The Greedy, the Naive, and the Optimal Multicast Routing: from Theory to Internet Protocols

Michalis Faloutsos

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

The Greedy, the Naive, and the Optimal Multicast Routing:
from Theory to Internet Protocols

Michalis Faloutsos

Doctor of Philosophy, March 1999

Computer Science
Univ. of Toronto

ABSTRACT

This work studies the problem of multicast routing both at a theoretical and a practical level. Multicasting involves the distribution of the same data to several receivers at the same time. Efficient multicast routing can reduce the communications cost of the distribution tree. In this document, we extend and study the most widely used routing algorithms in a variety of problems, and we compare their performance theoretically and through simulations. First, we study how the asymmetry of directed networks and the membership behavior affect the performance of these algorithms. We prove bounds for the various cases of these problems in a worst case analysis. Second, we propose algorithms for multicast connections where the users have different quality expectations from the multicast connection. We prove bounds for the performance of our algorithms, and examine their performance in a variety of scenarios through simulations. Third, we propose, QoSMIC, a multicast protocol for the Internet. A major advantage of QoSMIC is that the routing decisions consider the quality requirements of the users. In addition, the protocol limits the role of pre-configuration decisions, which cause problems in the employment of some previous protocols. Another important feature of the protocol is its flexibility; it can scale to large networks, and it can adapt and evolve with the needs of the applications and the requirements of the users.

ACKNOWLEDGMENTS

Approximately 15 years ago my mother, Sofia Kalamara-Faloutsou, decided that ”it would be good for the kids to get PhDs”. Being the third to finish with the fourth one in its final stages, the Plan is almost certain to complete. This just goes to show what a wicked people-reading intuition and a whole lot of optimism can do. My father, Nickos, was of course an accomplish in this business.

Academically speaking, I would like to thank the “Dream Team” which in order of appearance are: Rajesh Pankaj, for all the long discussions, his patience, and warmth, (I think it is about time he learned I had nicknamed him ”honey-bunny” for obvious reasons); Ken Sevcik, for his tremendous support and dedication, and for being the ultimate proof-reading machine, (also for being the only out of four supervisors that did not leave the university before me); Anindo Banerjea, for taking over from Rajesh and showing me that I can easily loose on either side of an argument (realizing that, I would ask him to take the side I would want to support). Part of the Dream Team is also my committee: Derek Corneil, Alan Borodin, Irene Katzela, and Al Leon-Garcia. Special thanks to my external advisor J.J. Garcia-Luna-Aceves, for his comments and positive attitude.

I want to thank Kathy Yen, the Ultimate Power and Supreme Ruler of the Department, for being a friend and a *wonderful* person. Special thanks to Winnie, Joan and Martha.

“One can not live life alone, because it gets lonely”. I would like to thank Petros, for putting up with me the last 28 years, for his unlimited hacking abilities, and for having Harrison Ford’s face and Roberto Benigni’s hair style. Special thanks to Christos, Christina, Maria, Antonis, little Christos, for being a dream family. Thanks to Lili, Klea, Dina, Kostas, and Christos for their love. Special thanks to Alexis for his nomination in the best cousin category.

Friendship is a valuable thing for various reasons according to numerous proverbs that I can’t recall right now. I do recall my friends though, and that’s a start. Special thanks to: Piotrek Gozdyra for all the laughs and alcohol we shared. Special thanks to Florine ”the bunny” Tseu, Birgit Rohde, Karen Block, Melanie Baljko, Joanna Everitt and Dave Modjeska for all the good times. Life in Toronto would not have been the same without all the “greekies”: Panayotis Tsaparas, for being a surrealist and a visioner, Themis Palpanas, for the sound of his laughter, Theo Garefalakis, for his goatee even when he shaves it, Vasilis Tzerpos, for optimizing the labour-gain trade-off of life, Dimitra Vista, for teaching me the fairy-tale of “Kyra-Kota Kyra-Kota”, Spiros Mancoridis, for being a major snake-dealer, Nick Koudas, for being “fevgatos”, Vasso Bartzoka, for laughing quicker than the speed of light, Nick Zahariadis, for his cute smile, Stergios Anastasiadis, for all the “zeimpekiko” dances, Yiannis Papoutsakis, Spyros Angelopoulos, for being the only person that ever read my work at his own will, and finally, the newest promising additions: Periklis “the tennis-player” Andritsos, Yiannis Velegrakis, and Tasos Kementsietsidis, for having the longest name.

And then there is Sherry. Here is to you, for being magical, for listening to all my fairy-tales (including the “Kyra-Kota Kyra-Kota” one), for the unlimited support and optimism, and, for deciding that you are the best.

This may be a long acknowledgment, but it has been a long four years, and I am not done yet.

I would also like to thank my die-hard friends in Greece: Yiannis Tsakos, Alexis Hadjigiannhs, Kostas Tsimidakis, Thanos Stefos, Kostas Stamos, Romina Tsouknida, Penny Anesti (currently in Strasbourg), Penny Strapatsaki, Sokratis Bambakos, Giorgos Sakelaropoulos (now in Luxemburg), Andreas Prodromidis.

I have to recognize the contribution of “Apollo 11” restaurant. Pavlo, thank you for all these Saturday morning breakfasts.

Special thanks to my right wrist for holding on during that final stretch in August of 1998. Finally, I am grateful to my stuffed dog Willy for not being judgemental no matter how badly I

goof up.

To my parents Niko and Sofia

Contents

Table of Contents **vi**

List of Tables **viii**

List of Figures **ix**

1 Introduction **1**
- 1.1 Background and Perspective 2
- 1.2 Motivation 4
- 1.3 Contributions 5

2 Background and Previous Work **8**
- 2.1 The On-line Directed Multicast Problem 10
- 2.2 The Heterogeneous Multicast Problem 12
- 2.3 Internet Multicast Protocols 16

3 Multicast Bounds in Directed Graphs **20**
- 3.1 The Extended Algorithms 22
- 3.2 The Static Problem 22
- 3.3 The Join-Only Problem 25
  - 3.3.1 The Join-Only Upper Bound 25
  - 3.3.2 A Lower Bound for the Join-Only Problem 28
- 3.4 The Join-Leave Problem 36
- 3.5 Conclusions 38

4 Heterogeneous Video Multicast **40**
- 4.1 The Fixed Heterogeneous Problem 40
  - 4.1.1 The Competitive Ratio of Our Algorithms 42
- 4.2 The General Heterogeneous Problem 47
- 4.3 Implementation and Complexity Analysis 49
- 4.4 Conclusions 50

5 The QoSMIC Internet Protocol **51**
- 5.1 Overview 52
- 5.2 Messages and Mechanisms 54
  - 5.2.1 The Search for Candidates 55
  - 5.2.2 Connection Establishment 57
  - 5.2.3 Leaving a Group 58

5.2.4 Candidate Selection. 58
5.2.5 Manager and Inter-domain Issues 59
5.3 Implementation Overview 60
5.3.1 Intree Router State 60
5.3.2 Router Actions 61
5.3.3 Source Specific State 63
5.4 Correctness 63
5.5 The Control Overhead 65
5.6 Conclusions 67

6 Simulation Results **68**
6.1 Overview of our Simulations 68
6.1.1 Graph Models and Scenarios 68
6.2 The Homogeneous Problem (Series I) 74
6.2.1 Observations for Series I 74
6.2.2 The Static Problem 75
6.2.3 The Join-Only Problem 78
6.2.4 The Join-Leave Problem 79
6.3 The Heterogeneous Problem 85
6.3.1 Observations for series II 87
6.3.2 Fixed Problem 87
6.3.3 General Problem 90
6.4 Multicast Internet Protocol 95
6.4.1 Routing Efficiency 95
6.4.2 Overhead Complexity 96
6.5 Conclusions 98

7 Conclusions **99**
7.1 Contributions 99
7.2 Future Work 101

A Additional Simulations for Series I **102**
A.1 The Static Problem 103
A.2 The Join-only Problem 107
A.3 The Join-Leave Problem 112

B Why the On-line Greedy Solution is Able to Exploit Asymmetry **120**

Bibliography **121**

List of Tables

2.1 Explanations of the main symbols and notation in this document. 9

3.1 Explanations of the main symbols and notation in this chapter. (For convenience, we repeat the symbols from chapter 2.) 21
3.2 The worst case bounds of the Greedy and the Naive algorithms for the various multicast problems. 37

4.1 Explanations of the main symbols in this chapter. 41

5.1 The different roles of the routers. 54
5.2 An explanation of the messages of our protocol. 55
5.3 Analysis parameters and their symbols. 64
5.4 Comparing the four Candidate selection methods in the Multicast Tree Search. . . . 66
5.5 A comparison of the search procedures of QoSMIC and YAM. Adaptable complexity type means that during the execution the protocol can limit the overhead while still offering at least one path per join. 66

6.1 The different series of simulations of our work. 69
6.2 Series I: An overview of our simulations for the homogeneous problem. 73
6.3 Series II: An overview of our simulations for the heterogeneous problem. 73
6.4 Series III: An overview of our simulations for the Internet protocols. 73
6.5 The protocols in our simulations. 95

List of Figures

1.1 The Naive, the Greedy and the optimal solutions. We do not show the non-participant nodes. . . . 2
1.2 The conceptual structure of our work. . . . 6

2.1 The behavior of different cost functions for $\beta = 0$ as a function of the used bandwidth. 13
2.2 Happiness values for different users. . . . 14
2.3 A graphic illustration of the optimization problem for one destination. (If we consider multiple destinations, the lines of the figures are not smooth and we have local minima.) 14
2.4 The architecture of the MBone/Internet network. . . . 17

3.1 An example of a bad case of a directed graph for Greedy in the static problem. . . . 24
3.2 A worst case example for the Naive algorithm. . . . 25
3.3 Graphic representation of the proof of theorem 5. . . . 26
3.4 The d-tree : a complete binary directed tree. . . . 28
3.5 The worst case example for highly asymmetric graphs. Edges from or towards the source are of cost c. Elsewhere, curved edges are of cost c, while their opposite edges are of cost ϵ. . . . 28
3.6 The worst case example for highly asymmetric graphs in the join-only problem for any algorithm. . . . 29
3.7 The family of F_g with $k = 2$: F_0 and F_1 30
3.8 The family of F_g with $k = 2$: F_2. The triangles represent d-trees in which the root is a node of the previous generation. The thick edges represent many pairs of edges that we merge for the clarity of the layout. . . . 30
3.9 A worst case on-line sequence for graph F_1 for less asymmetric graphs in the join-only problem for any algorithm. . . . 31
3.10 An example of a bad case for undirected and directed graphs for three join requests. The graph is shown as undirected, so for the directed case we can replace each edge with a pair of opposite directed edges. . . . 35
3.11 An example of a bad case for directed graphs. . . . 35
3.12 The impact of asymmetry: a graphical overview of our results. The bounds of Naive and Greedy for the static, join-only and join-leave multicast problems. . . . 38

4.1 How the heterogeneous problem solved by MaxStream relates to a number of homogeneous ones 43

5.1 An overview of QoSMIC. . . . 53
5.2 The state transition diagram of a link. (Protocol actions, messages or timer expirations, cause the transitions.) . . . 56

5.3 The Take-Over procedure: an intree router that lies in the path from the Candidate to the New router can replace the Candidate. 57
5.4 The entries at the routing table for Shared Trees, Source-Based Trees and their transition. The figure shows the flow of packets from the source. 61

6.1 The 84-MBone: The map of the major MBone routers in 1994 by S. Casner. 69
6.2 Tree Cost versus Asymmetry in the 32-MBone map for the static problem with group-size 8. 75
6.3 Path Length versus Asymmetry on the 32-MBone map for the static problem with group-size 8. 76
6.4 Tree Cost versus Asymmetry on the 32-Euclidean map for the static problem with group-size 8. 76
6.5 Path Length versus Asymmetry on the 32-Euclidean map for the static problem with group-size 8. 77
6.6 Tree Cost versus Group Size in the 32-MBone map for the static problem with asymmetry 10. 77
6.7 Path Length versus Group-Size on the 32-MBone map for the static problem with asymmetry 10. 78
6.8 Tree Cost versus Parameter a in the 32-Euclidean map for the join-only problem with asymmetry 10. 79
6.9 Path length versus Parameter a in the 32-Euclidean map for the join-only problem with asymmetry 10. 80
6.10 Tree Cost versus Parameter b in the 32-Euclidean map for the join-leave problem with asymmetry 10. 80
6.11 Path length versus Parameter b in the 32-Euclidean map for the join-leave problem with asymmetry 10 and group size 8. 81
6.12 Tree Cost versus Asymmetry in the 32-MBone map for the join-leave problem with group-size 8. 81
6.13 Path length versus Asymmetry on the 32-MBone map for the join-leave problem with group-size 8. 82
6.14 Tree Cost versus Group-Size in the 32-MBone map for the join-leave problem with asymmetry 10. 82
6.15 Path length versus Group-Size on the 32-MBone map for the join-leave problem with asymmetry 10. 83
6.16 Tree Cost versus number of requests in the 32-MBone map for the join-leave problem with group-size 8 and asymmetry 10. 83
6.17 Path length versus number of requests on the 32-MBone map for the join-leave problem with group-size 8 and with asymmetry 10. 84
6.18 Actual Group-Size versus number of requests on the 32-MBone map for the join-leave problem with group-size 8 and with asymmetry 10. 84
6.19 Fixed: Gain versus stream ratio in the 32-MBone graph with group-size 8. 88
6.20 Fixed: Tree cost versus stream ratio in the 32-MBone graph with group-size 8. . . . 88
6.21 Fixed: Gain versus asymmetry in the 32-MBone graph with group-size 8. 89
6.22 Fixed: Gain versus group size in the 32-MBone graph with group-size 8. 89
6.23 Fixed: Gain versus maximum number streams in the 32-MBone graph with group-size 8. 90
6.24 General: Gain versus stream ratio in the 32-MBone graph with group-size 8. 91

6.25 General: Total happiness versus stream ratio in the 32-MBone graph with group-size 8. 91
6.26 General: Gain versus stream ratio for the absolute profit Gain function ($Gain_2(H, C)$) in the 32-MBone graph with group-size 8. 92
6.27 General: Total happiness versus stream ratio for the absolute profit Gain function ($Gain_2(H, C)$) in the 32-MBone graph with group-size 8. 92
6.28 General: Gain versus asymmetry in the 32-MBone graph with group-size 8. 93
6.29 General: Gain versus group size in the 32-MBone graph with group-size 8. 93
6.30 General: Total happiness versus maximum number streams in the 32-MBone graph with group-size 8. 94
6.31 Routing Efficiency relative to the Offline multicast tree. 96
6.32 The join distance of a new join for YAM and QoSMIC. 96
6.33 The control overhead of the Candidate search in YAM and QoSMIC. 97

A.1 Tree Cost versus Asymmetry in the 84-MBone map for the static problem with group-size 10. 103
A.2 Path Length versus Asymmetry on the 84-MBone map for the static problem with group-size 10. 104
A.3 Tree Cost versus Asymmetry in the 84-Euclidean map for the static problem with group-size 10. 104
A.4 Path Length versus Asymmetry on the 84-Euclidean map for the static problem with group-size 10. 105
A.5 Tree Cost versus Group-Size on the 32-Euclidean map for the static problem with group-size 8. 105
A.6 Path Length versus Group-Size on the 32-Euclidean map for the static problem with group-size 8. 106
A.7 Tree Cost versus Asymmetry in the 32-MBone map for the join-only problem with group size 8. 107
A.8 Path Length versus Asymmetry on the 32-MBone map for the join-only problem with group size 8. 108
A.9 Tree Cost versus Asymmetry on the 32-Euclidean map for the join-only problem with group size 8. 108
A.10 Path Length versus Asymmetry on the 32-Euclidean map for the join-only problem with group size 8. 109
A.11 Tree Cost versus group Size in the 32-MBone map for the join-only problem with asymmetry 10. 109
A.12 Path Length versus group Size on the 32-MBone map for the join-only problem with asymmetry 10. 110
A.13 Tree Cost versus groupsize on the 32-Euclidean map for the join-only problem with asymmetry 10. 110
A.14 Path Length versus groupsize on the 32-Euclidean map for the join-only problem with asymmetry 10. 111
A.15 Tree Cost versus Asymmetry on the 32-Euclidean map for the join-leave problem with group-size 8. 112
A.16 Path Length versus Asymmetry on the 32-Euclidean map for the join-leave problem with group-size 8. 113

A.17 Tree Cost versus Asymmetry in the 84-MBone map for the join-leave problem with group-size 10. . . . 113
A.18 Path Length versus Asymmetry on the 84-MBone map for the join-leave problem with group-size 10. . . . 114
A.19 Tree Cost versus Asymmetry on the 84-Euclidean map for the join-leave problem with group-size 10. . . . 114
A.20 Path Length versus Asymmetry on the 84-Euclidean map for the join-leave problem with group-size 10. . . . 115
A.21 Tree Cost versus Group-Size on the 32-Euclidean map for the join-leave problem with asymmetry 10. . . . 115
A.22 Path Length versus Group-Size on the 32-Euclidean map for the join-leave problem with asymmetry 10. . . . 116
A.23 Tree Cost versus Group-Size on the 32-Euclidean map for the join-leave problem with asymmetry 10 and group-size 8. . . . 116
A.24 Path Length versus Group-Size on the 32-Euclidean map for the join-leave problem with asymmetry 10 and group size 8. . . . 117
A.25 Tree Cost versus Graph Size for the join-leave problem with asymmetry 10, average node degree and group size equal to one fourth of the nodes of the graph. . . . 117
A.26 Path Length versus Graph Size for the join-leave problem with asymmetry 10 and group size equal to one fourth of the nodes of the graph. . . . 118
A.27 Tree Cost versus average node degree in the 32-Euclidean map for the join-leave problem with asymmetry 10. . . . 118
A.28 Path Length versus average node degree in the 32-Euclidean map for the join-leave problem with asymmetry 10. . . . 119

B.1 On-line can be better than offline. . . . 120

Chapter 1

Introduction

The focus of this work is multicast routing: the distribution of the same data to several receivers at the same time. Many emerging applications are multicast in nature, such as tele-conferencing. We extend and study the most widely used routing algorithms in a variety of problems, and we compare their performance theoretically and through simulations. First, we study the effect of the topological and functional properties of the network on the performance of these algorithms. Second, we propose and evaluate efficient algorithms when the users have different quality expectations from the multicast connection. Third, we identify some limitations in the current multicast Internet protocols, and we propose a new protocol that considers the quality requirements of the users. Throughout our work, we show that different algorithms can vary in their performance significantly.

Multicasting is a more general communication paradigm than the simple one-to-one communication model. Multicasting involves the exchange of information among more than two entities. We can distinguish two cases of multicasting: one source and many receivers (one-to-many), and many sources and many receivers (many-to-many). In this document, we use the term multicast to denote the one-to-many information distribution, although in some cases we overload its meaning to denote also the many-to-many type of communication.

Multicast routing involves the identification and creation of distribution trees for multicast connections. Apart from routing, issues such as admission control, fault tolerance, and pricing are also important aspects for the support of multicast applications, but we deal with them only to the extent that they interact with routing. In routing, on the one hand, we want to minimize the cost of the resources used by each connection. On the other hand, routing must meet user requirements for quality and comply with several practical limitations (e.g., physical, financial). These limitations make the problem more challenging in practice. It can be seen as an optimization problem with various optimization parameters, which are often in trade-off relationships. In addition, in designing protocols, we have to consider issues such as current practices, industrial trends, and compatibility with existing solutions.

Most communication networks have to support multicasting. Several applications are of multicast nature (e.g., tele-education), and supporting them is facilitated by the existence of the multicast abstraction. First, without multicasting, applications have to perform multiple one-to-one transmissions. It is easy to see that this can lead to inefficient resource use or even inability to support the service in some cases. We can visualize the problem better if we assume a tree structured network. When the source performs multiple transmissions of the same information, the edges near the source become overloaded with traffic; destinations compete for the same resources. The number of one-to-one connections is proportional to the number of destinations. Thus, for many destinations, one such multicast connection can congest the links near the source to the point of

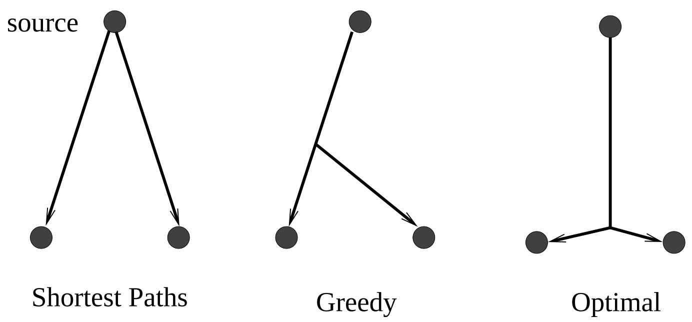


Figure 1.1: The Naive, the Greedy and the optimal solutions. We do not show the non-participant nodes.

failing to distribute the data to all destinations. Second, multicasting is an abstraction that can facilitate the communication. For example, in the Internet, a multicast group address functions as a meeting-place: any user can send information to the participants of the group, without having to know their identity or individual Internet addresses. The use of multiple one-to-one connections does not have this property, and constant update of the membership information could lead to a large control overhead in some implementations.

In the rest of this chapter, we present some general concepts and terminology in section 1.1. We discuss the motivation of our work in section 1.2. Finally, we present the contributions of our work in section 1.3

1.1 Background and Perspective

In this document, we use the term multicast session or connection to identify the distribution of some information. We use the terms **sender** or **source** to denote the origin of the information and **receiver** or **destination** to denote each end-point. A receiver can have expectations of the quality of the connection that we call **Quality of Service (QoS)**. QoS is a very broad concept that can be quantified by performance metrics such as packet-loss ratio, end-to-end delay, bandwidth or combinations of metrics. QoS may be defined differently among applications and users, and so we try to keep the definition of QoS as general as possible. The sender(s) and the receivers of a session are called **participants**. We use the term **network or service provider** to denote the authority that manages the network and provides the connections. We use the notion of algorithm in its usual meaning in computer science, while we use the term protocol to refer to an algorithmic solution closer to an actual implementation. We model a communications network by a directed graph, and a multicast connection can be supported by a tree, in other words, an acyclic connected graph, which is a subgraph of the initial graph.

In its simplest formulation, the constrained multicast routing problem is also known as the Steiner tree problem. In this problem, given a weighted undirected graph and subset of nodes, we want to find the minimum weight tree that spans these nodes. Even this problem is NP-hard [DW72], and so applied research has focused on polynomial approximation algorithms. In this document, we use the term **routing efficiency** of an algorithm to indicate how close its solution is to the optimal one. Efficiency relates to the amount of communication resources (e.g., bandwidth) that are allocated to a connection. In resource allocation, we consider the following two goals that are often conflicting: on the one hand, we want to minimize the overall communications resources we use (e.g., edges, and bandwidth), and on the other hand, we want to balance the utilization across the available resources. Note that resource allocation is tightly coupled with pricing or costing: the estimation of the relative value of a specific resource. For example, an edge may be topologically important, if its absence affects the connectivity of the network. In addition, costing can include other issues such as monetary value of the resources. In our work, we focus on the routing aspect of the problem. Namely, we assume that we are given a cost assignment (or cost functions) for our resources, and our task is to find a tree that minimizes the cost and meets other problem constraints (e.g., users' QoS requirements). Note that for any choice of cost functions, we still have to solve a routing problem any time we accommodate a session, and this is the problem we tackle. In discussing routing protocols, apart from the routing efficiency, we consider a number of issues such as scalability, and adaptability.

In practice, a number of constraints and limitations have to be considered on top of the Steiner problem formulation, as we already mentioned. We can group these constraints into the following categories:

1. *Resource constraints* reflect the attempt of the network manager to minimize the cost of the resources used.

2. *User requirements* express the QoS that the users expect (e.g., bound on expected end-to-end delay).

3. *Implementation limitations* are a consequence of established practices and the underlying technologies (e.g., available information, physical and fabrication considerations).

The first two types of constraints are conflicting since a user would be happy using all the resources for his/her connection, while the network manager may prefer to allocate minimal resources to each session to maximize the number of users. Finally, the third type of constraints affect the types of algorithms/protocols we can employ.

In this document, we study multicast routing for a variety of problems that stem from constraints that belong to all of the above categories. We begin with some theoretical issues that form the basis for the discussion of some more practical problems. We use the insight from the theoretical work in the design of algorithms and protocols.

Throughout our work, we focus on the two most important multicast algorithms from a practical point of view. The first algorithm is based on an efficient heuristic for the Steiner tree problem [TM80]. The main idea is that at each step we connect a destination with the minimum cost path to the existing multicast tree. When we have several destinations, we select the destination with the minimum cost path, and repeat this procedure until we connect all destinations. For the rest of this document, we call this algorithm **Greedy**. The second algorithm takes the union of the shortest paths between the source and each destination, and we call this algorithm **Naive**. Note that, if we break ties between paths of equal cost deterministically, both algorithms create trees.

In our research, we study a number of problems that stem from different constraints and functional requirements.

1. We chose directed graphs as the appropriate model for the underlying network, although most previous theoretical work assumes undirected graphs. We believe that directed graphs model better the independence of the communication links between pairs of nodes. For example, congestion on the link from point A to point B does not necessarily mean that the link from B to A is also congested. Undirected graphs cannot capture this scenario, which can, in general, affect the relative performance of the algorithms.

2. We wished to consider various scenarios for the membership behavior of the users. We examine three types of problems: (i) the *static* problem, where all destinations are known in advance, (ii) the *join-only* problem, where destinations join in an arbitrary way and stay until the end of the session, and (iii) the *join-leave* problem, where destinations can join and leave arbitrarily throughout the multicast session. In the order of presentation, each problem includes its predecessors as special cases. However, all three problems are significant, since they model applications with different functional requirements. We use the term on-line or dynamic to refer to the join-only and join-leave problems.

3. We study the case where we differentiate the quality of the service among users, who have different expectations or tolerances in general. A typical such case is the distribution of video data that has been hierarchically encoded. Hierarchical encoding is a layered way of encoding information, such as video, that can appear in different quality levels. The information is divided into an ordered set of levels and higher levels can only be used on top of the previous one. This way, participants receive subsets of the same information. We call these subsets **streams** and they correspond to different quality levels. Each receiver indicates a range of streams and the resource allocation strategy must offer to the receiver some stream in the range. In its generality, the problem consists of two interrelated sub-problems: a) selecting the quality level for each user, b) selecting a route. This constitutes the **general heterogeneous problem**. If we assume that each user requires one specific stream, we get the **fixed heterogeneous problem**. In contrast, we refer to the one-stream problem as the **homogeneous problem**.

4. In the more practical aspect of our work, we chose to focus on multicasting issues on the Internet. Our goal is to support multicast connections that could meet the QoS requirements of the users. The Internet is fundamentally a best-effort packet switching network. That is, there are no guarantees for the services and applications that run over it; applications may "starve", end-to-end delays may be arbitrarily large, and packets may be lost. This best-effort approach is one of the main weaknesses of the network, and lately a lot of research has been devoted to overcoming this problem. Regarding current research on multicasting, the Internet Engineering Task Force (the principal standards organization for the Internet) maintains a working group for large scale multicast routing under the name Inter-Domain Multicast Routing (IDMR). Several protocols have been proposed, but most of them do not scale well for large networks and groups. In addition, the support of QoS was not even a goal for most of the previous multicast protocols.

1.2 Motivation

Multicasting is necessary for the support of several applications, and a good multicast algorithm can minimize the cost of the communications resources in use. In recent years, there has been an increasing amount of research in multicasting in both scientific and industrial communities targeting applications like tele-conferencing, tele-education, and computer supported collaborative work. Several such applications have been implemented and they are expected to move from proof-of-concept prototypes to commonly used tools. Given that the Internet became a household commodity only in the last five years, and multicast applications are just now appearing as commercial services, we believe that we are at the beginning of a large growth of multicast services.

In our work, we focus on Greedy and Naive, because they are the dominant solutions in current practice. Both algorithms are simple in concept and easy to implement. They also lend themselves to a distributed implementation requiring only minimal information and limited synchronization. This simplicity of the algorithms turns out to be their strongest advantage; they are the predominant routing solutions (as a whole or as building blocks) in most existing and proposed multicast protocols [MS97] [EFH+97] [CC97a]. Between the two algorithms, Naive has been employed more widely.

Our work compares the two algorithms and questions the preference of Naive over Greedy in practice, and particularly in the Internet environment. We believe that the Naive has been the "obvious first" choice, because it introduces minimal complexity and requires minimal information (on top of the information needed for one-to-one connections). Greedy requires more complicated implementations compared to Naive, and we found it important to determine whether the additional complexity of Greedy is justified by the more efficient resource use it attains.

Addressing the heterogeneous problem can satisfy users with variable and conflicting requirements, and can optimize the use of resources. Some users may demand the highest quality and they may be willing to pay for it. Some other times, users may not need or be able to handle the highest quality. For example, a mobile battery-supported user in a video session may prefer to minimize the processing power consumption rather than having a high quality picture, since processing less data can prolong the duration of the battery. The preference in this trade-off may be reversed for a user at a workstation. In such cases, we want to be able to offer variable quality, because otherwise we displease or exclude from the session potential users. The existence of Naive-based algorithms motivated us to examine the use of Greedy for multicast problems with heterogeneous requirements. In the literature, most solutions have attacked the routing and stream-selection problems separately. Given interdependency, it is clear that addressing both problems simultaneously could lead to better performance.

The reason we chose to study the Internet is related to its indisputable significance and its impact in modern telecommunications. More specifically, multicasting over the Internet has been the subject of a lot of research in the networking community. The Internet Engineering Task Force (IETF) has created a working group in multicast routing under the main Inter-Domain Multicast Routing (IDMR), whose responsibility is to provide protocols and propose standards. Despite the intense research activity, there does not yet exist a multicast protocol that promises a "complete" solution.

Our work is motivated by the need to support QoS-sensitive multicast applications. The current protocols do not consider QoS in their routing, and this perpetuates the "no-guarantees" approach of the Internet. We believe that the marketability of any kind of service depends heavily on its ability to provide a level of quality. Currently, the services over the Internet are limited by the best-effort nature of the network. However, the scope of our work extends beyond the current use of the Internet to a fully-commercialized environment with competing service providers. We are convinced that such commercial services will need to guarantee their QoS, and that some users will be willing to pay to have such guarantees. The first step in this direction has already been taken; people pay for Internet access. It is only natural to expect that the requirements of these users will increase in terms of the desired quality. Going even further, we speculate that if the Internet does not manage to provide a commercial environment, it will inevitably be superseded by other networking technologies such as ATM according to the "survival of the fittest" law of evolution.

1.3 Contributions

We arrange our work in the three different sections that build on each other: worst case bounds, heterogeneous multicast algorithms, and Internet protocols (see Fig. 1.2). We can view the first

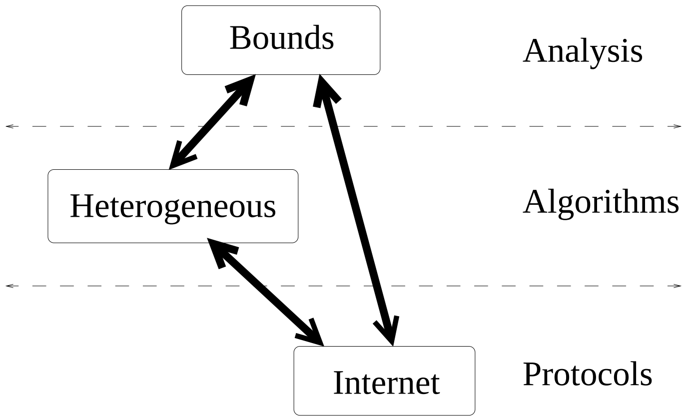


Figure 1.2: The conceptual structure of our work.

two studies as the foundation that guided and shaped the design of our Internet protocol, although the actual interactions were two-way. In all three problems, we propose and examine algorithms based on the Naive and the Greedy approaches, and compare their performance.

Multicast Bounds (Chapter 3). We study the effect of the asymmetry of the network on the performance of the algorithms. We first extend the algorithms to address the on-line problem on directed graphs. We then prove bounds on the performance of the algorithms as functions of the asymmetry and the size of the number of destinations. We prove that Naive has the same tight bound in the static, join-only and join-leave problems. For the first two problems, Greedy outperforms Naive. However, in the join-leave case, Greedy can perform "very" poorly compared to Naive. It is interesting to note that, in the join-only problem, Greedy has near-optimal performance compared to *any* on-line algorithm, which we consider as a strong result in favor of Greedy.

Heterogeneous Multicasting (Chapter 4). We examine how multicast routing can take advantage of hierarchical encoding. Focusing on the static problem, we propose algorithms that address stream optimization and routing problems simultaneously, and show the advantage of such an approach. We suggest the Greedy approach as a basis for more efficient algorithms compared to Naive-based algorithms. In the fixed heterogeneous problem (one stream per participant), we propose Greedy-based algorithms and show that they outperform the Naive-based ones. We formulate the general heterogeneous problem as a two-part optimization problem. We model the gain as the happiness of the users (or the price they pay) and the cost as the resources used in the connection. We demonstrate that optimizing the offered streams is crucial for optimizing the resource use. We show the advantage of our algorithms theoretically in a worst case analysis, and by simulations in a series of scenarios that attempt to model realistic cases.

Multicast Internet Protocol (Chapter 5). We present QoSMIC, a multicast protocol for the Internet that supports QoS-sensitive routing. QoS-sensitive routing is supported by: a) the choice among multiple routing paths, and b) dynamic information on these paths for the selection. This way, when a receiver joins the multicast tree, it is able to choose the most promising among several paths. An especially attractive feature is that QoSMIC includes mechanisms for collecting the necessary dynamic routing information, and so it does not rely on other protocols/mechanisms. In addition, the protocol is designed to be scalable and to minimize the importance of *a priori configuration* decisions, which cause problems in using some previous protocols. Furthermore, QoSMIC is able to adapt to the conditions of the network by balancing the resource trade-offs in a way that favours the scarce resources. For example, it can trade-off link-use efficiency against set-up delay by minimizing the routing selection phase. We present the architecture of the protocol and provide both analytical and experimental results to support our claims.

Chapter 2

Background and Previous Work

In this chapter, we present the necessary background and relevant previous work. The chapter also puts our work into perspective and shows how it fits in the bigger picture of multicast support. We start by presenting a general model that is common in all parts of our work, and we discuss the more specific issues of each part in separate sections: multicast bounds in section 2.1, the heterogeneous problem in section 2.2, and multicast Internet protocols in section 2.3.

The General Model. We assume a weighted graph $G(V, E)$. We denote the total number of nodes by N, and we denote by S, the set of the destinations and the source. We define M to be the number of the join requests plus one[1] and denote the source by s. We denote the **weight or cost of an edge** from node u to node w by the function $e(u, w) : V \times V \rightarrow \mathcal{R}^+$. The weight represents "how much we want to avoid using this edge", and it can depend on issues such as capacity, delay, current use, and monetary cost. In our work, we assume that the weight function is given as we have already mentioned. For the heterogeneous problem, we need to have the weight as a function of the used bandwidth, and we use the function $cost_{(u,w)}(Used, b)$ to denote the cost of using bandwidth b on edge (u, w) when bandwidth $Used$ is already occupied (see section 2.2). The cost of a path (tree) is the sum of the weights of the edges of the path (tree). The distance between two nodes is defined as the cost of their minimum-cost path, and we denote the distance by $d(u, w) : V \times V \rightarrow \mathcal{R}^+$. Similarly, the distance between a node and a tree is the minimum distance between the node and some node of the tree. The distance depends on the direction of a path, and we want to establish connections as directed paths from the source towards the destinations. For convenience, we summarize all the notation in Table 2.1.

A common measure of the quality of an approximation is the approximation ratio or competitive ratio (CR) of a multicast algorithm, which is defined as the maximum ratio of the cost of the tree produced by the algorithm, T, over the cost of the optimal tree, OPT, over all problem instances.

$$CR = \frac{T}{OPT}$$

Similarly, in the on-line problem, the competitive ratio is the maximum ratio of the cost of the tree created by the on-line algorithm over the cost of the tree created by the optimal off-line algorithm at any given moment of the execution.

In comparing the competitive ratio, we use the standard notation for asymptotic bounds: O for upper bounds, Ω for lower bounds, and Θ for tight bounds.

[1]The addition of one accounts for the source, and we do it for compatibility with the definition of M for the static problem.

Symbol	Explanation
$G(V, E)$	Weighted directed graph G with V nodes and E edges
N	Number of nodes, $\lvert V \rvert$.
M	Number of active participants. For the on-line problems: the number of join requests plus one.
S	The set of participants (destinations and source).
A	The asymmetry of the graph.
p	The number of leaves in the optimal tree
s	The source of the session.
$e(v, w)$	The cost of the edge from v to w.
$d(v, w)$	The distance from node v to node w
OPT	The cost of the optimal tree.
T	The cost of the tree created by an algorithm.
CR	The competitive ratio of an algorithm: the maximum T/OPT over all problem instances.
O, Ω, Θ	O upper, Ω lower, and Θ tight bounds.

