay performance, in terms of RMP and RAP respectively. The results show that MeshTree always outperforms TBCP and Banerjee et al.'s scheme. For group sizes from 32 to 256, it produces trees with lower RMP and similar RAP compared to the centralised CPT algorithm. For larger group sizes where we expect a centralised approach to be unsuitable, MeshTree still shows reasonably good delay properties.

It is interesting to observe an inconsistent trend in the RMP and RAP curves with the growing group size. We note that both RMP and RAP are ratios between the overlay delay with the unicast delay. The absolute values of the overlay delays observed actually increase with the group sizes. This can be seem in Figure 7.12 which plots the root-diameter of MeshTree and the corresponding unicast delay against the group size.

Figure 7.11 (c), (d) and (e) depict the worst-case and average link stress, and the tree cost ratio performance. We can now observe that CPT produces low delay trees at the expense of high traffic redundancy and network resource usage. The fact that its worst-case stress grows rapidly also suggests that it is not suitable for larger group sizes. MeshTree shows a much lower maximum stress performance, which is close to that of Banerjee et al.'s scheme. In fact, for power-law topologies (see Appendix C), we observed that MeshTree always results in maximum stress values that are smaller than those of TBCP and Banerjee et al.'s scheme. Interestingly, Banerjee et al.'s scheme yields the worst stress and tree cost performance for power-law topologies. Overall, MeshTree generally gives the lowest average link stress and tree cost properties. We conclude that MeshTree achieves good delay properties and results in reasonable traffic redundancy and resource usage.

In the results presented, each data point in the figures represents the average for experiments using a different set of randomly chosen members. (We make sure that all schemes are run with the similar set.) For an incremental improvement scheme, it is desirable that given the same data source, members and degree constraints, a solution should converge to the same point regardless of the joining sequence. We randomly chose a set of members and conducted 50 runs using different joining sequences. Figure 7.11 (f) depicts the result for RMP, which provides an indication on the root-diameter performance. From the figure, we can see that MeshTree consistently produces trees with about the same delay property, compared to Banerjee et al.'s scheme. This also suggests that our scheme can avoid an inefficient structure better than the delay-centric approach.

We also conduct experiments where the out-degree of the nodes follow a truncated binomial distribution with a minimum of 2 and maximum of 10, with different mean values (see Section 3.2.3). While not shown here, the observed trends are similar to the above results.

In Figure 7.13, we show the convergence property of MeshTree for a group size of 1024 members. In the experiment, all members randomly join the overlay within the first 50 seconds. We plot the evolution of the tree cost ratio of the backbone structure, and the RAP and RMP of the delivery tree. From the figure, we can see that the RAP, RMP and cost ratio increase quickly as members are joining the overlay. This is because the initial overlay is randomly connected. In the experiment, we set the periodic improvement period to 30 seconds (as before), for each of the receivers. Hence, the improvement process started soon after all members have joined. We can see that the RAP and RMP values rapidly decrease to a value less than 2 within the first 250 seconds, i.e. less than 10 improvement rounds per node. This indicates that MeshTree can converge very quickly. The result also shows that MeshTree can gradually improve its backbone tree cost, which suggests that the overlay contains a lot of short links between the members. This helps to reduce the delivery tree cost and link stress, as observed previously.

The high delay observed at the early stage is obviously undesirable. As mentioned earlier, a new MeshTree node can send multiple joining requests when trying to attach to the overlay. This parameter, number of initial join targets (NIJT), is configurable. This value affects the structure of the initial overlay. With an NIJT value of one (the setting used in our previous experiments), the initial overlay will have the form of a tree (until the nodes begin to add in extra links). Increasing the value of NIJT allows a newcomer to attach to more than one node, and results in a mesh overlay. As a mesh contains more links than a tree, one would expect it to increase the chances of including "good" links into the overlay which helps to improve the delivery tree. To confirm this assumption, we reran the convergence experiments above, but varying the NIJT for each node from 1 to 5. Figures 7.14 (a) and (b) show the RMP and RAP results. The results show that increasing NIJT (from 1 to 3) indeed improves the initial RMP and RAP values, as expected. Note that the curves for NIJT of 4 and 5 are coincident with the curve of NIJT of 3, which suggests that further increases the value do not bring any improvement. This is because, as the NIJT increases, the available degree for nodes in the overlay will quickly be occupied, and this restricts the number of links that can be connected by nodes joining at a later time. Also, changing the NIJT shows no significant effect on the converged state results (not shown in the region shown in the figures, which concentrate on the early stage).

7.3.7 Further Discussion: An Alternative Usage of MeshTree

MeshTree is designed for applications that require low delay trees. The low delay tree is embedded in a mesh overlay, which also contains a low cost backbone tree. This dual trees structure offers an alternative usage for applications that do not require fast distribution of their normal data, but, occasionally, need to dispatch some critical information to the members quickly. In this case, the backbone tree can be used for normal data delivery, while the low delay tree can be used to quickly deliver the critical information. The benefit of this approach is that frequent data transmission can be done over a low cost tree, which uses less network resources and results in lower packet redundancy.

We recall that in the original MeshTree, the backbone tree is only loosely maintained. Specifically, when a node loses its backbone tree parent, it will only try to reattach to the tree during its next improvement process. For the alternative which uses the backbone tree for data delivery, we require the repair process to start as soon as possible.

7.4 Chapter Summary

This chapter studied the problem of creating degree-constrained low delay overlay multicast trees. We approach the problem by analysing arguably the simplest distributed solution: parent switching. Two important issues were identified: (i) the greedy problem; and (ii) delay-cost trade-off, which can result in an inefficient overlay structure. We then introduced a concept called MeshTree to address the above issues. The main idea of MeshTree is to embed the delivery tree in a degree-bounded mesh containing many short links.

We devised a distributed protocol for MeshTree that exhibits the following desirable properties:

- It constructs overlay trees in a fully distributed manner using only local information maintained at the members. It also has fast convergence and good failure recovery properties.

- The constructed trees are degree-bounded based on each individual node’s capacity limitation. In addition, these trees have small delay from the root.

Our simulation experiments reveal that the MeshTree overlays have delay properties comparable with (and sometimes better than) the centralised compact tree algorithm, and always have lower delay than other decentralised schemes. In addition, MeshTree overlays consume fewer network resources. We point out that MeshTree maintains a mesh structure and thus uses more state information than comparable tree-based proposals. Consequently, its control overhead can be slightly higher than these proposals. However, the mesh-based approach has the advantage of added robustness, as shown in the previous chapter.

In this chapter, the MeshTree concept is applied to create a single delivery tree. In the next chapter, we will adapt the idea for the case of multiple trees. Specifically, we propose the use of multiple shared trees for many-to-many multicasting.

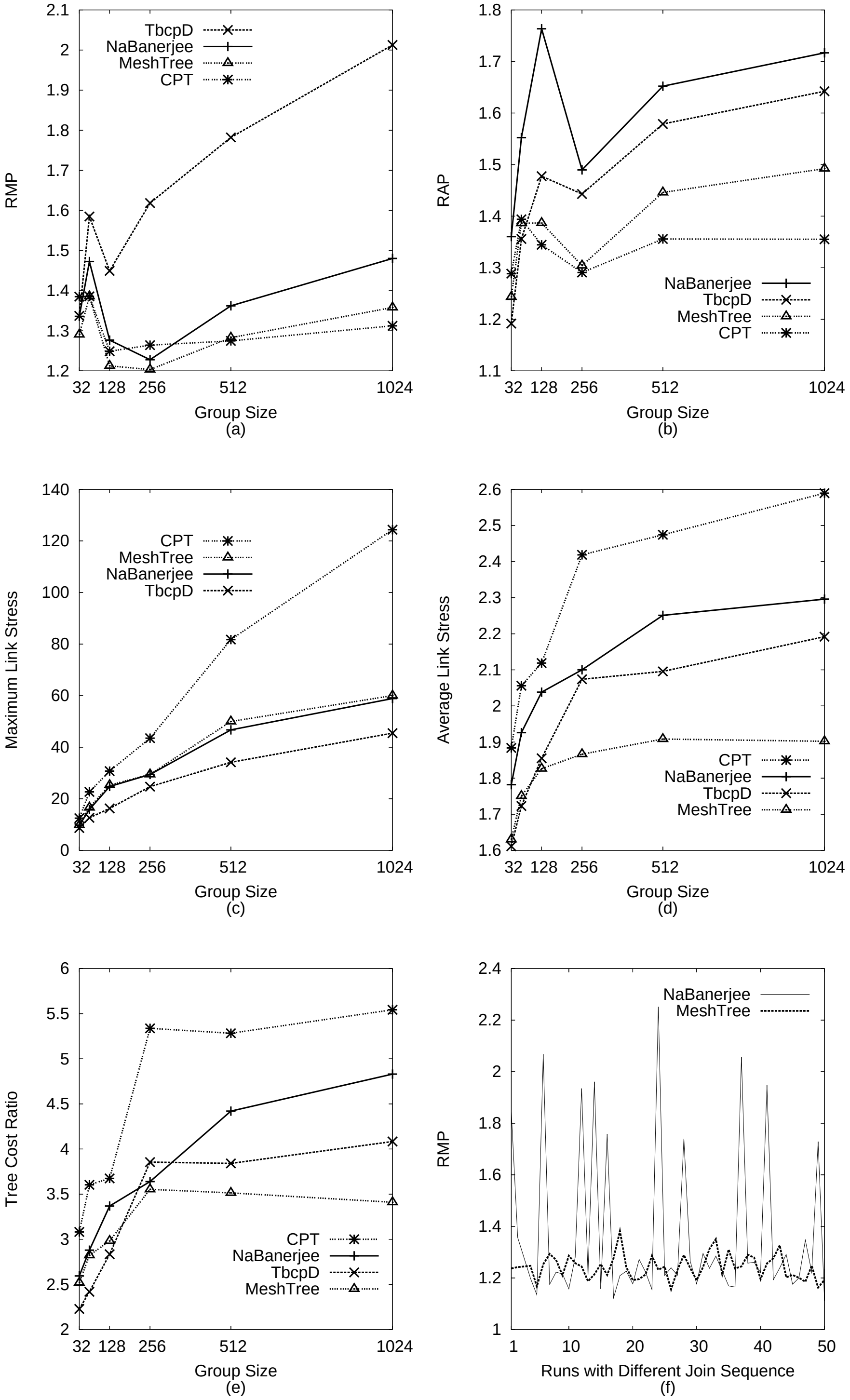


Figure 7.11: Comparison results: (a) RMP performance. (b) RAP performance. (c) Maximum link stress performance. (d) Average link stress performance. (e) Tree cost ratio performance. (f) Impacts of join sequence

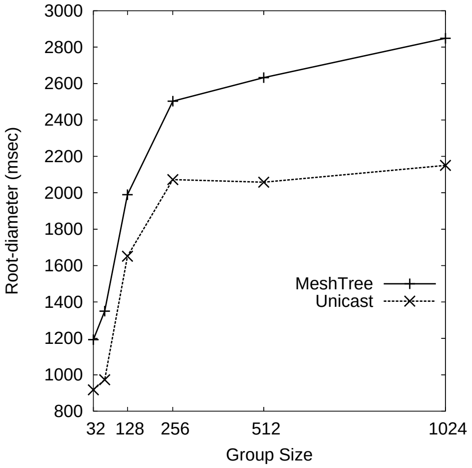


Figure 7.12: Root-diameter of MeshTree and unicast delay

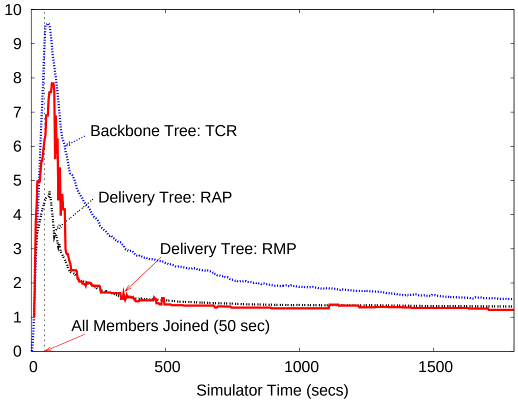


Figure 7.13: MeshTree convergence property

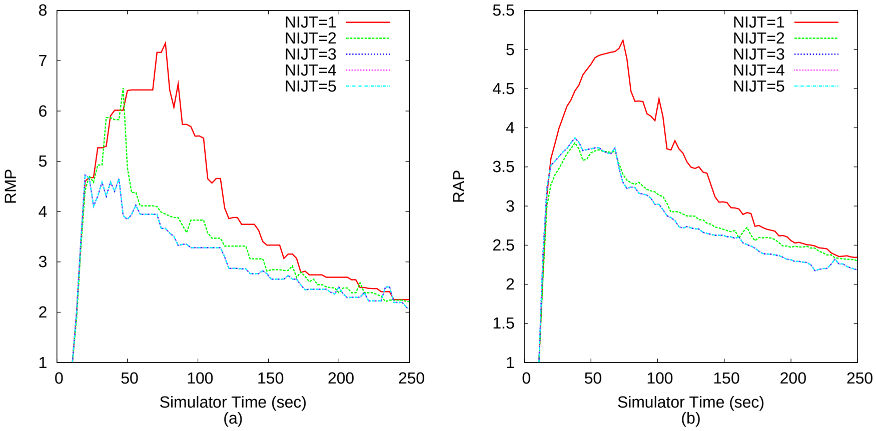


Figure 7.14: Impacts of the number of initial join targets

Chapter 8

A Multiple Shared Trees Approach for Many-to-many Multicasting

Many-to-many multicast offers a service for group communication applications that involve multiple active senders, for example, video conferencing and multi-party network gaming. Conventionally, multicasting in such groups is based on one of the two extreme cases: *single shared tree* or *source-specific trees*. In the shared tree approach, a single tree is used as a shared structure for communications among the members. On the other hand, the source-specific trees approach uses a separate tree for each of the data sources; thus the number of trees can be as high as the group size. Unsurprisingly, these two approaches complement each other in various aspects, e.g. delay, protocol overhead and robustness.

In this chapter, we investigate an intermediate solution to the above two extremes. Specifically, we consider a multiple shared trees solution that uses m trees for a group with n members, where $m > 1$ and $m \ll n$. We are particularly interested in showing that for a reasonably large n (e.g. up to thousands of nodes), a small m can provide good delay while still incurring low protocol overhead.

The rest of this chapter is structured as follows. The next section discusses the background and related research on the data delivery mechanism for many-to-many multicasting. In Section 8.2, we discuss some design issues pertinent to a multiple trees approach in ALM. We then present two versions of our proposal in Section 8.3, along with the performance evaluation. Finally, Section 8.5 summarises the work in this chapter.

8.1 Background

The notions of using a single shared tree and source-specific trees has long being investigated in studies of network layer multicast. Before discussing some such works, we first recall the working principle of the network layer multicast.

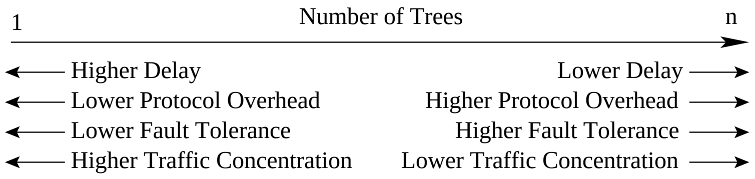


Figure 8.1: Single shared tree vs. multiple source-specific trees

In network layer multicast, every multicast group is identified by a unique multicast address. Each multicast-capable router discovers the multicast membership in its local network using the Internet Group Management Protocol (IGMP). These routers participate in a multicast routing protocol to form the delivery trees for groups for which they have active members. For a multicast group, there can be a single tree that is shared among the members, or a set of separate trees dedicated to each data source. Packets addressed to a group will be delivered over the corresponding tree and reach the members. Note that data sources do not need to be a member of the group to which they are sending.

Early works on network layer multicast, e.g. Deering's Distance Vector Multicast Routing Protocol (DVMRP) [25] and Moy's multicast extension to the Open Shortest Path First (OSPF) routing protocol [65], adopted the source-specific approach. In these protocols, each tree is calculated based on the (network layer) shortest routes between a source and members of a group. Due to this, the trees have low delay. However, this inevitably requires the routers to maintain per-source information for every multicast group for which they have members. This limits the scalability of these protocols.

Consequently, later works such as CBT (Core-based Tree) [5] and PIM-SM (Protocol Independent Multicast - Sparse Mode) [31], began to adopt the group-shared tree approach. As only one tree is maintained for a group, the shared tree approach significantly reduces the routing state overhead. Hence, it is also being used by BGMP [56], the inter-domain multicast routing protocol, to reduce state within the Internet backbone. The shared tree is rooted at a dedicated node, typically called a core.

Despite the scalability advantage, the shared tree has a number of drawbacks relative to the source-specific trees. In [103], Wei and Estrin conduct an extensive set of comparisons between these two approaches. Their simulation results show that the shared tree on average imposes a higher delay between a source and the group members. This is because packets from a source must travel over the shared structure to reach all other members, which in many cases does not involve the shortest path to those members. They also show that the shared tree approach may result in traffic concentration, in which some links in the network are much more heavily utilised than others. The shared tree approach is also less robust as the core node creates a single point of failure problem. Overall, there are trade-offs between the single tree and all sources trees approaches, as shown in Figure 8.1.

Observing these trade-offs, Zappala et al. [108] begin to look at an intermediate design that uses a small number of trees rather than the extreme of using one tree or all source trees. Their objective was to investigate if the multiple trees approach can provide lower end-to-end delay and improved fault

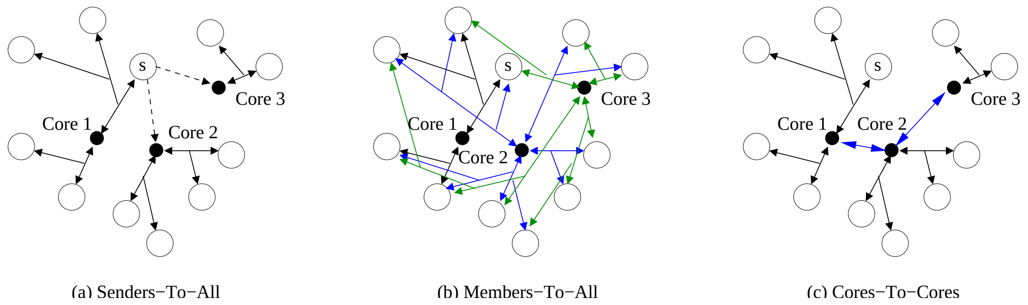


Figure 8.2: Alternative designs for network layer multiple shared trees

tolerance over the single tree approach, at reasonably low state overhead. They examine two multiple trees designs: (i) senders-to-all; and (ii) members-to-all. In both designs, a small number of nodes are selected as cores. Each core is the root of its own bidirectional, shared multicast tree, spanning some or all of the group members.

In the senders-to-all design, members join to only one of the cores, as shown in Figure 8.2 (a). In this figure there are three cores, each of which is the root of a separate, bidirectional tree connecting a subset of the group members. A sender, denoted as s, is also a member and has joined to core 1. To send a message to the group, s transmits 3 copies, one to each core. To receive packets, a member chooses a core and joins the core's shared tree. The members-to-all design, on the other hand, requires the members to join to all of the cores. This is depicted in Figure 8.2 (b). As in the previous example, there are three cores. In this case, all members join all the cores and this results in 3 trees (Note that the tree for core 1 is only partially drawn to avoid cluttering up the figure). To transmit data to the group, the sender, s, can use any of the three trees. Both designs improve the fault tolerance of single tree protocol by reducing the recovery delay when a core fails. This is because in the protocol, a sender or receiver (depending on the variation) can quickly switch to another core, which is already installed. With a single tree protocol, the members have to wait for the routing protocol to re-establish the new tree for the new core.

A third design, in which the senders and members both use only one core, the cores distribute multicast packets among them, is also sketched by Zappala et al. [108]. The data distribution among the cores could be done with a spanning tree, a ring or some other topology. Figure 8.2 (c) depicts an example where the cores are interconnected by a bidirectional shared tree. Following this, we will refer to the design as cores-to-cores. No performance investigation was carried out for this design by Zappala et al..

Zappala et al. carried out a set of experiments to evaluate their multiple trees designs. They first assumed that all cores are randomly selected, and are within a small fraction of the group size. The experiments ran on networks of 50 to 100 nodes, group size ranged from 5 to 50 and the number of cores varied from 1 to 8. Their results confirm that both designs are feasible alternatives to the shared tree and the source-specific trees approaches. In particular, their approaches can have lower delay than a single shared tree and cost comparable to source-specific trees. Between the two designs, the members-to-all variant has better performance in terms of cost and delay at the expense of more router state as all members need to join all of the trees.

Zappala et al. also examine the impacts of the placement of the cores. They view the multiple cores selection in two ways: (i) the minimum d-dominating set problem; and (ii) the minimum k-centre problem (see Section 8.2.2 for details). Both problems are NP-hard [37]. Zappala et al. introduce a centralised heuristic called the dominating set approximation algorithm that can be adapted for both the d-dominating set and k-centre problems. By comparing the d-dominating set and k-centre selections with a random cores selection, they show that a carefully chosen set of cores offers better performance.

Our work applies the multiple trees concept to ALM. While the principal idea — using multiple trees to strike a balance between the one or all approaches — is the same as Zappala et al.'s, the design and working of our protocol is necessarily different from their work due to the different system architecture. In the network layer approach, a multicast tree is formed by the network routers. The tree structure is principally limited by the physical links that interconnect these routers. On the other hand, ALM trees are created by the multicast members as an overlay on top of the physical network. Due to this, the overlay can have a more flexible structure, which is restricted by the degree constraints of the members. More importantly, the overlay is evolving for better performance. In addition, Zappala et al. used a centralised algorithm to select the multiple cores. This, however, is unsuitable for ALM as the overlay is changing over time. Instead, we focus on a simple distributed strategy to identify the cores.

Moving to application layer multicasting, we can generally still classify existing work as either based on the shared tree or source-specific trees concept. Single tree ALM protocols inherit the shared tree idea for scalability reasons. For examples, Yoid [36], HMTP [109], TBCP [62] and switch-trees [43]. Other protocols, e.g. Narada [21] and Gossamer [18], on the other hand, adopt the source-specific trees approach. Both of these protocols use the path-vector routing protocol to help obtain the multicast trees. Narada automatically derives trees for all members, and thus is effective only for small group sizes. Gossamer tries to reduce the routing overhead by deriving trees for active senders. However, this is only useful if the number of senders is relatively small, and the set of senders is consistent throughout the session. If the set of sender is dynamic, or nodes become active (in transmitting data) and passive (act as receiver only) intermittently, the total number of trees involved may still be large. This approach is therefore not suitable for all cases.

As mentioned above, ALM trees are overlays on top of the physical network. This allows more flexibility in organising the members into structures that simplify routing and management, and thus improve the scalability of source-specific trees. For example, NICE [7] creates a multiple-level clusters overlay; LARK [49] organises the members into inter-connected cliques; and the Delaunay triangulation protocol [58] embeds Delaunay triangulation into the overlay. With these structures, source-specific trees can be obtained easily by using only local information maintained at each overlay node. Unfortunately, the structures of these overlays is typically driven by some system-wide parameters. The parameters, rather than the actual degree constraints, will determine the number of overlay links that a node can maintain. For example, NICE uses a configurable parameter, k, to bound the cluster size. For an n-node overlay, the maximum degree in the overlay can be as high as $O(k \log n)$ as discussed in Chapter 2. In other words, to maintain the properties of the overlay, a node may have to serve more nodes than its degree constraint. Our proposed multiple shared trees protocol honours the degree constraints of the overlay nodes, and scales for moderately large groups.

Another related work is Wang et al.'s TMesh [100]. TMesh is an overlay optimisation technique designed for many-to-many applications with a small set of active senders. TMesh begins with a shared tree (created by existing protocols such as HMTP or Yoid); shortcut links are then added to the tree to form a mesh structure. The shortcut addition is initiated by the receivers so as to improve the average delay observed from the active senders in the session. To obtain the routing trees, TMesh uses the path-vector protocol. As in Gossamer, TMesh only creates trees rooted at the active senders. By using simulation, Wang et al. show that for multicast groups with a small fraction of senders (i.e. 10% of groups of up to 1000 nodes), TMesh outperforms Yoid, HMTP and Narada in terms of source-to-members delay. Due to its routing approach, TMesh has low protocol overhead if the number of senders is small, which however increases when more nodes begin to transmit data. In addition, it may need to create a large number of trees if the senders set is dynamic, as in the case with Gossamer. Our multiple trees approach, on the other hand, is designed to achieve low routing overhead regardless of the number of active senders. However, we note that the knowledge of the active senders may help to improve the data delivery trees. The technique for the addition of shortcut links proposed in TMesh can also be used by our proposal.

8.2 Multiple Shared Trees for Application Layer Multicasting

This chapter investigates the potential of the multiple trees approach in the context of ALM. In particular, we focus on a practical distributed solution that can offer a good balance between the one-tree and all-trees approaches. Consistent with our system model (Chapter 3), a potential solution must honour the capacity constraints of the members, i.e. the delivery trees are necessarily degree bounded based on individual node capability. In addition, the protocol overhead must be small so as to scale for a reasonably large multicast group. The rest of this section discusses some issues pertinent to the multiple trees approach in ALM. In the rest of this chapter, we will refer to the tree root as the core, following the convention used in network layer multicast. Also, the number of cores represents the number of trees, and vice versa.

8.2.1 Alternative Designs

Previously we have mentioned the three designs proposed by Zappala et al., namely senders-to-all, members-to-all and cores-to-cores [108]. We discuss the suitability of these designs in the context of ALM.

Senders-to-all This approach partitions the members into several disjoint sets. Members within each set join a tree rooted at one of the cores. A sender delivers data to all the cores, which in turn deliver the data to their members. Owing to this, a sender needs to transmit as many copies of a packet as there are cores. As the number of cores is a system-wide parameter, this inevitably requires all nodes to have the same sending capacity, i.e. fan-out. This is in contrast with our model where nodes may have heterogeneous capacity. Moreover, if the number of cores is reasonably large, e.g. tens of nodes, not all members can fulfil the fan-out requirement. We thus believe this design is not suitable for our study.

Cores-to-cores This design can be viewed as a variant of the senders-to-all, as members join to different trees. Unlike the previous approach, the sender transmits data onto the tree it has joined. The data is then distributed to other cores using a spanning tree, ring or other suitable topology. Hence, we can view this as a two level system — the top level consists of the interconnected core nodes and the lower level consists of other members forming shared trees rooted at each of the cores.

We believe that this approach has a number of drawbacks. First, it is vulnerable to the core failure problem. In particular, when a core fails, its tree is partitioned from other trees. A new core needs to be quickly elected to heal the partition. In this design, each member can only join one tree. As end systems have limited topological knowledge, it is unclear which tree a member should join. This problem becomes more complicated as more cores are involved, and it also applies to the senders-to-all design. Also, as the data is delivered from one tree to other trees via the cores, this approach may result in longer delivery path. Hence, we reject this design in our study.

Members-to-all We believe that this design is more appropriate for ALM. In this case, members join all trees, and the senders choose only one of them to transmit data. In [108], Zappala et al. actually favour the senders-to-all over this design. This is due to the following two reasons.

1. Members-to-all will require the routers to store more state information, as they need to join more trees. The senders-to-all, on the other hand, only requires each router to join one tree.
2. The senders-to-all approach gives the members control over choosing a tree, while in members-to-all, the sender is responsible for choosing a tree. They argued that in this way, senders-to-all allows the members to select the tree that best suit their performance.

In our opinion, these two shortcomings are rather irrelevant in the context of ALM. First, in network layer multicast, the trees are formed by the routers. There is a greater concern over the size of state information as it is kept at the network routers, which could need to support a large number of multicast groups. In contrast, ALM trees are formed directly among the members, which are end systems. Typically, an end system will only participate in one or a small number of multicast sessions. More importantly, storing more information at these systems does not affect the underlying routing infrastructure.

For their second concern, as discussed previously, the end systems have limited topology knowledge and thus face difficulty in deciding the best tree to join to.

If we assume that a subset of members has been chosen as cores, the members-to-all design can be achieved in two ways:

1. The members run a separate tree building protocol (e.g. HMTP, TBCP) to create and maintain each tree.

2. The members create a shared structure (i.e. a mesh) and derive the trees from the mesh.

The first approach is perhaps the most straightforward way to realise the multiple trees design. Also, each tree can be optimised separately. However, this approach requires every node to manage multiple instances of the tree building protocol. A complex coordination procedure may be needed to efficiently manage the information kept for each of the trees. With the second approach, we can easily use the path-vector routing protocol to establish trees rooted at each core node simultaneously. For this reason, we advocate the second approach in this work.

8.2.2 Core Selection

An important factor that affects the performance of the multiple trees approach is the selection of the core nodes. There are two major considerations here: (i) the number of cores; and (ii) the placement of the cores.

As discussed in Section 8.1, the number of trees is chosen to trade-off the performance and scalability. When the number of trees is small, the performance approaches the single shared tree solution. On the other hand, increasing the number of cores will drive the performance towards the source-specific trees approach. However, increasing the number of trees also increases the protocol overhead. It is crucial to find a point that balances these two conflicting metrics.

The core placement is not a new problem. In fact, much research attention has been devoted to the optimal core placement problem for the network layer single shared tree protocols. Selecting an optimal core under a dynamic environment is a non-trivial problem. Finding an optimal placement for *multiple* cores is more difficult. As discussed in Section 8.1, there are two ways to view the problem. The first is to view it as the minimum d-dominating set problem which tries to find the smallest number of cores such that the maximum distance from the cores to the nodes is at most d. The second is to view it as the minimum k-centre problem which locates k cores such that the maximum distance from a node to its nearest core is minimised. These two problems are both NP-hard [37].

In [108], Zappala et al. introduce a centralised heuristic called the dominating set approximation algorithm that can be adapted for both the d-dominating set and k-centre problems. The algorithm requires complete information about the members and their distance matrix. As our proposal works on top of the application layer that has limited topology information, we are interested in a simple distributed technique to do the selection. Specifically, we devise an incremental technique that adds-in cores until a predefined number is reached. The objective of the technique is to spread the cores evenly across the overlay. This aims to minimise the distance from the members to their nearest core. In other words, our technique attempts to approximate the objective of the k-centre problem.

The reason to approximate the k-centre problem is because we would like to have more control over the number of cores, which directly affects the scalability of the solution. In addition, it is more difficult to provide a good estimation of the distance parameter for the d-dominating set problem without full knowledge of the overlay membership and topology. Furthermore, the overlay is changing over the time.

8.3 Application Layer Multiple Shared Trees Protocol

In this section, we describe our solution to multiple shared trees application layer multicasting. The solution is a self-organising protocol which fulfils the requirements (degree-bounded trees and reasonably low protocol overhead) and the design considerations discussed previously. For conciseness, we refer to the protocol as MSTP.

In MSTP, the members self-organise into a connected mesh overlay. Out of all the members, m nodes are chosen as cores. By using the path-vector routing protocol, we derive shortest path trees rooted at these cores. These trees are used as bidirectional shared trees. When a member wishes to multicast data, it selects one of the trees and transmits the data by flooding the selected tree. With this approach, when a core node fails, the overlay is still connected as long as the underlying mesh is connected.