Table 2.1: Explanations of the main symbols and notation in this document.

As we already mentioned, various constraints and factors create a number of multicast problems each with a different focus. As a consequence, the simple model for the Steiner-tree problem becomes more complicated. We outline the different aspects (constraints and scenarios) of multicasting that we consider in our work.

- *Cost.* The cost or the weight of each edge is an indication of how much we want to avoid using the edge. The cost is a function of several factors including the utilization of the edge, and its capacity. The goal is to minimize the total cost of the tree. By defining appropriately the cost, minimizing the cost corresponds to optimizing the way we use our resources.
- *Asymmetry.* The graph that we use to represent the communications network is directed and asymmetric in general.
- *Dynamic Membership.* Participants may join and leave during the lifetime of the session.
- *Variable User Requirements.* Receivers participate at various levels of quality by receiving different amounts of data. This corresponds to the heterogeneous problem.

In addition, we examine Internet protocols in our work. The model becomes more complicated including all of the above variations as possible requirements as we will see in section 2.3.

In this document, we will use and refer to some algorithms that we discuss here.

We consider a Greedy-based algorithm for the static problem, that we call **Random Participant First (RPF)**. Given a graph and a set of participants, we start from the root, and at each step, we pick a destination randomly and connect it with the minimum cost path to the tree. We use the term **random order** to denote the order in which the destinations are selected. RPF corresponds to an instance of Greedy solving an on-line problem in which destinations join in the random order. Imase and Waxman [IW91] prove that the competitive ratio of RPF in an undirected graph is bounded as follows:

$$CR_{\text{RPF}} \leq \log(M)$$

RPF will facilitate the discussion of some proofs relating to the heterogeneous problem.

We consider an algorithm that is like Naive except that it considers the paths in the opposite direction than Naive. Namely, the algorithm finds the shortest path from each destination *towards* the source, and the multicast tree is the union of these paths. We call this algorithm **Reverse Shortest Paths**. Clearly, Reverse Shortest Paths and Naive create the same tree in a symmetric graph. However, in asymmetric graphs, Reverse Shortest Paths can perform poorly, since a low cost path one way may be very expensive when traversed the other way. Although the algorithm seems counter intuitive, it was the basis of the first multicast algorithm in the Internet [DC90] and it is still being used [MS97]. The reason is that, it is easier and faster for a destination to identify the destination-to-source path than the source-to-destination path (see section 2.3). It seems that the supporters of this protocol were expecting small asymmetry on the Internet.

2.1 The On-line Directed Multicast Problem

In this section, we quantify the asymmetry of a graph and define the on-line directed problem. This problem is a generalization of static and/or symmetric variations of the problem. We then present previous work on various special cases of the on-line directed problem.

In practice, a communications network can usually be modeled by a directed graph where every bidirectional link corresponds to two opposite directed edges. It is important to have an accurate model in designing and estimating the performance of algorithms. As we see in this work (chapter 3), the asymmetry has a significant impact on the performance of the algorithms.

We quantify the asymmetry of directed graphs with the following metric [Ram96] [FPS96].

Definition 1 *The asymmetry of a graph $G(V, E)$, A, is defined as*

$$A = \max_{v,w \in V} \left(\frac{e(v, w)}{e(w, v)} \right)$$

With the above definition, the undirected case corresponds to an asymmetry of $A = 1$. Without loss of generality, we can assume that there are no multiple edges in the same direction between each pair of nodes. We call **opposite** edges the pair of edges between two nodes, i.e., for $v, w \in V$, (v, w) and (w, v) are opposite edges.

In a dynamic environment, the participants of a session change during its lifetime. In practice, many applications and network environments are of a dynamic nature; destinations are allowed to join and/or leave during a session. According to the degrees of freedom allowed to the users, we have the static, the join-only and the join-leave problems. As the freedom allowed to the users increases, it becomes more difficult to find a good approximate solution.

PROBLEM DEFINITION. In its generality, the on-line directed multicast problem can be formulated as follows. Assume a directed weighted graph in which some node is designated as the source of the multicast connection. The requests arrive in a sequence of sets, and we have to accommodate each set before the next set arrives. We want to connect each joining node with a directed path from the source. The goal is to minimize the average cost of the multicast tree over the duration of the session. The static problem corresponds to the case where we have one set of join requests. The join-only problem has multiple joining sets but no leave requests.

Assumptions. We examine the worst case performance of the algorithms. We consider deterministic algorithms only, and so our worst case is equivalent to the case where we are so "unlucky" that our routing decisions are the worst possible with respect to the future requests. Thus, in our examples, once we have routed a destination, the on-line sequence will select the "worst" node as the next destination. Without loss of generality, we consider only on-line algorithms that create trees by adding a path for each new destination[2] [Bor97].

In addition, we do not consider rerouting in our work; once we establish a path from the source to a destination this path does not change, until the destination leaves the session. Rerouting can minimize the cost of the on-line tree, by rearranging the current distribution tree. However, rerouting interrupts the communication, it can cause loss of data, and it may degrade the performance below the user's QoS requirements where applicable. For these reasons, rerouting is highly undesirable or it is not provided for at all in many applications and network technologies (e.g. ATM networks) [DDC97]. Assuming that we are able to reroute, we need to identify the criteria and the extent (global versus local) of an efficient policy. This forms an interesting optimization problem which remains an open research problem [IW91] [KK95] [BV97].

We present the previous work for each of the three problems. Note that there are no previous results for directed graphs for either the join-only or the join-leave problems.

For the static directed problem, Ramanathan [Ram96] proved that $2{\cdot}A$ is an upper bound for the Greedy algorithm. That work also proves that a constant-factor approximation algorithm is highly unlikely to exist by a reduction from the Minimum Set Cover problem. In that work, Ramanathan proposed a greedy framework that unifies Naive and Greedy. In the unified algorithm, a parameter controls the number of nodes a new destination can consider joining to: only the source in Naive, and all connected nodes in Greedy. Wong [Won84] proposed a heuristic based on linear programming without offering any bounds. For the static undirected problem, the competitive ratio of the algorithm is bounded by two [TM80].

For the *undirected join-only* problem, Bharath-Kumar and Jaffe [BKJ83] and later Imase and Waxman [IW91] proved that the competitive ratio of the Greedy is tightly bounded by[3] $\log(M)$. Westbrook and Yan [WY95] refined this bound to $O(\log(\frac{d_{max}}{OPT}{\cdot}M)$, where d_{max} is the maximum distance among the destinations and/or the source. Taking a different angle, Awerbuch and Azar [AA95] study the join-only problem for multiple interleaved sessions on undirected graphs, and propose an $(\log N \log M)$-competitive algorithm with respect to the minimization of congestion on the bottleneck link. The goal there is to minimize the maximum edge congestion over a sequence of interleaved sessions. A survey is provided by Awerbuch [Awe96].

For the **undirected join-leave** problem, Imase and Waxman [IW91] prove that M is a lower bound. (We increase this bound to 2^M in section 3.4.) Several experimental studies of this problem exist [DL93] [Wax93] [Wax88]. In all these studies, the graphs topologies are based on Random Euclidean which are formed by randomly distributing points in a Euclidean plane and then introducing edges between nodes favoring proximity (for more details see section 6.1). Several studies [DL93] [Wax93] use hierarchical graphs where we start from a Random Euclidean and we replace each node with a lower level Random Euclidean graph. The selection of the destinations that join and leave is random. The studies suggest that Naive and Greedy have equivalent performance in the join-leave problem [Wax93], and that Naive performs between 10% and 60% worse than a

[2] An on-line algorithm could create more than one path for one node to facilitate future connections. However, we do not gain anything in the long run, since we can create the extra path (the part of it that is actually needed) the moment we need it.

[3] All the logarithmic functions in this chapter have a base of two. This is not important in asymptotic complexity but, as we mention at the end, the bounds that we discuss do not hide large constants.

Greedy off-line solution [DL93]. The maximum difference appeared for networks with high average degree (number of adjacent edges of destinations).

2.2 The Heterogeneous Multicast Problem

In this section, we define the heterogeneous problem. We need to define carefully the cost and the gain from offering a certain quality of service to a user. We also present previous work in the area.

In the hierarchical encoding scheme, the information is divided in **L levels** denoted by l_1 to l_L. Lower levels are fundamental and higher levels can only used on top of all lower levels. Stream s is defined as the union of the first s levels $s : l_1 \cup l_2 \cup ... \cup l_s$. Clearly, the resource requirements of stream s increase with the stream index. We define the **bandwidth ratio of streams (θ)** as the ratio of the bandwidth requirements of consecutive streams. For simplicity, we assume that all consecutive streams have the same bandwidth ratio. We assume that each destination defines a range of preferred streams and participates in the session with one of them. Given the preferred ranges, the network manager has to decide which stream to offer to each destination and how to route each one.

Defining the Routing Cost. We want to consider more carefully the definition of the routing cost, $cost_e()$ of an edge, e, in the heterogeneous problem. Considering the bandwidth, the routing cost depends on the relative values of the edge capacity Cap, the bandwidth in use $Used$, and the bandwidth we want to route b. In more detail, the routing cost becomes infinite when $Used + b > Cap$. Furthermore, it is reasonable to expect that the routing cost increases as bandwidth increases. Defining a cost function for every edge of a network, is a non-trivial task for the management of the network, as we already mentioned in section 1.1. However, for any choice of cost functions, when we route a new session, we have to solve a routing problem as we formulate it here. In other words, given routing functions for the edges, the routing algorithms will attempt to minimize the cost of the multicast tree.

There are several functions that one can use for the routing cost of an edge. The simplest one is to count each edge as one unit of cost:

$$cost_e(Used, b) = \begin{cases} 1, & \text{if } b + Used < Cap \\ \infty, & \text{otherwise} \end{cases} \tag{2.1}$$

This function leads to routing based only on path-length or hop-count independently of utilization. It may seem as a poor choice of function, since it does not balance the traffic load. However, it is being used in many Internet routing protocols. The advantage of this function is its simplicity in the calculation; we do not need to know the existing load of each edge.

Most work on admission control and costing considers the capacity and the current use of an edge in its cost [Plo95] [AA95] [GHP98] [GKR95]. Plotkin [Plo95] offers a comprehensive survey of the issue of competitive routing, admission control, costing, and different optimization models for these problems. We present the functions that are commonly used in their most general format.

The exponential function:

$$cost_e(Used, b) = \begin{cases} e^{\alpha\cdot(Used+b)} - \beta\cdot e^{\alpha\cdot Used} + \gamma, & \text{if } Used + b < Cap \text{ and } \alpha, \beta, \gamma \in \mathcal{R} \\ \infty, & \text{otherwise} \end{cases} \tag{2.2}$$

The inverse available bandwidth function:

$$cost_e(Used, b) = \begin{cases} \frac{\alpha}{Cap-(Used+b)} - \frac{\beta}{Cap-Used} + \gamma, & \text{if } Used + b < Cap \text{ and } \alpha, \beta, \gamma \in \mathcal{R} \\ \infty, & \text{otherwise} \end{cases} \tag{2.3}$$

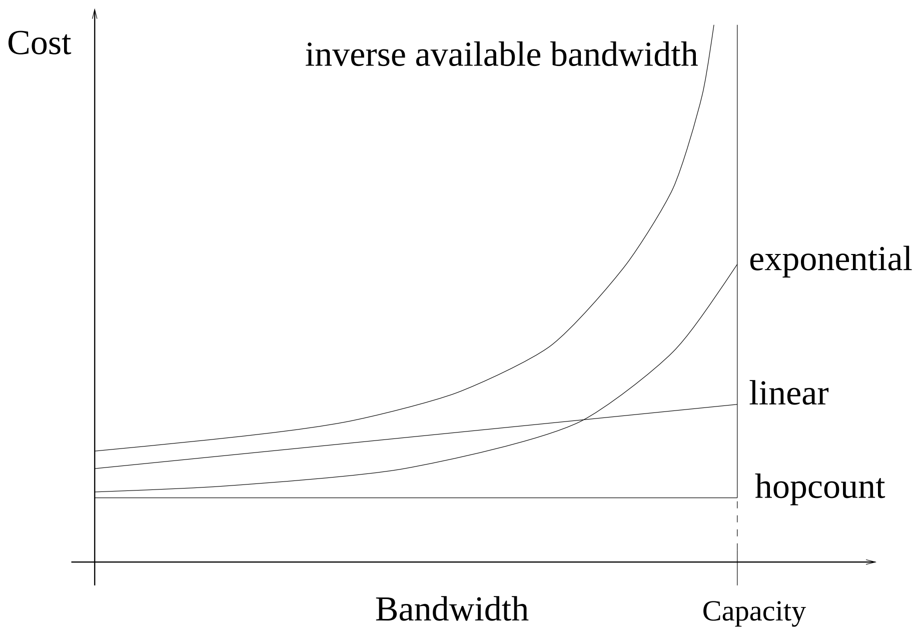


Figure 2.1: The behavior of different cost functions for $\beta = 0$ as a function of the used bandwidth.

The linear function:

$$cost_e(Used, b) = \begin{cases} \alpha \cdot (Used + b) - \beta \cdot Used + \gamma, & \text{if } Used + b < Cap \text{ and } \alpha, \beta, \gamma \in \mathcal{R} \\ \infty, & \text{otherwise} \end{cases} \quad (2.4)$$

In figure 2.1, we show the behavior of the functions above as a function of the used bandwidth for $\beta = 0$. Note that for $\alpha = \beta = 0$ and $\gamma = 1$ the above functions behave like the hop function 2.1. For $\alpha = \beta \neq 0$ and $\gamma = 0$, the functions return the difference of the new total bandwidth $(Used + b)$ minus the currently used bandwidth. Function 2.2 increases exponentially with the $Used$ bandwidth, and appears in many theoretical studies for competitive routing and admission control [Plo95] [AA95] [GHP98] [GKR95]. Function 2.3 demonstrates a smooth increase when used bandwidth increases from zero to Cap. Function 2.4 increases linearly with the bandwidth, and it has the property that routing equal amounts of bandwidth increases the cost of the edge by the same amount independently of the $Used$ bandwidth. The use of this function seems suitable only when the network does not operate near saturation; other mechanisms (e.g., monetary cost) will enforce a balance between the demand and the available resources [Max97]. In our work, we assume a cost function, and we try to minimize the cost of the multicast tree for that function. This way, our algorithms do not restrict the choice of a cost function.

We define the **joining cost** $cost_v(s, T)$ **of node** v to be the cost of connecting destination v with stream s to the multicast tree T. The joining cost is the sum of the cost of routing the required new bandwidth on each edge of the path. When a destination connects to a node of the tree w that carries a stream lower than the desired one, we have to bring the higher required levels to node w. The addition of levels along a tree branch is associated with a cost that we denote by **overrouting cost**. We define the joining cost to include the overrouting cost.

Defining Happiness. We need to include in our model the motivation of the network provider to offer higher streams. The service provider must have a benefit by offering higher streams or else s/he only needs to offer the lowest streams. We introduce the concept of **happiness**, which represents

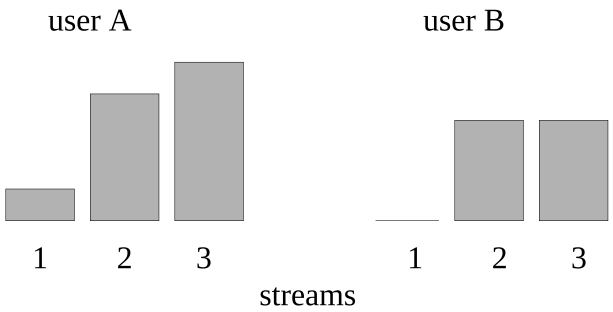


Figure 2.2: Happiness values for different users.

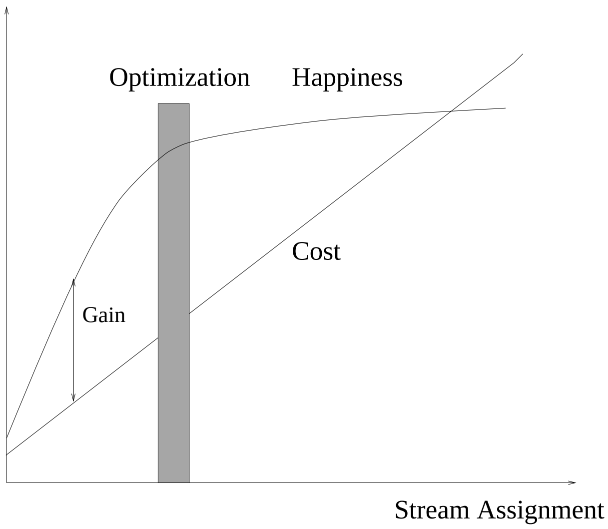


Figure 2.3: A graphic illustration of the optimization problem for one destination. (If we consider multiple destinations, the lines of the figures are not smooth and we have local minima.)

the user-satisfaction that corresponds to a certain stream. Alternatively, user-satisfaction can be interpreted as a price that the user is willing to pay for a certain stream. For each destination v, a happiness value $hap_v(s)$ is associated with stream s, and these values are non-decreasing with stream (see Fig 2.2). Furthermore, for different destinations the happiness values of a stream may not be the same. We assume that every stream for every destination has a happiness value. This is a broad formulation, since we can model various scenarios. For example, if we want to force the algorithm to offer a preferred stream to all destinations, streams below the preferred streams may have a happiness value of minus infinity. Naturally, the total happiness of a session is the sum of the happiness values of all destinations.

Given the above formulation, the general heterogeneous problem becomes an optimization problem with two competing parameters. On the one hand, we want to minimize the total cost, $Cost$, and on the other hand, we want to maximize the total happiness Hap. Given two different solutions, we need to be able to compare them. For this, we introduce the $\mathbf{Gain}(\mathbf{Hap}, \mathbf{Cost})$ function that measures the "gain" by combining the two parameters into a single objective value, which we denote by **Gain**. By definition, the function measures the efficiency of the solution: large happiness and low cost correspond to large values of the function. Figure 2.3 illustrates intuitively the stream-optimization problem by plotting the happiness and the cost of a solution (y-axis) versus the stream assignment. In our work, we assume that the function is given as part of the problem definition. In a real application, however, the service provider has to choose this function, and the choice is crucial for effective network use. The function should model the actual dynamics of the application: resource availability, service demand, number of expected sessions and customers per session.

PROBLEM DEFINITION. We can now define the heterogeneous problem in its full generality. The input is a directed weighted graph, a source, a set of destinations with happiness values for every stream, and an objective function $Gain()$. We want to find the tree and the stream assignment for which happiness and cost maximize the objective function. We assume that each user indicates a range of streams and we have to connect each user with one of its desired streams. This constitutes the **general heterogeneous problem**. If we assume that each user requires one specific stream, we get the **fixed heterogeneous problem**. In contrast, we refer to the one-stream problem as the **homogeneous problem**.

The fixed heterogeneous problem has received little attention until recently. The first work was conducted by Shacham [Sha92], and he proposed an algorithm that we call **Naive-high** or **Naive-hi**. The algorithm is an extension of Naive and thus its competitive ratio is at least that of the homogeneous case, which is $O(M)$. Each destination receives the highest stream that can possibly be routed from the source to the destination. This approach avoids the stream optimization problem by assuming that a higher stream provides enough more happiness to a user that we always offer that stream as long as there is enough bandwidth. This solution does not consider that efficient use of the network may require optimization of the offered streams. For example, offering the highest stream to a distant destination may be possible but not beneficial, because the routing cost may be higher than the related happiness. The complexity of the algorithm is polynomial and bounded above by the complexity of a shortest paths algorithm ($O(N^2)$). The heterogeneous on-line routing problem has not been studied in either the join-only or the join-leave variations.

Previous work on stream-optimization does not address the routing problem, which is assumed to be solved independently; the topology has already been reduced to a tree. A receiver-driven stream-optimization method has been proposed by McCanne et al. [MJV96] for the Internet environment. Receivers drop to lower streams when the path they use is congested, and upgrade to higher streams when the path has available bandwidth. The receiver bases the decision on the packet-loss ratio the connection is experiencing. The method is scalable and easy to implement, but it does not seem to be effective in large networks in practice, as shown by measurements on the MBone by Handley [Han97]. Finally, Shacham and Meditch [SM94] examine the problem of routing unrelated streams to destinations over a given tree, and they propose an algorithm, whose complexity is exponential in M.

The heterogeneous problem is based on the assumption that all the levels of a session have to use the same route. The assumption stems from constraints and practical issues that appear in real networks such as the Internet. All previous work [Sha92] [Max97] [FPS97b] makes the same assumption. If levels can follow different paths, we can perform a number of homogeneous multicast sessions one for each level. However, distinct paths will introduce different end-to-end delays, and the packets of different levels must be resequenced. This resequencing involves extra processing at the receiver and possibly degrades the performance when any of these paths is congested. In addition, maintaining distinct multicast trees requires more routing information that has to be kept in the routers of the network. On the other hand, routing the levels together can facilitate the network management through the enforcement of priority policies: higher levels will be discarded first, thus achieving graceful degradation of quality under heavy load.

2.3 Internet Multicast Protocols

In this section, we present a brief overview of the Internet, the current multicast protocols, and identify a set of parameters that we consider in the design of our protocol.

The Internet is a packet-switching network that principally provides best-effort service. That is, there are no guarantees for the services and applications that run over it; applications may "starve", end-to-end delays may be arbitrarily large, and packets may be lost. The routing is performed with the use of look-up tables residing in each router. Traditional Internet routing protocols do not consider Quality of Service (QoS) metrics. For the support of multicast connections, a multicast backbone (MBone) was introduced as a virtual network "on top" of the Internet in 1992 [Eri94]. The MBone is expected to be phased out, as the routers in the Internet become multicast capable. In the rest of this document, we use the term **router** to imply a "multicast capable Internet router", unless otherwise stated.

Conceptually, the structure of the Internet can be viewed as having three levels (see Fig.2.4). The network consists of routers that are switching devices and communication links. The workstations (hosts) of the users are connected to a Local Area Network (LAN) such as Ethernet. Each LAN has a *Designated router* which communicates directly with each host using the Internet Group Membership Protocol (IGMP) [Dee89]. This is the first or **LAN level**. We use the term **Destination** to refer to the designated router that has a group-member in its LAN. The designated router is connected with other routers forming domains, which is the second or **intra-domain level**. The domains are interconnected with routers that are called **Border** routers. The network of domains is the third or **inter-domain level**. This creates a three level hierarchy that allows the co-existence of different technologies and facilitates the scalability of the network. We can have different protocols at the intra-domain and inter-domain levels, as long as they can interoperate. Routing at the inter-domain level imposes more restrictions due to scalability considerations.

A multicast group is associated with a Class-D or multicast address. A host knows the address of the group it wants to join through an advertising or query mechanism such as the Session Description Protocol (SDP) [HJ95]. A multicast group can have multiple sources and the distribution of the packets can be done in two ways. First, each source can create its own distribution tree, called a **Source-Based Tree**, with itself as the root. Second, all sources can distribute their packets using the same tree, called a **Shared Tree**. Source-Based Trees have better end-to-end performance (e.g. lower delay), and distribute the traffic of each group across the network, but lead to large routing tables[4] [BCFG+97] [WE94]. On the other hand, Shared Trees concentrate the traffic of a group onto a few links in the network. This concentration is bad in the case where all sources are active simultaneously (e.g. Distributed Interactive Simulation), but it can be beneficial when sources take turns in transmitting (e.g. audio-conferencing). The two approaches have complementary behavior, and are both useful in different situations.

A multicast protocol that considers QoS in its routing phase can create a tree better suited to the needs of QoS-sensitive applications. However, the straightforward approach of exchanging dynamic link-state metrics to compute QoS based routes does not scale to large networks. We can distinguish two categories among Internet multicast protocols based on QoS considerations: QoS-oblivious protocols [Bal97] [EFH+97] [TEM97] [PGLA97] [SGLA97] [SGLA98]; and QoS-sensitive protocols [CC97a] [ZES97]. It is important to note that we usually associate the QoS support with scalability problems since QoS requires connection specific information. However, these concerns are justifiable only in point-to-point connections, for which routing tables do not maintain a connection-specific

[4]Source-Based Trees require a routing entry per source per group, while Shared Trees require a single routing entry per group.

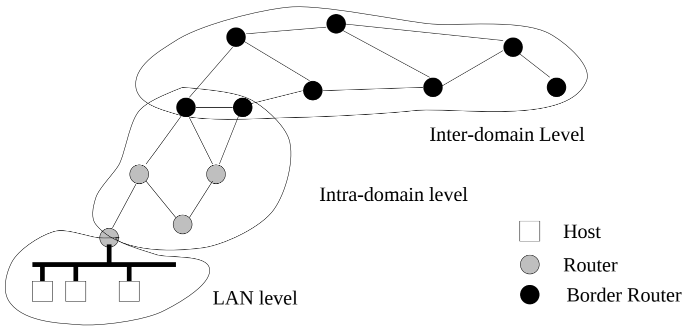


Figure 2.4: The architecture of the MBone/Internet network.

state. However, multicast tables keep connection specific state in routers anyway; namely, the multicast group address refers to a connection. Thus, adding QoS connection specific information is straightforward and increases the size of the routing table (where the routing state is kept) only by a small fraction.

We compare paths in terms of their ability to support an application at a specific QoS level. Quality of Service (QoS) denotes the user-perceived quality. Recently, Zappala *et al.* [ZES97] used the term **Quality of Route (QoR)** to refer to static parameters of the route (e.g. link capacity, delay, or reliability), and this was adopted in the YAM protocol [CC97a], which we discuss below.

In comparing protocols, one has to consider more parameters than just the routing efficiency of the algorithm they implement. The performance of protocols includes different aspects, which are often in trade-off relationships. Usually, protocols are optimized for a subset of such aspects, but not all of them. We list the most important aspects of multicast Internet protocols.

1. *QoS Support.* The protocol should be able to support the various QoS requirements of users.

2. We want to minimize the need for *pre-configuration decisions and manual intervention.* The protocol should not rely on decisions that are taken before the routing of sessions. In addition, we want to eliminate the manual intervention for operational and maintenance functions. We will see examples of such pre-configuration issues when we discuss the current protocols.

3. *Flexibility - Adaptability* measure the ability of the protocol to function efficiently in as wide a range of conditions as possible. The topology and the nature of networks change rapidly, applications have variable requirements and needs, and the behavior of users is difficult to predict.

4. The routing efficiency of a protocol can be measured by the *routing cost* of its distribution tree that we defined in the previous section.

5. The *end-to-end delay* is defined as the time it takes a packet to travel from the source to a destination router.

   The *maximum and average end-to-end delay* over all source and destinations pairs could be used to characterize the end-to-end performance of the tree.

6. The *traffic concentration* is defined as the maximum link load divided by the average link load. The load of a link may be defined in a number of ways, such as the total bandwidth used along a link, or the number of connections going through the link [BCFG$^+$97].

7. The *control overhead* represents the number of control packets that are required to establish and maintain a multicast connection.

8. The *routing table* is the table maintained by each router to support multicasting.

9. The *set-up time* is the time it takes a user to join a group, the time from the join request until the arrival of the first data packet. We can either try to minimize the average set-up time or have a bound on the maximum set-up delay.

10. *Scalability* expresses the ability of the protocol to scale well for large and dense networks, many groups, and large group-sizes.

11. *Robustness* represents the ability of the protocol to overcome failures and malfunctions.

12. *Loop-freedom.* A protocol should not create cycles.

QoS-oblivious protocols. These protocols provide one route when a new member joins, and QoS is not considered in the selection of the route. Examples of such protocols are Core Based Trees (CBT) [Bal97], Protocol Independent Multicasting - Sparse Mode (PIM-SM) [EFH$^+$97], Border Gateway Multicast Protocol (BGMP) [TEM97], Multicast Internet Protocol (MIP) [PGLA97], Ordered Core Based Trees [SGLA97], and HIP [SGLA98]. All these protocols share some characteristics. We describe the BGMP protocol in more detail [TEM97], and then present the other protocols.

BGMP creates **rooted** trees with a ***core* router** as the root of the tree: sources send data to the *core* and destinations receive the data from the *core*. In BGMP, before any multicast group action, we assign sets of multicast addresses to different domains according to the use they expect to have. This may be done with the MASC protocol [EHKT97] which is currently under development in the Internet Engineering Task Force (IETF) community. The domain manager assigns to a user that creates a multicast group a multicast address that "belongs" to the domain. When a new destination wants to join, it sends a join request to the multicast group. Given the address-location dependency, the message is forwarded to the "owner" domain. The path that the join request traveled is the shortest path from the destination to the core, and the join message installs routing state. The routing state is an entry in the routing table, so that the router will know where to forward incoming data packets. Eventually, the message reaches the core or another router in the tree, and data packets are forwarded to the new destination along that path. Similarly when a destination leaves a session, it sends a leave message along the same path, deleting the routing information at the routers that are not servicing other destinations. By default, the distribution tree is a Shared Tree . However, destinations can connect directly to the Source-Based Tree for highly active sources. A destination monitors the intensity of the traffic from each source of the group, and when this exceeds a threshold, the destination joins the Source-Based Tree of that source, and stops receiving the data from that source on the previous Shared Tree path (see section 5.3 and Fig. 5.4). The reason for this transition is that a small number of highly active sources may congest the edges of the Shared Tree[5].

Apart from BGMP, CBT, PIM-SM and MIP protocols create rooted trees. The *core* is chosen in the beginning, and it does not change until the end of the session. In all these protocols, the routing is based on the Reverse Shortest Paths algorithm: the tree is the union of the reverse shortest paths

[5]The reason for not having only Source-Based Trees is that this would increase the routing information that routers must maintain, from one entry per group (Shared Trees) to one entry per source per group (Source-Based Tree).

between the *core* and each destination. The protocol uses Reverse Shortest Paths, because it is easier than finding the forward shortest paths: each new destination can find the shortest path towards the source using the unicast routing database. CBT creates only Shared Trees, while PIM-SM and BGMP start with Shared Trees and permit the creation of Source-Based Trees for very active sources (high data rate). The MIP protocol introduces new approaches to administrative and correctness issues, and guarantees loop-free Shared Trees and Source-Based Trees. The OCBT and HIP protocols combine ideas from MIP and CBT and focus on creating loop-free trees in a hierarchical way.

The use of rooted trees and Reverse Shortest Paths simplifies routing, but has a number of disadvantages that stem from "tying" the distribution tree to a core. First, we need efficient ways to select the core, since the position of the core affects the performance of the tree. For example, consider a group whose participants are in Canada and the core is in Greece. Second, assuming dynamic membership, the initial choice of a core may be suboptimal later on. Note that we can not switch to another core easily, since this would involve removing the old tree and establishing a new one[6]. In our previous example, assume a group whose initial participants are in Greece but then they lose interest, while some Canadians become interested and continue the multicast group: having only one core is bound to be suboptimal for one of the two phases of the group. Third, if the neighborhood of the core is congested, the multicast group will suffer performance degradation, since it is not easy to change core. As we mentioned, PIM-SM and CBT rely on an external mechanism for the core selection. BGMP addresses the selection indirectly; each domain "owns" a number addresses and is the root of any distribution tree with such an address. This, however, raises the problem of partitioning the addresses among the domains. The problem is not trivial, since the needs of domains for addresses may vary over time [EHKT97].

QoS-sensitive protocols. These protocols offer multiple routes to a new member, who selects the best such path. Carlberg and Crowcroft [CC97a] [CC97b] suggested the **YAM** protocol, which creates a Shared Tree considering multiple routes. The destination router searches its neighborhood and finds routers that are part of the tree of the desired group. This way, the new router selects the most promising route. Routing in YAM relies mainly on this neighborhood search, but this procedure can increase the control overhead significantly. Zappala et al. [ZES97] suggested a multicast protocol that provides alternate routes to avoid a bottleneck link. This protocol does not seem applicable in the case where QoS suffers from several mildly congested links. Finally, both protocols use only static QoR metrics.

The YAM protocol can be seen to implement the Greedy heuristic. The Greedy routing scheme has been shown to outperform the commonly used Naive routing in various analytical [TM80] [IW91] [FPS97a] and experimental [Wax88] [DL93] [Wax93] studies, as we indicated in section 2.1.

[6]The current protocols follow this delete-all approach for simplicity. Re-using parts of the existing tree introduces the risk of creating cycles, but avoiding cycles can increase the complexity of a protocol significantly.

Chapter 3

Multicast Bounds in Directed Graphs

In this chapter, we study the bounds for the Greedy and Naive algorithms on directed graphs in a dynamic environment. We compare the algorithms with respect to the cost of the multicast trees they create. First, we show how the Greedy and Naive algorithms can be extended for these problems. Recall that in the multicast literature, there has been some work on the static problem [Ram96], but we have found no results for the dynamic problems on directed graphs.

The contributions of our work are summarized in the following points. Recall that M is the number of join requests and A is the asymmetry of the graph.

- For the *static* problem, we prove a tight bound, $\Theta(\min(A, M))$, for the Greedy algorithm.
- For the *join-only* problem, we prove that Greedy has near-optimal on-line performance. Specifically, Greedy is $\log(M+1)$-competitive compared to *any* on-line algorithm.
- For the *join-leave* problem, Greedy has a large competitive ratio. We prove that the Greedy competitive ratio is proportional to the asymmetry and exponential in M: $\max(A, 2^M)$.
- We prove that the competitive ratio of the Naive algorithm has the same tight bound of M for all problems, and show a worst case example.

In our conclusions in section 3.5, we present an overview of our results in Table 3.2 and an intuitive graphical representation of them in Fig. 3.12.

Putting things in perspective, our work is a generalization of the work of Imase and Waxman [IW91] on undirected graphs. Furthermore, our results subsume their results for symmetric graphs $(A = 1)$. Early versions of our work on the static problem [FPS96] and the on-line problem [FPS97a] have been presented in conferences.

This chapter is structured as follows. In section 3.1, we present our model and previous work. In section 3.2, we discuss the static problem and prove a tight bound for the Greedy algorithm. We also show the tight bound of the Naive algorithm for all problems. In section 3.3, we address the join-only problem. We present an upper bound for the Greedy algorithm (in Subsection 3.3.1), and we prove a lower bound for *any* on-line algorithm for the join-only problem in Subsection 3.3.2. In section 3.4, we discuss the join-leave problem, and we prove a lower bound for the Greedy algorithm. In section 3.5, we give an overview of all the known results and we extract some guidelines for the design of algorithms and protocols.

Symbol	Explanation
$G(V,E)$	Weighted directed graph G with V nodes and E edges
N	Number of nodes, $\lvert V \rvert$.
M	Number of active destinations plus the source. For the join-only: the number join requests plus one.
S	The set of destinations and the source
A	The asymmetry of the graph.
p	The number of leaves in the optimal tree
s	The source of the session.
$e(v,w)$	The cost of the edge from v to w.
$d(v,w)$	The distance from node v to node w
OPT	The cost of the optimal multicast tree.
T	The cost of the multicast tree created by an algorithm.
CR	The competitive ratio of an algorithm: the maximum T/OPT.
O, Ω, Θ	O upper, Ω lower, and Θ tight bounds.
d-tree	A complete binary directed tree
F_g	The graph of **generation** g from the family of graphs that are used in the derivation of the lower bound for the join-only problem.
g, g_m	The generation of an F_g graph and the maximum number of generations in a specific example.
$k+1$	The depth of the d-tree, used in the creation of the F_g graphs. Eventually, we set $k = A$.
l	The layer of a node in F_g defined as the distance in number of hops of the node from the source.
$v_{g,l,x}$	A node of F_g of layer l, and with an in-layer identifier x included only for completeness.

Table 3.1: Explanations of the main symbols and notation in this chapter. (For convenience, we repeat the symbols from chapter 2.)

3.1 The Extended Algorithms

In this section, we briefly remind the reader of our model and previous work. We present the generalizations of Greedy and Naive, since they were intended as solutions for the Steiner tree problem. We list the notation of this chapter in Table 3.1.

The Greedy algorithm was originally suggested by Takahashi and Matsuyama [TM80] for the Steiner problem. We generalize their algorithm for both the static and on-line problems and for directed graphs.

The General On-line Greedy Algorithm

1. START from the source.
2. REPEAT for each new set of requests (on-line)
   (a) Leave requests: Prune the edges of the tree that do not lead to current destinations.
   (b) Join requests: REPEAT for each Join request:
      i. SELECT among non-accommodated requests the one for which the destination is closest to some point of the multicast tree, v_{new}.
      ii. JOIN v_{new} to the tree by a path of minimum cost.

Sets of join and leave requests arrive in an on-line fashion, as we already mentioned. For each new set of requests, we accommodate first the leave requests by pruning the unneeded parts of the tree, i.e., parts of the tree that do not carry information to a destination. For the join requests, we select the destination with the minimum distance from the tree. Recall that we define the distance of between a node and a tree to be the cost of the minimum-cost path between the node and any node in the tree. We add this destination and the minimum-cost path to the tree, and we repeat this procedure for each destination in the set. The algorithm's computational complexity is $O(M \cdot N^2)$ [TM80]. The complexity is equivalent to M times the complexity of Dijkstra's algorithm for shortest paths [Dij59], since we need to know the distance between each destination and every other node in the graph. Note that, in the case of a single set of requests, the above algorithm behaves like static Greedy.