The development of MSTP has gone through two phases. The aim of the first phase was to quickly build a working prototype to investigate the feasibility of the multiple shared trees in ALM. To achieve this, we adapt the Narada protocol to implement the multiple trees design mentioned in Section 8.2. We refer to this version as MSTP-v1. With some limited simulations, we show that MSTP-v1 is a promising alternative for many-to-many multicasting [91]. In the second phase, we include lessons learned from our MeshTree protocol (Chapter 7) to further enhance the protocol. We call the second version MSTP-v2. It is worth pointing out that MSTP-v1 was developed prior to MeshTree, and the MeshTree concept was actually conceived during our attempt to enhance MSTP-v1. We first applied the MeshTree concept to create source-rooted trees for one-to-many multicasting, as in Chapter 7. Then, the idea was applied to MSTP-v2.

We next present the working of MSTP-v1. Section 8.3.2 then describes the improved version of the protocol, MSTP-v2. In Section 8.4, we examine the performance of both versions of MSTP.

8.3.1 MSTP-v1

The first version of the protocol can be viewed as an extension of Narada for creating multiple shared trees. It first creates a randomly connected mesh, which is degree bounded based on the limitation of each individual node. Initially, only one core exists in the mesh. The path-vector routing protocol is used to obtain the tree rooted at the core. The protocol then incrementally adds in new cores up to a predefined number. As all the trees are obtained from the degree bounded mesh, they are also degree bounded.

The protocol consists of five main components: (i) constructing the overlay; (ii) data delivery; (iii) selection of new cores; (iv) improving the overlay; and (v) overlay maintenance. These will be discussed in the following subsections.

8.3.1.1 Constructing the Overlay

MSTP-v1 first creates a randomly connected mesh. As in our previous proposals (dbMeshTree and MeshTree) and in most other ALM protocols, MSTP-v1 assumes that there exists a well-known Rendezvous Point (RP) as the bootstrapping entity for all new members. When a newcomer wishes to join a group, it first contacts the RP to obtain a list of existing members. It then tries to attach to some randomly selected members from the list. The joining procedure consists of the request, reply and acknowledge sequence as in dbMeshTree and MeshTree. A newcomer will regard the first member that provides a successful reply as its parent for all the trees currently in the overlay. Once the node attaches to the overlay, it participates in the maintenance and routing procedures. The routing procedure (explained in the next section) will establish the best routing paths for the node.

In MSTP-v1, the first member that joins in the overlay will be designated as the *lead core* by the RP. It will become the root of the first tree, and is responsible for the creation of additional cores (thus, new trees).

8.3.1.2 Data Delivery

MSTP uses the path-vector routing protocol to help establish the trees rooted at the cores. The routing process at each node consists of an incoming routing information base, path selection policy and the actual routing path, as discussed in Section 6.2.4. Unlike the mesh-based framework in Chapter 6, MSTP needs to obtain more than one tree from the mesh. Thus, its routing process uses the conventional approach to propagate the routing information, rather than the tree-based approach introduced in Section 6.2.4.

In the routing process, every member periodically exchanges its routing table with its mesh neighbours. For a node x, the routing table contains the identities of the cores, and the overlay paths and distances from x to the cores. The routing information is propagated across the overlay in the following manner. When x receives routing information from its neighbour, it first tries to validate the received paths. That is, if a path already contains x, x will mark the path as invalid. Otherwise, x will add the path into the incoming routing base. Then, x will use the shortest path first policy to select the best paths to each of the cores. As before, the next hop of the best path to a core becomes the parent of the tree rooted at the core. If there is a change of route after the recomputation, x will trigger the necessary response. For example, if a parent node is replaced by another node, x will trigger the new parent acknowledgement and old parent withdrawal procedures (see Section 6.2.4) to re-establish the relationship.

Given multiple trees, a source first needs to select one of the trees on which the data will be delivered. There are several possible selection criteria, such as random selection, or selecting a tree whose core is nearest (via the overlay) to the source member. Our cores selection approximates the k-centre problem, which tries to minimise the distance between nodes and their nearest cores. It is thus a natural choice to use nearest core selection. In our performance study, we also investigate the random selection strategy.

Now, assume that a source, x, has chosen a tree rooted at core, y. The chosen tree is used as a shared tree, i.e. a source node transmits packets to all of its tree neighbours (i.e. parent and children); when an on-tree node receives a packet, it will replicate and forward the copies to its other tree neighbours. In order to use the tree, x needs to include the identity of the core, y, in every packet that it transmits. When other nodes receive the packets from x, they can forward the packets on the chosen tree.

8.3.1.3 Core Selection

As discussed in Section 8.2.2, we are interested in a light-weight distributed strategy that approximates the k-centre problem. To achieve this, our protocol begins with a single tree (rooted at the lead core), and periodically increases the number of trees up to a configurable maximum value[1]. The first new core selection begins after a configurable period, e.g. 500 seconds. This is to allow the overlay to converge to a more stable structure.

In the new core selection process, the lead core first tries to identify a number of potential cores, which fulfil a certain requirement. Specifically, the lead core randomly transmits p copies of new core discovery message to its delivery tree children. A timer is set to wait for possible replies from the members. The value p is configurable. The discovery message contains a basic requirement for a potential core: h_m, the minimum overlay hop distance to the nearest core. A potential core must have a hop distance that is larger than h_m. This is an attempt to find a new core that is far away from existing cores, i.e. spreading the cores evenly across the overlay. The reason for specifying the distance in terms of hop count is simplicity. Instead of a predefined h_m value, one may consider a technique which adaptively adjust the value of h_m at each new core discovery period based on the potential cores' response. For example, if no potential core is found, the h_m value can be decremented. In our experiments, we typically set $p = 20$ and $h_m = 3$.

The discovery message also contains a sequence number. All messages transmitted by the lead core within each period have an identical sequence number. It is used to prevent a node replying more than once to the message.

A new core discovery message is forwarded down the lead core's delivery tree in the following

[1] If a group has very small number of members (e.g. 10), it is possible that all members could become cores. In this case, our proposal will work like the source-specific trees approach.

manner. Say a member, x, receives a message from its parent. If x has not seen the message (based on the sequence number), and x is not already a core, and x fulfils the h_m requirement, x will send a reply to the lead core. The reply message contains the distance between x and the core closest (via the overlay) to it. In the case that x is a leaf node which does not fulfil the h_m requirement, x will reply (as potential core) to the lead core if it has not done so. In other cases, x will forward the message to a randomly selected child on the lead core's tree.

Once the lead core receives all the replies (at most p copies) or the timer expires, the lead core will select a new core from the responding nodes. Specifically, it compares the distances of the potential cores to their respective nearest cores. The one that has the largest distance to its nearest core will be selected. In other words, the selection tries to maximise the distance among the cores. The lead core than sends an acknowledgement to the newly elected core, which will begin to advertise itself in the routing update. The routing update will take a while to reach the whole population. Hence, when a node first learns about a new core, the core will not be considered in the tree selection for a few cycles of the routing period (four cycles in our implementation).

8.3.1.4 Overlay Improvement

As the initial overlay is randomly structured, and should adapt to the changes in the membership and network conditions, the overlay needs to be reconfigured from time to time. MSTP-v1 employs a slightly modified version of Narada's overlay improvement technique.

In Narada, every member periodically tries to add a new link or delete an existing overlay link to improve its delay to other members. Consider a node, x. A new link will be added by x if it believes that the link will improve its delay. On the other hand, an existing link may be dropped by x if the link is considered ineffective. To add a new link, x randomly selects a non-neighbour node, and requests a copy of the routing table from the node. Assume that y is selected. Node x will compute the expected delay gain from x to other nodes if a link to y is added, using the utility function shown in Figure 2.10, Chapter 2. The link will be added if the gain exceeds a threshold. To drop an existing link, x estimates the consensus cost of links that it currently maintains. The consensus cost of a link is calculated as the maximum number of times that the link is used in data forwarding for both nodes involved in the link. A link with consensus cost lower than a threshold will be dropped.

To implement the above strategy in MSTP-v1, several modifications are needed. First, unlike Narada, the MSTP-v1 routing process only distributes the identities of cores over the overlay. As a result, every member only knows their immediate neighbours and the available cores. In order to discover other nodes in the overlay, we uses the gossip-style node discovery protocol that has been used in dbMeshTree and MeshTree (see 6.3.1.1 for details). Secondly, we modify the utility function so that the gain of a link is calculated with respect to the core nodes only. The new function is shown in Figure 8.3. The link add threshold is calculated using a function $\frac{m}{\max\{f_x, f_y\}}$ where m is the number of trees, f_x and f_y are the

```
**EvaluateUtility** $(y)$ begin
$utility = 0$
for each core $c$ $(where\ c \neq x)$ begin
    $CL$ = current latency between $x$ and $c$ along mesh
    $NL$ = new latency between $x$ and $c$ along mesh if edge $\langle x, y \rangle$ were added
    if $(NL < CL)$ then begin
        $utility \mathrel{+}= \frac{CL - NL}{CL}$
    end
end
return $utility$
```

Figure 8.3: Modified Narada utility function for MSTP-v1

available degree at the end points of the link, x and y, respectively. The link drop threshold is half of the add threshold.

As discussed in Section 5.1.1.1, Narada does not strictly enforce the degree constraint for the nodes. To avoid degree violation, MSTP-v1 adds in new rules in the link addition. In order to add a new link, a node must first ensure that it has spare degree for the new link. If the node has no more spare degree, it may still include a new link provided an existing link that it rendered ineffective can be dropped. Otherwise, no link can be added.

8.3.1.5 Overlay Maintenance

The objective of overlay maintenance is to ensure the connectivity of every tree in the overlay. We envisage that a small number of trees is sufficient for most cases. In fact, our performance study (Section 8.4) shows that with only 10 trees (for group size ranges from 32 to 1024), MSTP achieves delay as good as the centralised Compact Tree algorithm. We thus apply the mesh-based technique developed in Chapter 6 to manage each of the trees. As shown before, the technique provides fast recovery for degree-bounded trees. The detailed operations of the maintenance procedure (for a tree) can be found in Section 6.2.5. In our current implementation, the control messages for maintaining each of the delivery trees is handled separately. To further reduce the message overhead, one could aggregate the information of the different trees in the control messages.

Another issue to consider is the departure of the cores either because they fail or leave the group voluntarily. When a core departs, its tree is partitioned. A new core needs to be selected quickly to reconnect the tree. Assume that x is the core departing from the group. In order to minimise the disruption to the existing tree rooted at x, a new core is elected from x's children, as this moves the tree root to a nearby node. In addition, the children are the first to notice the departure of x (via the periodic heartbeat or explicit leave message from x).

As a preemptive measure, all existing cores elect their respective backup core from their children set. Note that only a non-core node is eligible to become a backup core. The selection can be done randomly, or based on the distance between the core and the children, or available degree of the children, etc. The identity of the backup core is made known to all children. When core x actually departs, its backup core (say y) will take over x's role: if x is the lead core, y will also become the lead core. The new core y will start advertising itself in the routing update, and x's other children will need to join to y's tree. To achieve fast restoration, we could use the proactive tree restoration technique by Yang and Fei [107] (a short description can be found in Section 6.3.2.2). In particular, the old core, x, could precompute parent-to-be using y as tree root, for all the other children.

8.3.2 MSTP-v2

Our early investigation on MSTP-v1 [91] shows that it is a promising strategy for many-to-many application layer multicasting. In the experiments, we consider MSTP-v1 with 10 trees, and show that it can achieve delay performance that is better than the single tree HMTP, and is comparable to NICE which uses source-specific trees, for groups up to 250 members, in a 2050 nodes transit-stub network. MSTP-v1 achieves the performance with reasonably low protocol overhead compared to Narada, which creates trees for all members.

We have later compared MSTP-v1 with other delay-optimised single tree protocols, i.e. TBCP (TbcpD) [62] and Banerjee et al.'s scheme (NaBanerjee) [9]. Surprisingly, we found that even using a larger number of trees (i.e. 10), MSTP-v1 performs worse than Banerjee et al.'s scheme (which uses a single shared tree) in end-to-end delay. Figure 8.4[2] illustrates the delay comparison in terms of RMP and RAP, obtained from experiments with topologies of 1000 nodes. In the experiments, all members are data sources, and are randomly assigned degrees in the range 2 to 10. For NICE, the lowest cluster size, k is equal to 3. From the results, we can see that MSTP-v1 gives good RMP for small group sizes (32 and 64) compared to other protocols. This is because the number of cores to group size ratio is relatively high in such group sizes. However, as the group size increases, its RMP also increases and performs less well than Banerjee et al.'s scheme and TBCP. In terms of RAP, MSTP-v1 consistently outperforms other protocols, except NICE which does not observe the individual node's degree limitation (see Section 5.2.7).

The better average delay (RAP) performance is expected as MSTP uses more trees than other protocols (except NICE). As the senders choose the tree with the nearest core, the sending quality is approaching the shortest path trees. Note that the shortest path trees are based on the overlay. The poor maximum end-to-end delay (RMP) performance suggests that there is room to further improve the overlay. Recall that the overlay improvement process is adopted from Narada. In Section 7.1.3, we have discussed that this technique is actually similar to switch-trees [43], which faces the greedy problem (Section 7.1.4). In Chapter 7, we devised the MeshTree protocol to overcome this problem.

We thus include the MeshTree overlay improvement process into our second version of MSTP, MSTP-v2. The new version retained most of the mechanisms of the first version. MSTP-v2 starts

[2] This is a replot of Figure 5.21 (a) and (b) to include the result for MSTP-v1.

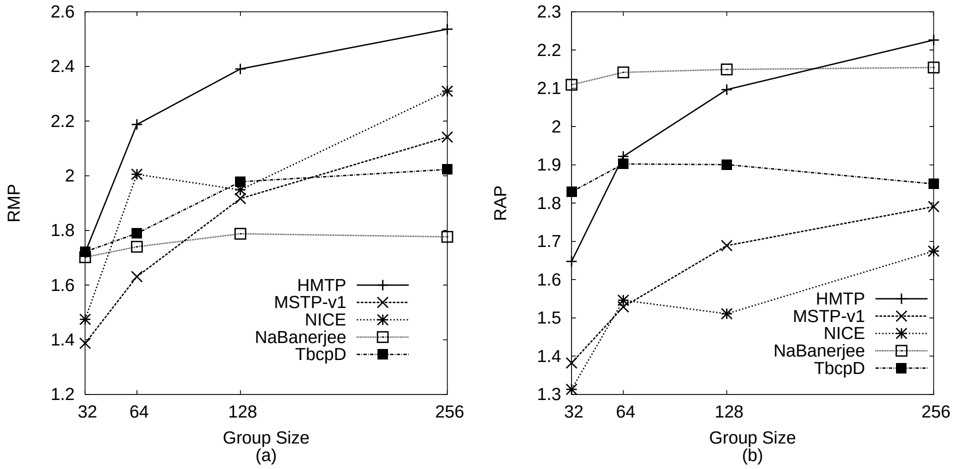


Figure 8.4: Comparing MSTP-v1 with other protocols

with a single tree rooted at the lead core, as in the first version. The MeshTree overlay construction (Section 7.3.2) and improvement (Section 7.3.3) procedures are used to create the initial overlay, and to improve the lead core's tree. With this, the overlay is optimised such that it contains a low cost backbone tree augmented with shortcut links that improves the lead core's delivery tree.