The Naive algorithm finds for each new destination the shortest path from the source to the destination, and adds to the multicast tree the part of the path that is not already in the tree. The leave requests again require simple pruning of the unneeded edges. Observe that we route every destination independently of the other destinations. For this reason, this algorithm is applicable to any on-line problem and in any kind of graph. Naturally, its computational complexity is at least $O(N^2)$, i.e., the complexity of Dijkstra's algorithm for finding shortest paths [Dij59].

3.2 The Static Problem

In this section, we study the static multicast problem on directed graphs, and we prove upper and lower bounds on the performance of Greedy. In addition, we prove a tight bound for the Naive algorithm. Note that the term approximation ratio is more appropriate for the static case than the term competitive ratio.

Theorem 1 (Upper bound I) *The following statements hold:*

a) For the static problem, the approximation ratio of the Greedy algorithm in any directed graph is no worse than

$$\frac{T}{OPT} \leq \frac{\sum_{v \in S} d(s,v)}{d_{max}} \leq M - 1$$

where d_{max} is the maximum distance of any destination from the source.

b) The same bound holds for the approximation ratio for Greedy in the join-only problem, and for Naive in the static, the join-only and the join-leave problems.

PROOF.

a) Let us first consider the static Greedy algorithm in the static problem. Recall that T is the cost of the Greedy tree, and OPT is the cost of the optimal tree. The cost T is bounded by the sum of the shortest paths between the root and each destination $v \in S$, since destination v will never join with a cost higher than $d(s,v)$:

$$T \leq \sum_{v \in S} d(s,v) \leq (M-1) \cdot d_{max}$$

Clearly, $d_{max} \leq OPT$ or $\frac{1}{OPT} \leq \frac{1}{d_{max}}$, and therefore we get: $\frac{T}{OPT} \leq \frac{\sum_{v \in S} d(s,v)}{d_{max}} \leq M - 1$.

b) In the proof of part (a), we did not assume anything about the order in which the destinations join. For this reason, the same arguments hold for the join-only Greedy. Furthermore, we have only considered source-to-destination paths; each destination connects to the source with cost $d(s, v_i)$. Hence, the above result holds for the Naive algorithm. Note also that departures of destinations do not affect the Naive algorithm; every destination is routed independently of the currently active destinations. Therefore, after removing a destination, the resulting tree is identical to the tree that would exist, if this destination had never joined the tree. ∎

Ramanathan [Ram96] proved an upper bound for Greedy with respect to the asymmetry of the graph.

Theorem 2 (Upper bound II) *The approximation ratio of the Greedy algorithm in any directed graph of asymmetry A is bounded above by*

$$\frac{T}{OPT} \leq 2 \cdot A$$

The detailed proof was given by Ramanathan [Ram96].

We combine the two upper bounds in the following corollary.

Corollary 1 *The approximation ratio of the directed Greedy algorithm is*

$$\frac{T}{OPT} = O\left(min\left(M, A\right)\right)$$

We prove that the above asymptotic bound is also a lower bound. We identify a family of graphs for which the approximation ratio of Greedy coincides with this upper bound. For convenience, we say a node v **reaches** node w if there is a directed edge or path from v to w.

Theorem 3 (Lower bound) *Given the asymmetry A, and the multicast group size, M, there exists a directed graph and a multicast group such that the approximation ratio of the Greedy algorithm is at least*

$$\frac{T}{OPT} = \Omega\left(\frac{(A)(M)}{A+M}\right)$$

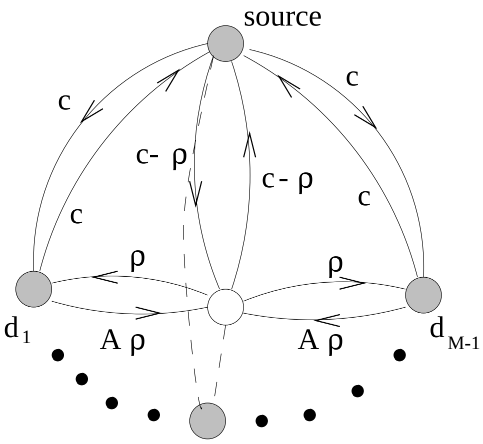


Figure 3.1: An example of a bad case of a directed graph for Greedy in the static problem.

PROOF. It is sufficient to identify a graph and a session that satisfies the inequality we want to prove. Consider the graph depicted in fig. 3.1. The source reaches all the destinations with an edge of cost c. There is only one non-participant node that we call the **middle node** (the white node in fig. 3.1). This node reaches all the participants with an edge of cost ρ, while the opposite edges are of cost $A\,\rho$. The source reaches the middle node with a cost of $c-\rho$, with $\rho \leq c$.

The optimal tree includes the middle node, since $\rho \leq c$. The optimal cost, OPT, is given by:

$$OPT = c - \rho + (M-1)\rho = c + (M-2)\cdot\rho \tag{3.1}$$

The worst case for Greedy is to miss the middle node, and not include it in the tree. This can actually happen if we assume:

$$c < \rho + A\rho = (A+1)\cdot\rho \tag{3.2}$$

The cost T of the tree of Greedy is given by:

$$T = (M-1)\cdot c \tag{3.3}$$

Thus, the approximation ratio of Greedy becomes:

$$\frac{T}{OPT} = \frac{(M-1)c}{c+(M-2)\rho} \tag{3.4}$$

The approximation ratio of Greedy is maximized for a large c and a small ρ. From eq. 3.2, the minimum value of ρ for a given c is : $c/(A+1) < \rho$. We complete the proof by taking the limit of the approximation ratio (eq. 3.4) for $\rho \to c/(A+1)$. ■

Note that $O(min(M,A)) = O\left(\frac{M\,A}{M+A}\right)$. Therefore, the upper and lower bounds coincide (corollary 1 and theorem 3) and we have the following tight bound.

Corollary 2 *For any directed graph of asymmetry A, and with a multicast group size M, the approximation ratio of Greedy has complexity*

$$\frac{T}{OPT} = \Theta\left(\min\left(M,A\right)\right)$$

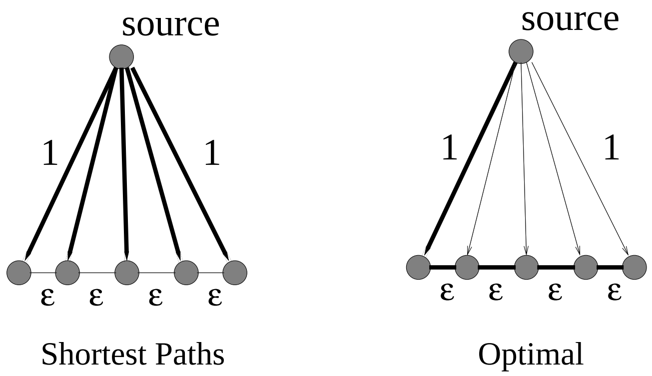


Figure 3.2: A worst case example for the Naive algorithm.

For the Naive algorithm we can prove the following tight bound.

Theorem 4 *In all multicast problems, the approximation ratio of the Naive algorithm is* $\Theta(M)$.

PROOF. Theorem 1 proves that M is an upper bound for Naive, so we only need to prove that it is a lower bound as well. Assume a graph where the source is connected with each destinations with an edge of unit cost (see fig. 3.2). Assume $M-2$ edges of infinitesimal weight ϵ between destinations, in a way that they form a path. Clearly, Naive creates a tree of cost $T = M-1$; while the optimal tree has a cost of $OPT = 1 + (M-2){\cdot}\epsilon$. Letting $\epsilon \to 0$ concludes the proof. ∎

Thus, for small asymmetry ($A \leq M$) Greedy has a better approximation ratio than Naive in the worst-case. However, the approximation ratio of Greedy becomes similar to that of the Naive in highly asymmetric graphs.

3.3 The Join-Only Problem

Here we study the join-only problem. We prove an upper bound for the Greedy algorithm, and then prove a lower bound for any on-line algorithm.

3.3.1 The Join-Only Upper Bound

We prove an upper bound for the Greedy on-line algorithm. First, we recall that Greedy is M-competitive (theorem 1). Second, we prove that $A \log M$ is another upper bound for Greedy. Both bounds are valid for any asymmetry value, and so we can combine them by taking the minimum of the bounds (see corollary 3). Note that in this section, we will use the subscript d for values that refer to the directed problem, while in the other sections we refer only to directed graphs and so the subscript is not needed.

Theorem 5 *For any weighted directed graph with asymmetry A and a multicast group of size M, the competitive ratio of the Greedy algorithm in the join-only problem is bounded above by:*

$$\frac{T_d}{OPT_d} \leq A{\cdot}\log M$$

where T_d is the cost of the Greedy tree and OPT_d the cost of the optimal one.

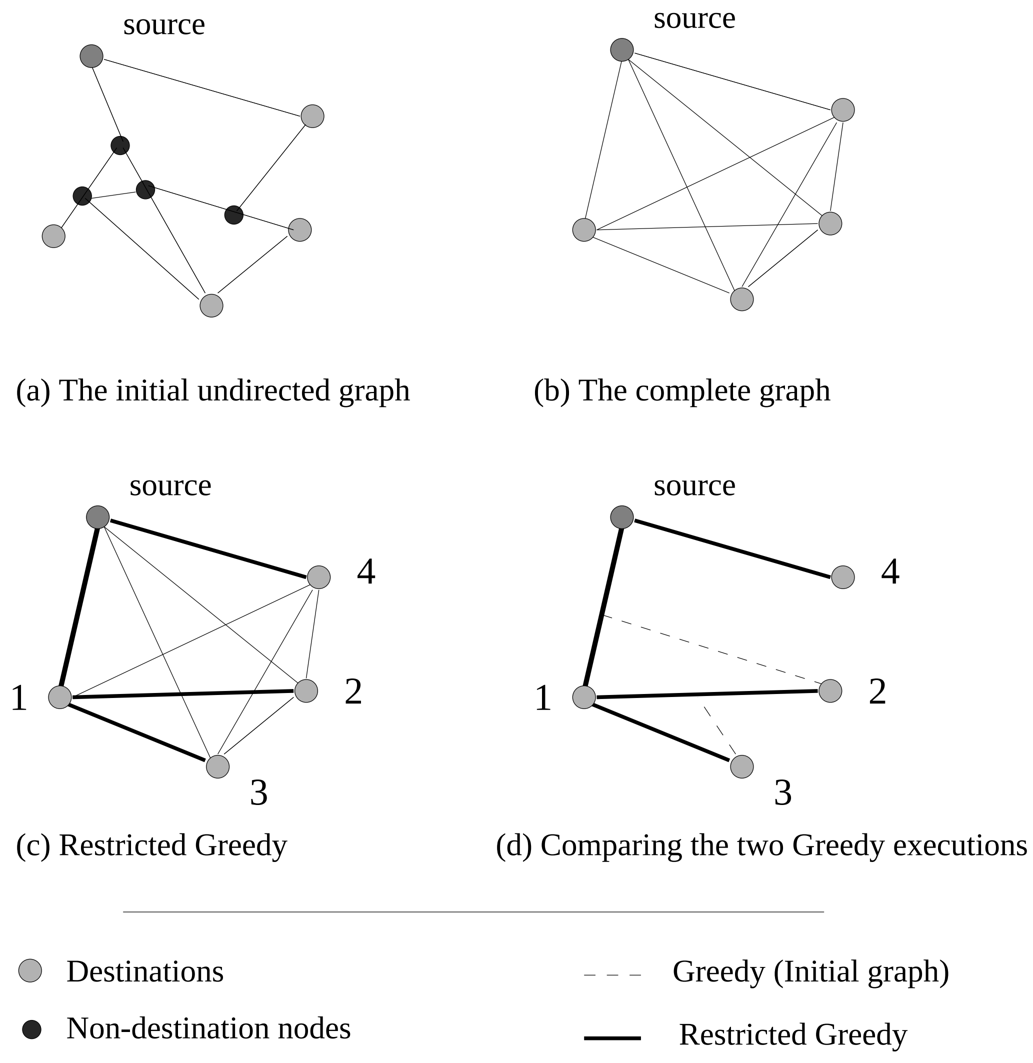


Figure 3.3: Graphic representation of the proof of theorem 5.

PROOF. In this proof, we transform the directed graph into an undirected graph in which the weight of each edge is the minimum of the two directed edges. Then, we apply a restricted version of the Greedy on-line algorithm to the undirected graph. The restricted Greedy algorithm is $O(\log M)$ competitive with respect to the undirected optimal solution, and the corresponding directed Greedy algorithm is $O(A)$ competitive with respect to the restricted undirected Greedy.

Let $G_d(V, E)$ be the directed graph. We denote T_d the cost of the tree produced by Greedy on G_d by T_d. Let us create an undirected graph $G_u(V, E_u)$ as follows. For every pair of opposite directed edges in G_d, we create an undirected edge in G_u with the minimum weight of the pair: $e_u(v, w) = \min(e_d(v, w), e_d(w, v))\,, \quad v, w \in V$. If OPT_u is the cost of the optimal tree on G_u, clearly

$$OPT_u \leq OPT_d \tag{3.5}$$

Let us execute a variation of Greedy for the on-line problem that we call **restricted** Greedy (see fig. 3.3). First, we consider the graph $G'_u(V'_u, E'_u)$, to be the complete undirected distance graph of the participant nodes of G_u, i.e., V'_u includes the source and the destinations S, and the weight of the edge $(v, w) \in E'_u$ is equal to the distance of the nodes in G_u. Then, we execute Greedy on G'_u for the on-line problem. We call T_u the cost of the resulting tree (see fig. 3.3 part (c)).

For the undirected on-line problem, Imase and Waxman (in theorem 2 [IW91]) proved that the cost of the restricted Greedy solution, T_u, is bounded as follows:

$$T_u \leq \log M \cdot OPT_u \tag{3.6}$$

Given the on-line nature of the problem, Greedy and restricted Greedy connect the destinations in the same order. Let us examine closely now the execution of Greedy on G_d, and recall that for each destination, we find the shortest path to the existing tree. For each destination, Greedy chooses a directed path that can be either similar to the path that restricted Greedy chose for the same destination, or a path of less cost compared to that directed path (see fig. 3.3 part (d)). In the worst case, each directed path is at most A times more expensive that the corresponding undirected path. The total cost is the sum of cost of these directed paths, and so the following inequality holds:

$$T_d \leq A \cdot T_u \tag{3.7}$$

We can combine equations (3.5), (3.6), (3.7), to conclude that:

$$T_d \leq A \cdot T_u \leq A \cdot \log M \cdot OPT_u \leq A \cdot \log M \cdot OPT_d$$

■

We combine the two upper bounds from theorem 1 and theorem 5 in the following corollary.

Corollary 3 *For any directed graph with asymmetry A and a multicast group of size M, the competitive ratio of the join-only Greedy algorithm in the multicast problem is no worse than*

$$\frac{T_d}{OPT_d} = O\left(\min(M, A \cdot \log M)\right)$$

We can see that for small asymmetry, $A \leq M/\log M$, the bound is $A \log M$, while, when the asymmetry becomes larger, the bound becomes M.

Recall that a join-only problem instance corresponds to an instance of the RPF algorithm (see the introduction of 2), when the random order coincides with the online sequence of requests. Therefore, the following corollary holds.

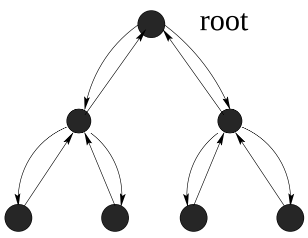


Figure 3.4: The d-tree : a complete binary directed tree.

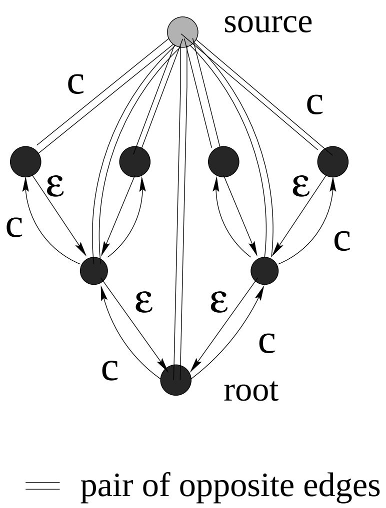


Figure 3.5: The worst case example for highly asymmetric graphs. Edges from or towards the source are of cost c. Elsewhere, curved edges are of cost c, while their opposite edges are of cost ϵ.

Corollary 4 *For any directed graph with asymmetry A and a multicast group of size M, the competitive ratio of RPF in the multicast problem is no worse than*

$$\frac{T_d}{OPT_d} = O\left(\min(M, A \cdot \log M)\right)$$

3.3.2 A Lower Bound for the Join-Only Problem

For the lower bound of the join-only problem, we have to distinguish the cases of: a) **highly asymmetric** ($A \geq M$), and b) **less asymmetric** ($A < M$) graphs. In both cases, we construct the worst case graph and an execution scenario. In highly asymmetric graphs, the worst case example is straightforward, while in less asymmetric graphs, we use the same example as a building block in a more elaborate construction.

Highly Asymmetric Graphs

In this subsection, we prove that, for highly asymmetric graphs, the worst case ratio of any on-line algorithm is proportional to the number of destinations. Specifically, we prove the following theorem.

Theorem 6 *Given a bound on the size of the multicast group, M, there exists a directed graph,*

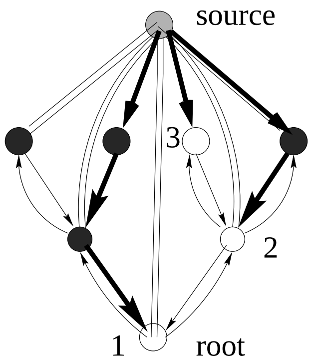


Figure 3.6: The worst case example for highly asymmetric graphs in the join-only problem for any algorithm.

and a sequence of join requests such that the competitive ratio of any on-line algorithm is at least

$$\frac{T}{OPT} \geq \frac{1}{2} \cdot (M-1)$$

where T is the cost of the tree of the algorithm and OPT is the optimal off-line cost.

PROOF. We will first present the intuition behind the graphs that we use. We want to create graphs that can be "bad" for the routing decision of any algorithm. For this, we have to ensure that the on-line algorithm must have at least two routing choices for every destination. In addition, for either routing choice, the arrival sequence must have the possibility to be equally bad. In other words, we need "two identical paths" for every arriving destination, and this suggests a binary tree configuration (see fig. 3.4).

Definition 2 *A* **d-tree** *(directed tree) is a complete binary directed tree that has pairs of opposite directed edges instead of undirected edges.*

Assume a d-tree with $M-1$ distinct levels (see fig. 3.4). We assign weight c to the edges "pointing away" from the root (curved edges), and weight ϵ to the opposite edges (straight edges). We will use the notion of a "child" as in undirected trees. Let the source be a node outside the d-tree and connected to each node of the d-tree with cost c (see fig. 3.5). We can assume that the opposite of the edges "from" the source are also of cost c.

For the on-line sequence, we choose the root of the d-tree as the first destination. In each step, we choose a child of the previous destination that does not belong to the multicast tree (see fig. 3.6). This way, every new destination increases the cost of the multicast tree by c. In more detail, each destination can join the multicast tree with a) a path along the d-tree, b) an edge from the source, or c) an edge from the previous destination. Each of these choices has cost of at least c, and the total cost becomes

$$T \geq (M-1) \cdot c \tag{3.8}$$

By the definition of the sequence, each new destination is a child of the previous destination, and therefore there exists a path from the source to the root of the d-tree that contains all the destinations. Therefore, the optimal cost is

$$OPT = c + (M-2) \cdot \epsilon \leq 2 \cdot c \tag{3.9}$$

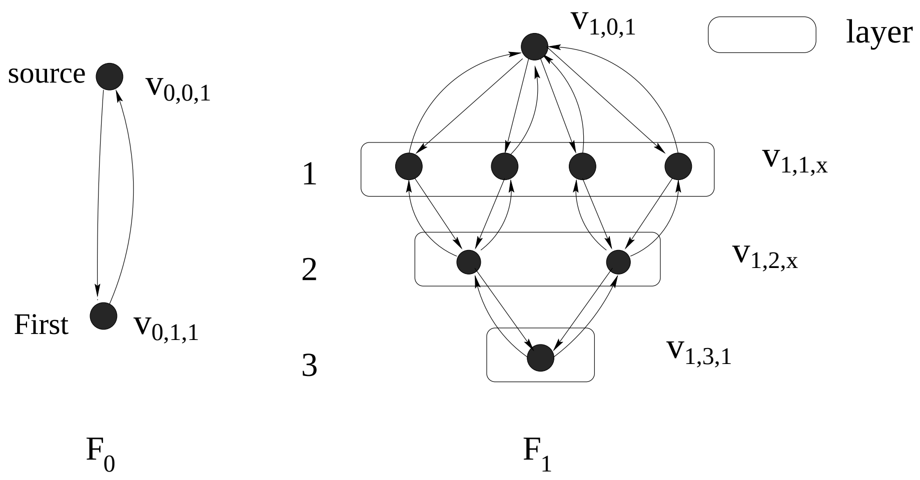


Figure 3.7: The family of F_g with $k = 2$: F_0 and F_1

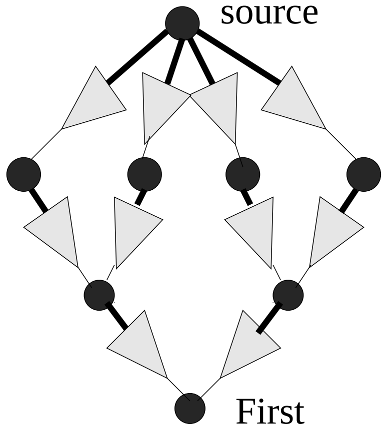


Figure 3.8: The family of F_g with $k = 2$: F_2. The triangles represent d-trees in which the root is a node of the previous generation. The thick edges represent many pairs of edges that we merge for the clarity of the layout.

For the last equality, we assume $\epsilon \leq c/M$. Using equations (3.8) and (3.9), we calculate the competitive ratio and complete the proof. ■

In the example above, the asymmetry is at least $\Omega(M)$. This asymmetry appears at the edges of the d-tree : c/ϵ, given that we assume $\epsilon \leq c/M$. This is the constraint that makes this example apply only to “highly asymmetric” graphs.

Less Asymmetric Graphs

As we saw in the upper bounds, the relative performance of the Greedy algorithm improves as the asymmetry decreases. In less asymmetric graphs ($A = O(M)$), the competitive ratio of Greedy is bounded above by $A \cdot \log M$. Two questions are of interest: i) how tight is this upper bound (for our Greedy algorithm), and ii) can we expect other on-line algorithms to do better? The answer to both questions is that the competitive ratio of *any* on-line algorithm cannot be better than $O(\log(A + 1))$ the competitive ratio of Greedy.

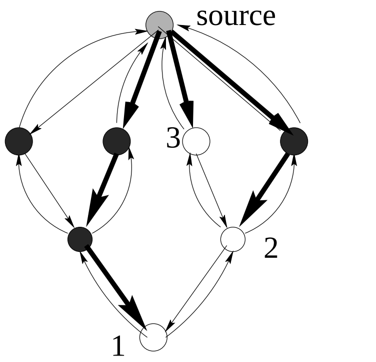


Figure 3.9: A worst case on-line sequence for graph F_1 for less asymmetric graphs in the join-only problem for any algorithm.

Theorem 7 *Given the size of the multicast group, M, and a bound on the asymmetry $A = O(M)$, there exists a graph and a sequence of join requests such that the competitive ratio of any on-line algorithm is at least*

$$\frac{T}{OPT} = \Omega\left(A\,\frac{\log M}{\log(A+1)}\right)$$

where T is the cost of the tree of the algorithm and OPT is the cost of the optimal tree.

PROOF. The proof consists of the following parts. First, we describe how we construct a graph given the asymmetry and multicast group size; then we define the on-line sequence in which nodes join, and calculate a lower bound on the cost of the tree.

PART I: Graph construction.

Before the detailed description, we present the intuition behind the **family of graphs** that we denote by F_g. The building block of our graphs is the d-tree defined in section 3.3.2. The d-tree guarantees that the on-line algorithm has two identical routing choices for each destination. We construct the graph in a recursive manner, until we reach the size we want. (The termination condition involves the group size and is specified later.) Each recursive step creates a **generation**. The number of generations in a particular graph is denoted by g. We start from a pair of nodes, and at each step we replace pairs of opposite edges with a d-tree of depth k. Note that k describes our building block (d-tree) and, for different values of k, we could get different graphs. However, in our example, k does not vary, because we fix $k = A$ eventually. For most of our description, we keep the expressions distinct to enhance clarity.

We need to identify nodes with their distance from the source, and for this we define the layer (see fig. 3.7).

Definition 3 *The* **layer** *of a node in the graph F_g is equal to the length (number of edges) of the shortest path from the source to the node.*

A node of the F_g graphs is denoted by $v_{g,l,x}$, where g is the generation, l is the index of the layer as previously defined, and x an index that identifies nodes of the same layer. For example, in fig. 3.7, in F_1 we have four nodes in layer one: $v_{1,1,1}, v_{1,1,2}, v_{1,1,3}, v_{1,1,4}$, and two nodes in layer two: $v_{1,2,1}, v_{1,2,2}$. As we will see later, it is usually not necessary to specify values for x.

We outline briefly the creation procedure. We start with a pair of nodes, one of which is the source. For every generation, we insert a binary tree between every pair of nodes of the previous generation (see fig. 3.7 and fig. 3.8). The edges are consistently asymmetric; from the source and "down" they are light (straight edges), while the opposite edges are heavy (curved edges). As we will see later, the weights of the edges are set in such a way that the distances are preserved: all paths between the two nodes of F_0 are of equal cost.

Initial Graph. The initial graph, F_0 has two nodes, $v_{0,0,1}$ and $v_{0,1,1}$ (see fig. 3.7). We consider $v_{0,0,1}$ to be the source of the multicast session. Node $v_{0,1,1}$ is denoted as **First**. There are two directed edges between the nodes with weights:

$$e(v_{0,0,1}, v_{0,1,1}) = \rho_0, \quad e(v_{0,1,1}, v_{0,0,1}) = A{\cdot}\rho_0$$

Creating the Next Generation. Given the graph F_{g-1}, we create the graph of the next generation F_g. For every pair of opposite edges in F_{g-1}, we substitute a d-tree (see fig. 3.7 and fig. 3.8). Assume the pair of nodes $v_{g-1,l,x}$ and $v_{g-1,l+1,x}$ of F_{g-1}. Node $v_{g-1,l+1,x}$ becomes the root of the new d-tree. Node $v_{g-1,l,x}$ is connected to all the leaves of the tree. We refer to the nodes introduced in generation g as **new-nodes** of the generation. We usually denote the nodes with the identifiers of their first appearance, as new-nodes. For uniformity reasons, all the "old" nodes can be identified alternatively with indices of the newer generations. Namely, a layer index is $k+1$ times the layer index of the previous generation: $v_{g-1,i,x} = v_{g,i(k+1),x}$

The weights of the edges between adjacent nodes are set as follows. In the equations below, x replaces indices of adjacent nodes.

$$\begin{aligned} e(v_{g,l,x}, v_{g,l+1,x}) &= \rho_g && (3.10)\\ e(v_{g,l+1,x}, v_{g,l,x}) &= A\ \rho_g && (3.11)\\ \rho_g &= \frac{\rho_0}{(k+1)^g} = \frac{\rho_{g-1}}{k+1} && (3.12) \end{aligned}$$

With this construction, we guarantee the following important properties of the F_g graphs. First, the asymmetry of the graph is A. Second, the distance between any pair of nodes is the same in each generation. As we can see in equation (3.12), we replace an edge of cost ρ_{g-1} with (multiple paths of) $k+1$ edges of $\rho_{g-1}/(k+1)$ cost. We can generalize the above as follows.

Observation 1 *All paths between the initial pair of nodes are of equal cost.*

Finally, we want to relate the number of layers to the generation index. Using induction on g, we can prove the following relation between the number of layers and the generation of a graph.

Observation 2 *The number of layers, l, of graph F_g is exponential in the generation index g:*

$$l = (k+1)^g + 1$$

PART II: The on-line sequence of joining destinations.

Intuitively, our on-line sequence follows the recursive structure of the graph. We choose one node from each layer, and we route it before selecting the next one. First, we select nodes from the first generation in the worst case sequence. in a similar way to the worst case example of the previous subsection (see fig. 3.9). Then, we move to the next generation: we select the d-trees of the new generation that lie between destinations. We select the destinations for each of these d-trees and repeat the procedure. Note that the relative distances in each generation decrease (eq. 3.12), but the number of the destinations in each generation increases. This makes the joining cost of each generation constant. The details will become more clear when we calculate the cost of the tree.

In more detail, given a graph F_g, we choose one node from each layer along a path from the First (first node to join) to the source. Namely, we choose the nodes in order of increasing generation number, and within the same generation, in order of decreasing layer. For example, we present the beginning of the sequence of nodes

$$\underbrace{v_{0,1,1}}_{g=0}\ \underbrace{v_{1,k,x}\ v_{1,k-1,x}\ \cdots\ v_{1,1,x}}_{g=1}\ \underbrace{v_{2,m,x}\ v_{2,m-1,x}\ \cdots\ v_{2,1,x}}_{g=2}\ \cdots$$

where $m = k{\cdot}(k+1)$, except for the nodes that correspond to destinations of the previous generation, which have already joined.

Given the above sequence, the total number of destinations is equal to the number of layers. Using Observation 2, the relation between the number of generations and the number of destinations is given below.

Observation 3 *If the number of participants of the graph F_g is M, the maximum number of generations g_m is*

$$g_m = \Theta\left(\frac{\log M}{\log(k+1)}\right)$$

In more detail, each destination, say $v_{g,l,x}$, can connect to: a) the source with a path, or b) the most recently connected destination with an edge. (The joining procedure is illustrated in fig. 3.9). After the routing decision is made, we choose a node of the next (smaller by one) layer, $v_{g,l-1,x}$ that is adjacent to $v_{g,l,x}$. There are two such nodes: the two children of $v_{g,l,x}$ in the d-tree. If $v_{g,l,x}$ has been connected to the source, then one of the children is already in the tree, and we choose the other child. If $v_{g,l,x}$ has been connected to its previous destination, we can choose either child. After choosing nodes from all the levels of a generation, we choose nodes from the levels of the next generation. Namely, we choose the d-trees of the next generation that are between destinations of the previous generation. This way, it is easy to see the following:

Observation 4 *Given the above on-line sequence on graph F_g, there exists a path from the source to the First node that contains all the destinations.*

PART III: Calculation of the Cost

For the above on-line sequence, we will find a **lower bound**, T, on the cost for any on-line algorithm. We also denote the cost of connecting the new-nodes of generation g by T_g. As we said above, there exists a path from source to First that contains all the destinations. Given Observations 1 and 4, the cost of the optimal solution is given by:

$$OPT = \rho_0 \tag{3.13}$$

To calculate the cost, we need to specify the value[1] of k:

$$k = A \tag{3.14}$$

Given that from each layer we select a destination[2], k is less than M. Combining this with equation (3.14), we get an **upper bound on the asymmetry of the graph**:

$$A \leq M \tag{3.15}$$

[1]If the asymmetry is not an integer number, we take its floor.

[2]Recall that M is equal to the number of layers, l, and that $k \leq l$ (Observation 2 for $g \geq 1$), thus, $k \leq M$.

The lower bound T of the cost of any on-line algorithm is equal to the sum of the lower bounds of the costs of each generation, T_g :

$$T = T_0 + \sum_{g=1}^{g_m} T_g \tag{3.16}$$

It is easy to see that $T_0 = \rho_0$. We will now calculate the value of T_1 (cost of the destinations added in F_1), and then generalize the result for the other generations. Consider the destination of level l, and note that $l \geq 1$, since zero level corresponds to the source. This destination can either connect: a) to the source or b) to another destination (see fig. 3.9). In the first case, the cost is $(k+1-l)\cdot\rho_1$, while in the second case, the cost is $A\cdot\rho_1$. Given that $A = k$ (see eq. 3.14), the minimum cost connection is $(k+1-l)\cdot\rho_1$ for all layers $(l \geq 1)$. Furthermore, all possible paths are edge-disjoint, because of the way we choose our destinations. Therefore, the sum of all these paths is a lower bound of the cost of this generation:

$$T_1 = \sum_{i=1}^{k} i\cdot\rho_1 = \frac{k(k-1)}{2}\cdot\rho_1 = \frac{k(k-1)}{2}\cdot\frac{\rho_0}{k+1} \tag{3.17}$$

For the cost of generation g, it is not difficult to see that for every added d-tree, we can calculate the cost in a similar way, and we will end up with an expression similar to the one above only with ρ_g instead of ρ_1. At each step, the number of these d-trees (that contain destinations) is equal to the number of F_{g-1} destinations minus one. Using Observation 2, we have $(k+1)^g$ such trees. Thus, the total cost of generation g is given below:

$$T_g = (k+1)^{g-1}\cdot\rho_g\cdot\frac{k(k-1)}{2} = \frac{\rho_0}{k+1}\cdot\frac{k(k-1)}{2} \tag{3.18}$$

In the above equation, T_g is constant for all the generations. For $k = A$, the total cost becomes:

$$T = T_0 + \sum_{g=1}^{g_m} T_g = \rho_0 + \frac{\log M}{\log(k+1)}\cdot\frac{\rho_0}{k+1}\cdot\frac{k(k-1)}{2} \Rightarrow$$

$$T = \Omega\left(A\cdot\frac{\log M}{\log(A+1)}\right)\cdot\rho_0 \tag{3.19}$$

Recall that $OPT = \rho_0$ (eq. 3.13), and that T is a lower bound for any on-line algorithm. Therefore, equation 3.19 completes the proof. ■

Note that the two lower bounds correspond to different kinds of graphs: $\Omega(M)$ for highly asymmetric graphs $(A > M)$ and $\Theta\left(A\cdot\frac{\log M}{\log(A+1)}\right)$ for less asymmetric ones $(A \leq M)$. The high asymmetry bound does not hold for less asymmetric graphs and vice versa. Given this, the lower bound for any specific graph is the minimum of the two bounds.

Corollary 5 *(Lower Bound) Given the size of the multicast group, M, and the asymmetry A, there exists a graph and a sequence of join requests such that the competitive ratio of any on-line algorithm is*

$$\frac{T}{OPT} = \Omega\left(\min\left(M, A\cdot\frac{\log M}{\log(A+1)}\right)\right)$$

Recall the upper bound from Corollary 3, $O(\min(M, A\log M))$. The upper and lower bound differ only by the term $\log(A+1)$ in the second clause, in the case where $A \leq M$.

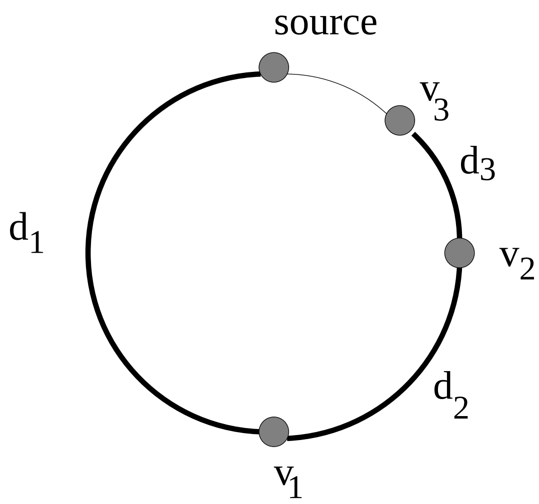


Figure 3.10: An example of a bad case for undirected and directed graphs for three join requests. The graph is shown as undirected, so for the directed case we can replace each edge with a pair of opposite directed edges.

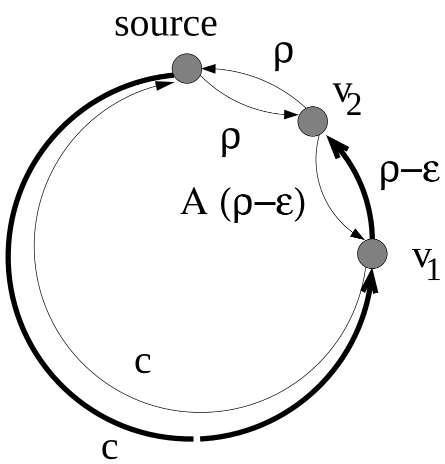


Figure 3.11: An example of a bad case for directed graphs.

3.4 The Join-Leave Problem

In this section, we study the join-leave on-line multicast problem, in which participants can join and leave anytime during the session. We find a lower bound for the Greedy algorithm. First, we improve the lower bound for the undirected case [IW91], which applies in the directed case as well. Then, we prove that the competitive ratio of the Greedy algorithm can be proportional to the asymmetry. Combining the two lower bounds, we conclude that the Greedy algorithm can lead to very poor multicast trees in asymmetric graphs.

First, we consider undirected graphs. Recall that we denote by $M-1$, the number of join requests. The competitive ratio of the Greedy algorithm is proven to be as large as M [IW91]. Here, we prove that the competitive ratio can be exponential in M.