By design, the rules and conditions used to improve the overlay in MeshTree consider only one delivery tree. To accommodate more than one tree, more complex rules have to be introduced. For simplicity, we instead consider the technique used in MSTP-v1: using a utility function to calculate the gain of adding a new link. We modify the original function (Figure 8.3) to take into account the low cost backbone tree. The new function is shown in Figure 8.5. Consider that x is trying to add a link to y; the new function first computes the utility as before. Then, if y is not a descendant of x's backbone tree (this can be determined from y's backbone root path, which is included in y's reply to x's request) and if the link also improves the cost of the backbone tree (i.e. the distance between x and y is smaller than the distance between x and its backbone tree parent), a constant value is added to the utility. The constant value should be larger than the add link threshold so as to include the new link into the overlay. As before, a link can only be added if it will not violate the degree constraints of the nodes.

Other components of MSTP-v2 such as tree derivations, maintenance, cores selection stay as MSTP-v1. The next section provides further evaluation of both versions.

8.4 Performance Evaluation

This section examines the performance of our multiple shared trees proposals. The evaluations are divided into two parts: (i) MSTP properties; and (ii) comparison of MSTP with other proposals.

For all the results shown (except Figure 8.8 and 8.12), each data point is obtained as the average of

```
**EvaluateUtility** $(y)$ begin
$utility = 0$
for each core $c$ $(where\ c \neq x)$ begin
    $CL$ = current latency between $x$ and $c$ along mesh
    $NL$ = new latency between $x$ and $c$ along mesh if edge $\langle x, y \rangle$ were added
    if $(NL < CL)$ then begin
            $utility \mathrel{+}= \frac{CL-NL}{CL}$
    end
end
$/*\ new\ condition\ */$
if ($y$ is not $x$'s backbone tree decendant) then begin
      $C$ = a constant
      $CBTL$ = current latency between $x$ and its backbone tree parent
      $d(x, y)$ = latency between $x$ and $y$
      if $(d(x, y) < CBTL)$ then begin
                $utility \mathrel{+}= C$
      end
end
return $utility$
```

Figure 8.5: Utility function for MSTP-v2

50 independent runs. Unless specified otherwise, we consider every member as a potential data source. This is to prevent biases in the results due to the location of the senders with respect to the overlay (see Section 3.2.3). The members' degree bounds are uniformly distributed in the range 2 to 10. For all distributed protocols, the overlay improvement period is set to 30 seconds. For MSTP, the periodic refresh between two neighbours is 5 seconds, the routing update period is 30 seconds, and the gossip-style nodes discovery runs every 30 seconds. Again, the quality of the overlays built is judged by the following metrics: RMP and RAP for the maximum and average delay penalties, tree cost ratio (TCR) and link stress.

8.4.1 MSTP Properties

This section studies the various aspects of MSTP: the trade-off between the number of trees and the overlay quality, the strategy used to choose among the trees, the strategy used to place the cores and a comparison of the two versions of MSTP. The experiments were conducted with topologies of 1000 nodes (see Section 3.2.2). We show the representative results from a transit-stub topology (TS1k-0). Unless specified otherwise, the results are obtained with MSTP-v1 using the following settings: 10 trees, senders choose tree with the nearest core and the cores are randomly placed.

8.4.1.1 Effects of the Number of Trees

A natural question in using a multiple trees approach is: how many trees are needed? Intuitively, more trees will give better delivery quality, but at the same time increase the maintenance overhead. This section examines this quality versus overhead trade-off. We consider a group of 256 members, and vary the number of trees from 1 up to 64 in the experiments.

The results in Figure 8.6 clearly show that the delay (both RAP and RMP) improves as the number

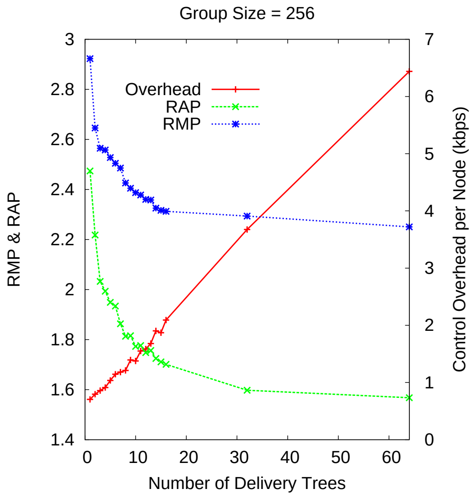


Figure 8.6: Impacts of number of delivery trees

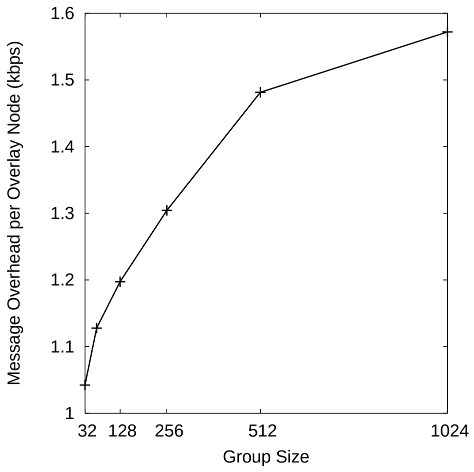


Figure 8.7: MSTP (with 10 trees) protocol overhead

of trees is increased. In particular, large improvement can be observed initially, but the improvement rate diminishes as the number of trees increases further. Unsurprisingly, the protocol overhead (bytes of control messages sent and received per overlay node) grows linearly with the number of trees. It is interesting to see that for up to 10 trees, the per-node message overhead is still less than 1.5 kbps. We believe that the overhead with up to 50 trees (approximately less than 5 kbps), is still acceptable for some applications. Considering the overhead (about 23 kbps) of Narada (which uses source-specific trees) for the same group size, as shown in Figure 5.30 (Chapter 5), it is clear that MSTP has better scalability. The comparison of the overlay quality built with MSTP and other techniques will be given in Section 8.4.2.

Figure 8.7 shows the scaling of 10 trees MSTP with the group size range from 32 to 1024, running on a 10000-node transit-stub topology. While the per node message overhead grows with the group size (which is doubled at each step), the overhead is still well below 2 kbps when there are 1024 members.

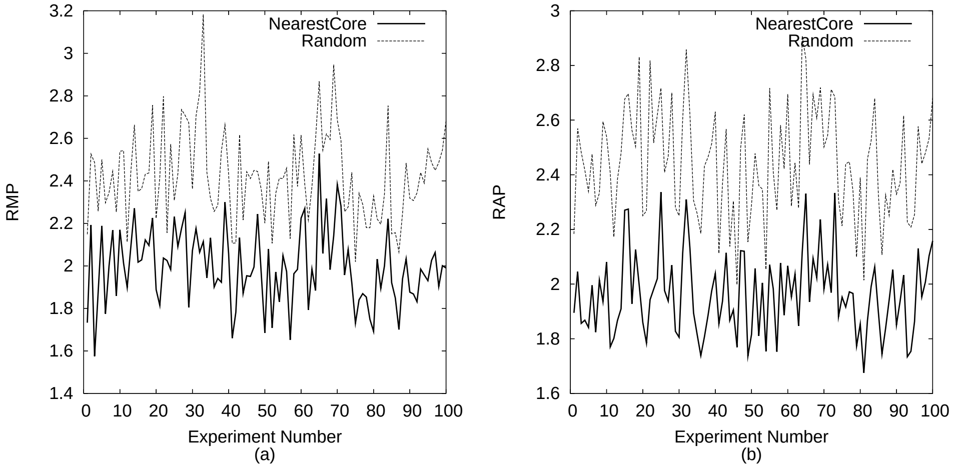


Figure 8.8: Impacts of tree selection strategy

This shows the protocol is reasonably scalable, considering the performance advantage over single tree proposals (see Section 8.4.2).

8.4.1.2 Tree Selection Strategies

As explained in Section 8.3.1.2, given multiple trees, a source will pick the tree with the core that is closest to it. Here we compare this strategy with a naive selection where the source randomly picks one of the trees.

To compare the strategies, we use MSTP-v1 to create 100 overlays, each with 256 members. For each overlay, we measure the RMP and RAP obtained using both tree selection strategies. Figure 8.8 (a) and (b) depict the results. From the figure, it is clear that the nearest core selection always provides lower delay. We recall that each of the trees is the shortest path tree (with respect to the overlay) of its core. Hence, the closer (with respect to the tree) a sender is to the core, the closer the delay will approach that of the shortest path tree. For other metrics (i.e. TCR and stress), there is no difference between the two techniques. This is because these metrics are affected by the structure of the trees, but not by which tree is chosen.

8.4.1.3 Impacts of Cores Placement

A key factor that may influence the performance of MSTP is where the cores are placed in the overlay. Here we examine two cases: (i) the strategies used to place the cores; and (ii) sender-aware core selection.

A. Core Selection Strategies Assume that the total number of cores desired is k, and the lead core has been identified, the following strategies will locate the remaining $k-1$ cores.

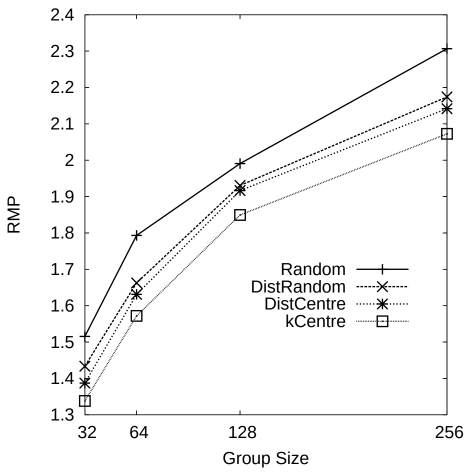


Figure 8.9: Comparing different cores selection techniques

1. *Random.* Given the number of members, n, this strategy randomly picks $k - 1$ nodes from the members list with a probability of $1/n$. This strategy disregards any available information about the members (e.g. distances), and thus acts as the worst-case scenario.

2. *Distributed cores selection.* This refers to the distributed randomised technique proposed in Section 8.3.1.3, which consists of two parts. First, the new core discovery message is randomly distributed to identify a number of potential cores. Then, the lead core will select one of them as a new core. We examine two ways that a new core can be chosen:

   - *Random selection.* A node is randomly selected from the set of potential cores. We refer to this technique as DistRandom. This version is used to study the advantage of the following more informed selection algorithm.

   - *Centre selection.* The lead core selects the node that has the maximum distance to its nearest core. The intuition behind this technique is to spread the cores across the overlay as much as possible. We refer to this technique as DistCentre.

3. *Centralised k-Centre.* In this case, we use a centralised k-centre algorithm to compute the cores. As discussed previously, the k-centre problem is NP-hard. We have experimented with three simple heuristic solutions (see Appendix D), and decided to use the k-mean algorithm which offers the best performance. The algorithm takes as input the number of desired centres (i.e. cores) and the complete distance matrix of the members, and outputs a set of centres that minimise the maximum distance of nodes to their nearest centre.

The results in terms of RMP (Figure 8.9) shows that the random cores selection performs the worst, while the centralised k-centre algorithm performs the best. This indicates that placing the cores at strategic location helps to provide better performance. The fact that our distributed cores selection techniques

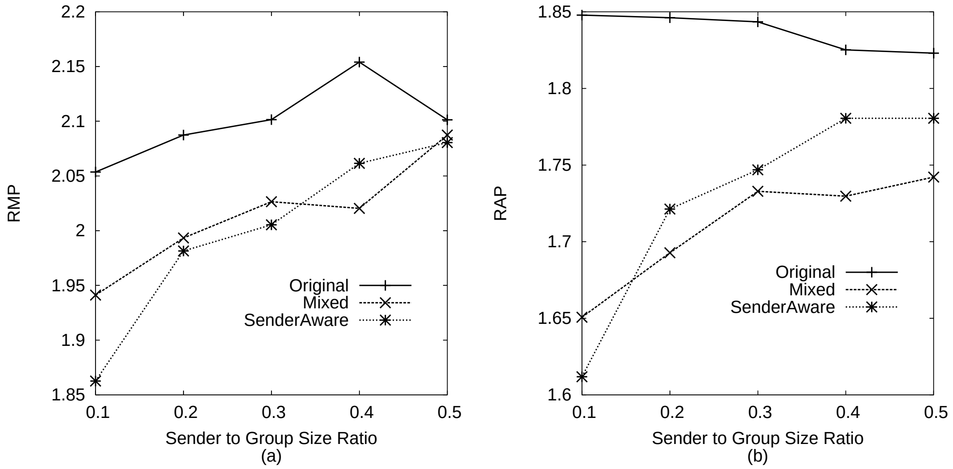


Figure 8.10: Performance of sender-aware cores selection

(DistRandom and DistCentre) perform better than the random strategy shows that our randomised distribution strategy indeed helps in placing the cores. This is rather encouraging as the technique uses only limited knowledge available at the members. Between the two distributed strategies, the DistCentre is slightly better than the DistRandom. No significant differences have been observed for other metrics: TCR and link stress.

B. Sender-aware Cores Selection In some many-to-many multicasting applications such as video conferencing, the number of members that actually take part in the conversation can be small relative to the total number of members. Here, we study the impact of using the knowledge of the data sources in core selection.

We have implemented a simple sender-aware core selection strategy. The strategy requires the members (most importantly, the lead core) to keep the identities of the active sources in the session. This is a reasonable assumption as in a multi-party conversation, the upper-level applications need to know the identity of the data source so as to perform functions such as flow control and error recovery. Thus, every member knows all the active sources. At each core selection round, the lead core randomly picks a number of nodes from the senders list as potential cores, and requests their distances to the existing cores. The lead core then selects a new core according to the centre selection criterion.

We also consider a variant which combines the sender-aware and -unaware (i.e. original) core selection technique. In particular, half of the cores are selected based on the original technique, while the other half are chosen with the sender-aware technique. We refer to this strategy as *mixed*.

We have conducted experiments with a group of 256 members. Recall that MSTP uses 10 trees for the purpose of evaluation. We vary the fraction of active senders from 10% to 50% of the group size. Figure 8.10 (a) and (b) illustrate the results of RAP and RMP. The x-axis represents the ratio

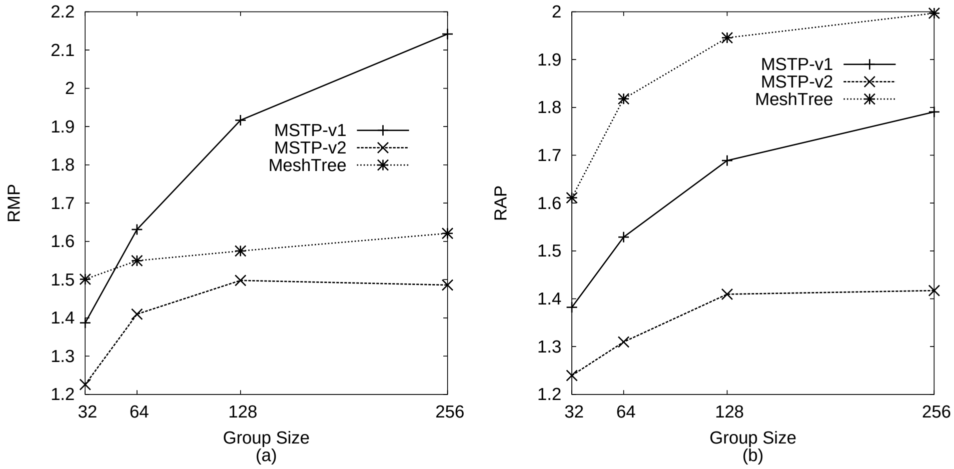


Figure 8.11: Comparing MSTP-v1 and MSTP-v2

between the number of senders and the group size. We first compare the performance of the original (sender-unaware) and sender-aware approaches. It is clear that sender-aware yields much better delay when the number of senders is small. When the number of trees is close to the number of senders, the trees are rooted at most senders, the performance is thus approximating the shortest path trees rooted at the chosen senders. Unsurprisingly, the performance advantage diminishes as the number of senders increases. Now, consider the mixed technique. Interestingly, we can see that its performance is closer to sender-aware than to the original. For a small number of senders, it trails the sender-aware technique. However, as the number of senders increases, it is on par with sender-aware in terms of RMP, and outperforms sender-aware in terms of RAP.

Overall, the results show that if the number of senders is small, e.g. tens of nodes, the sender-aware technique can be used to improve the delay performance. In practice, this could be useful for (audio or video) conferencing applications. In particular, a conference may consist of a large audience, even though the number of active speakers (i.e. data sources) could be small. If the average delay to the members is of particular concern, one might consider combining the sender-aware and unaware strategies (i.e. mixed). This mixed technique provides some cores that are independent of the senders, and hence may be useful for applications that have a highly dynamic senders set.

8.4.1.4 Comparing MSTP-v1 and MSTP-v2

We are now in the position to compare the two version of MSTP. This comparison is used to evaluate the advantage of including the MeshTree overlay improvement into MSTP-v2. The comparison also includes a version of MeshTree that uses shared tree delivery.

Figure 8.11 (a) and (b) depict the RAP and RMP performance. It is clear that MSTP-v2 always yields trees with lower RAP and RMP than MeshTree and MSTP-v1. As MSTP-v2 can be viewed as an

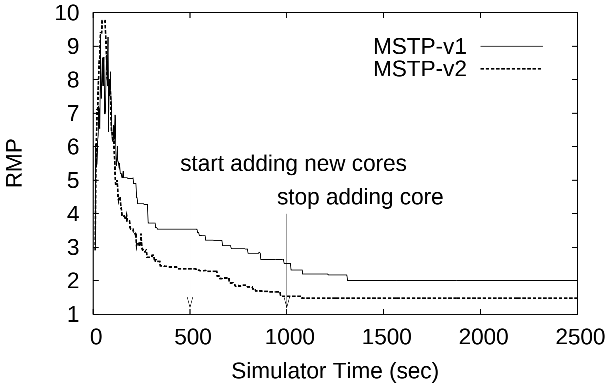


Figure 8.12: Convergence of MSTP-v1 and MSTP-v2

extension of MeshTree with additional trees, the results confirm the advantage of using multiple shared trees. While MSTP-v1 trails MeshTree in RMP (except for 32-node overlay, where the cores to group size ratio is rather large), its RAP is significantly better than MeshTree's. This shows another advantage of using multiple trees. The results again prove a combination of low cost tree and shortcut links can improve the delay.

In Figure 8.12, we compare the evolution of RMP for both versions of MSTP for a group of 256 members. In the experiments, all members randomly join the overlay within the first 50 seconds. The first member automatically becomes the lead core. The lead core begins to add in new cores after 500 seconds, one every 50 seconds until the total number of cores is 10.

From the figure, we can see RMP values increase quickly as members are joining the overlay. This is because the initial overlay is randomly connected. After all members have joined, the delay quickly improves due to the improvement process, until about 400 seconds. At this point, we can see that MSTP-v2 already produces a better quality overlay. This again confirms the advantage of the MeshTree improvement process. The overlay then continues to improve as new cores are added into the overlay.

8.4.2 Comparing MSTP with Other Techniques

This section compares MSTP-v2 (with 10 trees) against three distributed protocols: TBCP, Banerjee et al.'s scheme and HMTP. These protocols have been shown to perform well in their class: TBCP provides low average delay between the members; Banerjee et al.'s scheme gives low member-to-member maximum delay; and HMTP yields low cost trees. NICE and Narada are excluded as they cannot strictly limit the nodes' degree (see Section 5.2.7 and 5.2.2). The centralised Compact Tree [87] and GreedyMesh algorithms (see Chapter 4) are included as benchmarks. We use the 5000- and 10000-node topologies (Section 3.2.2) for the comparison, and show the representative results from a transit-stub topology (TS10k-0). The results for a 5000-node power-law topology can be found in Appendix C.

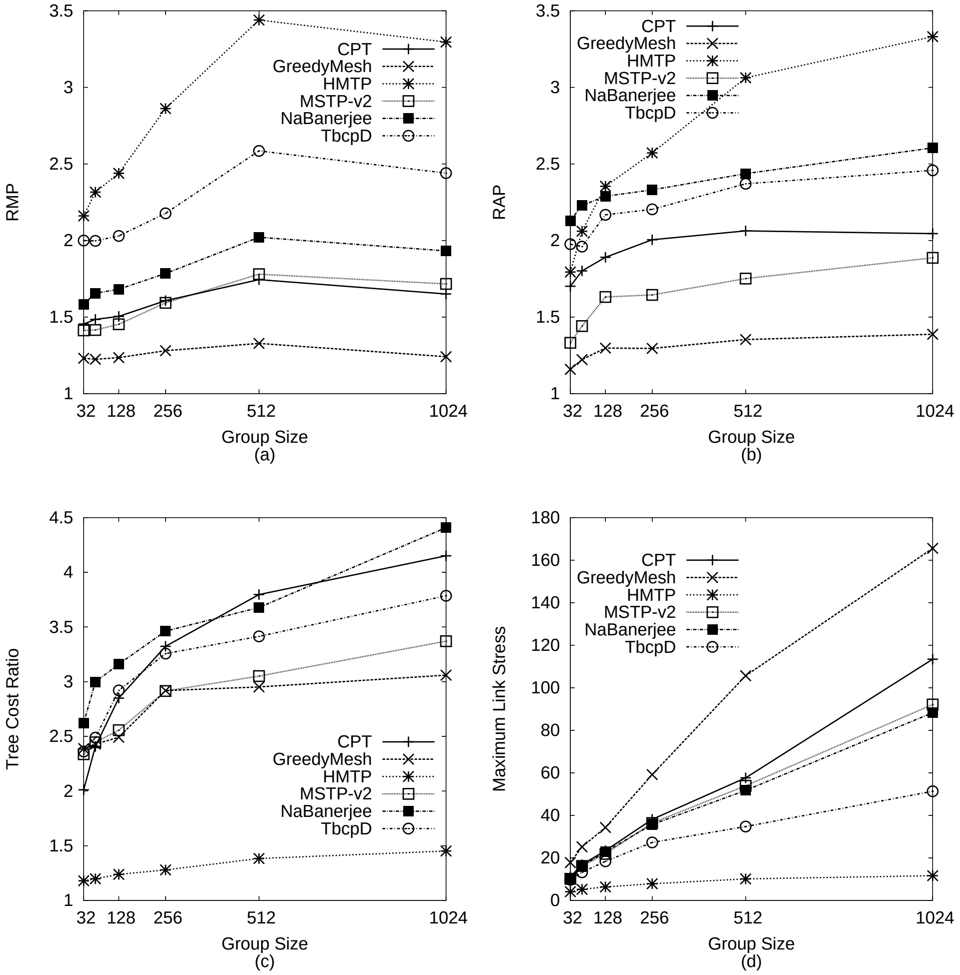


Figure 8.13: Comparison results: (a) RMP; (b) RAP; (c) Tree cost ratio; and (d) Maximum link stress

Figure 8.13 (a) to (d) depict the RMP, RAP, TCR and maximum link stress results, respectively. We first point out that the results for techniques studied in Chapter 5 are similar to those found in Section 5.2.2, where they were discussed in detail. So we focus on the behaviour of MSTP-v2. In terms of delay, we can see that MSTP-v2 provides RMP that is better than other distributed protocols (HMTP, TBCP and Banerjee et al.'s scheme), and is close to the centralised CPT. Its RAP outperforms all these techniques. The centralised GreedyMesh that creates source-specific trees always gives the lowest RAP and RMP, as expected. The observed result is very encouraging considering that MSTP-v2 uses only 10 trees (even for 1024-node overlays), which requires reasonably low overhead. We note the experiments consider all members as senders. If the sender population is much smaller than the group size, we could further improve MSTP-v2 by choosing the cores from the senders set, as discussed in Section 8.4.1.3.

In terms of TCR, generally we can see that MSTP-v2 performs better than other techniques, except HMTP and GreedyMesh. HMTP is a cost-optimised protocol, and it has the worst delay performance.

The lower tree cost of MSTP-v2 and GreedyMesh (with respect to other delay-optimised protocols) is because they both include a low cost tree in their overlay. Finally, Figure 8.13 shows that MSTP-v2 gives a moderate maximum link stress performance, which is close to those of Banerjee et al.'s scheme. Overall, the results prove that our multiple shared trees approach is a promising technique for many-to-many multicasting.

8.5 Chapter Summary

This chapter investigates the case of using multiple shared trees for many-to-many multicasting in ALM. The motivation is to use multiple trees to bridge the performance and quality trade-off in the traditional one tree or all source trees. We have considered a number of design issues in achieving multiple shared trees in ALM, and chosen to use a simple mesh-based approach: the members self-organise into a degree-bounded mesh; the trees, rooted at nodes that we called cores, are derived from the mesh with the path-vector routing protocol. In our design, all overlay nodes attach themselves to each of the trees. When one node is transmitting, it delivers data over one of the trees. The core nodes are added into the overlay in an incremental manner, using a simple distributed technique. We refer to our protocol as MSTP.

The development of MSTP has gone through two phases. The first version is a simple variation of the Narada protocol to study the feasibility of the multiple trees approach. The second version combines the lesson learned in our MeshTree proposal to further improve the protocol. Our performance evaluation shows that by using a reasonably small number of trees, i.e. 10, MSTP outperforms single trees techniques such as TBCP and Banerjee et al.'s scheme, and is on par with the centralised compact tree algorithm in terms of delay, whilst incurring a reasonably small protocol overhead.

Chapter 9

Conclusions and Further Work

In this chapter, we conclude the thesis by revisiting the contributions and lessons learned from this work, and presenting directions for future work.

9.1 Thesis Contributions

The focus of this thesis has been the construction of efficient ALM overlay trees for one-to-many and many-to-many data delivery. We have restricted our investigation to building low cost and low delay delivery trees, subject to the degree constraint impose by each individual member. For scalability reasons, we are interested in distributed proposals that use limited information about the overlay members and the underlying network, and that require only limited coordination between the members in building the overlays. The rest of this section discusses the contributions made by this thesis.

The past few years has seen a growing interest in using ALM to provide multicast services over the Internet. Unsurprisingly, a wide variety of ALM construction proposals have emerged. As a first step in this work, we have conducted a detailed performance comparison study for some existing proposals using simulation. We have chosen proposals with different characteristics. For example, proposals that use only a tree structure and proposals that derive trees out of a richly connected mesh; proposals that use either a single shared tree or multiple source-specific trees in many-to-many data delivery; proposals that use simple overlay reconfiguration technique such as parent switching and proposals that use a more elaborated transformation scheme. The chosen proposals include: HMTP [109], TBCP [62], variants of switch-trees [43] (which includes a version of HostCast [57]), NICE [7], AOM [104], Scribe [15], Banerjee et al.'s scheme [9] and Narada [21].

Two by-products arise from the comparison study. First, to serve as a standard and controlled platform for comparing the proposals, we have developed a simple yet flexible and extensible simulator which we called `ALMSim`. Secondly, we have devised a centralised degree-bounded overlay mesh creation algorithm, which we called GreedyMesh. Our evaluation shows that GreedyMesh can create degree-bounded mesh with low diameter. In this thesis, GreedyMesh is primarily used as a benchmark for many-to-many distributed ALM proposals.

By analysing the behaviour of the various proposals, we are able to identify their strengths and weaknesses. This allowed us to provide enhancements to switch-trees and TBCP. For switch-trees, we have proposed a mixed local and random node selection scheme, which combines the precision of informed parent choice in local-scoped selection and the exploration power of random selection. The new scheme is generic and is applicable for other overlay construction proposals. For TBCP, we have proposed a new tie-breaking rule and a new score function for overlay tree reconfiguration. We have shown that these extensions to the proposals perform better than the original versions.

From the comparison study, we have shown that for tree cost optimisation, existing proposals such as HMTP can achieve results close to a centralised algorithm; however, there is still room for improvement for delay optimisation, for both one-to-many and many-to-many proposals. Thus, our own proposals have been designed to provide low delay trees for both one-to-many and many-to-many delivery models.

As a basis for our own proposals, we have presented a distributed mesh-based framework for overlay tree construction and maintenance. The framework provides basic procedures for creating and maintaining a degree-bounded overlay tree, embedded in a mesh topology. The mesh-based approach offers fast and robust failure recovery, as well as offering more flexibility in overlay reconfiguration. In addition, by exploiting the structure of the delivery tree, we can lower the maintenance overheads for the mesh. The framework is generic and can be used to improve the robustness of some existing ALM proposals.

For one-to-many data delivery, we have proposed MeshTree, a distributed proposal for creating low root-diameter, degree-bounded overlay trees. The design of MeshTree is inspired by the greedy problem and delay-cost trade-off that happens in some delay-based distributed proposals. MeshTree approaches the problem by creating a structure consisting of a low cost tree and some additional links to improve the delay performance. Our evaluation shows that MeshTree outperforms other distributed proposals in providing trees with low root-diameter and low average root-to-receivers delay.

For many-to-many data delivery, we have proposed a distributed solution based on the concept of using multiple shared trees. This is based on the observation that existing proposals either use a single shared tree or source-specific trees for data delivery. The single shared tree approach is scalable, but provides poorer delay performance; the source-specific trees approach gives better delay performance, but does not scale well. By using a small number of shared trees, we show that our multiple shared trees approach can provide reasonably good delay performance while maintaining low control overhead.

9.2 Further Work

In the course of the investigations reported in this thesis, a number of interesting avenues have been uncovered which merit further research.

9.2.1 On the Techniques Proposed by the Thesis

Besides continuing the efforts to further enhance the performance of our overlay construction proposals (GreedyMesh, MeshTree and the multiple shared trees protocol (MSTP)), there are other areas that are worth additional attention.

- Improve the running time of the GreedyMesh algorithm: $O(\max\{\lambda, \Delta n^3 m \log n\})$, where λ is the running time of the tree building algorithm used, Δ is the maximum spare degree of all vertices, and n and m are the number of vertices and edges respectively.

- Consider other alternative design choices for the GreedyMesh algorithm. In particular, in line 8 of the algorithm (see Figure 4.1), when adding a new edge to the partial mesh, a vertex is picked in a round robin manner from all vertices with spare degree. It is interesting to study the impact of other node selection strategies, such as random selection or selection of node with the largest spare degree first.

- Incorporate the GreedyMesh algorithm into a centralised ALM protocol (e.g. ALMI [72]) for small-scale multicast applications. In this case, a central controller is used to distribute the overlay mesh computed by the algorithm to the group members. To obtain data delivery trees from the mesh, a distributed routing mechanism (e.g. the path vector routing protocol) may be needed.

- In Section 3.1.2, we discussed two ways that can be used to limit the impact of the dynamic variations of the delay metric to the overlay structure: (i) cache and threshold; and (ii) quantise the delay value. It is interesting to integrate these two techniques with the proposed distributed protocols and study their performance implication in a real-world environment (see next section).

- In Section 8.4.1.3, we have shown that the core placement strategy used has an effect on the performance of MSTP. Our chosen strategy consists of two parts. First, the core discovery messages are randomly distributed to the members to select a number of candidate cores. Then, a new core is elected based solely on the overlay distances between the candidates and the existing cores. The random message distribution technique is used for its simplicity. In the future, it is worth considering alternative techniques that improve the distribution of the messages to the members. In addition, other factors such as the capability of the nodes could be taken into consideration in the core selection process.

9.2.2 Real-world Testing and Applications

The performance evaluation in this thesis is based heavily on simulation. Simulation is suitable for comparing and for examining the detailed working of the proposals being studied. However, the simulation environment has abstracted away several real-world characteristics (such as congestion and system load) of a practical environment. Hence, the immediate extension of this work is to subject our proposals to a real-world testing in a wide-area testbed such as PlanetLab [74]. Lessons could be learned from such testing to further improve our proposals.