Theorem 8 *Given the number of join requests, M, there exists an undirected graph and a sequence of join-leave requests such that the competitive ratio of the Greedy algorithm is*

$$\frac{T}{OPT} = \Omega\left(2^M\right)$$

PROOF. Let us suppose that the network is a cycle of M nodes $v_0, ..., v_{M-1}$ in that order, and let $v_0 = s$ be the source (see Fig. 3.10). Assume that the sum of all the weights of the cycle is C. Assume that the weights of the edges between nodes $d_i = e(v_i, v_{i-1}), i = 1, ..., M-1$ decrease by half as follows: $d_1 = \frac{C-\epsilon}{2}$, $d_2 = \frac{C-\epsilon}{4}$ and in general $d_i = \frac{C-\epsilon}{2^i}$, where ϵ is an arbitrarily small constant. The purpose of ϵ is to bias node v_i to be closer to v_{i-1} than the source s: $d(v_{i-1}, v_i) < d(s, v_i), \forall i > 2$.

We distinguish two phases: the joining and the leaving phase.

Joining Phase. Assume that the nodes join in order of increasing index: $v_1, ..., v_{M-1}$. In the Greedy algorithm, the first node will join to the source. Given the way we set the weights, each new destination joins to the destination that joined just before, e.g., v_i joins to $v_{i-1}, i = 1, ..., M-1$ etc.

Leaving Phase. We can assume that all destinations leave the session except for v_{M-1}. So, in the Greedy tree, we have the destination v_{M-1} connected to just the source along the longest part of the cycle: $T = C - d(s, v_{M-1})$. The optimal tree consists of the edge between the source and the last node in the tree v_{M-1}: $OPT = d(s, v_{M-1})$.

By taking the ratio of the costs of the two trees we get:

$$\frac{T}{OPT} = \frac{C}{d(s, v_{M-1})} - 1$$

Finally, for $\epsilon \to 0$, we calculate that: $d(s, v_{M-1}) = C - \sum_{i=1}^{M-1} d_i = C - \sum_{i=1}^{M-1} \frac{C-\epsilon}{2^i} \simeq \frac{C}{2^{M-1}}$. ■

Now let us consider directed graphs and examine how the asymmetry of the directed graphs can degrade the efficiency of the solutions. Since an undirected graph corresponds to a directed graph of asymmetry $A = 1$, the bound of theorem 8 holds: the competitive ratio of Greedy can be exponential in M. Furthermore, we prove that the competitive ratio can be proportional to the asymmetry A.

Theorem 9 *Given a number of join requests, M, and an asymmetry, A, there exists a directed graph and a sequence of join-leave requests such that the competitive ratio of the Greedy algorithm is*

$$\frac{T}{OPT} = \Omega\left(A\right)$$

	Undirected			Directed		
	Static	Join	Join-Leave	Static	Join	Join-Leave
Naive	M	M	M	M	M	M
Greedy Up	2	$\log M$	–	$\min(A, M)$	$\min(M, A\cdot \log M)$	–
Greedy Low	2	$\log M$	2^M	$\min(A, M)$	$\min(M, A\cdot \frac{\log M}{\log A+1})$	$\max(A, 2^M)$

Table 3.2: The worst case bounds of the Greedy and the Naive algorithms for the various multicast problems.

PROOF. Intuitively, using the asymmetry, we don't need a large M to create the previous worst case example. Let us consider the source and two destinations v_1, v_2 (see fig. 3.11). The weights of the edges between v_1 and the source are c both ways. The weights of the edges between v_2 and the source are ρ. The asymmetry appears only between v_1 and v_2:

$$e(v_1, v_2) = \rho - \epsilon, \qquad e(v_2, v_1) = A\cdot(\rho - \epsilon), \qquad \rho > \epsilon, \ \ \rho, \epsilon \in \mathcal{R}^+ \tag{3.20}$$

We choose c to be of lower cost than the path from the source to v_1 that includes v_2:

$$c = A\cdot(\rho - \epsilon) \tag{3.21}$$

Joining. Node v_1 joins first and connects directly to the source (s, v_1) according to Greedy. When connecting v_2, Greedy chooses edge (v_1, v_2), which is less costly than (s, v_2).

Leaving. We remove destination v_1. The cost of the Greedy tree is:

$$T = c + (\rho - \epsilon) \stackrel{Eq.3.21}{\Longrightarrow} T = (A+1)\cdot(\rho - \epsilon) \tag{3.22}$$

Clearly, the optimal solution is:

$$OPT = \rho \tag{3.23}$$

For $\epsilon \to 0$, the competitive ratio becomes:

$$\frac{T}{OPT} = A + 1$$

■

The significance of the above theorem is that when the asymmetry is unbounded, the competitive ratio of Greedy is also unbounded.

Both previous bounds are applicable for any asymmetry value, since both examples apply to any directed graph without any assumptions about the asymmetry. Thus, the combined lower bound is the maximum of the two bounds.

Corollary 6 *The competitive ratio of the Greedy algorithm in the on-line join-leave problem on directed graphs is*

$$\frac{T}{OPT} = \Omega\left(\max(A, 2^M)\right)$$

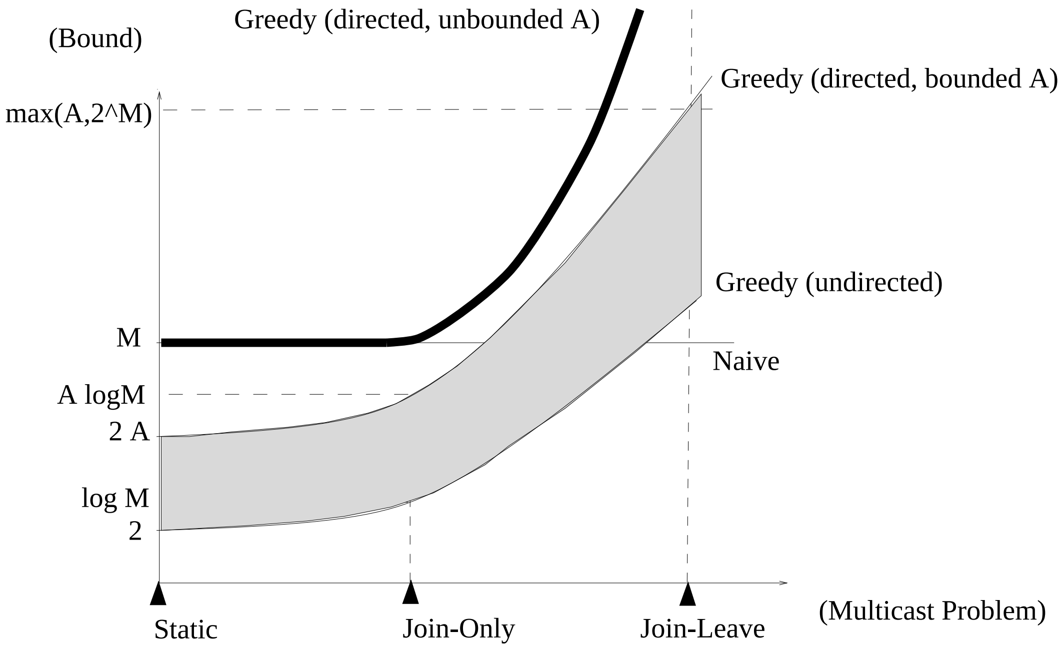


Figure 3.12: The impact of asymmetry: a graphical overview of our results. The bounds of Naive and Greedy for the static, join-only and join-leave multicast problems.

3.5 Conclusions

In this chapter, we studied the performance of the Greedy and the Naive algorithms for the multicast problem on directed graphs. In brief, for the join-only problem the Greedy bounds are better than those of Naive. However, for the join-leave problem, this is reversed; the competitive ratio of Greedy is unbounded, while that of Naive is not affected by the early departures. The reason is that the Greedy approach is based on the idea that we "make our best choice given the current state", but the state can change (destinations leave) and the choice can turn out to be "wrong". On the other hand, Naive routes each destination independently of the other destinations, and remains unaffected by the departures.

In Table 3.2, we present the bounds on the competitive ratio for the Greedy and the Naive algorithms for all multicast problems. The bounds for Naive are tight, while for the Greedy algorithm, we have separate lines for the upper and lower bounds. Note that the asymptotic bounds of the table do not hide "big constants": in most cases a factor of two is missing and all the logarithmic functions have a base of two, as we can see in most theorems before we focus on the asymptotic behavior of the bounds in the corollaries.

In Fig. 3.12, we plot the bounds of Table 3.2 graphically for a visual "feel" for the upper bounds of the two algorithms. The x-axis corresponds to the different problems, according to membership variation, and the y-axis shows bounds on the performance. Naturally, the x-axis is discrete, but we take the liberty to interpolate the values in order to make the figure easier to read. Note that for Greedy, we have three different lines corresponding to the undirected case, the bounded-asymmetry directed case and the unbounded-asymmetry directed case. The thicker line represents the unbounded-asymmetry case: all Greedy bounds are below this line. The grey area shows the change in the bounds due to the increase of the asymmetry from one to A (from symmetric to bounded asymmetric case). We make the following observations. First, the Greedy performance is a factor of A worse than that of the undirected case. Second, the Greedy competitive ratio is bounded by M for any (even unbounded) asymmetry in the static and join-only problem. Finally, Greedy performs poorly in the join-leave problem: the bound can become as large as the asymmetry.

Our work leads to the following conclusions that should be considered in the design of algorithms and protocols:

- Our network models should be directed graphs. Asymmetry has a great impact on the performance of multicast routing.
- The Greedy algorithm has near-optimal on-line performance for the join-only problem.
- In the join-leave problem, it is the leave requests that present a problem. For this reason, Greedy can perform poorly in the worst case, and we should be careful in handling sessions with frequent joins and leaves. Namely, when we identify an application with active membership behavior, we should examine whether the application runs the risk of performing poorly. This depends on the join-leave patterns and the topology of the graph. Our worst-case examples are not necessarily typical cases for all networking environments (see 6.2).

Note that, Greedy may be competitive even for the join-leave case on the average. However, the concept of "average case" is difficult to define: we need to define the "average" graph and the "average" join and leave scenario, but these can vary significantly with the application and the underlying network. Therefore, average case observations are meaningful only within a well-defined environment.

For the join-only problem, for less asymmetric graphs, we see a discrepancy between the lower and upper bound of the Greedy algorithm. It is likely that the upper bound could be reduced. If we consider only one destination, it could be the case that the directed edges towards it are the higher cost edges (of each opposite pair). In this case, even the optimal solution would have included high cost edges, so the difference between the two solutions is not going to be proportional to the asymmetry.

Chapter 4

Heterogeneous Video Multicast

In this chapter, we study the static heterogeneous problem where the QoS requirements in bandwidth vary among users. Let us recall the basic concepts of the heterogeneous problem that we presented in section 2.2. Consider the distribution of video data that has been hierarchically encoded. Hierarchical encoding is a layered way of encoding information, such as video, that can appear in different quality levels. The information is divided into an ordered set of levels and higher levels can only be used on top of the previous one. This way, participants receive subsets of the same information. We call these subsets streams and they correspond to different quality levels. Each receiver indicates a range of streams and the resource allocation strategy must offer to the receiver some stream in the range. In its generality, the problem consists of two interrelated sub-problems: a) selecting the quality level for each user, and b) selecting a route. This constitutes the general heterogeneous problem. If we assume that each user requires one specific stream, we get the fixed heterogeneous problem. In contrast, we refer to the one-stream problem as the homogeneous problem.

In the general heterogeneous problem, we have to select: a) a stream for each user, and b) a route. We address both problems at the same time, because they are interrelated. We propose Greedy-based algorithms and show their advantage over previous Naive-based algorithms. In section 4.1, we propose algorithms for the fixed heterogeneous problem and prove bounds for the competitive ratio of our algorithms. In section 4.2, we discuss the general heterogeneous problem and propose algorithms that address the stream optimization and routing problems in parallel. In section 4.4, we present the conclusions of our work on the heterogeneous problem. Note that in section 6.3, we examine the performance of our algorithms in a series of simulations.

For convenience, we list the symbols that appear in this chapter in Table 4.1, repeating some of the symbols from chapter 2. In this chapter, we recall the RPF algorithm that we discussed in the introduction of chapter 2. RPF picks a destination randomly and connects it with the minimum cost path to the tree. We use the term *random order* to denote the order in which the destinations are selected.

4.1 The Fixed Heterogeneous Problem

Let us recall the definition of the fixed heterogeneous problem: given a directed graph, a source and a set of destinations with one preferred stream each, we want to identify the minimum cost tree from the source that offers the preferred stream to each destination.

Our first algorithm for the heterogeneous problem is based on the observation that lower streams are contained within higher streams, and therefore, lower stream destinations can tap into the higher

Symbol	Explanation
$G(V, E)$	Weighted directed graph G with V nodes and E edges.
N	Number of nodes, $\|V\|$.
M	Number of destinations plus the source.
S	The set of destinations and the source.
S'	The set of destinations that are not yet part of the multicast tree.
S_i	The set of destinations that request stream i. (We use it in the proofs in the fixed problem.)
A	The asymmetry of the graph.
OPT	The cost of the optimal multicast tree.
T	The cost of the multicast tree created by an algorithm.
CR	The competitive ratio of an algorithm: the maximum T/OPT.
level	A distinct subset of the hierarchically encoded data.
stream	A continuous range of levels including the base one. It corresponds to a particular quality of service (e.g., quality of presentation).
$cost_v(s, T)$	The joining cost: the cost of connecting destination v to the tree T with stream s.
overrouting cost	The part of the joining cost that is due to routing higher streams over an existing part of the multicast tree.
$hap_v(s)$	The happiness of destination v for receiving stream s.
L	The total number of levels of the session.

Table 4.1: Explanations of the main symbols in this chapter.

streams and acquire the levels they want. Therefore, we route the higher stream destinations first. Among destinations of the higher stream, we choose the one nearest to the partial multicast tree. This strategy gives rise to the **Maximum Stream Participant First** or **MaxStream** algorithm. Maxemchuck [Max97] proposed independently a similar algorithm for the fixed heterogeneous problem [FPS97b], providing some experimental evaluation.

Maximum Stream Participant First Algorithm (MaxStream)

1. START from the source.
2. REPEAT until all destinations are reached:
   (a) SELECT destination: Find the set of unreached destinations that have maximum preferred stream. From this set, select the nearest (minimum cost) destination to some point of the partial multicast tree, v;
   (b) JOIN: v to the tree by a path of minimum cost.

In another approach, we can change the selection criterion and choose the destination that can be added with minimum cost at each step (independent of its preferred stream). Observe that low stream nodes have an advantage over the high stream ones in the selection; higher streams are more costly even along the same path. We call this algorithm **Minimum Cost Participant First** or **MinCost**.

Minimum Cost Participant First Algorithm (MinCost)

1. START from the source.
2. REPEAT until all destinations are reached:
   (a) SELECT among unreached destinations the destination v that can connect to the tree with the minimum cost path;
   (b) JOIN: v to the tree by a path of minimum cost.

Recall that the joining cost includes the overrouting cost; the cost of bringing higher required levels to the node of the tree where the new destination connects to.

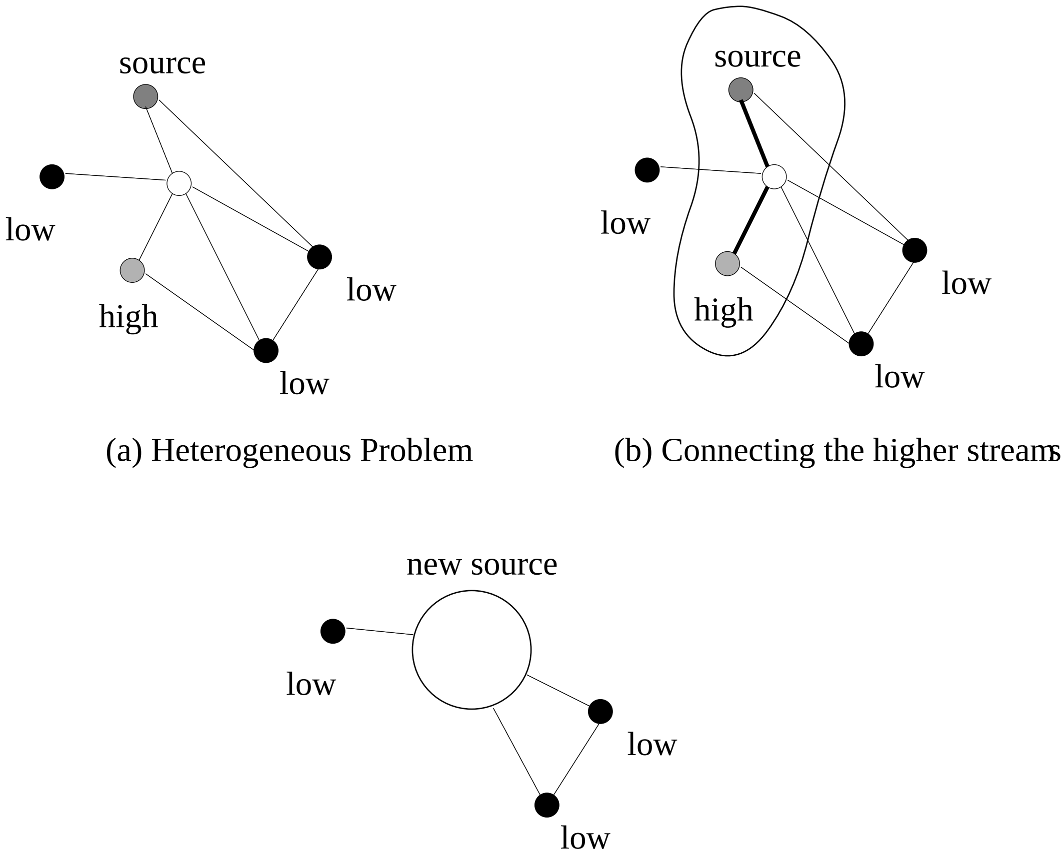
The computational complexity of the centralised implementations of each algorithm is polynomial. Efficient implementations of the algorithms have complexity bounded by $O(M \cdot N^2 + L \cdot N \cdot \min(M, L))$ (see Appendix 4.3).

4.1.1 The Competitive Ratio of Our Algorithms

Using bounds for Greedy in the homogeneous problem, we can prove the following bound on the performance of both algorithms.

Theorem 10 *For any directed graph of asymmetry A, and any multicast session of size M with L distinct streams, the competitive ratio of MaxStream and MinCost in the fixed heterogeneous problem is bounded by*

$$CR \leq \min(2 \cdot A \cdot L, M)$$



(a) Heterogeneous Problem

(b) Connecting the higher streams

(c) The equivalent homogeneous problem

Figure 4.1: How the heterogeneous problem solved by MaxStream relates to a number of homogeneous ones

PROOF. Intuitively, the theorem can be explained as follows. If we group the destinations according to their stream, we have at most L such groups. If we perform Greedy for each such group, we have L homogeneous problems. The heterogeneous Greedy solution is never worse than L homogeneous Greedy solutions, one for each distinct stream. Let us first prove the theorem for MaxStream.

MaxStream. Let us denote the cost of the tree for MaxStream by T_{het}, and the optimal heterogeneous cost by OPT_{het}. Let us denote by $OPT(DEST, STR)$ the cost of the optimal tree for destinations $DEST$ in a homogeneous problem offering stream STR.

Let us partition the destinations into sets S_i consisting of destinations that require stream s_i. We can decompose the cost of the heterogeneous tree T_{het} among the S_i sets; we denote by $T_{het,i}$ the cost that is the result of joining destinations of S_i.

$$T_{het} = \sum_{k=1}^{L} T_{het,k}$$

In MaxStream, the joining cost of the destinations in S_i can never be worse than the worst case of a homogeneous problem for the same nodes and the same offered stream. Let us consider

MaxStream. First, it routes the destinations in S_L and creates the same tree as Greedy solving the similar homogeneous problem. Then, it will route the destinations in S_{L-1}. In routing this set, MaxStream will use the existing tree. We can consider the existing tree as a "large" source (see fig. 4.1): we can replace the whole tree by a "new source" and keep the minimum cost edge between the "new source" and every node that is adjacent[1] to the tree. Routing S_{L-1} on the new graph with Greedy is equivalent to running MaxStream. Note that the "new source" may be closer to some of the destinations than the initial source, and therefore, the optimal solution assuming the "new source" can never be worse than the optimal cost $OPT(S_{L-1}, s_{L-1})$. Given this, if we recall the upper bound for Greedy from theorem 2, the following inequality holds for every $T_{het,i}$:

$$T_{het,i} \leq 2{\cdot}A{\cdot}OPT(S_i, s_i), \quad i = 1, ..., L$$

If we sum the previous inequality for all streams $i = 1, ..., L$, we get:

$$\begin{aligned} \sum_{k=1}^{L} T_{het,i} \quad &\leq \quad 2{\cdot}A{\cdot}\sum_{k=1}^{L} OPT\left(S_k, s_k\right) \Rightarrow \\ \Rightarrow T_{het} \leq 2{\cdot}A{\cdot}\sum_{k=1}^{L} OPT(S_k, s_k) & \end{aligned} \tag{4.1}$$

An optimal distribution tree for a subset S_i can never cost more than the solution for all the destinations:

$$\begin{aligned} OPT(S_i, s_i) \quad &\leq \quad OPT_{het}, \;\; i = 1, ..., L \;\Rightarrow \\ \Rightarrow \sum_{k=1}^{L} OPT(S_k, s_k) \quad &\leq \quad L{\cdot}OPT_{het} \end{aligned} \tag{4.2}$$

We combine inequalities 4.1 and 4.2 as follows:

$$T_{het,i} \leq 2{\cdot}A{\cdot}\sum_{k=1}^{L} OPT(S_k, s_k) \leq 2{\cdot}A{\cdot}L{\cdot}OPT_{het} \tag{4.3}$$

Finally, MaxStream will never do worse than M times the optimal cost. We can prove this with arguments similar to those used in the proof of theorem 1 in section 3.2; the cost of the tree of each algorithm will never be more expensive than M times the cost between the source and the most costly destination, and the optimal cost will be at least as large as this cost.

$$T_{het} \leq M{\cdot}OPT_{het} \tag{4.4}$$

We can combine inequalities 4.3, and 4.4 as follows:

$$T_{het} \leq \min(2{\cdot}A{\cdot}L, M){\cdot}OPT_{het}$$

[1]A node of a graph is adjacent a subgraph if and only if the node shares an edge with a node of the subgraph.

MinCost. The proof for MinCost is similar. We partition the destinations into sets S_i destinations that require stream s_i. We decompose the cost of the tree produced by MinCost, T_{het}, among the S_i sets, and we denote by $T_{het,i}$ the cost that is the result of joining destinations of S_i.

We can show that the joining cost of the destinations in S_i can never be worse than the worst case of a homogeneous problem for the same nodes and the same offered stream. To prove this, we use arguments as the ones we used in the proof for MaxStream (see fig. 4.1). Therefore, the following inequality holds:

$$T_{het,i} \leq 2 \cdot A \cdot L \cdot OPT_{het} \tag{4.5}$$

In addition, the cost of the tree of each algorithm will never be more expensive than M times the cost between the source and the most costly destination, and the optimal cost will be at least as large as this cost.

$$T_{het} \leq M \cdot OPT_{het} \tag{4.6}$$

By combining inequalities 4.5 and 4.6, we complete the proof. ∎

We can prove a another bound for MaxStream under some conditions. In describing these conditions, we need the following definition.

Definition 4 *A network with a cost function for each edge is* **bandwidth-invariant** *for a range of bandwidth at a given instant of its operation if and only if the minimum-cost path between each pair of nodes is the same for the connection of any bandwidth in the given range.*

Intuitively, the above definition ensures that the minimum-cost path for the lowest stream is the minimum cost path for the highest stream as well. In other words, as the bandwidth that we route increases, the cost of the paths increases in a way that maintains the relative order of cost. Note that this assumption may not hold in the cases where a) some edges are near congestion, b) there is a great relative difference of bandwidth among the higher and lower streams, and c) the bandwidth of a session is comparable to the capacity or available bandwidth of the edges. In other words, we can expect the assumption to hold for low to medium network load, and for sessions whose bandwidth is much smaller than the capacity of the edges.

The following theorem holds for MaxStream.

Theorem 11 *Assume that the cost of an edge is proportional to its used bandwidth and a directed graph is bandwidth-invariant for a bandwidth range. For any graph of asymmetry A, and any session of M participants with bandwidth in the above range, the competitive ratio of MaxStream in the fixed heterogeneous problem is bounded by*

$$CR \leq \min(M, A \cdot \log M)$$

PROOF. We can explain the theorem intuitively by observing that MaxStream "shuffles" the joining order; it selects the destinations in an order that seems arbitrary compared to the minimum-cost order. The theorem says that this arbitrary order in the heterogeneous problem corresponds to an arbitrary joining order in a homogeneous case, and therefore, it is bounded by the competitive ratio of RPF.

Recall that we denote the levels by l_i and the streams by s_i for $i = 1, ..., L$, such that s_i is the union of l_1 to l_i. Let us partition the destinations into sets S_i destinations that require stream s_i. Let us denote B_i the destinations that require a stream higher or equal to s_i, i.e., $B_i = \bigcup_{j=i}^{L} S_j$.

Let T_{het} be the cost of the tree of MaxStream. Let us denote by $ALG(DEST, STR)$ the cost of the tree of algorithm ALG for destinations $DEST$ in the homogeneous problem offering stream STR; ALG is replaced by OPT for the optimal solution, and by RPF for RPF.

In the fixed problem, we have to deliver the required levels to each destination with the constraint that levels for the same destination must follow the same path. If we remove this constraint, we can offer each level, l_i, in separate trees and only to the destinations that require it. In other words, we perform L separate homogeneous multicast problems one for each level. The optimal solution for each such homogeneous problem is the minimum cost way to deliver the related level; any other distribution is at least as expensive. The cost of OPT_{het} consists of the cost of delivering each level to the destinations that require it. Therefore, the sum of these optimal homogeneous solutions is never worse than the optimal heterogeneous tree, OPT_{het}.

$$OPT(B_1, l_1) + OPT(B_2, l_2) + ... + OPT(B_L, l_L) \leq OPT_{het}$$

Consider the order of the destinations as they join in MaxStream, and let us denote this order by Q_{MxS}. The MaxStream algorithm can be described as several executions of RPF algorithms, where the destinations are chosen in the Q_{MxS} order in RPF. In the rest of this section, we will focus on executions of RPF that select destinations in the Q_{MxS} order. Having assumed that the cost is linear in the bandwidth, the cost of routing multiple connections on the same edge is equal to the sum of the costs: $cost(e, u + w) = cost(e, u) + cost(e, w)$, for any edge e.

1. Repeat for $k = L$ down to $k = 1$, decreasing k by one
   (a) Execute RPF for the S_k destinations offering stream s_k. Note that at each step, we use the previously routed higher streams.

According to this, the cost of MaxStream can described by the following formula:

$$T_{het} = RPF(S_L, s_L) + ... + RPF(S_2, s_2) + RPF(S_1, s_1) \tag{4.7}$$

Note that, for a given set of destinations, the multicast tree is independent of the stream we route, given our assumption that the network is bandwidth-invariant. Considering that RPF follows the Q_{MxS} order, we can re-describe MaxStream as follows:

1. Repeat for $k = L$ down to $k = 1$, decreasing k by one
   (a) Execute RPF for all the destinations B_k that use level l_k offering this level following the Q_{MxS} order

According to the above structure and the additive property, the cost of the algorithm is expressed as follows:

$$T_{het} = RPF(B_L, l_L) + ... + RPF(B_2, l_2) + RPF(B_1, l_1) \tag{4.8}$$

Each RPF execution corresponds to a homogeneous problem, with a multicast group $K_i = |B_i| + 1$. Note that $|B_i| + 1 \leq M\ \forall i$, and $|B_1| + 1 = M$ since all destinations receive level one. If we recall the upper bound of RPF (Corollary 4) for each level $i = 1, ..., L,$, we have the following inequality:

$$\begin{aligned} RPF(B_i, l_i) &\le \min(A \cdot \log K_i, K_i) \cdot OPT(B_i, l_i) \Rightarrow \\ \Rightarrow RPF(B_i, l_i) &\le \min(A \cdot \log M, M) \cdot OPT(B_i, l_i) \end{aligned}$$

Using the above inequality for every level i, we can substitute the cost of RPF in Equation 4.8.

$$T_{het} \le \min(A \cdot \log M, M) \cdot [OPT(B_1, l_1) + OPT(B_{l_2}, l_2) + ... + OPT(B_L, l_L)]$$

Finally, using inequality 4.7, we can replace the sum in the above formula with OPT_{het}:

$$T_{het} \le \min(A \cdot \log M, M) \cdot OPT_{het}$$

■

For comparison, recall that the Naive-based algorithm [Sha92] has a competitive ratio of $(\Theta(M))$.

4.2 The General Heterogeneous Problem

Recall the function $Gain(H, C)$ that combines the happiness H and routing cost C into a single objective value. We have defined the general heterogeneous problem as follows: given a directed graph, a source, a set of destinations with a range of preferred streams each, and an objective function $Gain()$, we want to identify the tree rooted at the source that offers to each destination one of its preferred streams in a way that the happiness and cost of the tree maximize the objective function.

Let us focus on the stream assignment problem, and consider a trivial multicast problem with only one destination. Clearly, the route is the shortest path between the source and the destination, and we only have to choose one of the preferred streams. Recalling figure 2.3, the stream-optimization problem is trivial to solve for one destination: we choose the preferred stream that maximizes the objective function, which we call **chosen stream**. In this section, we use the term "chosen" stream to denote the optimal stream in a one-to-one sense: if we assume that the destination is the last to connect to the tree, the chosen stream is the optimal choice. However, when other destinations connect later, they may share the routing cost and may make higher stream more profitable to route. The chosen stream is given by the following procedure:

Choose-Stream: Stream Selection Procedure for One Node

For a destination v, choose the stream $s_{o,v}$ that maximizes the objective function:

$$s_{o,v} : \max_s [Gain(hap_v(s), cost_v(s, T))]$$

where $cost_v(s, T)$ is the cost of joining cost of v with stream s.

We can combine the above Choose-Stream procedure with the Naive routing scheme into the following algorithm. We consider the multicast tre to be only the source. For each destination, we calculate the an algorithm for the chosen stream. We offer to each destination its chosen stream along a path of minimum cost. When paths share edges, we route on the edge only the maximum of the requested streams. Note that the algorithm creates the same multicast tree independently of the order with which we process the destinations. We call this algorithm the **Chosen Stream Naive** or **Naive-cho**. The competitive ratioof the algorithm is $\Theta(M)$, since it uses Naive routing (see section 3.2). The computational complexity of the algorithm is $O(L{\cdot}N^2)$, since we need to calculate a shortest paths tree from the source for each distinct stream.

We can use the Choose-Stream with Greedy routing. An algorithm we propose is an extension of the MaxStream algorithm that we saw in the previous section. Namely, at each selection step, we find the chosen stream for each destination not already in the tree. Then, among destinations of maximum chosen stream, we choose the one nearest to the partial multicast tree. We call this algorithm the **Maximum Chosen Stream Participant First** or **MaxChoStr** and we present it below.

Maximum Chosen Stream Participant First Algorithm (MaxChoStr)

1. INITIALIZE the partial multicast tree: $T = \{source\}$.
2. REPEAT until all destinations are reached:
   (a) FIND the chosen stream $s_{o,v}$ for each destination v using the Choose-Stream procedure;
   (b) SELECT destination: Find the set of unreached destinations with maximum chosen stream (maximum stream index): $\{x : s_{o,x} = \max_{w \in S'}(s_{o,w})\}$. From this set, select the destination, v, with minimum joining cost;
   (c) JOIN: Include v and the related path in the tree T with stream $s_{o,v}$.

It is easy to see that for a fixed problem this algorithm behaves like MaxStream.

In another approach, we observe that, at each step, we have to select a destination-stream pair. Therefore, we choose the pair whose gain (from the added happiness and cost) maximize the objective function. We call this algorithm **Maximum Gain Participant First** or **MaxGain**.

Chosen Participant First Algorithm (MaxGain)

1. INITIALIZE the partial multicast tree: $T = \{source\}$.
2. REPEAT until all destinations are reached:
   (a) FIND the chosen stream $s_{o,v}$ for each destination v using the Choose-Stream procedure;
   (b) SELECT destination: Find the destination whose gain for its chosen stream maximize the objective function:
   $$\max_{w \in S'} \left[Gain(hap_w(s_{o,w}), cost_w(s_{o,w}, T)) \right];$$
   (c) JOIN v in the tree offering stream s_o by the minimum cost path.

The complexity of each algorithm is bounded by $O(L{\cdot}M{\cdot}N^2 + L{\cdot}M{\cdot}N)$, where L is the number of distinct streams (see section 4.3). For every destination, we have to keep track of the minimum cost path for every preferred stream. Note that the preferred streams of a node may have different minimum cost paths. Thus, for every node added in the tree, we have to re-examine the joining cost of each destination for each preferred stream.

4.3 Implementation and Complexity Analysis

We present a low complexity implementation of the Greedy framework of MaxChoStr and MaxGain. We also calculate the complexity of the fixed problem algorithms MaxStream and MinCost that follow the same framework. We distinguish two phases.

Phase 1. Minimum Distances Calculation.

We want to find the minimum-cost paths between each participant and all other nodes using Dijkstra's algorithm. Note that we find the distances in the direction from a node towards the destination. In the general problem, this computation has to be repeated for each stream, and the computational complexity of this phase is $O(L{\cdot}M{\cdot}N^2)$.

For the fixed problem, we need to calculate the shortest paths for one stream per destination so the complexity is $O(M{\cdot}N^2)$.

Phase 2. Tree creation.

To make the algorithm more efficient, we keep track of the overrouting cost for each stream for every node of the multicast tree. Whenever an existing node in the tree receives a higher stream (due to a new destination that requires it), the overrouting cost of all its descendants (in the tree) is modified.

1. Initialize.

   newnode = *source*, the node that will join in the next step.

   newstream = *maxstream*, the stream offered to the *newnode*, we assume that the *source* "receives" the maximum stream of the session, *maxstream*.

   newpath = {*source*}, the minimum cost path from the tree to the *newnode* providing *newstream*.

2. REPEAT until all destinations are reached:

   (a) Join *newnode* offering *newstream* using *newpath*.

   (b) Update the overrouting cost of nodes in the tree. Start from the *source* and move towards the *newnode* increasing the existing stream of each visited node to the *newstream*. If such an increase happens, update the overrouting cost of all its descendents in the tree.

   For each tree node, the updates are limited to $\min(M, L)$, since each update is related to a stream increase, but also a stream increase happens when a new destination is added in the tree. In each update, we modify the overrouting cost of every possible stream L.

   For an execution, the computational complexity of this step is $O(L{\cdot}N{\cdot}\min(M, L))$. for both the general and the fixed problem.

   (c) Update the joining information of unreached destinations. For each node of the path $O(N)$, we update the joining information of each unreached destination $O(M)$, for all preferred streams of each destination, L_{pref}. In doing so, we keep track of our current best destination, *bestnode*, according to the selection criterion of the algorithm, and the related stream *beststream* and path *bestpath*. Note that L_{pref} is one in the fixed problem and bounded by L in the general problem.

For the general problem, the computational complexity of this step is $O(L \cdot M \cdot N)$ for an execution.

For the fixed problem, the complexity of this step is $O(M \cdot N)$ for the whole execution.

(d) Set:
$newnode = bestnode$
$newstream = beststream$
$newpath = bestpath$

For the general problem, the total computational complexity of the algorithm is bounded by $O(L \cdot M \cdot N^2 + L \cdot N \cdot \min(M, L))$. If we assume that the number of streams is $O(N)$, the complexity of the algorithms is bounded by $O(M \cdot N^3)$.

For the fixed problem, the complexity is bounded by $O(M \cdot N^2 + L \cdot N \cdot \min(M, L))$. If we assume that the number of streams is $O(N)$, the complexity of the algorithms is bounded by $O(M \cdot N^2)$.

4.4 Conclusions

In this chapter, we have examined ways to exploit hierarchical encoding in multicast routing. This leads to the formulation of the heterogeneous multicast problem in its full generality. In this general problem, apart from the routing problem, we have to solve a stream selection problem. We have to strike the balance in the trade-off between the benefit (happiness) and the cost (use of resources) of a session or else maximize the $Gain()$ objective function. We propose stream-selecting algorithms for the static problem on directed graphs and show their advantage over other solutions.

The main observation is that stream-selection techniques and Greedy routing can greatly improve the solution. The stream-selecting algorithms that we propose outperform the previous algorithms in series of simulations (see section 6.3). The advantage of Greedy over Naive routing is demonstrated both theoretically in this chapter, and experimentally in section 6.3. In the fixed problem, we saw that the competitive ratio of MaxStream and MinCost is $\min(2 \cdot A \cdot L, M)$. In addition, the performance of MaxStream is also bounded above by $\min(M, A \cdot \log M)$, if we assume a linear relationship between bandwidth and routing cost, and the graph is bandwidth-invariant. Recall that the competitive ratio of previous Naive-based solutions is $\Theta(M)$. We see that the competitive ratio of our Greedy-based algorithms is not larger than that of the previous Naive algorithms: our algorithms have an advantage for small asymmetry and small number of levels ($A \leq M/\log M$ or $A \cdot L \leq M$).