In addition, it would be meaningful to apply our proposals to some real life applications. For example, MeshTree could be useful for single-source applications such as streaming media, while the multiple shared trees protocol could be used by multi-source applications such as video conferencing or multi-party network gaming. Such applications would require additional components such as security, reliability of data, flow control and congestion control.

9.2.3 Network Address Translators (NATs) and Firewalls

In this work, we model the overlay network as a complete graph, in which every member can reach every other members as long as the address information is available. However, this may not be the case in a practical environment due to the use of NATs and firewalls. Specifically, hosts separated by NATs and/or firewalls may not be able to communicate directly with one another. For example, this has been observed by Chu et al. in their experience [20] with the End System Multicast project [75], an early wide-area deployment of ALM for live events broadcast. They report that over 20 – 30% of viewers attempting to join the broadcast need to be turned down due to NATs and firewalls.

Early this year, Wang et al. [99] proposed a generic protocol called e^* to address the limited connectivity problem. In e^*, nodes are classified into two types: *open* hosts and *guarded* hosts. Open hosts are nodes that allow both incoming and outgoing TCP connections, while guarded hosts are nodes that only allow incoming TCP connections. In other words, a guarded host can only serve as a leaf node. The concept of e^* is to cluster the nodes into a two level overlay. The bottom-level consists of all nodes which are grouped into several clusters. Each cluster has a cluster leader, which must be an open host. The guarded hosts attach to one of the clusters as leaf nodes. All the cluster leaders form the top-level overlay. Any existing overlay building protocol (e.g. HMTP, TBCP, Narada) can be used to create the top-level overlay. It is interesting future work to incorporate e^* into our proposals.

9.2.4 Trust

In this work, we have considered a cooperative environment in overlay construction. In particular, we assume that the overlay members are honest and trust each other by sharing their information in building efficient overlays. Unfortunately, in the real-world, there are incentives for a member to violate this assumption. For instance, a selfish member may be reluctant to contribute its own bandwidth to the system. Such a member will refuse any request to become its child, even if it still has sufficient resources. In [61], Mathy et al. study the impacts of some simple cheating strategies on several ALM proposals, i.e. Narada, NICE, TBCP and HBM [80]. Their findings show that cheating always has a negative impact, either on the quality of the data delivery perceived by the members, or on the underlying physical network, or on both. How to prevent nodes from taking advantage by cheating is interesting future work.

Appendix A

Acronym

AGCS Alternative Group Communication Service.

ALM Application Layer Multicast.

BG Border Gateway.

BGMP Border Gateway Multicast Protocol.

BGP Border Gateway Protocol.

BN Broadcast Network.

CBT Core Based Tree.

CoP Cost-optimised Protocol.

CPT Compact Tree.

dbMST Degree-bounded Minimum Spanning Tree.

DFS Depth first search.

DHT Distributed Hash Table.

DoP Delay-optimised Protocol.

DT Delaunay Triangulation.

DVMRP Distance Vector Multicast Routing Protocol.

GNP Global Network Positioning.

HBM Host Based Multicast.

HMTP Host Multicast Tree Protocol.

IETF Internet Engineering Task Force.

IGMP Internet Group Management Protocol.

IP Internet Protocol.

MAAA Multicast Address Allocation Architecture.

MBGP Multicast extension for the Border Gateway Protocol.

MDC Multiple Description Coding.

MOSPF Multicast extension for Open Shortest Path First routing protocol.

MSDP Multicast Source Discovery Protocol.

MSTP Multiple Shared Trees Protocol.

NAT Network Address Translator.

non-TNRP Non-tree Neighbours' Root Path.

OMNI Overlay Multicast Network Infrastructure.

PIM-DM Protocol Independent Multicast — Dense Mode.

PIM-SM Protocol Independent Multicast — Sparse Mode.

P2P Peer-to-Peer.

PRM Probabilistic Reliable Multicast.

RAMA Root Addressed Multicast Architecture.

RDP Relative Delay Penalty.

RP Rendezvous Point.

RPF Reverse Path Forwarding.

SM Simple Multicast.

TBCP Tree Building Control Protocol.

TCR Tree cost ratio.

TCRP Tree Children's Root Path.

Appendix B

Annotated Publications List

This appendix lists the papers written during the course of this PhD, and how they relate to this thesis.

Conference and Workshop

1. *Su-Wei Tan, Gill Waters and John Crawford, ”MeshTree: Reliable Low Delay Degree-bounded Multicast Overlays”, The 1st International Workshop on Distributed, Parallel and Network Applications (DPNA’05), Fukuoka, Japan, 20-22 July 2005.*

   This work presents the MeshTree proposal (Chapter 7) for creating low root-diameter degree-bounded ALM trees.

2. *Su-Wei Tan, Gill Waters and John Crawford, ”A Study of Distributed Low Latency Application Layer Multicast Tree Construction”, London Communications Symposium, London, UK, 13-14 Sept 2004.*

   In this work, we present comparison study of several distributed one-to-many ALM proposals (Chapter 5), in the light of achieving low cost and low delay ALM trees.

3. *Su-Wei Tan, Gill Waters and John Crawford, ”A Multiple Shared Trees Approach for Application Layer Multicasting”, The 39th annual IEEE International Conference on Communications (ICC), Paris, France, 20-24 June 2004.*

   This paper presents the first version of our multiple shared trees approach (i.e. MSTP-v1) for many-to-many multicasting (Chapter 8).

4. *Su-Wei Tan, Gill Waters and John Crawford, ”A Multiple Shared Trees Approach for Application Layer Multicasting”, The 8th Radicals Workshop, Cabernet, Corsica, France, Oct 2003.*

   This paper introduces the concept behind the multiple shared trees approach, its challenges and design considerations (Chapter 8).

5. *Su-Wei Tan and Gill Waters, ”Building Low Delay Application Layer Multicast Trees”, The 4th Annual PostGraduate Symposium (PGNet), Liverpool, UK, 2003.*

   In this paper, we describe and evaluate our enhanced version of Tree Building Control Protocol (TBCP) (Chapter 5).

Technical Reports

1. *Su-Wei Tan, Gill Waters, and John Crawford, ”MeshTree: A Delay-optimised Overlay Multicast Tree Building Protocol”, University of Kent Technical Report, TR 5-05, April 2005.*

   This paper is an extended version of our DPNA’05 paper, for the MeshTree proposal (Chapter 7).

2. *Su-Wei Tan, Gill Waters, and John Crawford, ”A Survey and Performance Evaluation of Scalable Tree-based Application Layer Multicast Protocols”, University of Kent Technical Report, TR 9-03, July 2003.*

   This work is our initial comparison study of various existing ALM proposals, focusing only on tree-based proposals, for both one-to-many and many-to-many data delivery models (Chapter 5).

Under Review

1. *Su-Wei Tan, Gill Waters, and John Crawford, ”A Performance Comparison of Self-organising Application Layer Multicast Overlay Construction Techniques”, submitted to Computer Communications Journal, Elsevier Science.*

   This work extends the above technical report (TR 9-03) by including new proposal (i.e. Narada) and consists of a larger set of experiments (Chapter 5).

Appendix C

Additional Results

In this appendix, we provide some results omitted from the main text.

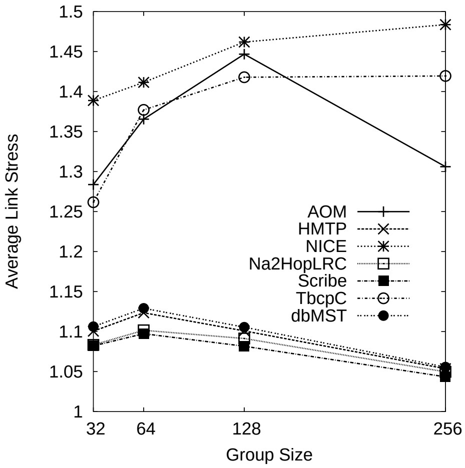


Figure C.1: Average link stress performance of some cost-optimised ALM proposals, Section 5.2.1.1

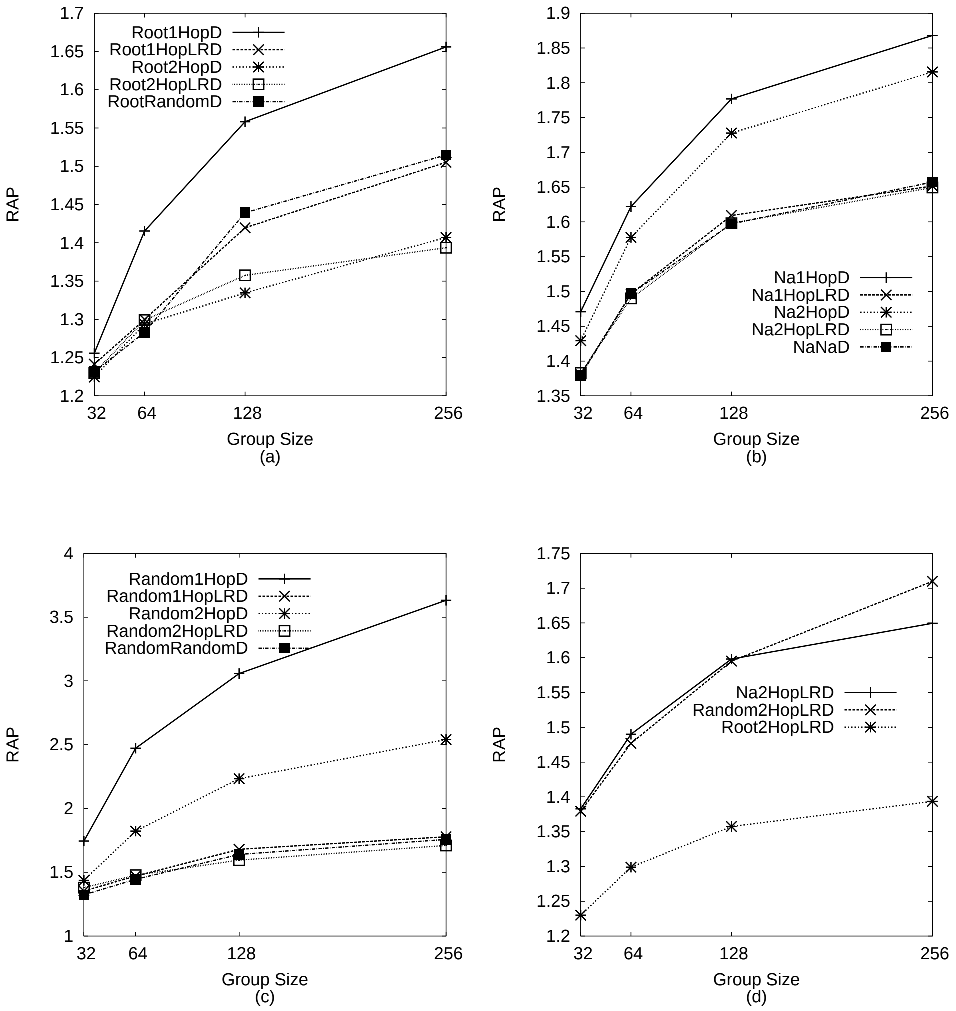


Figure C.2: RAP performance for delay-optimised switch-trees variants, Section 5.2.1.2

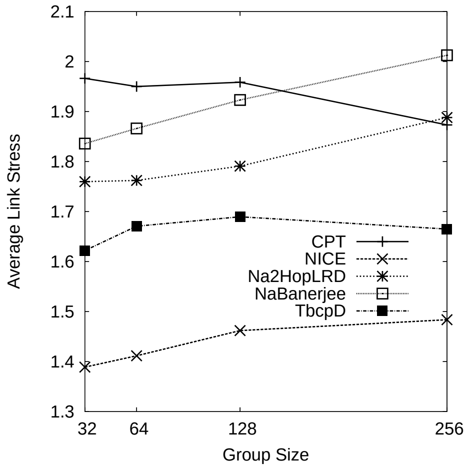
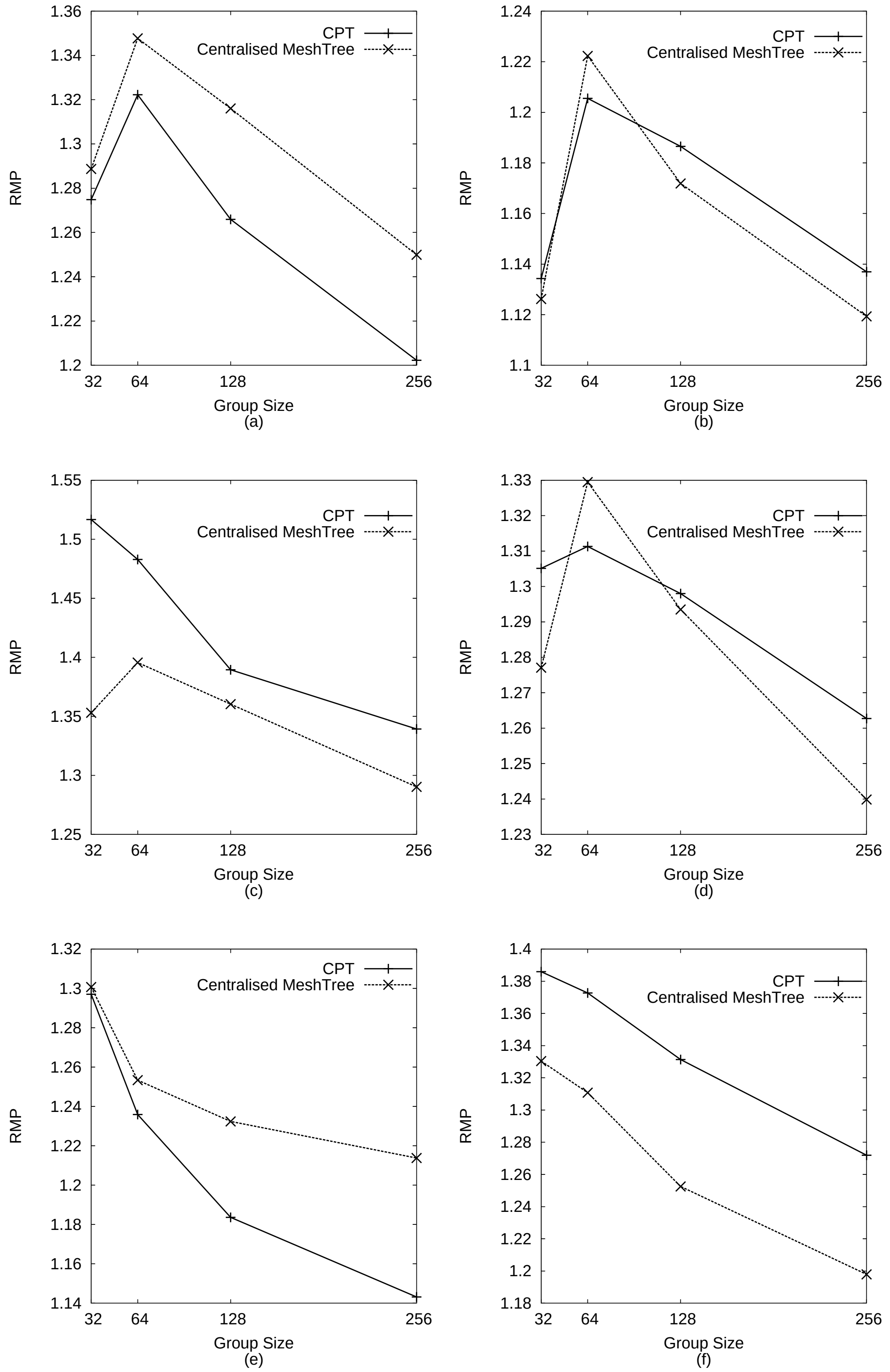