Second, we observe that the asymmetry of a graph has a significant impact on the relative performance of the algorithms. This is a strong indication that a network model should include the asymmetry in order to provide reliable results. Theoretically, in the fixed problem, we saw that the competitive ratio of MaxStream and MinCost is linear in asymmetry (when asymmetry is $O(M)$). In our simulations, however, the asymmetry of a graph does not seem to affect the relative performance of the algorithms in most of the cases (see section 6.3).

In conclusion, we are convinced that there is a great benefit from effective stream-selection and Greedy-based heterogeneous algorithms. The trade-off of our approach is a small increase in the computational complexity, namely by a factor of $L \cdot M$, compared to previous solutions. However, the benefit that we show in our theoretical and experimental work (see section 6.3) outweighs the increase in the complexity in most cases.

Chapter 5

The QoSMIC Internet Protocol

In this chapter, we propose, QoSMIC, a protocol for the Internet that supports QoS-sensitive multicast applications. A multicast protocol that considers QoS in its routing phase can create trees better suited to the needs of QoS-sensitive applications. In section 5.2, we offer a more detailed description of the protocol. In section 5.4, we prove that our protocol creates loop-free distribution trees. In section 5.6, we summarize our work. In section 6.4 of chapter 6, we compare the routing efficiency and control overhead complexity of our protocol with that of other protocols.

Recall that some of the terms of this chapter differ from that of the previous ones due to the more practical focus of the work (see section 2.3). For example, we use the term router instead of nodes, and links instead of edges. A conceptual difference is that in the Internet, we may have multiple sources in a session (many-to-many communication). The data may be distributed over a) one distribution tree (Shared Tree) for all sources, or b) each source may create a separate tree (Source-Based Tree), or c) a combination of a Shared Tree and a number of Source-Based Trees. Another difference is that we are interested in more aspects of the performance of the protocol in addition to the routing efficiency (which has been the main concern in the previous chapters).

Our protocol was designed with the following primary goals.

- *QoS Support.* We want to provide a framework to support a wide range of QoS requirements such end-to-end delay. We support QoS in two major ways. First, we offer multiple paths with dynamic information when a destination joins, so the destination can choose the path that fits its needs best. An everyday analogy is driving a car: we want to have the option of many ways to go someplace and up to date information (usually through a radio station) on the traffic condition of each of the alternatives. Second, when a destination is not satisfied with the end-to-end QoS from the source through the Shared Tree, it can join the Source-Based Tree of that source, and this would most likely provide better end-to-end performance.

- *Limited Impact of Pre-configuration Decisions.* We want to limit the impact of any *a priori* configuration decisions, such as the choice of special status routers (PIM-SM, CBT, and BGMP), or a special partitioning of the multicast address space (BGMP [TEM97] [EHKT97]).

In our effort to propose an improved protocol, we identified several of weaknesses of some of the current and proposed MBone protocols. We transformed these weaknesses into the following secondary design guidelines for our protocol.

1. *Efficiency.* It should construct distribution trees that use the network resources efficiently.

2. *Application sensitivity.* It should accommodate diverse application types with minimal user input. For example, a real-time video connection has different requirements from a banking transaction: real-time video requires bounded end-to-end delay and is tolerant of packet losses, while the banking transaction is intolerant of losses but much more tolerant of delay.

3. *Scalability.* It should scale well for large networks, many groups, and large group-sizes.

4. *Robustness.* It should be robust to failures, even of special status components, e.g. *core* routers.

5. *Loop-freedom.* It should not create cycles.

In selecting among paths, we suggest the use of dynamic metrics, e.g. available bandwidth, current delay. These provide paths that can meet QoS requirements at a given moment. Furthermore, routing with dynamic metrics can respond pro-actively to link congestion. The reasons why previous protocols did not suggest the use of dynamic information is that they would require a supplementary protocol to provide the information, and such a protocol would cause problems in scalability. In QoSMIC, we propose a way to collect the up to date information without relying on an external protocol to provide this QoS information of a path. Note, however, that we assume the existence of unicast routing information, which is necessary anyway for unicast routing. For the rest of this document, the term "QoS of a route" denotes the quality that the route is expected to deliver to a connection given its current state, and is calculated from dynamic metrics of properties such as bandwidth or delay. However, the terms "distance" and "proximity" for routers are defined using static metrics of the same properties. It is important to stress that our protocol is compatible with any metric of the routing quality of a path, and the specific metric to use is application dependent. Recall the example of real-time video that requires bounded end-to-end delay, and the banking transactions that are intolerant of packet losses that we mentioned earlier. To summarize, our protocol offers the choice among multiple paths with up to date information. It is then up to the application and the receiver to select the path that suits its needs.

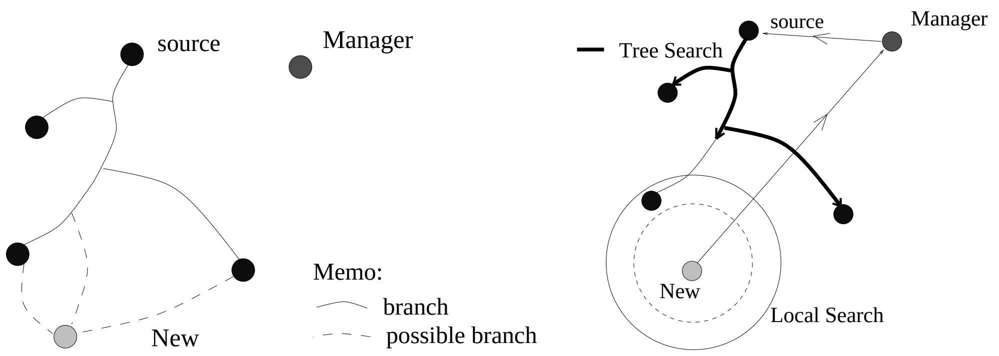
QoSMIC constructs trees based on a Greedy heuristic [TM80], that connects each user to the "closest" branch of the existing tree and leads to efficient resource use[1]. Our protocol offers alternate paths to enable the support of QoS requirements. The protocol collects dynamic routing information, without assuming an underlying protocol to provide it. Another innovation is that QoSMIC does not require pre-configuration decisions; its efficiency does not depend on the selection of a special status router, or a special partitioning of the multicast address space. Finally, our protocol can be seen as a flexible framework that encompasses the behaviour of most of the previous protocols, when sensitivity to QoS is not required.

5.1 Overview

Our protocol creates Shared Trees by default and Source-Based Trees when needed. In both cases, the protocol offers alternate paths for each connection (see fig. 5.1(a)).

We introduce the notion of a **Manager router** of a group. The Manager administers a specific multicast group, and facilitates the joining of the new group members. The fundamental difference between a *core* router and a Manager is that the distribution tree is not necessarily rooted at the Manager. This way, the selection of the Manager has marginal effect on the topology of the tree.

[1] QoSMIC can easily shift to a core-like Naive routing, as we will see in section 5.2. However, at this point, we have indications that argue strongly in favour of the Greedy approach as we will see in chapter 6.



(a) Identifying multiple routes.

(b) The Local Search and the Multicast Tree Search procedures.

Figure 5.1: An overview of QoSMIC.

Furthermore, we can have multiple Managers and change the Managers during the lifetime of the group without any data loss, as we will see later in this section.

Joining a group. The Designated router of the new member, which we call **New router**, will try to connect to the most promising router of the tree, according to QoS metrics defined by the application. The New router identifies several intree routers as **Candidates** or potential points to join to the tree. This search is conducted by two procedures: one from the side of the New router (Local Search), and one from the side of the tree (Multicast Tree Search) (see fig. 5.1.b), both operating in parallel. For the moment, we consider a purely intra-domain or purely inter-domain scenario.

1. *The Local Search Procedure.* The New router searches its neighborhood, using of probe messages, with scope limited by use of the time-to-live (TTL) field for its closest intree router. Every intree router that receives the probe message, and responds with an "advertisement" message unicast to the New router. This message lets the New router know that the intree router is a Candidate, i.e., a possible point to join the tree. This procedure is also used in the YAM protocol.

2. *The Multicast Tree Search Procedure.* The New router contacts the Manager router of the group, and the Manager router "informs" the tree of the New router. Some intree routers are selected as Candidates. They advertise themselves to the New router with a unicast "advertisement" message. In the next section, we propose several mechanisms, centralized and distributed, to select Candidates.

Eventually, the New router compares all the identified connection paths, and selects the best among them according to the needs of the application. The New router sends one more message (JOIN) towards the selected Candidate, to set up the routing state along the path and the chosen router starts forwarding the data. The routing state is fixed or pinned (in the Internet community jargon), and the route does not change once it has been established. It should be clear from the above description that our protocol corresponds to an implementation of the Greedy algorithm. The New router searches for the "closest" point of the tree and connects there.

The path chosen by the advertisement messages depends on the **static** QoR metrics of the point-to-point routing table. The specific metric used by the routing protocol depends on the

Router Name	Role
Manager router	Supervises the group and facilitates the joining procedure; does not necessarily participate in the data distribution tree.
Candidate router	Considered as possible joining point for a new connection.
Designated router	Connected to a set of users (e.g. LAN), receives requests from users (e.g., using IGMP) and initiates searches.
New router	Designated router of the LAN containing the new member.
Destination router	Designated router of a LAN that has active group members.
Intree router	All routers that are part of the distribution tree.
Intermediate router	Intree router that is not a Destination or a Source.
Border router	Router that connects multiple domains.

Table 5.1: The different roles of the routers.

needs of the application. For example, real-time applications will prefer paths of minimum end-to-end delay, while data transfers may prefer paths of maximum bandwidth or low packet-loss ratio. Current point-to-point routing protocols already consider multiple static metrics.

However, the advertisement messages can collect up-to-date **dynamic** QoS metrics along the path on which they travel (e.g, by interacting with RSVP [ZDE$^+$93], or using measurement based metrics). Thus, although the paths that the Candidates choose is restricted by the static information in the routing information base, the New router selects among these paths using dynamic routing information.

Source-Based Trees. Using Shared Trees minimizes the routing table information to an entry per group. However, Shared Trees may fail in the following two cases. First, when a group has many highly active sources simultaneously, the bandwidth of the shared links may not be able to accommodate all the traffic. Second, when the QoS requirements of a user are not met along the Shared Tree, we have to find a different source-to-destination path. In both these cases, we resort to Source-Based Trees. The switch from a Shared Tree to a Source-Based Tree of a specific source is initiated by the receiver. The procedure for establishing the Source-Based Tree is similar to the procedure of the Shared Tree and uses the Local Search and the Multicast Tree Search procedure to identify routers *in the Source-Based Tree.* To avoid packet duplication, the Shared Tree is pruned on a source-specific basis, for the sources for which Source-Based Trees have been established. More details for the co-existence the two types of trees is provided in section 5.3.

5.2 Messages and Mechanisms

In this section, we describe the messages and mechanisms used by our protocol in detail. After that, we explain the process of Candidate selection, Manager selection, and interdomain operation. Table 5.1 lists the different roles of the routers. Table 5.2 contains a list of the protocol messages.

Every router maintains multicast routing information, which enables it to forward each multicast packet to an appropriate number of links. In more detail, each router maintains a routing table for each link (see fig. 5.2). A link has an entry in its table for each group for which the link is part of the distribution tree. We denote such an entry for the Shared Tree of group G by (*, G); the symbol "*" represents "all sources". A data packet of a group is forwarded to all links that have the (*, G) entry, except from the link that it arrived from. A router discards a data packet

MESSAGE	Explanation
BID-REQ	New router searches locally for intree neighbors
BID	Candidate "proposes" to the New router; Message collects dynamic QoS metrics along the path
M-JOIN	New router contacts the Manager who initiates a Multicast Tree Search
BID-ORDER	The Manager "orders" the intree nodes to send bids
JOIN	The New router sends a message up its selected path to the tree to establish/refresh routing state
PRUNE	Departing Destination tears down unneeded part of tree; can be for a source or for a group

Table 5.2: An explanation of the messages of our protocol.

that arrives on a link that does not have a (*,G) entry, which means that is not part of the tree. Furthermore, a table entry can indicate a direction: it may permit receiving data, sending data or both. Furthermore, a table entry can refer to a specific source of a multicast group (source specific information), and we denote these entries by (S,G) for source S of group G. A link with a (*,G) entry may have a source-specific exception: the link carries data from all source of the group except a specific source. We denote this state by "($\bar{S}$, G)". This way, we can implement Source-Based Trees and have them exist in parallel with Shared Trees without creating loops in the distribution of data (see 5.3). Finally, the state of a link, (*,S) or (S,G), may be tentative: the link is considered in the distribution tree, but no data is forwarded on that link. The state of a link is modified by protocol control messages, as we will see in this section. In addition, state can expire: adjacent nodes exchange messages periodically, and refresh the state of their common links. If such a state refresh does not happen (due to link or router failure), the state of the link expires.

If a router has links with (*,G), we say that it belongs to the (*,G) Shared Tree. If it has (S,G) links, then it belongs to the (S,G) Source-Based Tree. A router can belong in both the Shared Tree and the Source-Based Tree at the same.

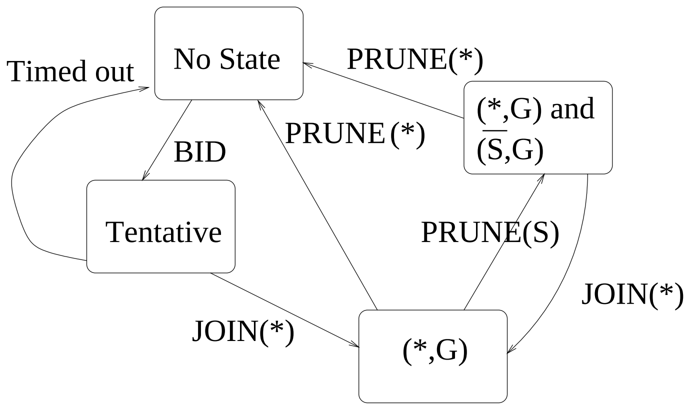
5.2.1 The Search for Candidates

We assume that the New router receives a request to join a multicast group. To simplify the presentation, we assume that the new member is a receiver[2]. The Join is received via IGMP on a LAN at the intra-domain level, or from an intra-domain protocol at a Border router at the inter-domain level. If the New router is part of the group already, the connection is established locally. If the New router is not part of the group, the Local Search and the Multicast Tree Search procedures are employed.

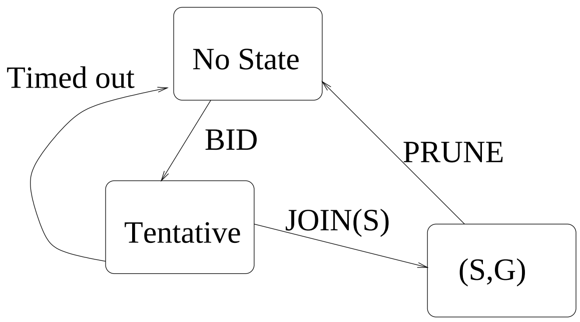
Local Search. The New router tries to identify neighboring intree routers.

1. The New router "floods" a BID-REQ message in its neighborhood. The extent of the search is controlled with the use of a special field in header of the Internet packets, the Time To Live (TTL) field. This field is initialised to a value, it is decreased by one at each hop, and

[2] A source joins a group in a way similar to a receiver; it connects to the closest router of the Shared Tree. A difference is that the dynamic metrics of the path are collected "towards the tree", i.e., in the direction the data will flow.



a) For Shared Trees



b) For Source Based Trees

Figure 5.2: The state transition diagram of a link. (Protocol actions, messages or timer expirations, cause the transitions.)

when it reaches zero, the messages expires. Furthermore, the search does not have to be a flooding of the neighborhood. Reverse path broadcasting can be used to avoid the exchange of redundant messages [DC90]. In more detail, a router ignores probe messages that do not come from the edge that is on the shortest paths from the node towards the New router. This way, each node forwards a specific message only once. This procedure is the same as proposed in YAM, but because we also have the Multicast Tree Search, we can keep the scope of the Local Search. This advantage is quantified later analytically and experimentally.

2. Every intree router that receives a BID-REQ message becomes a Candidate router, and replies with a BID message, which is unicast to the New router. The BID message informs the New router of the option of joining the sending router. Furthermore, the BID message on its way collects information on the expected performance of the path, based on dynamic QoS metrics[3]. The Candidate router considers the New router as a tentative dependant. Having a tentative dependant means that the router cannot leave the tree (even if all real dependants leave), but the router does not forward data towards a tentative dependant. The tentative

[3]A Candidate does not actively "bid" since it just makes its existence known. However, from the New router's point of view, the messages resemble bids with the dynamic information being the "bidding" values.

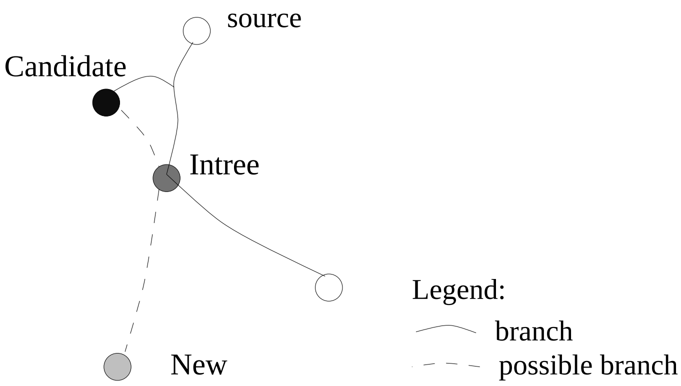


Figure 5.3: The Take-Over procedure: an intree router that lies in the path from the Candidate to the New router can replace the Candidate.

status either times out or is confirmed and a connection is established, as we will see later in this section.

3. The New router collects the BID messages. The procedure terminates unsuccessfully if the New router does not receive any replies before the expiration of a timer set for this purpose. Otherwise, we enter the phase of establishing the connection (see section 5.2.2).

Multicast Tree Search. The New router contacts the Manager and the Manager causes some of the intree nodes to propose themselves as Candidate routers. The selection of Candidates is an important aspect of our protocol and it is discussed later in this section. The sequence of actions is as follows:

1. New router sends an M-JOIN message to the Manager of the group.

2. The Manager "orders" a bidding session with a BID-ORDER message. Some subset of the routers that receive the BID-ORDER are selected as Candidates.

3. The Candidates unicast BIDs to the New router. The BIDs are identical to the BIDs in the Local Search.

For both the Local and the Multicast Tree Search, if an intree router receives a BID message (for the same tree), the router takes the place of the Candidate by "dumping" the original BID and initiating a new BID message. This procedure is called **Take-Over** (see fig. 5.3), and it is essential in preventing cycles. The main idea is that if an intree router (or one with a tentative dependant) find itself between a Candidate and a New router, it interferes to prevent loops as we will see later in this section.

5.2.2 Connection Establishment

Having performed the BIDding phase, we will examine how the connection is established.

1. The New router selects the best Candidate according to the dynamic QoS metrics collected by the BIDs.

2. The New router sends a JOIN message to the Candidate that sent the best BID. This message traverses in the opposite direction the path used by the BID message, and establishes routing state along the path.

3. When the chosen Candidate receives the JOIN message, it starts transmitting data packets on the newly set up path towards the New router.

4. If an intree router receives a JOIN message, it performs a Take-Over, terminates the message and starts forwarding data on the newly set up path. This way we avoid the creation of cycles (see fig. 5.3).

5.2.3 Leaving a Group

A user may request to leave from a distribution tree that can be either a Shared Tree or a Source-Based Tree using the IGMP protocol (see section 2.3). The directly attached router has to take care of this request. When the router realizes[4] the absence of participants in the direction of one of its links for some distribution tree, it removes that link from the distribution tree. Then, the router checks whether it has become a leaf in the related tree, in other words, it checks whether it has only one link in the tree. If this is the case, it sends a PRUNE message along its last remaining link of the tree and removes the state from that link, thereby ceasing to be an intree router for that tree.

5.2.4 Candidate Selection.

During the Multicast Tree Search, the Manager must select an appropriate subset of the intree routers as Candidates. This can proceed in a centralised or a distributed way.

1. *Centralised Selection.* If the Manager has sufficient knowledge of the tree and the network topology, the manager can select the set of Candidates directly. This can happen if the domain is running a link state protocol, or if the manager has access to a routing information database. In this case, the Manager identifies promising Candidates, either on demand or statically. Then, the Manager unicasts the BID-ORDER to each of them. If the selection is based on accurate information, we can find the right Candidates with limited overhead cost. Note, that with this mechanism, our protocol can behave like CBT or BGMP; a Manager can trivially act as a *core* router by considering itself as the only Candidate. Note that this implements the Naive routing where all nodes join at the *core*. Furthermore, the Manager can preselect a number of routers as designated *core*s and Candidates. This approach leads to a hybrid routing scheme, i.e., a combination of Naive and Greedy.

2. *Distributed Selection.* In the absence of centralised information, each intree router has to decide whether or not to become a Candidate in a distributed way. We assume that each router has a static estimate of its distance to the New router. We have the Manager join the tree as a source[5] and multicast the BID-ORDER along the tree. All intree routers receive the BID-ORDER and decide whether to become Candidates. Naturally, the more intree routers that become Candidates, the more "accurately" we find the closest Candidate to the New

[4]The membership change can be the result of a request of directly-connected user (IGMP protocol) or a PRUNE message from a neighboring router.

[5]Note, that this does not contradict the flexibility of the role of the Manager, since it only transmits control messages and receives nothing. For example, the Manager can change without disrupting the data distribution.

router, but the greater the control overhead. We propose the following orthogonal mechanisms that can be used in combination to strike the balance in this trade-off.

(a) *Directivity.* We discourage distant routers from becoming Candidates. The BID-ORDER keeps track of the smallest distance yet detected between the tree and the New router. A router with a distance greater than this minimum does not become a Candidate, and, if the distance of the router exceeds a threshold, the BID-ORDER is not forwarded further[6].

(b) *Local Minima.* In absence of global knowledge, we select locally optimal routers as Candidates. The BID-ORDER message carries the distance to the New router of the two previously visited routers, which we denote by $i, i-1$. Router $i+1$ compares the two distances with its own. If router $i+1$ sees that router i was closer to the destination than both routers $i+1$ and $i-1$, it sends a message back to router i, which becomes a Candidate.

(c) *Fractional Choice.* We can choose as Candidates a fraction $(1/n)$ of either all intree routers or the ones that meet the other two criteria. For the implementation, we only need a $\log n$-bit wrap-around counter in the BID-ORDER message. At each step, the counter decreases and when it reaches zero, the router becomes a Candidate, and re-initializes the counter (the initial value is also included in the message). Note that the number of Candidates is approximately $1/n$ (and the exact number depends on the topology of the tree). Choosing a value for the counter depends on various parameters such as the size of the network, and the current or expected size of the group. Note that we can have other ways of implementing this mechanism, for example, we can have every router decide randomly: most switches have random number generators. The random number can follow a uniform distribution, and with the use of threshold value in the message, we can control the number of Candidates on average.

Choice of Mechanisms. We identify combinations of the previous mechanisms that seem more promising. The final choice will have to consider the network topology and the traffic behavior. Some preliminary analysis is presented in section 5.5.

If topology information for the entire domain is available, Manager Selection offers the lowest control overhead for a reasonable selection of Candidates. In the absence of such information, the combination of Fractional Choice and Directivity together is simple to implement, and may lead to a satisfactory solution with a careful choice of parameters. The combination of Local Minima, Fractional Choice and Directivity is more sophisticated and can improve the Candidate selection procedure.

5.2.5 Manager and Inter-domain Issues

Manager Address Distribution. For the Multicast Tree Search, the New router needs to know both the address of the Manager and of the group, and the two addresses can be made known through the same mechanism (e.g., Session Directory tool [HJ95]). Alternatively, the address of a local Manager within a domain can be broadcasted via the Domain Wide Reporting [Fen97] or can be provided on demand (by the Border routers or an administrative database).

[6] Note that a similar mechanism appears in YAM [CC97a], but it is used to reduce the overhead of its Local Search. YAM needs to assume rooted trees for this.

Manager versus Core Router. The main purpose of the Managers is to invoke the Multicast Tree Search. Since Managers are not key routers for data distribution, we can replace a Manager during a session by simply providing notification of a new Manager. In contrast to a *core* change, a Manager change does not cause any data loss or any change in the distribution tree. For a smooth transition, the old Manager can "resign", after the new Manager has been around for a sufficiently long time (depending on the size of the network).

Multi-Domain Multi-Level Operation. Our protocol can be used at both the intra-domain and the inter-domain level. If the domain already has active group members, the search for Candidate routers is carried out within the domain. If not, the connection has an intra-domain and an inter-domain part. Inside the domain, the New router contacts one or more Border routers. The Border routers find their best path to the group at the inter-domain level. If the inter-domain protocol is QoSMIC, then the Border routers initiate searches using the procedures of QoSMIC. Each Border router that finds a path to the tree proposes itself to the New router (BID message). The New router chooses the "best" Border router. The selected Border router becomes the Designated Border router for this group (or Source-Based Tree). Finally, if one of the levels is running a different protocol, an interface similar to that presented by Thaler et al. [TEM97] will allow interoperation.

Selecting Manager Routers. In practice, we want to have multiple Managers that will co-operate for efficient and scalable solutions with reduced set-up time. For this, we have developed the following scheme. We have at least one Manager per domain, and more Managers in large domains. This way the intra-domain and inter-domain protocols can be independent. We prefer that Managers be Border routers. This way, the Manager can communicate with routers inside and outside the domain more easily. Thus, by default, we select the Designated Border router of the group to be the Manager in a domain. In the creation of a session, the Border router nearest to the session initiator becomes the Designated router. For a domain that joins the distribution tree, the selection of the Designated router was described in the previous paragraph. For a Source-Based Tree, we select the source as the Manager in its own domain to simplify the administration of the tree. It is important to note that these choices are the default ones, but as the execution progresses, we can change to more appropriate Managers (e.g., when the Manager fails) without having to disrupt the data distribution.

5.3 Implementation Overview

The goal of this section is to highlight some implementation details of our protocol.

5.3.1 Intree Router State

We indicated the different states that link can be in the previous section (see fig. 5.2).

- A (*, G) entry corresponds to the Shared Tree of a group. By default, it is bidirectional.
- An (S, G) entry corresponds to a Source-Based Tree with source S and group G. It is by definition uni-directional, so we can distinguish IN-only (S, G, IN) and OUT-only (S, G, OUT) entries.
- We enable source-specific information. More specifically, we can exclude a source from the Shared Tree . We denote this state by ($\bar{S}$, G). A packet from source S that arrives from a link with such state is dropped. The main reason is to avoid packet duplication for destinations that have joined an Source-Based Tree as we explain in section 5.3.3.

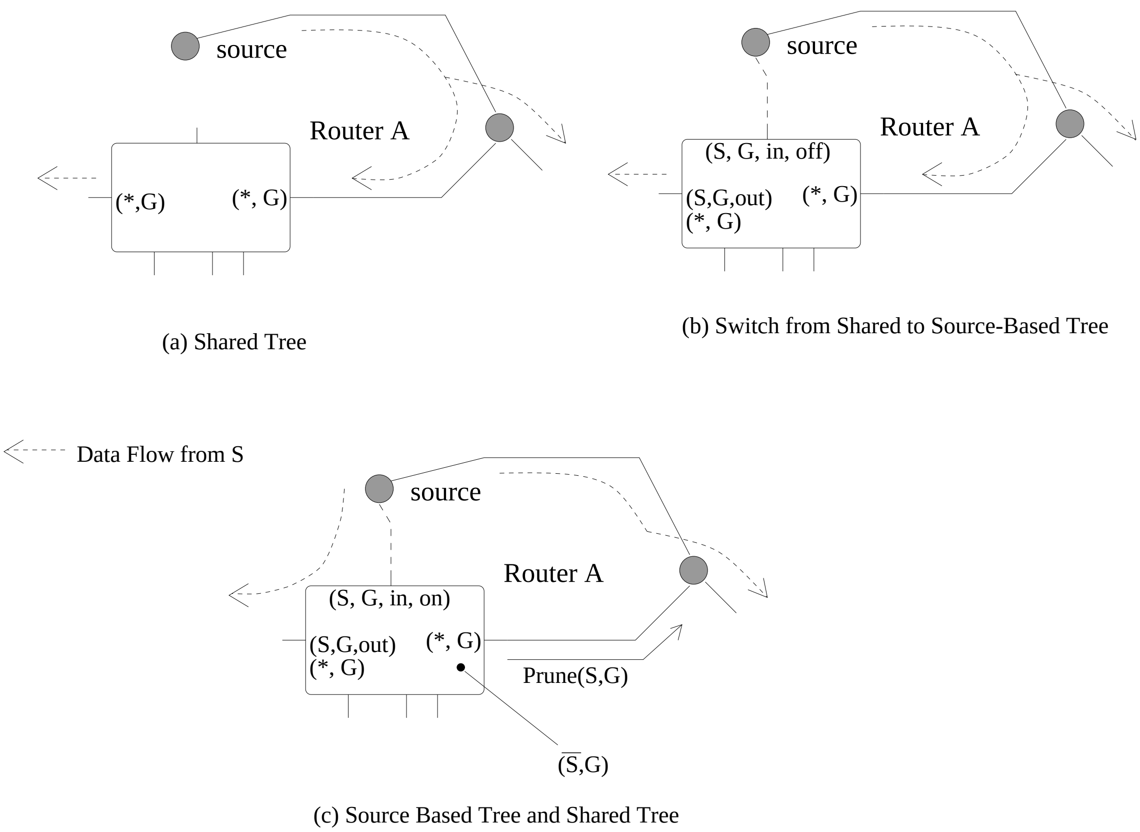


Figure 5.4: The entries at the routing table for Shared Trees, Source-Based Trees and their transition. The figure shows the flow of packets from the source.

We use (*/S, G) as an abbreviation of the expression (*, G) or (S,G). The control messages always refer to a specific distribution tree that is either the Shared Tree, (*,G), or the Source-Based Tree, (S,G). This way, we use the expression "a (*,G) control-message" in the place of the longer "a control- message that refers to the (*,G) distribution tree".

5.3.2 Router Actions

Here we study the actions of a router as triggered by control messages.

Joining a session

The request (*/S, G) can be either to the Designated router (via the IGMP protocol) or the border router (via an intradomain protocol).

If the router is already part of the requested (*,G) or (S,G) tree, the join is accommodated there. If not, the router considers itself a destination router that we denote as New router, and performs the Local Search and Multicast Tree Search procedures. For the Multicast Tree Search, the New router sends an M-JOIN(*/S, G, New router) to the Manager of the group. For the Local Search, the New router broadcasts BID-REQ(*/S, G, New router, TTL) message in expanding rings. The messages follow a reverse path multicast. The TTL is initialised to a value as we saw before. A timer is set to monitor the responses. If the timer expires before any responses, then a new broadcast (ring) takes place with an increased TTL. The value of the TTL is bounded by a default value or is explicitly specified. When the timer expires for the maximum TTL value, without any responses, the procedure terminates unsuccessfully.

BID-REQ(*/S, G, New router, TTL) message.

On reception of a (*/S,G) BID-REQmessage, a router that is part of the tree in question replies with a BID message. In a Source-Based Tree, the Candidate incorporates in the message estimates of the QoS it experiences (quality of the path from the source to the Candidate), when these are available. Note, that this way, we can maximize the end-to-end QoS that the New router will receive. In more detail, the BID message is of the form BID(*/S, G, New router, Candidate, Metrics): it includes the address of the Candidate, and estimates of the QoS of the path. The BID message on its way updates its metrics at each router to estimate the quality of the path. Examples of QoS metrics are the minimum available bandwidth of a path, and end-to-end hop-count as we mentioned in the introduction of this chapter 5.

MJOIN(*/S, G, New router) message.

This message is unicast to the Manager of the group. The Manager sends BID-ORDER(*/S, G, bid-mechanism). The "bid-mechanism" represents the selection mechanisms that we saw earlier. The BID-ORDER can be either multicast over the tree, or unicast to pre-selected routers depending on the mechanism in use.

BID-ORDER(*/S, G, New router, bid-mechanism) message.

The actions on this message depend on the "bid-mechanism". In general, the message is either unicasted to the chosen Candidate or multicasted over the tree. In the first case, on receiving the message, the router unicasts a BID(*/S, G, Metrics) to the New router. In the second case, the router examines the criteria defined by the bid-mechanism, and if they are fulfilled, a BID message is sent to the New router, and the BID-ORDER is forwarded further along the tree.

BID(*/S, G, New router, Candidate, Metrics) message.

On receiving this message, if the router is the New router specified in the message, it updates a list of the best BIDs. An intree router invokes the Take-Over procedure, and the router replace the Candidate. Any other router that receives a BID message, updates the Metrics field and forwards the packet. Typical metrics of interest are the minimum available bandwidth of a path, hop count, average delay.

JOIN(*/S, G, New router, Candidate) message.

This message travels towards the selected Candidate. When the Candidate router receives the message, it considers the connection as "established" (as opposed to "tentative", see fig. 5.2) and sets-up the path. Any intree router that receives the message invokes the Take-Over procedure and starts forwarding data.

Leaving a Group

The leave request (*/S, G) can be either to the Designated router (via the IGMP protocol), the border router (via an intradomain protocol), or with a PRUNE message, or the result of state time out. If the request arrived from a LAN with active group members, the router does not do anything. In any other case, the link is removed from the distribution tree. Then, the router determines whether it has become a leaf of the tree.

1. If the router has become a leaf of the (*/S,G) tree[7], it sends a PRUNE(*/S,G) message along its only tree link.

2. If the router has two or more links in the tree (tentative or real), it takes no further action.

[7]Note that leaf here implies also that the router does not have any hosts attached to its LAN.

The Take-Over procedure.

An intree router that receives a message between the Candidate and New router, may be able to take the position of the Candidate (see fig. 5.3). This could lead to shorter delays and more efficient trees. This interception can happen in the messages: BID, JOIN.

- The message concerns the (*, G) tree.
  - If the router is in (*, G), then Take-Over.
  - In any other case, forward the message. This applies even if the if the router is in a Source-Based Tree of that group.
- The message concerns the (S, G) tree.
  - If the router is in (S,G), then Take-Over. Optional refinement: if the current QoS from the source to here is worse than the expected QoS along the bidding path, then switch to the best path.
  - If the router is in (*,G), then if the current QoS is not worse than the QoS of the bidding path, then Take-Over, otherwise just forward the message.

5.3.3 Source Specific State

In QoSMIC, we can have source-specific pruning on the Shared Tree (see fig. 5.4). A destination joins the Source-Based Tree for a specific source, S, as we already mentioned. In brief, the destination joins the Source-Based Tree of S, and sends a source-specific prune (PRUNE(S)) message along the Shared Tree. This way, data packet from S arrive along the Source-Based Tree, while packets from other sources of group G continue to arrive over the Shared Tree (see fig. 5.4). A router that is part of both the (S, G) and the (*, G) tree, ignores packets from source S that do not arrive from the (S,G, IN) link. Furthermore, when a router receives an (S,G) packet from another link (the link that the Shared Tree used for the packets from source S), it drops the packet. We denote by (*,G)($\bar{S}$, G) a Shared Tree link that does not "allow" (S,G) packets.

In more detail, when we switch from the (*, G) to the (S, G) tree, we want to minimize the packet loss. For this reason, we drop packets from source S over an (*,G) link only when data starts to arrive from the (S,G) tree. In this procedure, we use a one-bit flag that we denote by SBT-FLAG. When the request for Source-Based Tree is sent, we install an (S,G, SBT-FLAG= OFF) entry, and we continue to forward packets from S that we receive from the (*,G) link. The first data packet (from S) on the (S,G) link sets the flag (SBT-FLAG= ON). When this happens, we drop any packets from S coming from the (*, S) link, and we send a source-specific PRUNE(S) messages along that link to stop the transmission of the packets from S even earlier in the Shared Tree tree.

5.4 Correctness

In this section, we prove that our protocol creates loop-free distribution trees, independently of loops the unicast routing tables may have. Among the protocols that we discuss, the MIP, OCBT, and HIP protocols have the same property [PGLA97] [SGLA97] [SGLA98]. Note that in this section we do not consider failures and we assume that the timers for QoSMIC are set appropriately.

Lemma 1 *A tentative path is loop-free independent of the underlying unicast protocol.*

Symbol	Explanation
t	The maximum TTL value of the Local Search
c	The number of Candidates
f	The fraction of routers that become Candidates according to the Fractional method (approximately 1/(initial value of wrap-around counter))
h	The fraction of the tree that participates in the Multicast Tree Search in the Directivity method
w	The average degree of a router
$\|T\|$	The average size of a multicast tree
CR	A Candidate router
MR	The Manager router
NR	The New router
$\text{Hop}(a,b)$	The average distance in hops from a to b
Hop_{avg}	The average hop distance between Manager and Candidates
$Join(c)$	The message complexity of the joining part with c Candidates

Table 5.3: Analysis parameters and their symbols.