Figure C.3: Average link stress performance of some delay-optimised ALM proposals, Section 5.2.1.2


1.36
1.34
1.32
1.3
1.28
1.26
1.24
1.22
1.2
RMP
CPT
Centralised MeshTree
32
64
128
256
Group Size
1.24
1.22
1.2
1.18
1.16
1.14
1.12
1.1
RMP
CPT
Centralised MeshTree
32
64
128
256
Group Size
1.55
1.5
1.45
1.4
1.35
1.3
1.25
RMP
CPT
Centralised MeshTree
32
64
128
256
Group Size
1.33
1.32
1.31
1.3
1.29
1.28
1.27
1.26
1.25
1.24
1.23
RMP
CPT
Centralised MeshTree
32
64
128
256
Group Size
1.32
1.3
1.28
1.26
1.24
1.22
1.2
1.18
1.16
1.14
RMP
CPT
Centralised MeshTree
32
64
128
256
Group Size
1.4
1.38
1.36
1.34
1.32
1.3
1.28
1.26
1.24
1.22
1.2
1.18
RMP
CPT
Centralised MeshTree
32
64
128
256
Group Size

(a)

(b)

(c)

(d)

(e)

(f)

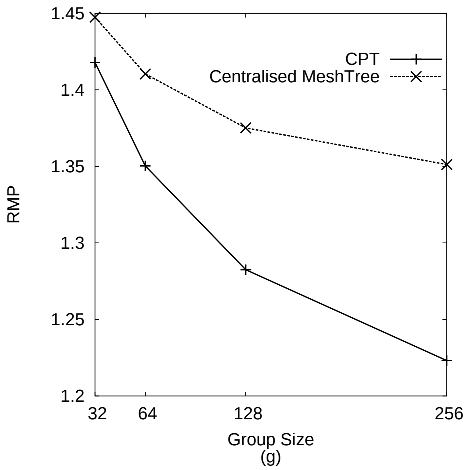
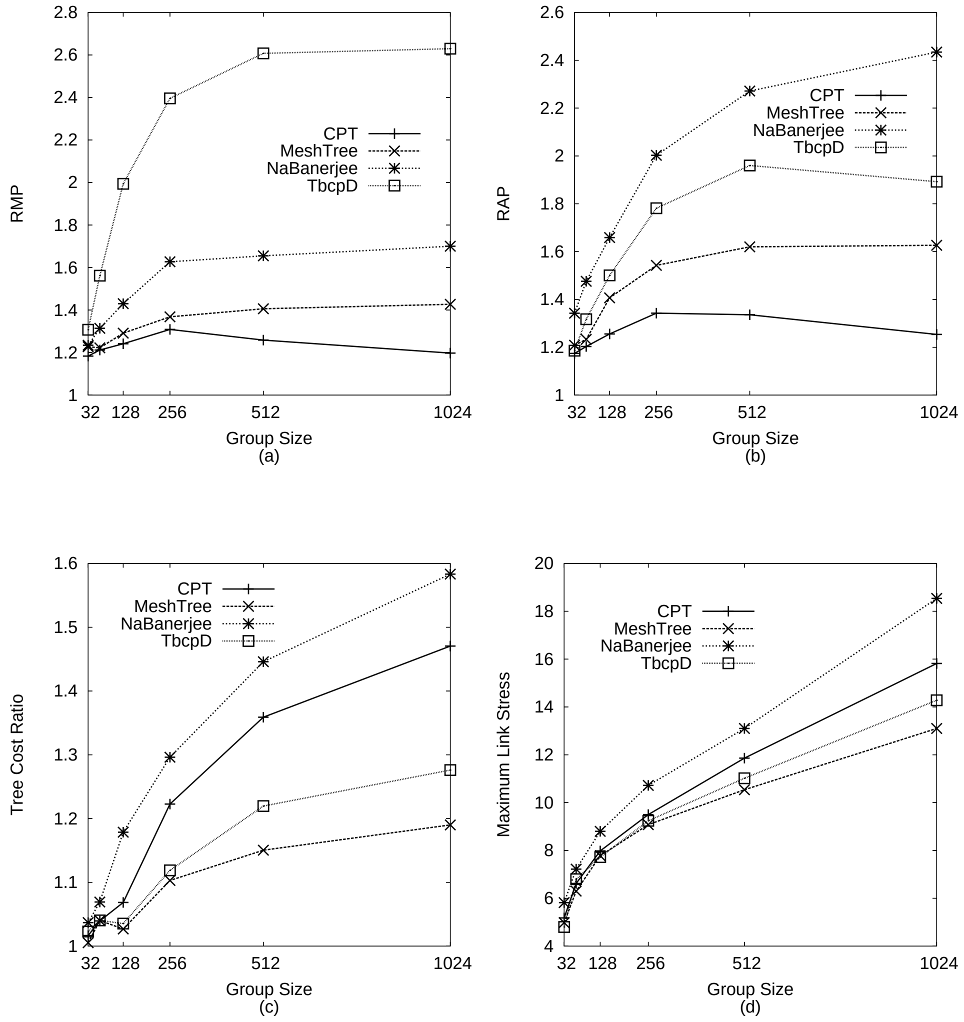


Figure C.4: Delay (RMP) performance of the centralised MeshTree and CPT, for different topologies: (a) PL1k-1; (b) PL1k-2; (c) TS1k-1; (d) TS1k-2; (e) WM1k-0; (f) WM1k-1; and (g) WM1k-2, Section 7.2.2



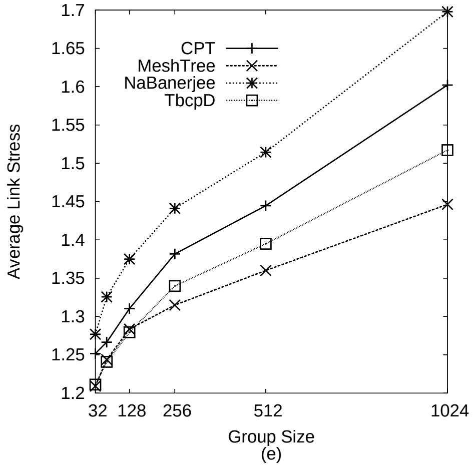
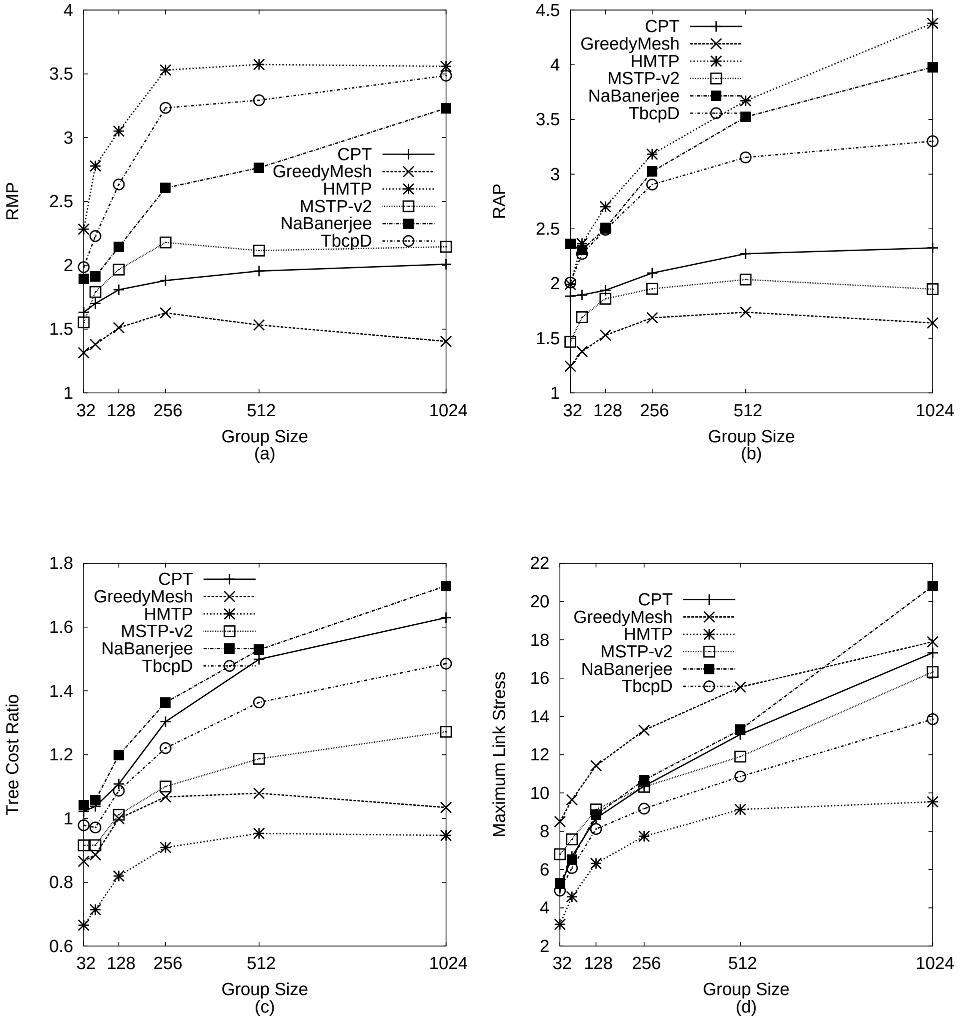


Figure C.5: Comparison of MeshTree with other proposals in a power-law based topology (PL5k-0), Section 7.3.6



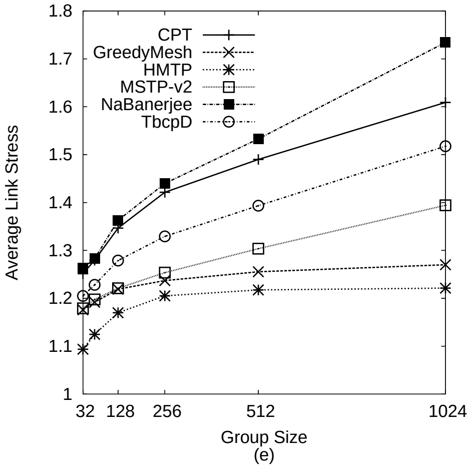


Figure C.6: Comparison of MSTP-v2 with other proposals in a power-law based topology (PL5k-0), Section 8.4

Appendix D

k-Centre Problem

The k-centre problem is a basic facility location problem, where we are asked to locate k facilities in a graph and to assign vertices to facilities, so as to minimise the maximum distance from a vertex to the facility to which it is assigned. In Section 8.4.1.3, we used k-mean [85], a heuristic for the k-centre problem to investigate the impacts of cores location in our multiple shared trees protocol. This appendix describes some comparison results for three simple k-centre selection algorithms: random, k-mean and a 2-approximation algorithm [98]. Note that this is not a comprehensive comparison. We are only interested in technique that yields reasonably good performance.

D.1 The Problem and Solutions

Formally, the k-centre problem can be stated as follows [98].

> Let $G = (V, E)$ be a complete undirected graph with edge costs satisfying the triangle inequality, and k be a positive integer. For any set $S \subseteq V$ and vertex $v \in V$, define $connect(v, S)$ to be the cost of the cheapest edge from v to a vertex in S. The problem is to find a set $S \subseteq V$, with $|S|= k$, so as to minimise $max_v\{connect(v, S)\}$.

Unfortunately, the above problem is NP-hard [37]. We are interested in simple heuristic that performs reasonably well for the problem. We consider the following three strategies:

- *Random*. This technique does not uses any knowledge of the underlying graph to compute the centres. Rather, it randomly picks centres up to the required number. Thus, it serves as the worst-case scenario.

- *2-approximation*. We consider a 2-approximation algorithm, which outputs solution that is at most twice as bad as the optimal solution. This is the best possible case in the sense that no r-approximation algorithm exists with $r < 2$, unless $P = NP$ [98]. We use an algorithm by

Hochbaum and Shmoys [98]. In the algorithm, edges are initially sorted in a nondecreasing order based on their distances. For each edge length, l, the graph is pruned by removing edges with cost greater than l. The aim is to find a minimum dominating set in the pruned graph, i.e. the smallest set D of vertices such that every vertex not in D is adjacent to one of the vertices in D. If the cardinality of the minimum dominating set of the pruned graph is at most k, then such a dominating set is also the solution for the k-centre problem. This algorithm has also been considered in other networking research, e.g. the Internet distance service, IDMaps [35], for placing distance tracers in the network.

- *k-mean*. k-mean is a popular clustering algorithm that divides a given data set into k subsets. For the k-centre problem, the vertices are treated as the data set. The algorithm begins by selecting k nodes as initial seeds or centroids. For simplicity, we randomly select the initial centroids. Other nodes are then assigned to their nearest centroids. This results in k clusters. After that, the algorithm computes new centroid for each cluster, and reassigns the nodes to new clusters. For a given cluster, there are several ways to compute the centroid, e.g. average-within, nearest-within and farthest-away (see [85] for details). We use the average-within technique which selects a centroid, $i \in C$ that minimises the intra-cluster distance:

$$d(i, C) = \sum_{j \in C} \frac{d_{ij}}{|C|} \tag{D.1}$$

where C represents nodes in a cluster, and $d(i, j)$ is the distance between node i and j.

The above process continues until the solution converges, i.e. the latest solution is similar to the previous one. We also set a maximum number of iterations that the algorithm can run (100 in our implementation). The output of the algorithm is nodes grouped in k clusters. The desired k centres are selected as the most central node (i.e. the node with the smallest maximum distance to other nodes in the same cluster) from each cluster. The use of k-mean was drawn to our attention by work that applies the technique to create hierarchical multicast trees [59].

D.2 Performance Evaluation

We are interested in the following performance metrics:

- The maximum distance from a node to its nearest centre, i.e. the objective function of the k-centre problem:

$$max_v\{connect(v, S) : v \in V - S\}$$

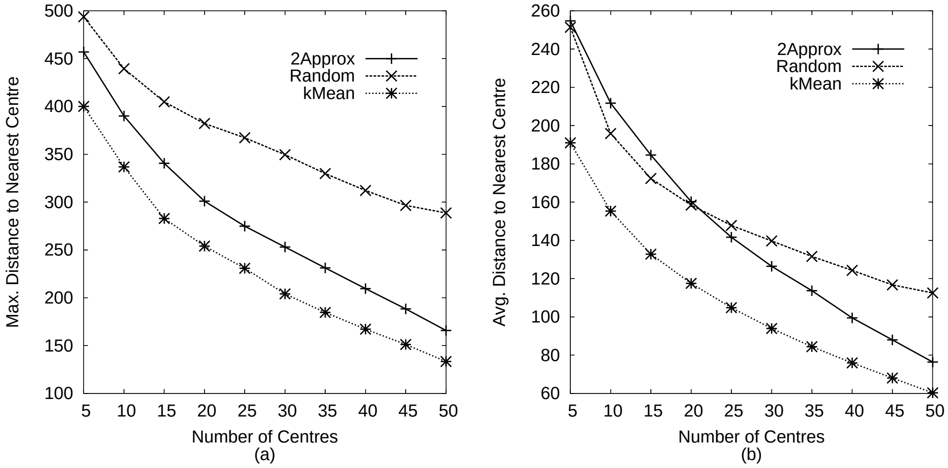


Figure D.7: Comparing three k-centre heuristics

- The average distance from the set of nodes to their nearest centres:

$$\frac{1}{|V| - |S|} \sum_{\forall v \in V-S} connect(v, S)$$

We have ran some experiments on top of a 1000-node transit-stub topology (TS1k-0 in Section 3.2.2). In each experiment, we chose 100 nodes and constructed a distance matrix using the Floyd-Warshall all-pair shortest path algorithm [23]. The distance matrix is taken as the input graph to each of the algorithms listed above. For each value of k, we conduct 50 independent runs and report the average.