PROOF. Recall that a tentative path is established by a BID message. The routers of the path store the addresses of the group, the Candidate, the New router, and the source for a Source-Based Tree, as part of the tentative state. If the BID message returns to a previously visited router, the router ignores the message. This way although there is a loop formed, the path never reaches the destination, so it never qualifies for a "tentative" path. Eventually, the tentative state is timed out. ■

Theorem 12 *The routing table information on all the routers for a group (Shared Tree) or for a source of a group (Source-Based Tree) constitutes a forest*[8] *at every instant.*

PROOF. It is sufficient to prove that the union of a joining path (the part of the tree added to include a new destination) and the distribution forest is also a forest, assuming that the distribution tree is initially a forest.

First, a joining path is loop free, since it follows the path set up by a tentative path, which is loop free. Second, a joining path contains only one intree node at its end. The JOIN message traverses the tentative path and it stops at the first intree router that it encounters (Take-Over procedure). This applies even for the case where two joins happen simultaneously and their joining paths intersect. In such cases, at the first router common to the two paths, one of the JOIN messages is processed first. Therefore, when the second message arrives the router already belongs to the tree and the Take-Over procedure terminates the second path there.

In case the tree becomes disconnected due to a failure, we end up with a forest. The re-join procedures are handled like join attempts and do not threaten the forest or loop-free properties. ■

5.5 The Control Overhead

We compare the control overhead of YAM and QoSMIC with analytical methods. We do not consider other protocols, since they do not try to achieve QoS and resource optimized trees. Table 5.3 lists the parameters and their symbols. Most of these parameters depend on the topology and the membership behavior of the applications. In Table 5.3, the first three parameters can be altered by the protocol. Parameter t depends on protocol decisions solely in all cases. Parameter f is the fraction of routers that become Candidates according to the Fractional method. Parameter c represents the number of Candidates for the Multicast Tree Search is directly defined by the protocol for the centralized case, and indirectly controlled for the distributed cases. Note that the protocol can modify its parameters during the life-time of the group, and this accounts for the adaptability of our protocol. The fraction of the tree that participates in the Multicast Tree Search in the Directivity method varies on each execution, and we denote it by h. We denote by CR the Candidate, by MR the Manager, and by NR the New router.

The Complexity of Selecting Candidates.

The Local Search. Assume w to be the average degree of a router. For simplicity, we consider only one search with a maximum TTL value of t. We consider the complexity to be a function of the t, since the average degree, w, is given. In a broadcast, a router that receives a message will forward it to $w-1$ links on average. Assuming a reverse path multicasting method, every router in the neighborhood receives such a message exactly once. This way, the approximate complexity[9] of the Local Search with one expanding search is as follows

$$Local(t) = w\cdot(w-1)^{t-1}$$

While for multiple expanding rings the complexity becomes

$$Local(t) = \sum_{i}^{t} w\cdot(w-1)^{i-1}$$

Since YAM depends entirely on the Local Search to find the Candidates, we can assume that YAM will need to perform expanding-ring searches to keep the complexity under control, while QoSMIC can perform a single search with a small t, which explains the complexity difference in Table 5.5.

$$Local(t) = \sum_{i}^{t} w\cdot(w-1)^{i-1}$$

The Multicast Tree Search. This procedure exists only in QoSMIC, and can differ depending on which Candidate mechanism is used. Let us denote by Hop_{avg} the average hop distance between the Manager and the Candidates. The complexity is $c\cdot\text{Hop}_{avg}$ for the Centralised mechanism. For the distributed mechanisms, recall that the BID-REQ message propagate along the whole tree, and the size of the tree, $|T|$, is an upper-bound on the number of messages. In more detail, Table 5.4 demonstrates the complexity of the selection procedure and the relative advantages for each of the four Candidate mechanisms we saw earlier. In the Fractional mechanism, the BID-REQ message visits the whole tree, and the number of candidates is a fraction of the intree routers, $fr\cdot|T|$. If on top of that we assume the Directivity mechanism, distant parts of the tree do not participate

[8] A forest is defined to be a union of disjoint trees.

[9] We calculate an upper bound of the complexity based on the average degree of a the graph. A tighter bound would depend on the topological properties of the graph.

Label	Method	Messages	Candidates	Pros	Cons
C	Centralised	$c{\cdot}\mathrm{Hop}_{avg}$	c	Low control overhead; Local Search can be omitted	Requires some aggregation of information
F	Fractional	$\|T\|$	$f{\cdot}\|T\|$	Simple	Random selection of Candidates
FD	Fract., Directivity.	$h{\cdot}\|T\|$	$h{\cdot}f{\cdot}\|T\|$	Simple; Limits the extent of the Multicast Tree Search	
LM	Local Minima, Fract., Direct.	$h{\cdot}(\|T\|+l)$	$f{\cdot}h{\cdot}l$	Intelligent selection of Candidates	Control of the number of Candidates

Table 5.4: Comparing the four Candidate selection methods in the Multicast Tree Search.

Protocol	Type	Routing Information	Number of Messages	Message Complexity
QoSMIC-Centr.	Central./Distr., Adaptable	Dynamic	$w{\cdot}(w-1)^{t-1}+c{\cdot}\mathrm{Hop}_{avg}$	Low
QoSMIC-Distr.	Distributed, Adaptable	Dynamic	$w{\cdot}(w-1)^{t-1}+\|T\|$	Medium
YAM	Distributed	Static	$\sum_i^{t'} w{\cdot}(w-1)^{i-1},\ \mathbf{t'} \gg \mathbf{t}$	High

Table 5.5: A comparison of the search procedures of QoSMIC and YAM. Adaptable complexity type means that during the execution the protocol can limit the overhead while still offering at least one path per join.

in the Candidate selection phase. We denote by h the fraction of Candidates over the number of Candidates that would exist, if Directivity was not employed. Finally, adding the Local Minima mechanism increases the message complexity by one message for each local minima, and restricts the Candidates only to routers that are locally closest to the New router, as we explained in section 5.2.

The Complexity of Joining. In any case, the joining overhead is low compared to the overhead of the search. Assume we have c Candidates. Each of them sends a BID message to the New router. The New router sends a JOIN message to one selected Candidate. The associated complexity is: $Join(c) = (c{+}1){\cdot}\mathrm{Hop}(CR, NR)$. This complexity is approximately the same in both YAM and QoSMIC as long as the number of candidates selected is kept roughly the same.

We compare the performance of the selection of Candidates in QoSMIC and YAM protocols according to: the message complexity of searching, the type of the selection, the information used in the selection. The comparison is shown in Table 5.5. In terms of messages, QoSMIC can reduce drastically the complexity given that it can rely on the Multicast Tree Search procedure. In the centralised case, it can be limited arbitrarily (down to one Candidate), and if necessary, we can also dispense completely with the Local Search. In the distributed case, the Multicast Tree Search complexity is bounded above by the size of the tree. YAM relies uniquely on the Local Search procedure whose extent should be large enough to reach the tree, and thus, we claim that $t' \gg t$.

5.6 Conclusions

We propose QoSMIC, a protocol for supporting QoS-sensitive multicast applications over the M-Bone. Our protocol identifies multiple paths, and selects the most promising one using dynamic information that the control messages collect. Our protocol creates Shared Trees by default and destinations switch to Source-Based Trees in order to accommodate the needs of different applications. This switch can take place for highly active sources or when the QoS requirements of receivers are not met.

Our protocol includes the main concepts of the YAM protocol [CC97a], and introduces several new ideas. Below, we list the characteristics of our protocol that differentiate it from YAM and the other protocols.

1. *Limited Impact of Pre-configuration Decisions.* We separate the management from the data transfer functions, by introducing the concept of the group Manager in the place of the *core* router (PIM, CBT, BGMP, MIP). The choice of the Manager does not affect the distribution tree. This way we eliminate the problems of *core* router selection and the address partitioning problems of other protocols.

2. *Dynamic Routing Information.* Our protocol uses dynamic routing information without relying on external mechanisms/protocols for information exchange.

3. *Scalability.* We introduce the Multicast Tree Search, which allows us to reduce the complexity of the Local Search. This complexity appears to be a major weakness in YAM.

4. *Efficient Resource Use.* QoSMIC combines Greedy and Naive routing. Here we focus on Greedy routing, since Greedy outperforms Reverse Shortest Paths routing used by other protocols.

5. *Sensitivity to Direction.* QoSMIC computes the performance metrics of the paths in the direction that the data will flow. This ensures better performance in asymmetric networks.

6. *Robustness.* The only special status routers of QoSMIC are the Managers. If they fail, the recovery can be fast and with minimal data loss. Having multiple managers further enhances the robustness.

7. *Adaptivity During Execution Time.* In the MBone it is rather difficult to foresee the behavior of the application and the users. Our protocol can adapt to changing network conditions and requirements during the life of the group. This flexibility appears in three different aspects. First, we can trade-off control overhead with the number of multiple paths. Second, the routing scheme can vary from a Greedy scheme to a core-based Naive scheme. Finally, we can locate and relocate Managers to adapt to the evolution of the distribution tree.

8. *Loop Freedom.* We proved that QoSMIC creates loop free trees independently of the underlying unicast protocol.

The analysis in this chapter and our simulations in section 6.4 support the above claims.

Chapter 6

Simulation Results

In this chapter, we simulate the performance of the algorithms we have discussed in this document. First, we compare the performance of Naive and Greedy in various scenarios. Second, we examine the heterogeneous algorithms in a variety of scenarios. Third, we compare the QoSMIC protocol we propose against previous solutions. Our chosen test scenarios are based on the Internet environment. Given our work on Internet protocols, we decided to derive our scenarios from the Internet environment. We use an actual Internet topology, and, in parallel, we experimented with a family of graphs that current research in the networking community suggests as models of real topologies.

We present three series of simulations which correspond to three parts of our work (see table 6.1). The simulations for the homogeneous problem are presented in section 6.2, and we call them series I. The simulations for the heterogeneous problem, are presented in section 6.3, and we call them series II. The simulations for the Internet protocols are shown in section 6.4 , and we call them series III. In each section, we present the observations from the simulations in the beginning, so that the reader can observe better the interesting aspects of each plot. In section 6.1, we present an overview of our simulations and discuss previous experimental work. In section 6.5, we present an overview of the conclusions from all our series of simulations.

6.1 Overview of our Simulations

In this section, we present our simulation model and justify our choices. In doing so, we refer to previous experimental work.

6.1.1 Graph Models and Scenarios

In selecting graph models, we would prefer to have a number of real topologies. We were able to find one actual real graph of Internet multicast routers. For widening our range of topologies, we had to turn to artificial topologies.

As we already mentioned, the Internet is a hierarchical network. Different domains or levels may be administered by different managers and may employ different protocols. For example, our protocol may be employed at the interdomain level, while different protocols may be running in each domain. At the interdomain level, each domain appears as one entity, and the interdomain protocols do not control the multicast trees within the domains. The task for an interdomain protocol is to create multicast trees between the Border routers of domains with active members (see section 2.3). On the other hand, it may be possible to employ in a domain the same protocol that runs

Series	Problem
I	Homogeneous
II	Heterogeneous
III	Internet Protocols

Table 6.1: The different series of simulations of our work.

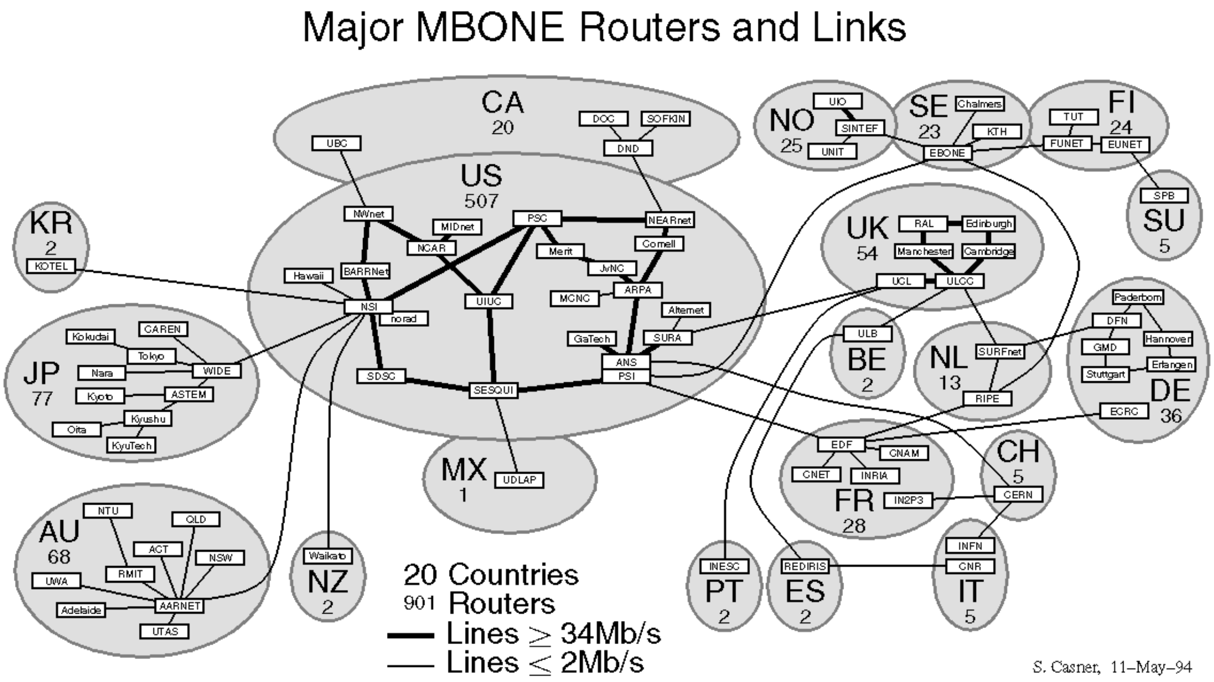


Figure 6.1: The 84-MBone: The map of the major MBone routers in 1994 by S. Casner.

in the interdomain level. For this reason, we want to examine the performance of protocols and algorithms in: a) intra-domain topologies, b) inter-domain topologies, and c) the whole network.

The most recent and comprehensive study of creating graphs that model Internet topologies is conducted by Zegura et al. [ZCD97]. Their work compares several ways of creating realistic models for both domains and the network as a whole.

Inter-domain topologies. Zegura et al. suggest the **Random Euclidean** family of graphs, which was originally introduced by Waxman [Wax88]. These graphs have been widely used in studies in the communications networks community [DL93] [SRV97] [BV95] [BV97] [KPP93] [FK97]. We can describe these graphs by explaining the way we create them. We distribute nodes on a Euclidean plane uniformly, and add edges between nodes on a probabilistic basis that favors pairs of nodes of small Euclidean distance from each other. For nodes $v, w \in V$, the probability of being connected with an edge is given below:

$$Prob\_Connect(v,w) = b \cdot e^{\left(\frac{-Dist(v,w)}{a \cdot D_{max}}\right)}, \quad a, b \in (0,1]$$

In the above formula, $Dist()$ is the Euclidean distance, and D_{max} is the maximum possible distance between nodes. Large values of parameter a increase the number of long edges (in a

Euclidean distance sense). Large values of parameter b increase the total number of edges in the graph.

The whole network. Zegura et al. create hierarchical graphs whose higher level (interdomain) is a Random Euclidean graph, and their lower level domains (intradomain) are Random Euclidean graphs or trees. This work introduces the term **transit-stub** graphs to describe hierarchical graphs that have a Random Euclidean graph at the higher level (**transit graph**) and tree structures for the lower level (**stub graphs**).

Casner provides an actual topology of the multicast-capable Internet routers in 1994 [Cas94]. This graph represents an actual instance of the multicast backbone (MBone) network. In other words, the employed multicast Internet protocols were running on this graph in 1994. The graph has 84 nodes (or routers) and 92 undirected edges, which leads to an average degree of 2.2 (see fig. 6.1). We create a directed version of the graph by replacing every unidirectional edge with a pair of opposite directed edges. We call this graph the **84-MBone graph**. The graph resembles a transit-stub graph: there is a core component of nodes and a large number of stubs (trees). We can eliminate the stubs by removing nodes with only one incoming directed edge. If we repeat this recursively until we can not find any such nodes, we end up with a graph with 32 nodes, 40 pairs of opposite edges, and an average degree of 2.5. We call this graph the **32-MBone graph**, and we consider it a real model of an interdomain graph.

Simulations on multicast protocols by Keshav and Paul [KP98] and Zegura et al. [ZCD97] lead to the following two observations:

1. *The performance of algorithms in hierarchical graphs is determined by the performance of the algorithms in each domain.* Intuitively, we can see that in a hierarchy, the points that connect the levels act as bottleneck points that all algorithms have to use. Therefore, the differences of the algorithms appear only within levels: in each domain, and at the interdomain level [KP98] [ZCD97].

2. *The size of the graph has little effect on the performance of algorithms in hierarchical graphs.* An intuitive explanation relates this phenomenon to the fact that the diameter of a hierarchical graph increases only logarithmically with its size [KP98].

In our work, we examine the performance of our algorithms on topologies that attempt to model the topologies of intradomains or the whole network. In both cases, we experiment with real graph instances, and various Random Euclidean graphs. In more detail, we experiment with the following topologies.

- *Intradomain and Interdomain separately.* In the case of domains with tree structure, all intradomain routing decisions are uniquely defined, so there is no point in studying it further. The other suggested choice for domains is Random Euclidean graphs [ZCD97]. Similarly, for interdomain topologies, most of the existing and most recent research suggests the Random Euclidean graphs, as we already mentioned.
  - We use the 32-MBone graph, which we described above.
  - We experiment with a large number of 32-node Random Euclidean graphs by varying the a and b parameters. For a wide range of such graphs, the relative performance of the Greedy and the Naive is comparable to that of the 32-MBone graph.
- *The whole network.*

– We use the 84-MBone graph, which is based on the map by Casner [Cas94].
– We use a range of 84-node Random Euclidean graphs. First, we vary the graph parameters, and we create Random Euclidean graphs for which the relative performance of the algorithms is as similar to that of the 84-MBone. We also vary parameters a and b and experiment with a range of Random Euclidean graphs with 84 nodes.

Our simulations provide an indication that what we observe on the 32-MBone graph does not seem to be the consequence of an idiosyncrasy of the 32-MBone graph: there are ranges of Random Euclidean graphs that support these observations. In this chapter, we present the experiments on the 32-MBone graph, in order to focus on the comparison of the performance of the algorithms. We also show some of the experiments on Random Euclidean graphs, and the effect of the graph parameters on the relative performance. In appendix A, we present the rest of our experiments.

Weight Assignment. Our graph model is a weighted directed graph, and for this, we need a way to assign weights on the edges of our graphs. Recall that the weights represent "how much we want to avoid using an edge" and it is a functions of many parameters such as capacity, available bandwidth, and monetary cost, to name a few. For most of the simulations, we initialize the weights of each pair of opposite directed edges to a constant, w_o. We also examine the effect of the weight assignment: we vary the range in which the weights are uniformly distributed in a number of simulations.

After assigning the initial weight to each pair of opposites edges, we introduce asymmetry. We select the asymmetry of the graph, A, and then we select the asymmetry for each pair of opposites edges. We experiment with two different ways of choosing the asymmetry of a pair. First, the asymmetry of each pair asy is uniformly distributed between one and the maximum asymmetry of the graph unless otherwise stated. We set the weight of one of the edges equal to the initial weight w, and we set the other to $asy{\cdot}w$. Second, we set the asymmetry of each pair equal to the average asymmetry ($A/2$). The asymmetry distribution did not have a great impact in the relative performance of the algorithms, and the simulations we show here are with the uniform distribution of the asymmetry.

Membership Scenarios. In creating a session, we control the group size, which is defined to be the number of active destinations plus the source.

- In the *static problem*, we select a source and the desired number of destinations pseudo-randomly: every node is considered with equal probability.
- In the *join-only problem*, we select destinations pseudo-randomly and join them one by one until we reach the group size we want.
- In the *join-leave problem*, we have three phases: a) build-up, b) fluctuations, and c) tear-down of the session. This behavior was observed in real multicasts over the Internet by Almeroth and Ammar [AA97]. The build-up is like a join-only session. Once the group reaches the specified size, we add and remove nodes pseudo-randomly: first, we select pseudo-randomly with equal probability between a join and a leave, and then we select an eligible node to associate with the request. The number of fluctuations is set approximately three times the size of the graph, i.e., every node in the graph joins and leaves at least twice on the average. As we will see, the number of fluctuations does not have a significant impact on the results. In the tear-down of the session, we remove pseudo-randomly the active destinations one by one[1].

[1]Note that the number of requests of the tear-down phase varies among sessions, because of the way we do the fluctuations.

Note that in the simulations where we study the effect of the number of destinations, we make the fluctuations strictly alternate between leaves and joins. This way, we can isolate better the effect of having a specific group-size.

Performance Metrics. In our simulations we keep track of the following metrics.

- The *tree cost* is the sum of the routing costs of all its edges, which quantifies the routing efficiency as we have defined in chapter 2.
- The *end-to-end average path length* is the number of edges of the path along the tree from the source to a destination. We take the average end-to-end length over all destinations that participate in a session. This average length is the average depth of the tree, which is important to monitor, since it describes the tree and thus the effect of the algorithms.
- The *Gain* is the value of the objective function, $Gain()$, in the heterogeneous problem as we defined in section 2.2.

We use the term "relative performance" to denote how the performances of Naive and Greedy compare. For the tree cost, we consider the ratio of the cost of the tree of Naive over that of the Greedy: T_{Naive}/T_{Greedy}. For the average path length, we refer to the absolute difference in edges: $P_{Greedy}-P_{Naive}$, where P_x is the average path length of the tree of algorithm x. In the heterogeneous problem, we consider the ratio of the Gain of an algorithm over the Gain of Naive-high [Sha92].

Varying Parameters. We study the effect of several parameters on the performance of the algorithms.

- The group size (Grp) is the number of destinations in a session. For the static and join-only problems, we define it to be equal to the number of join requests. For the join-leave problem, we define it to be the average group-size during the phase of fluctuations.
- The *asymmetry* (A) is defined as the maximum ratio of the weights of opposite edges between a pair of nodes, for all such pairs.
- *Number of requests - Time.* For the join-only and join-leave problems, we examine the evolution of a session as requests arrive. This gives us the evolution of a tree over time assuming that the inter-arrival time between requests is constant.
- The *average node degree of a graph* is related to the number of different paths that exist between a pair of nodes. For example, if edges appear in pairs of opposite edges, average node degree of approximately two means that the graph is a tree, and there is only one path between each pair of nodes (assuming a connected and large graph).
- The *graph size* is the number of nodes in a graph.
- The *number of streams* is the number of distinct streams that destinations request in a fixed heterogeneous problem, and the maximum number of distinct preferred streams in the general problem.
- The *bandwidth ratio of streams (θ)* is defined as the ratio of the bandwidth requirements of consecutive streams as we defined in 2.2.

Problem	Graph	Measure	Vary	Versus
Static	32-MBone	Tree Cost	Initial Weight Assignment	Asymmetry
Join-Only	32-Euclidean	Path Length	Asymmetry Distribution	Group-Size
Join-Leave	84-MBone			Avg. Node Degree
	84-Euclidean			Requests - Time
				Graph-Size
				a, b of Random Euclidean graphs

Table 6.2: Series I: An overview of our simulations for the homogeneous problem.

Problem	Graph	Measure	Vary	Versus
Fixed	32-MBone	Gain	$Gain()$ Function	Asymmetry
General		Tree Cost		Group-Size
		Happiness		Avg. Node Degree
				No. of Streams
				Bandwidth Increase of streams

Table 6.3: Series II: An overview of our simulations for the heterogeneous problem.

Problem	Graph	Measure	Versus
Join-only	32-MBone	Tree Cost	Asymmetry
		Added-branch length distribution	Group-Size
		Control overhead	Avg. Node Degree

Table 6.4: Series III: An overview of our simulations for the Internet protocols.

In tables 6.2, 6.3, and 6.4 we present an overview of the experiments of each series of simulations.

Accuracy of results. In our results, each reported value of tree cost is the average value of a number of different multicast instances, so that the width of the 95% confidence interval of every average value we plot is less than 5% of its value. For each execution, we use one or more graphs and we average over a sufficiently large and fixed number of sessions. At the end, we verify that the experiment achieves the confidence interval above.

Sensitivity of our results. An interesting question is how sensitive our results of our simulations are to small changes of the environment and the scenario. For this, we conducted a number of experiments in series I, some of which we already mentioned (see appendix A).

- We varied the range of the initial weight assignment. We also varied the value of the constants of the initial weight assignment.
- We have also done simulations where we plot the average over multiple graphs and multiple sessions and the results were consistent with the other experiments (see fig. 6.8, 6.9, 6.10, and 6.11).
- We varied the Random Euclidean parameters, a, and b.
- We varied the average degree of the nodes by varying both parameters a and b (see figs. A.27 A.28).

Regarding the sensitivity, a general observation from all these experiments was that small changes had only a small effect on the outcome in almost all cases. A second observation is that in *almost* all variations, algorithms did not change the relative order of performance: the tree cost of Greedy was no larger than the tree cost of Naive, and was typically smaller by $10 - 20\%$.

6.2 The Homogeneous Problem (Series I)

In this section, we study the homogeneous problem in the static (section 6.2.2), the join-only (section 6.2.3), and join-leave problems (section 6.2.4). In section 6.2.1, we summarize our observations from series I. For every simulation, we present a plot of the tree cost and the average end-to-end path length. In each plot, we vary a parameter. We have the following default values; asymmetry is 10, and group size is 8.

6.2.1 Observations for Series I

The following observations are of interest in series I simulations. In this section, by the term performance, we mean the tree cost, which is to be minimized as the objective in our problems.

1. Greedy performs better than Naive in almost all cases. In some cases, they have similar performance, but often Greedy has smaller tree cost by 10% to 30% (see figures 6.6, 6.2, 6.16, and 6.16). In our experiments with the a and b parameters of the Random Euclidean graphs, the relative performance of the algorithms was rather insensitive to small changes of the parameters. Parameter b did not seem to affect the relative performance, while a increased slightly the difference between the algorithms.
2. Greedy trades off end-to-end path length with tree cost. Greedy has longer end-to-end paths, but has smaller total cost.

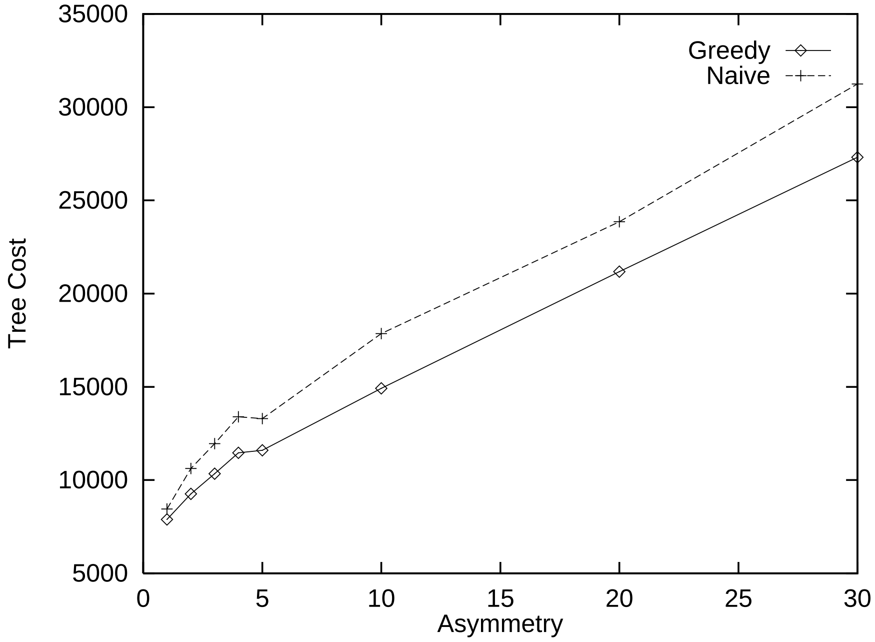


Figure 6.2: Tree Cost versus Asymmetry in the 32-MBone map for the static problem with group-size 8.

3. The performance of Greedy does not seem to degrade even with high membership activity in the join-leave problem. This suggests that the worst case scenarios presented in chapter 3 do not appear in our simulations. Note that Naive is insensitive to the membership behavior by the nature of its approach.

4. Greedy performs better than Naive for all the asymmetry values (see figure 6.2). At first, as asymmetry increases from one, the relative difference increases. We can explain this, if we think that as asymmetry increases the weights of the edges vary. Greedy can take advantage of these variations, since it considers more possible paths at each join compared to Naive. This way, Greedy is more likely to find low cost edges. However, as the asymmetry increases further, many edges become "prohibitively" large and the options for both algorithms are reduced. Also, the overall cost increases, which makes the percentage of the difference look smaller.

5. For larger group sizes, the relative performance of Greedy over Naive increases. This observation holds for group sizes smaller than half the size of the graph. We can understand this intuitively, if we think that the larger a group is, the more choices we have for a new join, and this way, Greedy is able to use edges of low cost. On the other hand, as group sizes increase further (greater or equal to half of the nodes of the graph), the trees increase in size and cost and this reduces the relative difference. It seems that, for larger trees, both algorithms end up choosing the same edges, since the graphs have low average degrees.

6.2.2 The Static Problem

Asymmetry. We plot the tree cost and end-to-end path length versus the asymmetry of the graph in figures 6.2, 6.3 respectively. We observe that Naive performs worse than Greedy for the

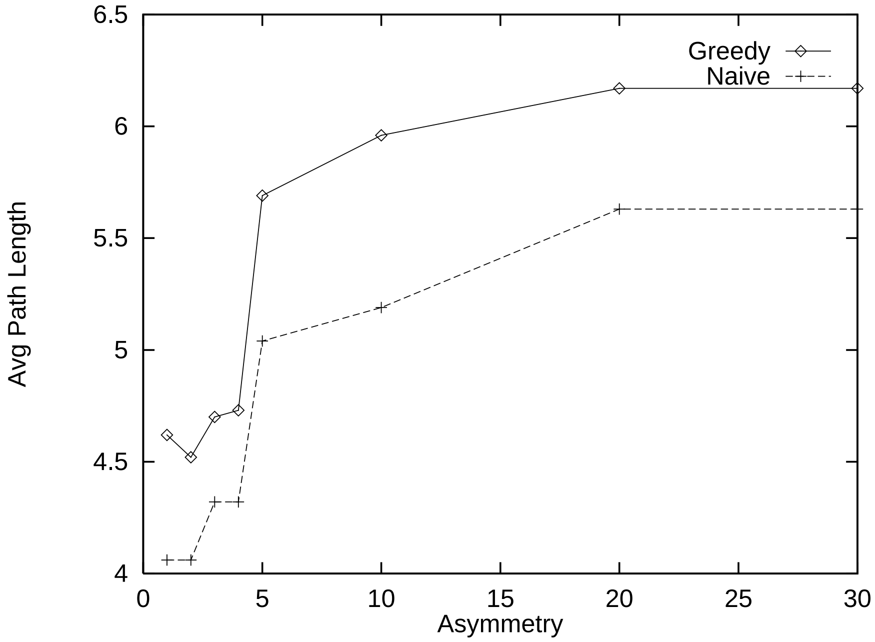


Figure 6.3: Path Length versus Asymmetry on the 32-MBone map for the static problem with group-size 8.

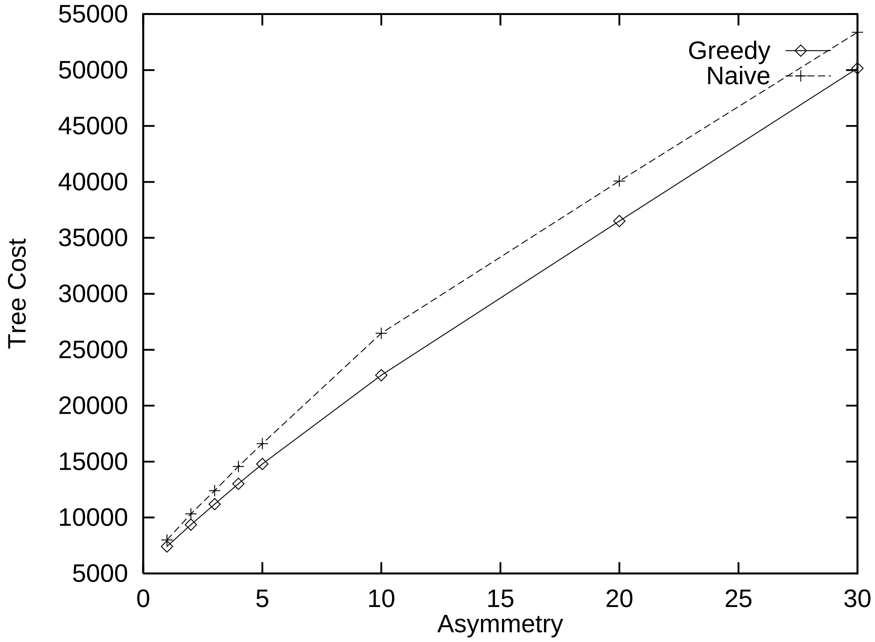


Figure 6.4: Tree Cost versus Asymmetry on the 32-Euclidean map for the static problem with group-size 8.

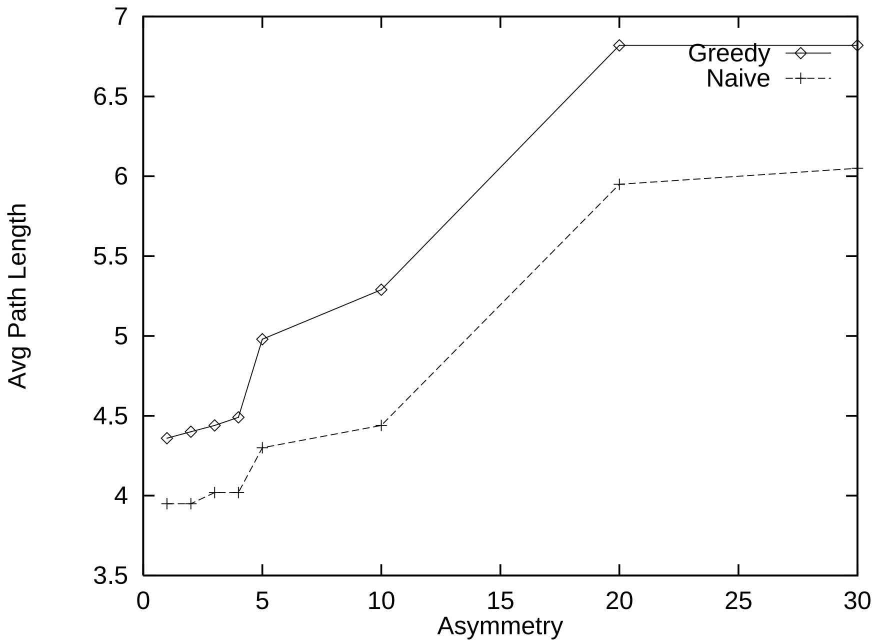


Figure 6.5: Path Length versus Asymmetry on the 32-Euclidean map for the static problem with group-size 8.

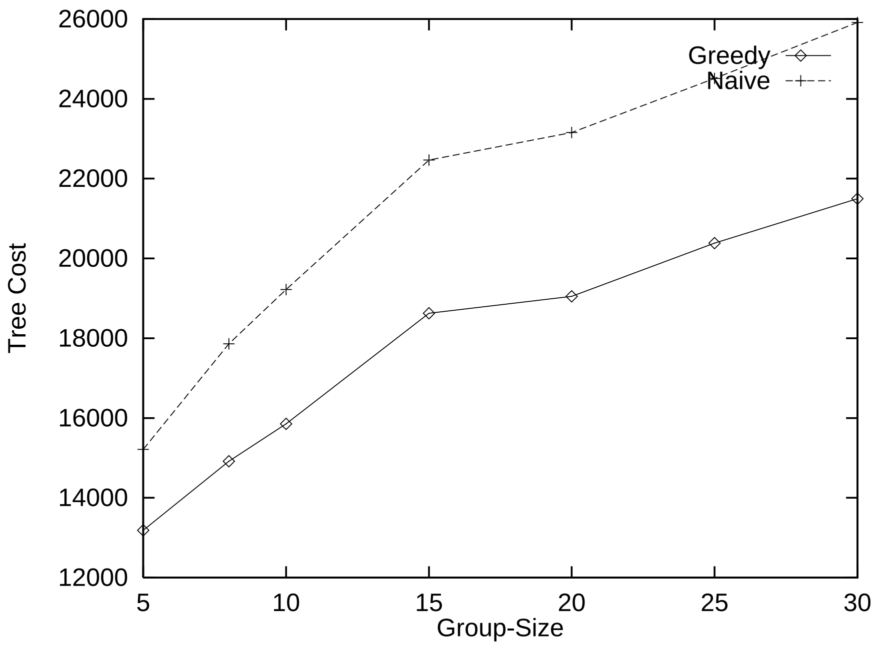


Figure 6.6: Tree Cost versus Group Size in the 32-MBone map for the static problem with asymmetry 10.