The results are shown in Figures D.7 (a) and (b). In both metrics, k-mean always gives the best performance. The 2-approximation (2Approx) algorithm performs reasonably well in the maximum distance, but poorer than the random approach in terms of average distance for a small number of centres, i.e. $k \leq 25$. Consequently, we choose to use k-mean in Chapter 8.

Bibliography

[1] A. Adams, J. Nicholas, and W. Siadak. Protocol independent multicast-dense mode (PIM-DM): Protocol specification (revised). IETF RFC 3973.

[2] K. C. Almeroth. The evolution of multicast: from the MBone to inter-domain multicast to Internet2 deployment. *IEEE Network*, 14(1):10–20, Jan/Feb 2000.

[3] NAM: Network Animator. Available at http://www.isi.edu/nsnam/nam.

[4] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi. Quality of service based routing: A performance perspective. In *ACM SIGCOMM*, pages 17–28, Vancouver, 1998.

[5] A.J. Ballardie, P. Francis, and J. Crowcroft. Core Based Trees (CBT). In *ACM SIGCOMM*, pages 85–95, San Francisco, 1993.

[6] S. Banerjee and B. Bhattacharjee. A comparative study of application layer multicast protocols. Submitted for review, Department of Computer Science, University of Maryland, 2002.

[7] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *ACM SIGCOMM*, pages 205–217, Pittsburgh, PA, 2002.

[8] S. Banerjee, B. Bhattacharjee, and C Kommareddy. Scalable application layer multicast. Technical Report UMIACS TR-2002-53, Department of Computer Science, University of Maryland, May 2002.

[9] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *IEEE INFOCOM*, pages 1521–1531, San Francisco, USA, 2003.

[10] S. Banerjee, S. Lee, , B. Bhattacharjee, and A Srinivasan. Resilient multicast using overlays. In *ACM SIGMETRICS*, pages 102–113, San Diego, CA, 2003.

[11] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[12] T. Bates, R. Chandra, D. Katz, and Y. Rekhter. Multiprotocol extensions for BGP-4, 1998. IETF RFC 2283.

[13] S. Bhattacharyya, C. Diot, J. Jetcheva, and N. Taft. POP-level access-link-level traffic dynamics in a tier-1 POP. In *ACM SIGCOMM Internet Measurement Workshop*, San Francisco, USA, 2001.

[14] R. Boivie, N. Feldman, Y. Imai, W. Livens, D. Ooms, O. Paridaens, and E. Muramato. Explicit multicast (XCast) basic specification, 2004. IETF Internet Draft.

[15] M. Casto, P. Druschel, A. M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralised application-level multicast infrastructure. *IEEE JSAC*, 20(8):1489–1499, 2002.

[16] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 298–313, Bolton Landing, New York, Oct 2003.

[17] M. Castro, M. B. Jones, A. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-lavel multicast built using peer-to-peer networks. In *IEEE INFOCOM*, San Francisco, 2003.

[18] Y. Chawathe. *An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, University of California, 2000.

[19] Y. Chawathe and M. Seshadri. Broadcast federation: An application-layer broadcast internetwork. In *NOSSDAV*, pages 117–126, Miami, Florida, USA, 2002.

[20] Y. Chu, A. Ganjam, N. Eugene, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early experience with an Internet broadcast system based on overlay multicast. In *USENIX Annual Technical Conference*, pages 155–170, Boston, USA, June 2004.

[21] Y. H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS*, pages 1–12, Santa Clara, CA, 2000.

[22] R. Cohen and G. Kaempfer. A unicast-based approach for streaming media. In *IEEE INFOCOM*, pages 440–448, Alaska, 2001.

[23] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithm*. McGraw-Hill, 1991.

[24] Y. K. Dalal and R. Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, 21(12):1040–48, 1978.

[25] S. Deering and D. R. Cheriton. Multicast routing in datagram inter-networks and extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, 1990.

[26] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over a peer-to-peer network. Technical Report CS-2001-31, Department of Computer Science, 2001.

[27] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, Jan 2000.

[28] J. Doar. *Multicast in the Asynchronous Transfer Mode Environment*. PhD thesis, University of Cambridge, January 1993. Chapter 8.

[29] A. El-Sayed, V. Roca, and L. Mathy. A survey of proposals for an alternative group communication service. *IEEE Network Special Issue on Multicasting: An Enabling Tehcnology*, 17(1):46–51, Jan 2003.

[30] H. Eriksson. MBONE: The multicast backbone. *Communications of the ACM*, 37(8):54–60, 1994.

[31] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol independent multicast-sparse mode (PIM-SM): Protocol specification. IETF RFC 2362.

[32] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. In *ACM SIGCOMM*, pages 251–262, Cambridge, MA., 1999.

[33] B. Fenner and D. Meyer. Multicast source discovery protocol (MSDP), 2003. IETF RFC 3618.

[34] W. Fenner. Internet group management protocol, version 2. IETF RFC 2236.

[35] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A global Internet host distance estimation service. *IEEE/ACM Transaction on Networking*, 9(5):525–540, Oct 2001.

[36] P. Francis, Y. Pryadkin, P. Radoslavov, R. Govindan, and B. Lndell. Yoid: Your own internet distribution. Unpublished work in progress, ISI, 2000.

[37] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.

[38] BRITE Topology Generator. Available at http://www.cs.bu.edu/brite.

[39] GT-ITM Topology Generator. Available at http://www.cc.gatech.edu/fac/ellen.zegura/gt-itm/gt-itm.tar.gz.

[40] Gnutella. http://www.gnutella.com.

[41] M Harchol-Balter, T. Leighton, and D. Lewin. Resource discovery in distributed networks. In *18th Annual ACM-SIGACT/SIGOPS Symposium on Principles of Distributed Computing, Atlanta*, pages 229–238, May 1999.

[42] D. A. Helder and S. Jamin. Banana tree protocol, an end-host multicast protocol. Technical Report TR-429-00, Department of Computer Science, University of Michigan, July 2000.

[43] D. A. Helder and S. Jamin. End-host multicast communication using switch-trees protocols. In *Workshop on Global and Peer to Peer Computing on Large Scale Distributed System (GP2PC)*, page 419, Washington, DC, USA, 2002.

[44] H. Holbrook and B. Cain. Source-specific multicast for IP, 2000. IETF Internet Draft.

[45] H. Holbrook and D Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. In *ACM SIGCOMM*, pages 65–78, Cambridge, MA., 1999.

[46] J-Sim. Available at http://www.j-sim.org.

[47] R. Jain, S. Mahajan and D. Wetherall. A study of the performance potential of DHT-based overlays. In *4th USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, March 2003.

[48] J. Jannotti, D. DGifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: Reliable multicast with an overlay network. In *Symposium on Operating System Design and Implementation*, pages 197–212, San Diego, CA, 2000.

[49] S. Kandula, J. K. Lee, J. C. Hou, and S. Kalyanaraman. LARK: A lightweight resilient application-level multicast protocol. In *IEEE Annual Computer Communications Workshop*, Dana Point, California, 2003.

[50] J. Konemann. *Approximation Algorithms for Minimum-cost Low-degree Subgraphs*. PhD thesis, Carnegie Mellon University, 2003.

[51] J. Konemann, A. Levin, and A. Sinha. Approximating the degree-bounded minimum diameter spanning tree problem. In *6th International Workshop on Approximation Algorithms for Combinarorial Optimization Problems*, 2003.

[52] Guy Kortsarz and David Peleg. Generating low-degree 2-spanners. *SIAM Journal on Computing*, 27(5):1438–56, 1998.

[53] D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A. Vahdat. Using random subsets to build scalable network services. In *4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[54] D. Kostic, A. Rodriguez, and A. Vahdat. Scalability and adaptivity in two-metric overlays. Technical Report CS-2002-10, Department of Computer Science, Duke University, May 2002.

[55] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *11th Canadian Conference of Computational Geometry (CCCG)*, pages 51–54, Vancouver, Canada, 1999.

[56] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D. Estrin, and M. Handley. The MASC/BGMP architecture for inter-domain multicast routing. In *ACM SIGCOMM*, pages 93–104, Vancouver, Canada, 1998.

[57] Z. Li and P. Mohapatra. HostCast: A new overlay multicast protocol. In *IEEE ICC*, Anchorage, Alaska, USA, 2003.

[58] J. Liebeherr, M. Nahas, and Si. W. Application-layer multicasting with delaunay triangulation overlays. *IEEE JSAC*, 20(8):1472–1488, 2004.

[59] S. G. Lim. Constructing hierarchical multicast trees. MSc thesis, Computing Laboratory, University of Kent, Sept 2002.

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

[61] L. Mathy, N. Blundell, V. Roca, and A. El-Sayed. Impact of simple cheating in application-level multicast. In *IEEE INFOCOM*, Hong Kong, March 2004.

[62] L. Mathy, R. Canonico, and D Hutchison. An overlay tree building control protocol. In *Third International Workshop on Networked Group Communication (NGC)*, pages 78–87, London, UK, 2001.

[63] L. Mathy, R. Canonico, S. Simpson, and D. Hutchison. Scalable adaptive hierarchical clustering. *IEEE Communication Letters*, 6(3):117–119, 2002.

[64] D. Meyer and P. Lothberg. Static allocations in 233/8, 1999. IETF Internet Draft.

[65] J. Moy. Multicast extensions to OSPF. IETF RFC 1584.

[66] myns simulator. Available at http://www.cs.umd.edu/ suman/research/myns.

[67] S. C. Narula and C. A. Ho. Degree-constrained minimum spanning trees. *Computers and Operation Research*, 7(4):239–249, 1980.

[68] T. S. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *IEEE INFOCOM*, pages 170–179, New York, 2002.

[69] Network Simulator: ns version 2. Available at http://www.isi.edu/nsnam/ns.

[70] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. Distributed streaming media content using cooperative networking. In *ACM NOSSDAV*, pages 177–186, Miami Beach, FL, USA, May 2002.

[71] P. Parnes, K. Synnes, and D. Schefstrom. Lightweight application level multicast tunneling using mtunnel. *Computer Communications*, 21(15):1295–1301, Apr 1998.

[72] D. Pendarakis, S. Y. Shi, D. Verma, and M. Waldvogel. ALMI: An application layer multicast infrastructure. In *3rd USENIX Symposium on Internet Technologies and Systems*, 2001.

[73] R. Perlman, C. Lee, T. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, J. Thoo, and M. Green. Simple multicast: A design fro simple, low-overhead multicast, 1999. IETF Internet Draft.

[74] PlanetLab. http://www.planet-lab.org.

[75] End System Multicast Project. http://esm.cs.cmu.edu.

[76] Jungle Monkey Project. http://www.junglemonkey.net.

[77] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, pages 161–172, San Diego, Carlifornia, 2001.

[78] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Third International Workshop onNetworked Group Communication (NGC)*, pages 14–29, London, UK, 2001.

[79] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). IETF RFC 1771.

[80] V. Roca and A. El-Sayed. A host-based multicast (HBM) solution for group communications. In *1st IEEE International Conference on Networking (ICN'01)*, pages 610–619, Colmar, France, 2001.

[81] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.

[82] H. Salama, Y. Viniotis, and D. Reeves. An efficient delay constrained minimum spanning tree heuristic. In *5th International Conference on Computer Communications and Networks*, 1996.

[83] J. Saltzer, D. Reed, and D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):195–206, 1984.

[84] K. Saroiu, K. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Conferencing and Networking*, San Jose, CA, Jan 2002.

[85] S. Sharma. *Applied Multivariate Techniques*. John Wiley, 1996. Chapter 7.

[86] S. Y. Shi. *Design of Overlay Networks For Internet Multicast*. PhD thesis, University of Washington, Aug 2002.

[87] S. Y. Shi, J. S. Turner, and M. Waldvogel. Dimensioning server access bandwidth and multicast routing in overlay networks. In *NOSSDAV*, pages 83–91, Port Jefferson, New York, USA, 2001.

[88] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, and H. Balakrishnam. Chord: A scalable peer-to-peer lookup service for Internet applications. In *ACM SIGCOMM*, pages 149–160, San Deigo, US, 2001.

[89] S. W. Tan and Gill Waters. Building low delay application layer multicast tree. In *4th Annual Post-Graduate Symposium: The Convergence of Telecommunications, Networking and Broadcasting*, pages 27–32, Liverpool, UK, 2003. Liverpool John Moore University.

[90] S. W. Tan, Gill Waters, and J. Crawford. A survey and performance evaluation of scalable tree-based application layer multicast protocols. Technical Report 9-03, Computing Laboratory, University of Kent, July 2003.

[91] S. W. Tan, Gill Waters, and J. Crawford. A multiple shared trees approach for application layer multicasting. In *IEEE ICC*, Paris, France, 2004.

[92] S. W. Tan, Gill Waters, and J. Crawford. A study of distributed low latency application layer multicast tree construction. In *London Communication Symposium*, London, UK, 2004.

[93] A. S. Tanenbaum. *Computer Network*. Prentice Hall, 2002.

[94] D. Thaler, M. Handley, and D. Estrin. The internet multicast address allocation architecture, 2000. IETF RFC 2908.

[95] Duc A. Tran, Kien A. Hua, and Tai T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *IEEE INFOCOM*, pages 1283–1292, San Francisco, USA, 2003.

[96] Duc A. Tran, Kien A. Hua, and Tai T. Do. A peer-to-peer architecture for media streaming. *IEEE JSAC*, 22(1):121–133, 2004.

[97] R. Van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Middleware'98*, pages 55–70. IFIP, 1998.

[98] V. Vazirani. *Approximation Algorithms*. Springer, 2001. Chapter 5: k-Center.

[99] Wenjie Wang, Jin Chen, and Sugih Jamin. Network overlay construction under limited end-to-end addressability. In *IEEE INFOCOM*, Miami, March 2005.

[100] Wenjie Wang, D. Helder, S. Jamn, and L. Zhang. Overlay optimizations for end-host multicast. In *Fourth International Workshop on Networked Group Communication (NGC)*, Boston, USA, Oct 2002.

[101] Z. Wang and J. Crowcroft. Quality of service routing for supporting multimedia applications. *IEEE JSAC*, 14(7):1228–1234, 1996.

[102] B. M Waxman. Routing of multipoint connections. *IEEE JSAC*, 6(9):1617–1622, 1988.

[103] Liming Wei and Deborah Estrin. A comparison of multicast trees and algorithms. In *IEEE INFOCOM*, 1994.

[104] S. Wu and S. Banerjee. Improving the performance of overlay with dynamic adaptation. In *IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, 2004.

[105] J. Xie, R. R. Talpade, A. Mcauley, and M. Liu. AMRoute: Adhoc multicast routing protocol. *Mobile Networks and Applications (MONET)*, 7(6):429–36, Dec 2002.

[106] Yahoo! MarketTracker. http://finance.yahoo.com.

[107] M. Yang and Z. Fei. A proactive approach to reconstructing overlay multicast trees. In *IEEE INFOCOM*, Hong Kong, 2004.

[108] D. Zappala, A. Fabbri, and V. Lo. An evaluation of shared multicast trees with multiple cores. *Journal of Telecommunication Systems*, 19(3):461–479, 2002.

[109] B. Zhang, S. Jamin, and L. Zhang. Host multicast: A framework for delivering multicast to end users. In *IEEE INFOCOM*, pages 1366–1375, New York, USA, 2002.

[110] B. Zhang, S. Jamin, and L. Zhang. Universal IP multicast delivery. In *Network Grouped Communication (NGC)*, Boston, USA, 2002.

[111] B. Zhao, J. kubiatowicz, and A Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California, April 2001.

[112] S. Zhuang, B. Zhao, A. Joseph, R. Ratz, and J. kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV*, pages 11–20, Port Jefferson, New York, USA, 2001.