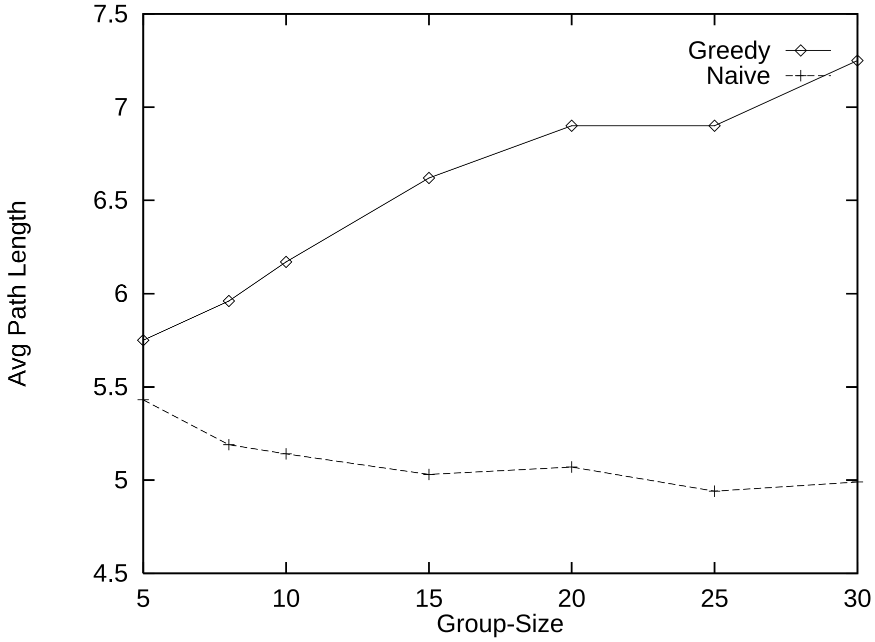


Figure 6.7: Path Length versus Group-Size on the 32-MBone map for the static problem with asymmetry 10.

whole range of asymmetries. We also note that as asymmetry increases from one, at first the difference increases up to 22% for asymmetry 10. After that point, the difference decreases to 16% for asymmetry 30. We already presented an intuitive explanation of this phenomenon in section 6.2.1. Note the sudden increase in the average path length when asymmetry increases from 4 to 5. This phenomenon appeared rather consistently in a number of simulations (for example see 6.5, and A.18). This suggests that this may not be an artifact, but it may relate to the behavior of the algorithms in conjunction with the topology, though we were not able to identify a precise explanation.

We compare the relative performance for our graphs of size 32: the 32-MBone in figures 6.2, 6.3, and a 32-Euclidean with $a = 0.25$, and $b = 0.3$ in figures 6.4, 6.5. We observe that the relative difference appears slightly smaller in the Random Euclidean graph. The effect of the asymmetry on the tree cost and average path length shows similar general trends. In appendix A, we present more plots of the two topologies for both the static and the join-leave problem.

Group Size. We plot the tree cost and end-to-end path length versus the group size in figures 6.6, 6.7 respectively. Initially, the increase in the size increases the performance difference of Greedy over Naive (26% relative tree cost for group size 10). Then, as the size increases further the relative tree cost difference drops (18% relative tree cost for group size 30). Note that the average path length increases from a difference of half an edge to two edges: the average path length of Greedy increases significantly, while the average path length of Naive reduces sightly.

6.2.3 The Join-Only Problem

The performance of the algorithms on the join-only problem shows similar trends to that of the join-leave problem, as we can see in appendix A in figures A.7, and A.8). In the join-only problem, Greedy outperforms Naive by a percentage that varies between 2-22%. Note also that, in figures

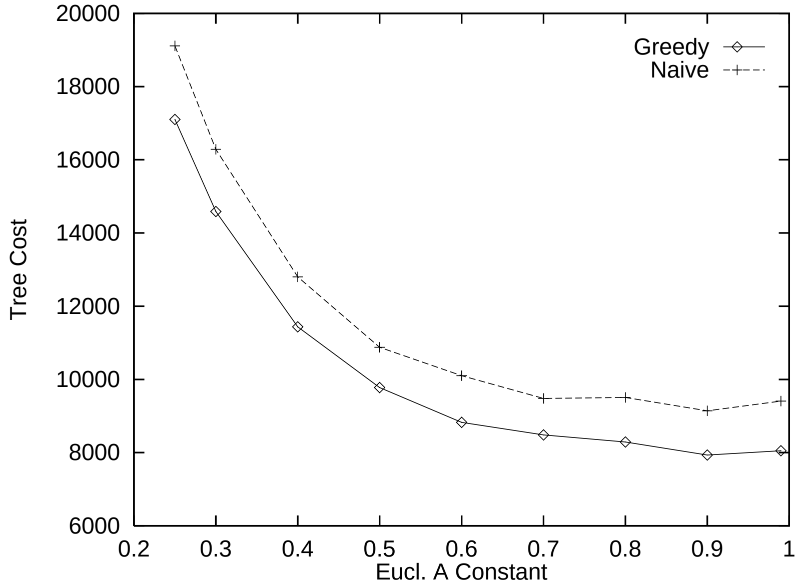


Figure 6.8: Tree Cost versus Parameter a in the 32-Euclidean map for the join-only problem with asymmetry 10.

6.16, 6.17, the build-up of the session corresponds to a join-only problem, and at the end of this phase, the relative difference reaches its highest difference: 33% for 8 requests, while the next largest difference is 30% for 106 requests, and on average the difference is 23%. This is something that agrees with the intuition from theoretical worst case analysis (see section 3.5).

Random Euclidean Parameters.

We examine the effect of a and b parameters on the Random Euclidean graphs. We create 30 graphs for each pair of values of the parameters and run simulations of multiple sessions on each graph.

Parameter a. Recall that as parameter a increases, we have more longer edges (in the Euclidean distance sense). We set the value of $b = 0.3$ to isolate the effect of a. We plot the tree cost and end-to-end path length versus the asymmetry of the graph in figures 6.8, 6.9 respectively. We observe that the relative performance increases with a: from 13% for $a = 0.25$ to 18% for $a = 0.99$. An explanation of this is that by having more edges Greedy has more routing options that it can use for reducing the tree cost.

Parameter b. Recall that as parameter b increases the total number of edges increases. We keep value of $a = 0.25$, and study the effect of parameter b. We plot the tree cost and end-to-end path length versus the asymmetry of the graph in figures 6.10, 6.11 respectively. We see that both costs reduced as more edges are added, but the relative difference of the tree cost stays between 9% and 12%.

6.2.4 The Join-Leave Problem

In this section we study the performance of our algorithms in the join-leave problem.

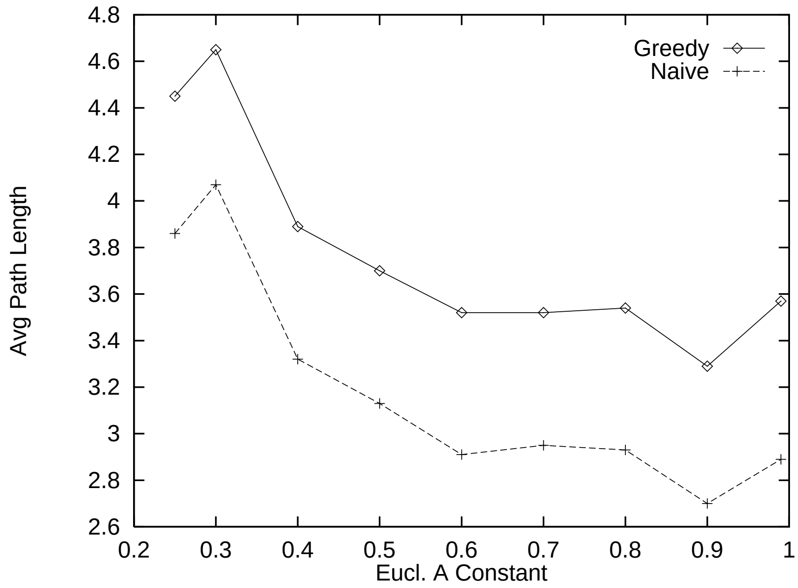


Figure 6.9: Path length versus Parameter a in the 32-Euclidean map for the join-only problem with asymmetry 10.

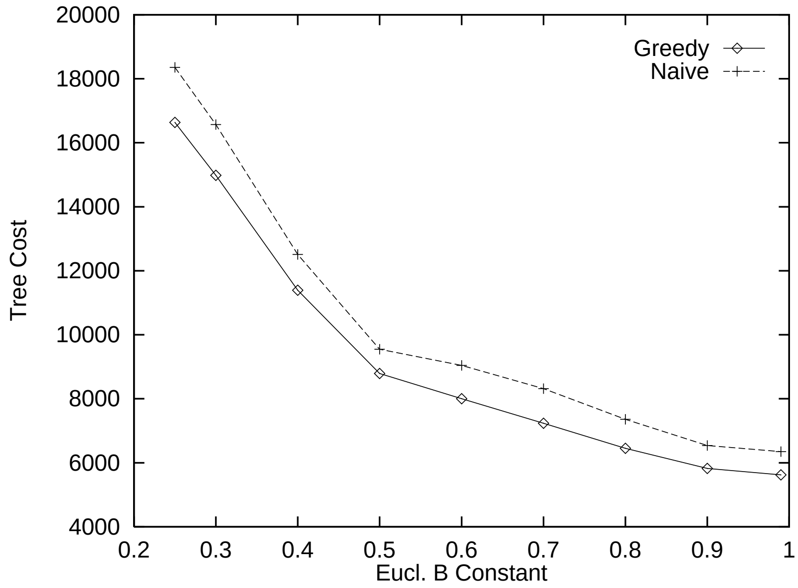


Figure 6.10: Tree Cost versus Parameter b in the 32-Euclidean map for the join-leave problem with asymmetry 10.

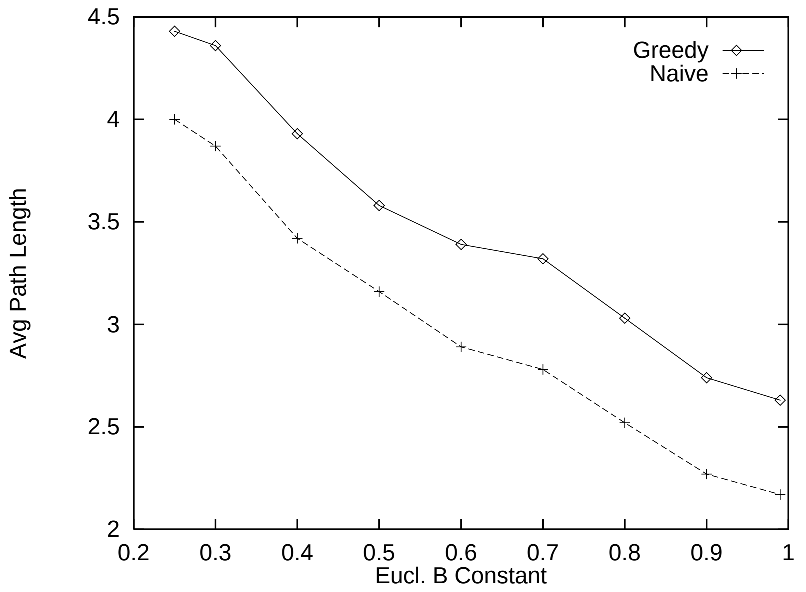


Figure 6.11: Path length versus Parameter b in the 32-Euclidean map for the join-leave problem with asymmetry 10 and group size 8.

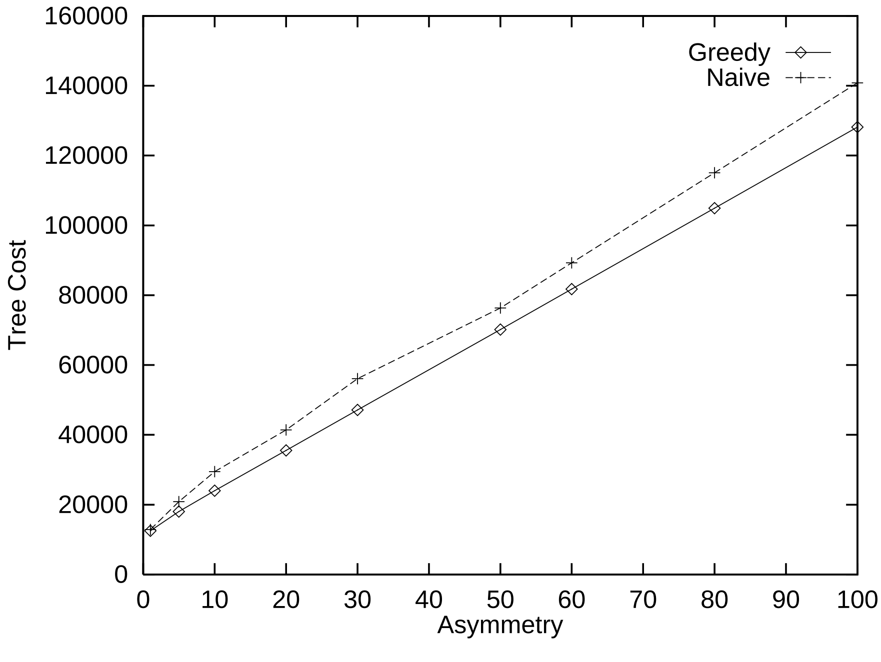


Figure 6.12: Tree Cost versus Asymmetry in the 32-MBone map for the join-leave problem with group-size 8.

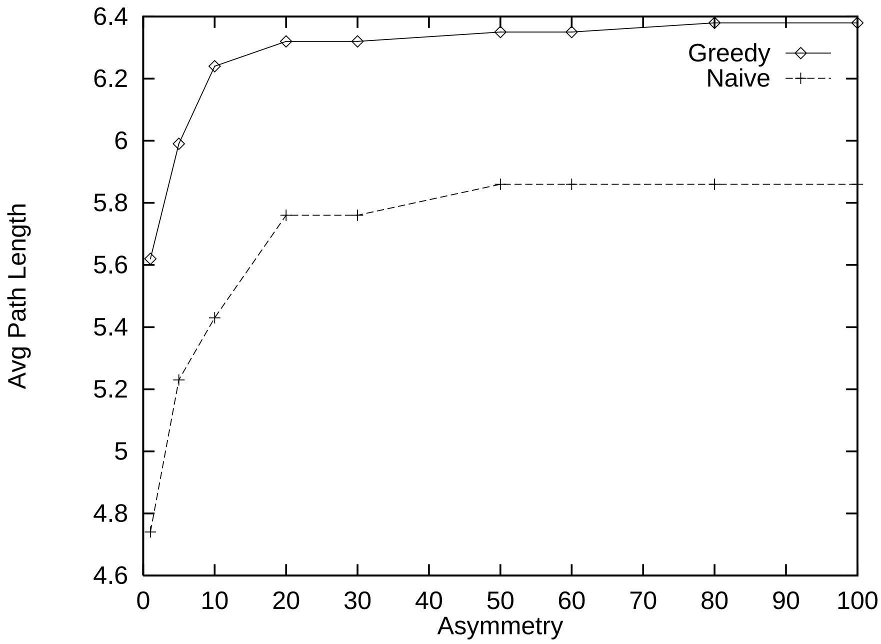


Figure 6.13: Path length versus Asymmetry on the 32-MBone map for the join-leave problem with group-size 8.

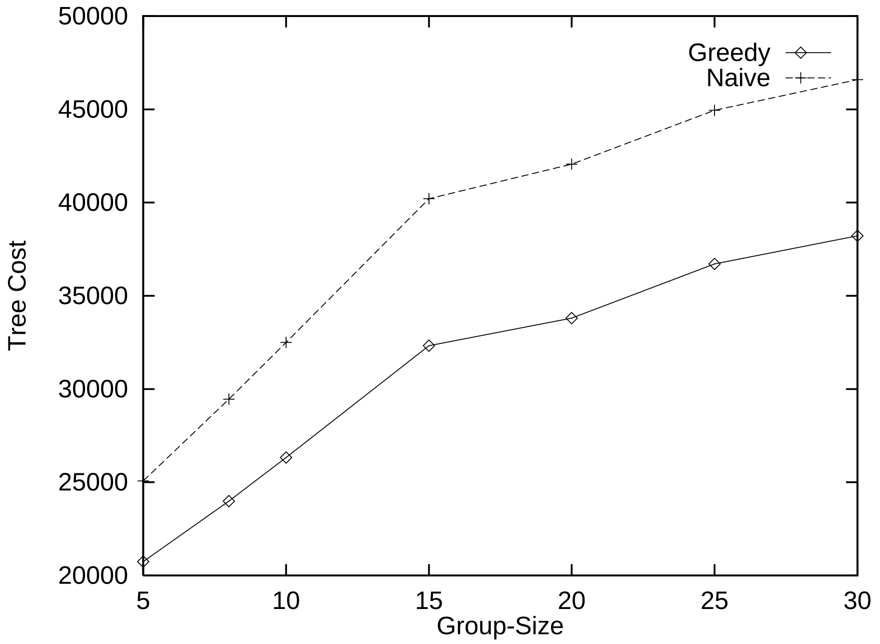


Figure 6.14: Tree Cost versus Group-Size in the 32-MBone map for the join-leave problem with asymmetry 10.

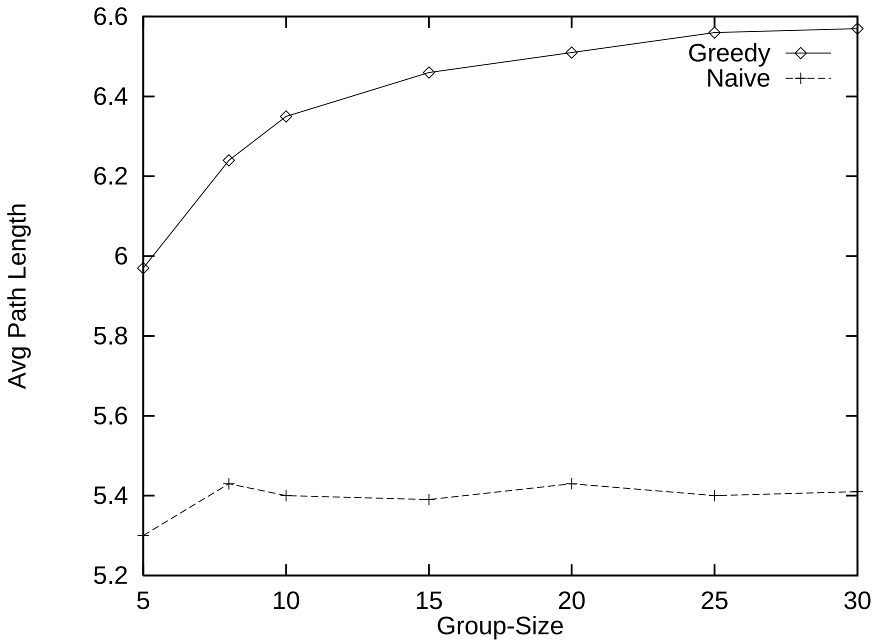


Figure 6.15: Path length versus Group-Size on the 32-MBone map for the join-leave problem with asymmetry 10.

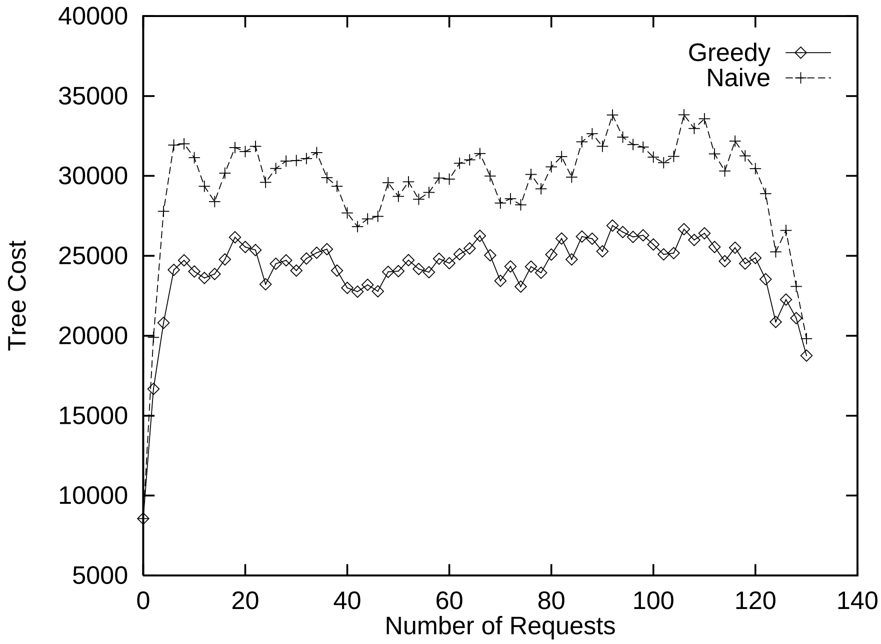


Figure 6.16: Tree Cost versus number of requests in the 32-MBone map for the join-leave problem with group-size 8 and asymmetry 10.

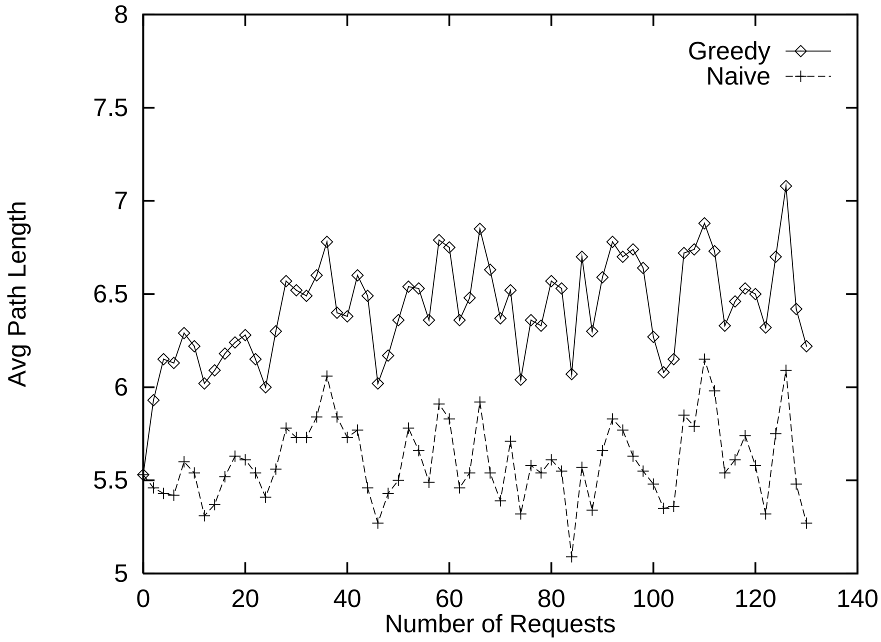


Figure 6.17: Path length versus number of requests on the 32-MBone map for the join-leave problem with group-size 8 and with asymmetry 10.

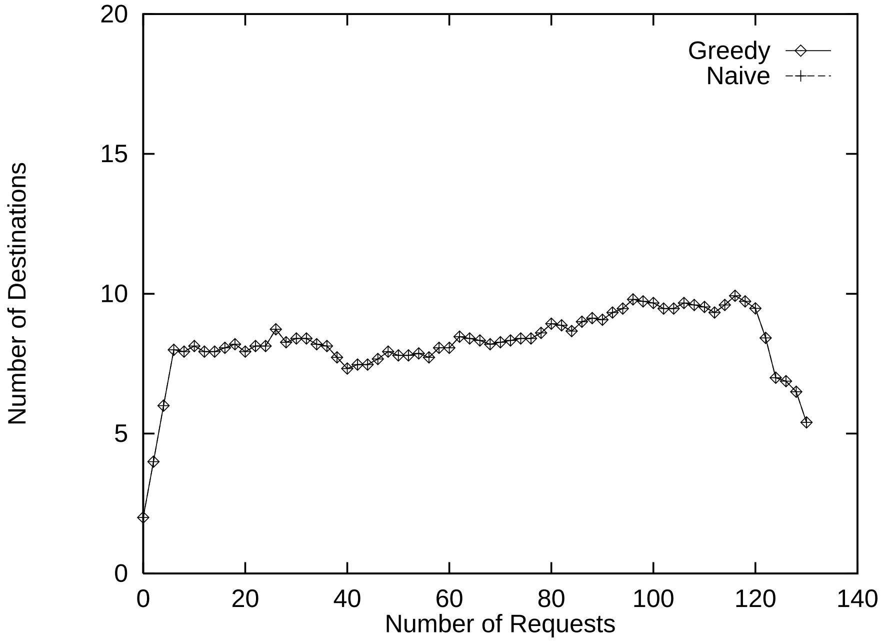


Figure 6.18: Actual Group-Size versus number of requests on the 32-MBone map for the join-leave problem with group-size 8 and with asymmetry 10.

Asymmetry. We plot the tree cost and end-to-end path length versus the asymmetry of the graph in figures 6.12, 6.13 respectively. The asymmetry has a similar effect in the join-leave problem as in the static problem. For asymmetry one we have 2% difference for asymmetry 10 we have 23% difference and for asymmetry 30 the difference drops to 19%. The difference in the average path length starts at 0.8 of an edge and decreasese to 0.6 gradually.

Group Size. We plot the tree cost and end-to-end path length versus the group size in figures 6.14, 6.15 respectively. The relative tree cost is 18% for group size 10, and the drops to 14% for group size 30. If we recall the corresponding relative tree cost values of the static problem (26% for group size 10 and 18% for group size 30 in fig. 6.2), the relative difference seems to reduce in the join-leave problem.

Number of Requests. We plot the tree cost and end-to-end path length versus the number of requests or time in figures 6.16, 6.17 respectively. In figure 6.18, we plot the number of destinations. We plot up to the average number of requests which in these graphs are approximately 130. Recall that the number of requests untill the end of the session vary, given the probabilistic way we do the fluctuations.

The main observation here is that the performance of Greedy does not degrade as the number of requests increases. If we recall the worst case analysis in chapter 3: Greedy has a very large lower bound in the join-leave case, making it conceivable that the performance of the Greedy would degrade as the number of requests increases. In addition, the difference in the average path length of the two algorithms does not seem to increase over time either, which suggests that the tree is in a stable state (on average).

In all our simulations, we never observed a case where the performance of Greedy degrades with time (see A.23, A.24).

6.3 The Heterogeneous Problem

In this section, we examine the performance of algorithms for the heterogeneous problem. The environment for our simulations in these series is similar to that of the homogeneous problem (see section 6.1), but has also the following additional characteristics. At this point, we should stress again that our algorithms do not make any assumptions on the choice of functions that we discuss below. However, we need to make some choices for the sake of the simulations. Given that we are solving an optimization problem, our concern was to create a large solution space, which would demonstrate the differences in the behavior of the algorithms.

In the creation of sessions, we assume four distinct streams in a session, unless we state otherwise. We choose the preferred stream range as follows. In the fixed problem, the single preferred stream is uniformly distributed in the range of the lowest and maximum stream of the session, i.e., the stream index is in $[1, L]$, where L is the maximum stream index for the session. In the general problem, the preferred range of each destination is between the lowest stream of the session and the highest preferred stream is selected as in the fixed problem, i.e., the range is $[1, l]$, where $l \in [1, L]$. The happiness is a logarithmic function in the stream index s ($hap(s) = 10 \cdot \log(10 \cdot s + 1) + 10$) for all destinations, with the rationale that the user satisfaction for (or the price of) higher streams is expected to increase rapidly in the beginning, but only marginally for higher streams. We assume the ratio of consecutive streams θ to be constant for every pair of consecutive streams in a session. Unless otherwise stated, we set $\theta = 2$, because this value creates a large solution space for the optimization problem, as we will explain later. This way, the required bandwidth b of stream s is an exponential function of the stream index.: $b = \theta^s$. We choose one function for the routing cost for all edges. Among the possible choices (see section 2.2), we select the proportional function

[Max97], i.e., the cost of a directed edge e is proportional to its bandwidth f: $cost_e(f) = v{\cdot}f$, where v is the weight of the directed edge.

We use the following two functions that take as parameters the happiness Hap and cost $Cost$:

The relative profit or rate of return function:

$$Gain_1(Hap, Cost) = \frac{Hap}{Cost}$$

An intuitive justification is that, if we consider the network resources as our capital and the satisfaction as our revenue, then for every session we want to increase the "rate of return" on our capital. This makes more sense if we consider that we want to economize resources for future sessions. As we will see later, this function facilitates the comparison of algorithms, and it is the function we use, unless otherwise stated.

The absolute profit function:

$$Gain_2(Hap, Cost) = k{\cdot}Hap - Cost, k \in \mathcal{R}$$

The intuition behind this function is straight-forward: if happiness is our profit and the routing cost is our expense, the function wants to maximize our profit. Note that in this functions we want to have the happiness and the cost are expressed in compatible units: parameter k express the value of a happiness unit relative to a cost unit. For this reason, this function is sensitive to the choice of parameter k, which makes it more difficult to compare the Gain values of the algorithms (see figure 6.26).

The Simulated Algorithms

We examine the performance of the algorithms that we defined in chapter 4 and some homogeneous solutions as points of reference.

1. *Naive-hi:* We perform Naive and offer the maximum preferred stream to destinations. Recall that Naive-hi corresponds to the algorithm suggested by Shacham [Sha92].

2. *Naive-cho:* We perform Chosen Stream Naive , i.e, we perform Naive and offer to each destination the chosen stream from the Choose-Stream procedure.

3. *MinCost:* We perform MinCost as described in section 4.1 in the fixed heterogeneous problem: the algorithm involves Greedy routing, selecting first the minimum cost destination.

4. *MaxStream:* We perform MaxChoStr as described in section 4.2 in the fixed heterogeneous problem: the algorithm involves Greedy routing, selecting the destination with the maximum chosen streams.

5. *MaxGain:* We perform MaxGain as described in section 4.2 in the general heterogeneous problem: the algorithm involves Greedy routing, selecting the destination that adds maximum Gain to the solution.

6. *MaxChoStr:* We perform MaxChoStr as described in section 4.2 in the general heterogeneous problem: the algorithm involves Greedy routing, selecting the destination with the maximum chosen streams.

7. *Greedy-low:* We perform Greedy and offer the minimum stream of the session to all destinations. This algorithm addresses the homogeneous problem, and offers a conservative solution, but we use it as a point of reference.

8. *Greedy-hi:* We perform Greedy and, to every destination, we offer its maximum preferred stream. In the fixed problem, MinCost is identical to Greedy-hi.

At this point, we can repeat the default choices, which is four distinct streams, consecutive stream ratio 2, asymmetry 10, and group size 8, and the objective function $Gain_1(Hap, Cost) = \frac{Hap}{Cost}$. Now we can justify these choices: these values correspond to the case where the gain value of Greedy-low and Greedy-hi are close. In other words, this set-up is balanced in the sense that neither of the two extreme stream assignments has an advantage. This way, our heterogeneous solutions show their ability to make effective stream selections, as we will see.

Note that we use the 32-MBone graph in our examples. We have also conducted similar experiments on Random Euclidean graphs and the results were comparable, which was expected given our simulations of series I.

6.3.1 Observations for series II

Our simulations lead to the following observations. Note that in the general problem, our observations focus mainly on the stream-selecting algorithms.

1. Our Choose-Stream procedure strikes the balance in the trade-off between happiness and routing cost and adapts effectively as the relative cost of the streams increase (see 6.24, 6.26, 6.25, and 6.27). In addition, the combination of our stream-selection and Greedy routing can greatly improve the heterogeneous solution.

2. Greedy routing creates solutions that have higher gain compared to Naive. In the general problem, the difference is independent of the consecutive stream ratio (see fig. 6.24), while, in the fixed problem,the difference decreases with the consecutive stream ratio (see fig. 6.19).

3. Asymmetry reduces the relative difference of the algorithms in the heterogeneous problem (see fig. 6.21, and 6.28). This is a phenomenon that we have observed and discussed in series I (see section 6.2.1).

4. In the fixed problem, MaxStream performs better than MinCost for higher consecutive stream ratios, (greater than 1.6), while for smaller ratios MinCost seems to perform better (see fig. 6.21). Intuitively, we can understand this if we think that MaxStream gives priority to higher streams, and this is "justified" only when the higher streams are significantly more expensive to route. In other words, as the routing cost differences of the streams disappears, the problem resembles a homogeneous problem, and the MinCost approach prevails, since it behaves as a homogeneous Greedy, while MaxStream selects the destinations in the maximum stream order and corresponds to an execution of Random Participant First. It is interesting to note that Maxemchuck [Max97] observed this phenomenon and reported a threshold of 1.5 in the consecutive stream ratio.

6.3.2 Fixed Problem

In this section, we describe our simulations on the fixed problem. In figure 6.19, we plot the Gain versus the consecutive stream ratio. In figure 6.20, we plot the tree cost versus the consecutive stream ratio. Note that the cost is directly related to the Gain in the previous figure. Note the straightforward dependency of the two plots, which we expect since all fixed problem algorithms offer the same happiness. In addition, we consider this as a good indication of the Gain function

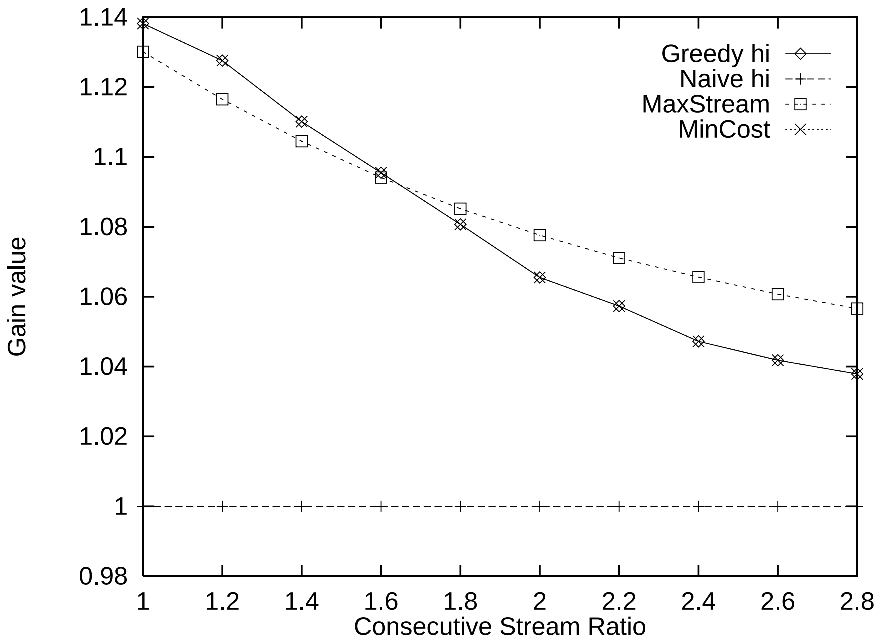


Figure 6.19: Fixed: Gain versus stream ratio in the 32-MBone graph with group-size 8.

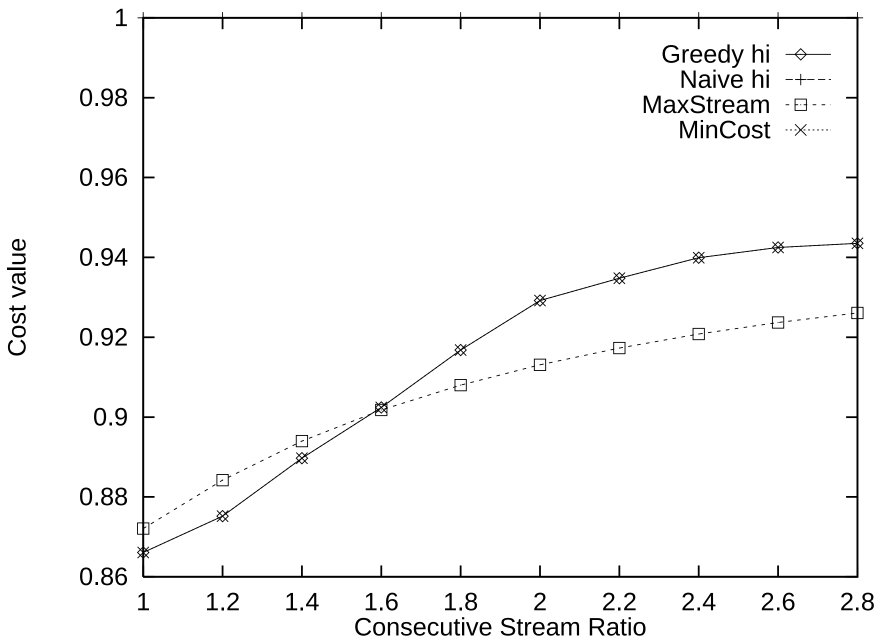


Figure 6.20: Fixed: Tree cost versus stream ratio in the 32-MBone graph with group-size 8.

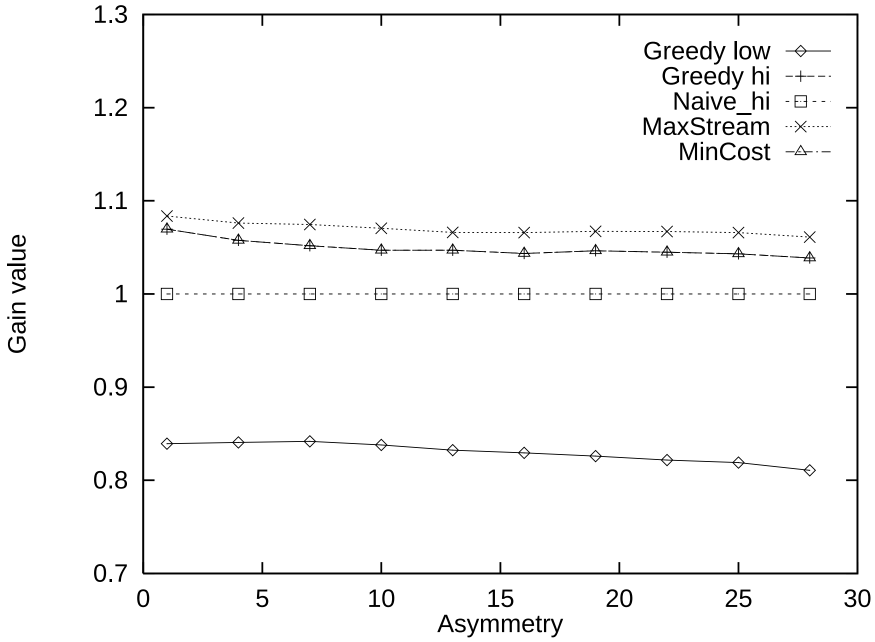


Figure 6.21: Fixed: Gain versus asymmetry in the 32-MBone graph with group-size 8.

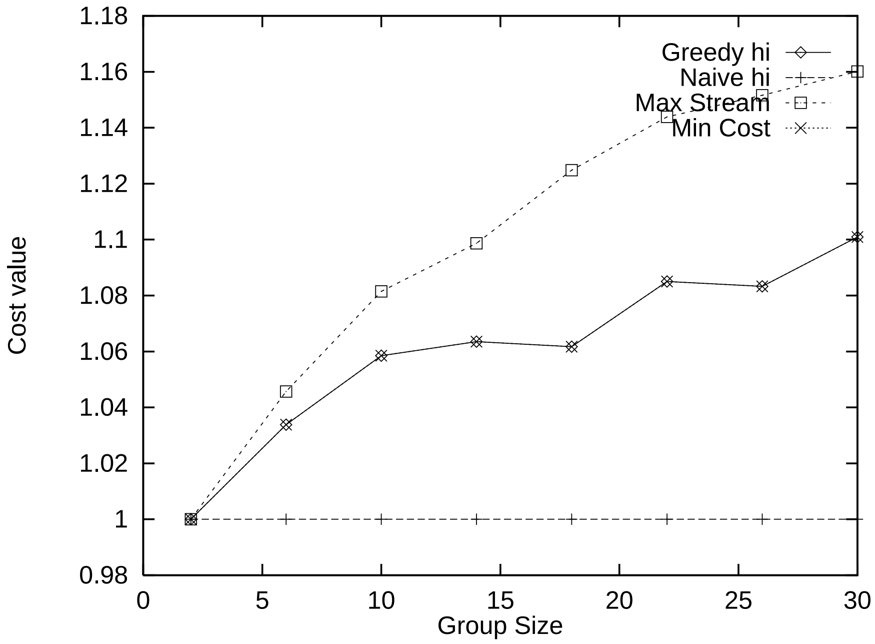


Figure 6.22: Fixed: Gain versus group size in the 32-MBone graph with group-size 8.

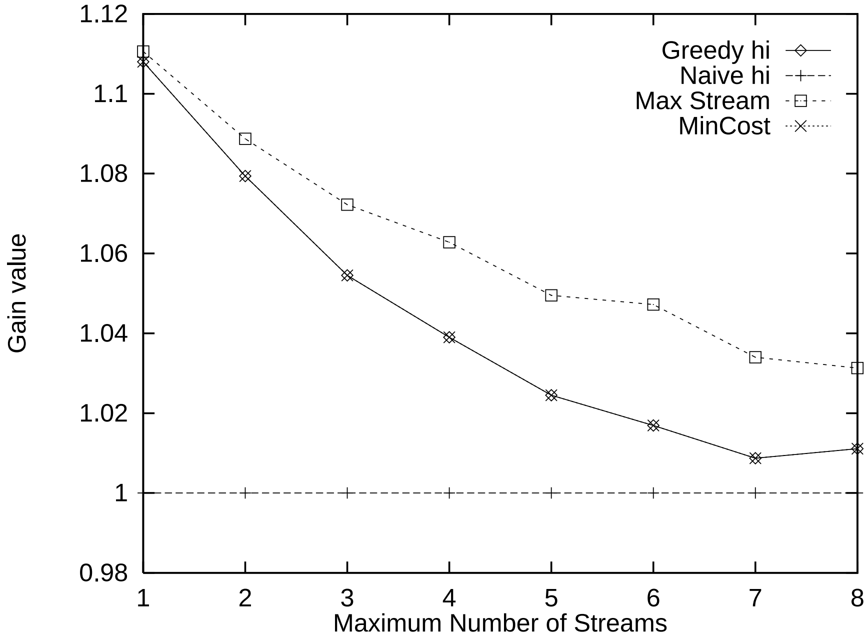


Figure 6.23: Fixed: Gain versus maximum number streams in the 32-MBone graph with group-size 8.

that we use $Gain_1(Hap, Cost) = \frac{Hap}{Cost}$ and the way we compare the performances: comparing the rations of the Gain values does not distort the the differences of the algorithms.

In figure 6.21, we plot the Gain versus the asymmetry of the graph. The relative performance decreases slightly with the asymmetry, as we already discussed in section 6.3.1.

In figure 6.22, we plot the Gain versus the group size. We see that the Greedy-based algorithms perform better than the Naive-based ones for large group sizes, which we also observed and discussed in series I. We see that MaxStream benefits more than MinCost from the larger group sizes. This can be explained, by the fact that MaxStream routes the high stream nodes first and expects low stream nodes to use the high stream connections: having more destinations justifies this approach.

In figure 6.23, we plot the Gain versus the maximum number of streams. We see that by having more streams, we have more expensive streams to route, and so it is the few high streams nodes that "define" the cost of the tree. This corresponds to having a smaller effective group size, and smaller group sizes reduce the difference (see fig. 6.22).

6.3.3 General Problem

In this section, we describe our simulations for the general problem. We focus our comparison on the algorithms for the general heterogeneous problem, but we also plot the fixed or homogeneous solutions for reference. The corresponding fixed problem here delivers to each destination with preferred range $[1, l]$, the maximum stream of the range l. In figure 6.24, we plot the Gain versus the consecutive stream ratio. We can see that MaxChoStr and MaxGain outperform the other algorithms for all the range of stream ratios. Naive-cho performs worse than those two algorithms by 15%. Note that for a stream ratio of two, Greedy-hi and Greedy-low have comparable Gain values. These two algorithms represent the extreme homogeneous solutions, and this is a point of balance, which is the reason why we chose $\theta = 2$ as the default value. In figure 6.25, we show the

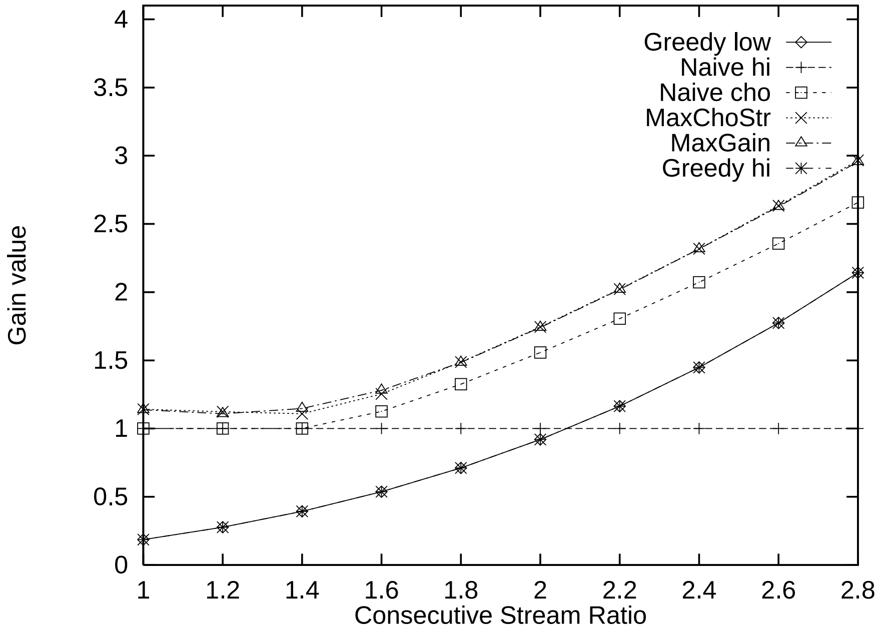


Figure 6.24: General: Gain versus stream ratio in the 32-MBone graph with group-size 8.

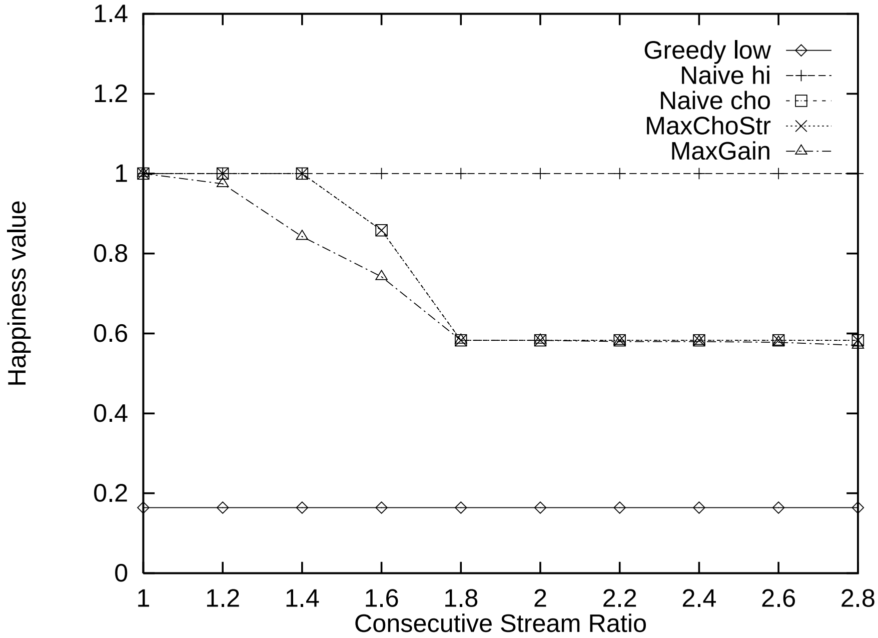


Figure 6.25: General: Total happiness versus stream ratio in the 32-MBone graph with group-size 8.

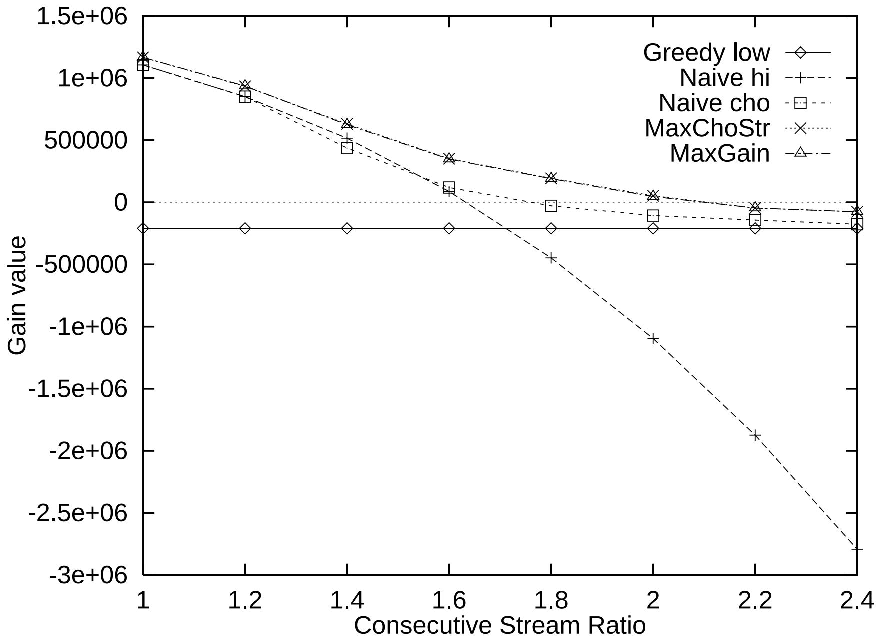


Figure 6.26: General: Gain versus stream ratio for the absolute profit Gain function ($Gain_2(H, C)$) in the 32-MBone graph with group-size 8.

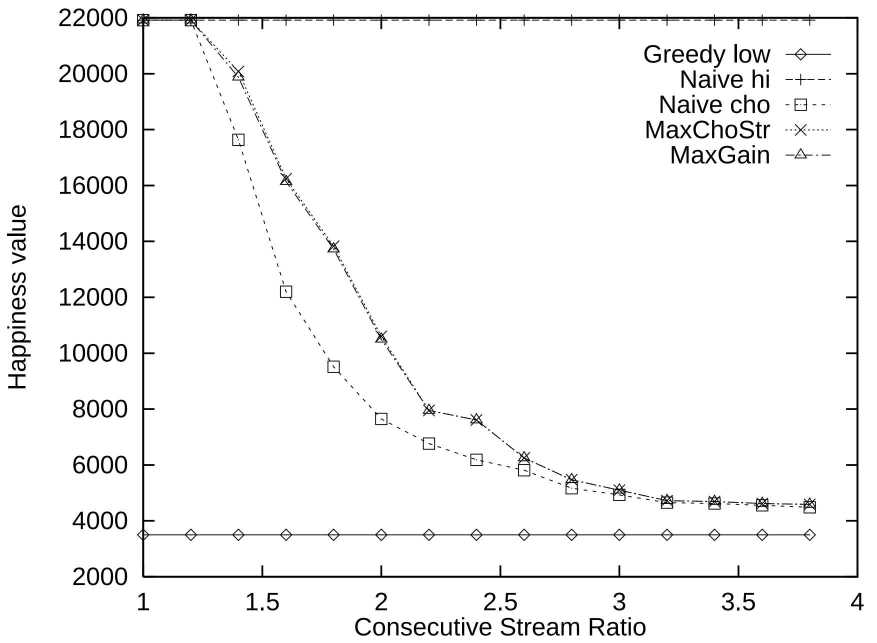


Figure 6.27: General: Total happiness versus stream ratio for the absolute profit Gain function ($Gain_2(H, C)$) in the 32-MBone graph with group-size 8.

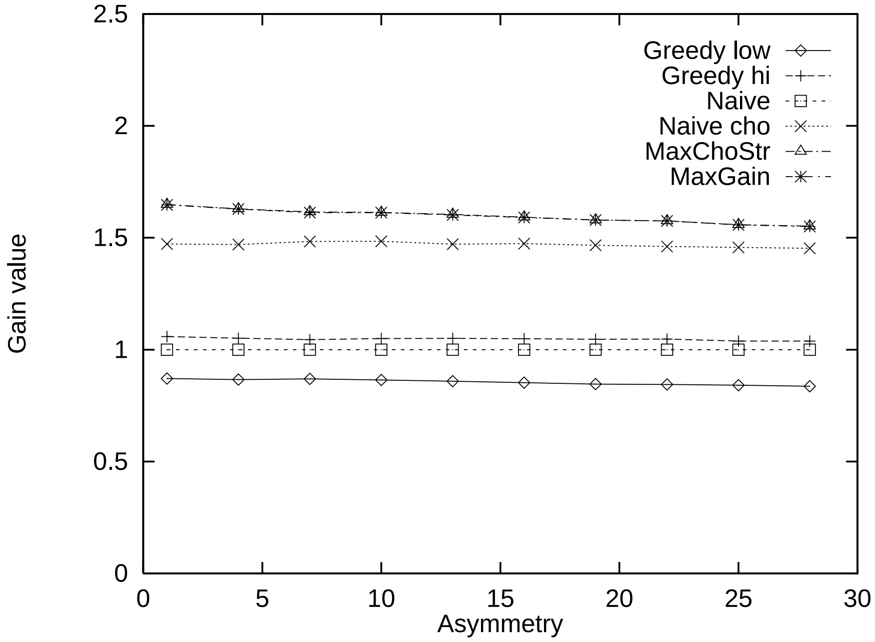


Figure 6.28: General: Gain versus asymmetry in the 32-MBone graph with group-size 8.

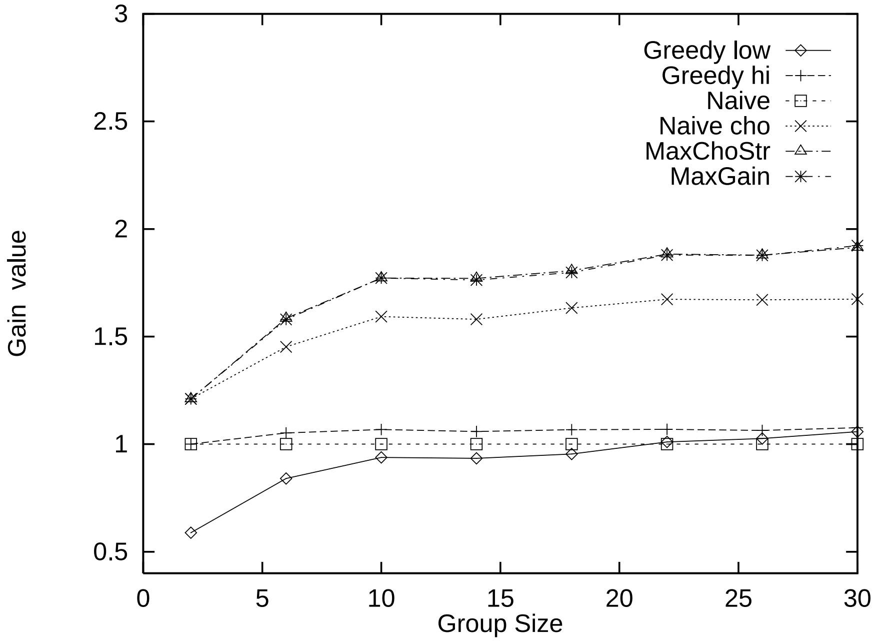


Figure 6.29: General: Gain versus group size in the 32-MBone graph with group-size 8.

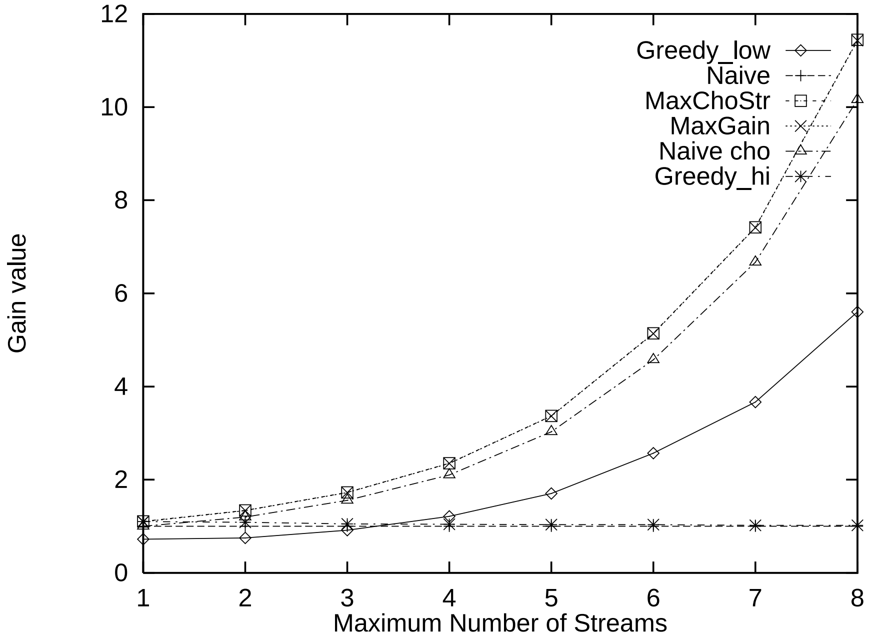


Figure 6.30: General: Total happiness versus maximum number streams in the 32-MBone graph with group-size 8.

average total happiness of a session versus the consecutive stream ratio. We see that the the stream-selecting algorithms adapt to the increase of the cost of higher streams by offering lower streams, which corresponds to lower total happiness. Note that MaxGain reduce the offered happiness earlier than MaxChoStr and Naive-cho. But, MaxGain offers less happiness for less cost, and the Gain value, which is the objective, is comparable with that of MaxChoStr(fig. 6.24). In other words, the two algorithms find two distinct local minima in the optimization problem.

We repeat the above experiments with our second cost function: $Gain_2(Hap, Cost) = k{\cdot}Hap - Cost, k \in \mathcal{R}$. In figure 6.26, we plot the Gain versus the consecutive stream ratio for $k = 75$, which balances the scales of the happiness and cost values. Note that, for stream ratio 1.7, Naive-hi and Greedy-low have the same Gain. In figure 6.27, we show the average total happiness of a session versus the consecutive stream ratio for $Gain_2(Hap, Cost)$. We can see similar general trends as in figure 6.25. A difference is that, Naive-cho reduces first the happiness it offers compared to MaxChoStr and MaxGain. This difference can be explained if we consider the advantage of the Greedy routing.

In figure 6.28, we plot the Gain versus the asymmetry of the graph. The relative difference of the algorithms reduces slightly with the asymmetry (see section 6.3.1).

In figure 6.29, we plot the Gain versus the group size. The relative difference of the stream-selecting algorithms does not change significantly.

In figure 6.30, we plot the Gain versus the maximum number of distinct streams in the session. It is interesting to note that for 4 maximum number of streams, the Greedy-low, Greedy-hi, and Naive-hi have roughly the same Gain value and the stream-selecting algorithms have almost double the Gain.

Symbol	Protocol	Comments
Offline	—	The off-line Greedy algorithm used for reference
QoSMIC	QoSMIC, YAM	Greedy on-line using minimum cost join path
RSP	PIM-SM, CBT, BGMP	Reverse Naive using a hop metric.
SP	—	PIM-SM, CBT, BGMP could do Naive routing.

Table 6.5: The protocols in our simulations.

6.4 Multicast Internet Protocol

In this section, we compare the tree cost of QoSMICwith that of other protocols. Note that we assume that QoSMIC is identical to Greedy routing; in practice, QoSMIC may not find the minimum cost path for a new destination. In addition, we compare the overhead complexity of YAM and QoSMIC. We focus on the message complexity of the Local Search, which dominates the complexity of both protocols. In this series, we consider homogeneous sessions as in series I (section 6.2).

The protocols we simulate are presented in Table 6.5. In particular, we focus on the Shared Tree that is created between one *core* (PIM, CBT) or one source (YAM, QoSMIC, BGMP) and a set of receivers. Recall that PIM, CBT, BGMP create their Shared Tree using the reverse shortest paths (receiver towards source) based on the *hop count metric* that we denote by **Reverse Shortest Paths**. We use hop counts rather than the cost metric for Reverse Shortest Paths, because that is what these protocols do, and also because it is better than using the actual cost in the wrong direction in an asymmetric link. However, we think that these three protocols can be modified to become cost and direction sensitive, which would correspond to the Naive approach (**SP**).

In creating a session, we choose the participants and the *core*/source randomly. We start from the *core*/source and add one receiver at a time. We simulate the join-only scenario, but as we saw in section 6.2, the performance of the algorithms was comparable in the join-only and join-leave problems.

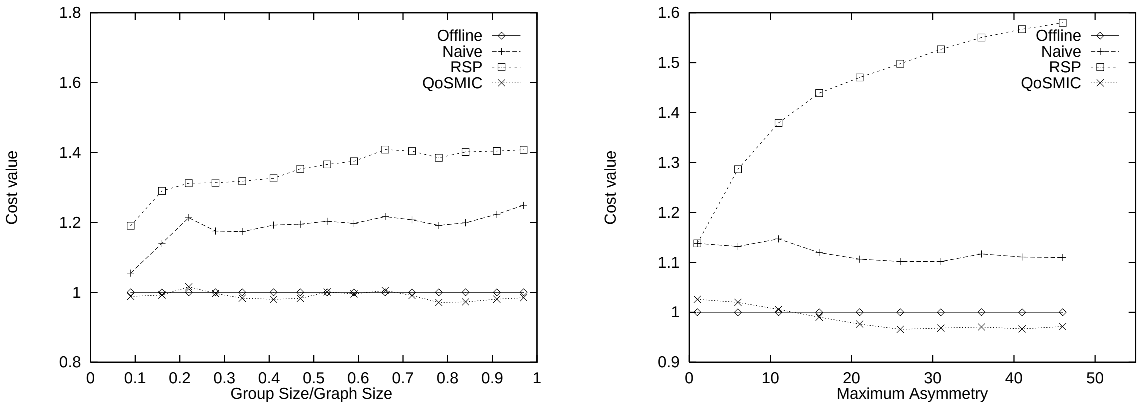
1. *Greedy routing creates more efficient trees than Naive.* This way, QoSMIC and YAM have an advantage over PIM-SM, CBT, BGMP.

2. *In asymmetric environments, protocols should consider the direction of the path.* This suggests that we should be careful in employing the direction insensitive protocols (PIM, CBT, and BGMP) in domains of the Internet with high asymmetry.

3. *QoSMIC avoids the control overhead explosion in its search phase even for networks with high average degrees.* In comparison, YAM seems to have high increase in its message complexity as the average node degree increases.

6.4.1 Routing Efficiency

In Figure 6.31, we study the cost of the distribution tree of the different protocols. Figure 6.31(a) shows the tree cost versus the group size for an asymmetry of ten. In Figure 6.31(b), we plot the tree cost versus the asymmetry for a group size of six. Given the uniform distribution of the asymmetry, the average asymmetry is $(A + 1)/2 \sim A/2$. Based on these figures, we make the following observations.



(a) Tree Cost versus Group Size

(b) Tree Cost versus Asymmetry

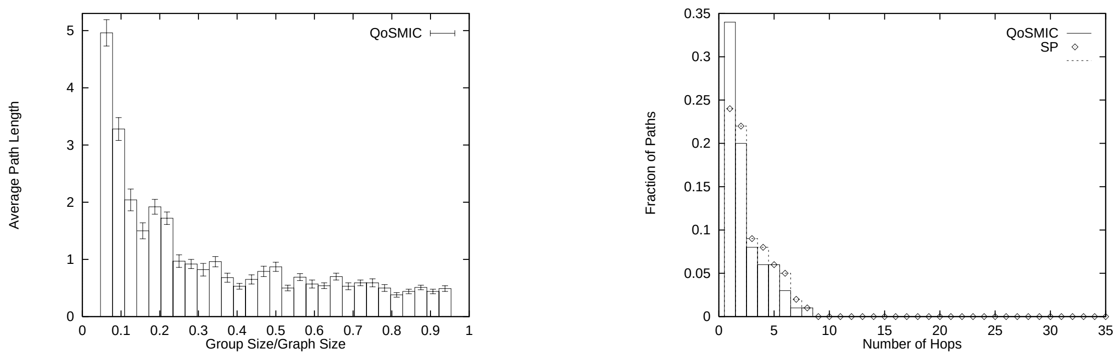
Figure 6.31: Routing Efficiency relative to the Offline multicast tree.

QoSMIC and YAM create more efficient trees. The Greedy routing of QoSMIC and YAM outperforms SP by 20% for most group sizes when the average asymmetry is five (Fig.6.31(a)). If we vary the asymmetry, QoSMIC outperforms Reverse Shortest Paths by 13% for $A = 1$ and rises to 60% as maximum asymmetry increases to $A = 45$ (average asymmetry is 45/2). (fig. 6.31(b)), in agreement with previous literature (see sec. 5.1).

Direction-aware routing is crucial. In asymmetric environments the awareness of the data flow is significant. The direction-aware Naive approach, SP, outperforms Reverse Shortest Paths by as much 42%. This suggests that the PIM, CBT, BGMP can benefit greatly from a consideration of the directed cost.

The online Greedy can perform better than the off-line Greedy in asymmetric environments. For large asymmetry, the online Greedy solution (QoSMIC) outperforms the offline Greedy solution. This is an interesting and counter-intuitive point that we discuss in Appendix B.

6.4.2 Overhead Complexity



(a) Average join distance versus group size. $(A = 1, Grp = 2 - 32)$

(b) The number of joins versus the join distance $(A = 1, Grp = 2 - 16)$

Figure 6.32: The join distance of a new join for YAM and QoSMIC.

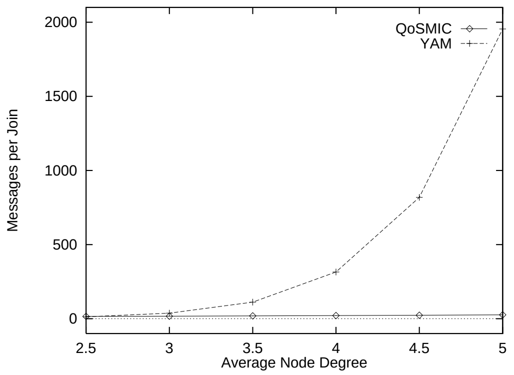


Figure 6.33: The control overhead of the Candidate search in YAM and QoSMIC.

The complexity of Local Search depends on the TTL value, which should be at least as large as the expected hop-distance between the New router and the closest point to the tree. In Figure 6.32, we study the paths that QoSMIC and YAM use for their joins. We define **join distance** to be the length of the minimum hop path between the new destination and the existing multicast tree. In Figure 6.32(a), we plot the average join distance versus the group size for a symmetric network. In Figure 6.32(b), we plot the distribution of join-paths according to their length for groups up to half the size of the network. For comparison, we plot the path distribution of Naive. We extract the following two conclusions.

Using Local Search pays off. In Figure 6.32(a), join distance reduces quickly as the group size increases. This way, the Local Search can be useful even for relatively sparse groups. This is supported also by the evidence in Figure 6.32(b): more than 50% of the join-paths (first two columns) have length less than or equal to two. Therefore, we conclude that a Local Search even with a small TTL value can be beneficial.

The Importance of Multicast Tree Search. In Figure 6.32(b), we see that a small percentage of the paths have lengths as large as eight. If we rely uniquely on the Local Search (as YAM does) then the TTL of the search should be at least as large or else some routers would be excluded from some distant multicast sessions. This suggests that Local Search alone would need large TTL values, which increases the overhead complexity significantly.

As a conclusion, we want to have a limited Local Search to take advantage of the many "short" joins, but we need a second search mechanism to provide solutions when the Local Search falls short. For this environment, a single Local Search of two hops seems a reasonable choice for QoSMIC, while YAM would have to go up to eight.

Scalability of message complexity.

Using data from our simulations, we calculate of the overhead of the searches of YAM and QoSMIC using the analysis of Section 5.5. Namely, from the experiment of Figure 6.32(b), we get the number of joins, and the distance of the node from the tree.

YAM: The maximum TTL of a Local Search should be at least as large as the maximum join distance or else some routers would be excluded from some distant multicasts. Thus, the maximum TTL is set equal to the path length of the join. We increase the diameter of each expanding ring by one. This way, although the set-up delay may suffer, we attempt to minimize the message complexity of YAM.

QoSMIC: The Local Search does not have to succeed for every join, since we have the Multicast

Tree Search to fall back on. This way, we choose a maximum TTL value of two, as suggested earlier. In addition, we over-estimate the messages of Multicast Tree Search by considering it equal to the number of intree nodes. However, using the mechanisms, the number of messages should be smaller in practice.

In Figure 6.33, we calculate the message complexity per join for various average degrees. The message complexity of YAM increases dramatically with the average degree, due to its extensive Local Search. In comparison, QoSMIC avoids this problem: having the Multicast Tree Search to fall back to we can afford to keep the Local Search small. Similarly, YAM works poorly for large networks, where the TTL value of the Local Search would have to be large. As a conclusion, YAM can be applied in small or sparse networks, but QoSMIC scales better to dense or large networks.

6.5 Conclusions

Our simulations lead to a number of conclusions that we already mentioned in section 6.2.1. Here we present a higher level discussion of the conclusions of each series of simulations.

- Greedy creates more efficient trees than Naive in a variety of scenarios and graph topologies. In our simulations in series I, the tree cost of Greedy is no larger than that of Naive, and it can be less by up to 25%. In addition, the number of joins and leaves do not affect the performance of Greedy in the join-leave problem. This conclusion challenges the intuition that stems from the worst case analysis of chapter 3. In addition, the Internet community has been reluctant to use Greedy-based multicast routing, and one of the main concerns was a possible tree degradation with time [Tha98]. All of the above suggest that in selecting a routing algorithm, it is worth considering Greedy.

- In the heterogeneous case, stream-selection is crucial for an effective use of our resources. We can also improve the solution further, if we address the stream-selection and the routing problem at the same time. In addition, our Choose-Stream procedure and the Greedy-based algorithms we proposed seem promising.

- Multicast Internet protocols should consider the asymmetry of the network in their routing. Protocols using Reverse Shortest Paths routing lead to higher tree cost by 20% for average asymmetry five compared to the cost of Naive. The role of routing is to provide paths that can actually support the traffic a connection. In addition, there are cases where the links are physically asymmetric or even one-way, e.g., satellite links.

- The flexibility of QoSMIC makes it possible to control its message complexity. Therefore, the protocol scales well even in large and dense networks. The message complexity of QoSMIC increases more slowly than that of the YAM protocol with the average degree of the graph. Even for small average degrees ($4-5$ out-degree), the number of messages required by YAM becomes an order of magnitude larger than the number required by QoSMIC.

Chapter 7

Conclusions

In our work, we have studied algorithms and protocols for routing multicast connections. We extended and compared the Greedy and the Naive algorithms, which are the most widely used routing solutions. The algorithms are intuitive and simple in concept, and they can easily be implemented in a distributed way. It is for these reasons that they dominate practical research and employed solutions. We study these algorithms in three variations of the multicast problem theoretically, and with the use of simulations. First, we study the effect of the asymmetry of the communication graph on the performance of the algorithms. Second, we examine how hierarchical encoding can improve the routing of multicast sessions (e.g., video data distribution). Finally, we propose a protocol for the Internet that can support QoS requirements. The protocol is based on insight gained from the previous investigations, but extends further to address practical considerations of the Internet environment.

7.1 Contributions

The results from our work lead to the following conclusions and observations.

1. *Directed asymmetric graphs are a more accurate model for communications networks than undirected or symmetric graphs.* Our work shows that asymmetry affects the multicast routing issues. First, asymmetry makes it more difficult to approximate the exact solution of the problem. Second, asymmetry changes the relative performance of algorithms. The effect of asymmetry is visible in both the worst case analysis and simulations of some representative cases. For example, recall that the competitive ratio of Naive is $\Theta(M)$ and that of Greedy is $O(\min(A, M))$ for the static problem: for $A = 1$, Greedy outperforms Naive while for $A = M$ their competitive ratios are the same. This suggests that directed graphs should be used in the design and analysis of algorithms. Furthermore, even if the directed graph is perfectly symmetric with all pairs of reverse edges having identical capacities, still the directed graph is needed for realism, since traffic in one direction does not influence or impede traffic in the opposite direction between two nodes. As an example, recall that, for average asymmetry of five, Reverse Shortest Paths routing leads to higher tree cost by 20% compared to the cost of Naive and by 30% compared to Greedy.

2. *Greedy is a good approximation algorithm for several networks and scenarios.* Our results provide the first comprehensive bounds on the performance of Greedy and Naive for directed graphs. For the static and join-only problems, Greedy performs much better than Naive.

Furthermore, Greedy is almost as good as *any* on-line algorithm, which is a very strong argument in its favour. In the join-leave case, the performance of Greedy can be very bad in the worst case. However, the observed performance of Greedy does not approach its worst case performance in several experimental studies including our experiments. Actually, Greedy usually does better than Naive even in the join-leave case, as we saw in section 6.2 and in other studies [DL93] [Wax93].

Given the dominant use of these algorithms, the results can be used for the interpretation and performance prediction of applied algorithms and protocols. In our research, the intuition from these results has played an important role in the rest of this work.

3. *Heterogeneous multicasting can be more efficient with Greedy-based stream-selecting algorithms.* We formulate the heterogeneous problem in full generality, and propose Greedy-based algorithms that outperform previous solutions. The general heterogeneous problem consists of two conflicting and closely related problems: use minimization and quality maximization. We address these problems simultaneously for the first time, and we show the benefit from such a joint solution. The gain is a result of a) the improved Greedy routing, and b) the stream (or quality level) selection techniques we propose. In more detail, our Greedy-based algorithms, MaxStream and MaxChoStr, outperform previous Naive-based solutions. We demonstrate their advantage by proving bounds in the worst case. In addition, we show the relative benefit in representative cases in a series of simulations. In the fixed problem, our algorithms based on Greedy create trees whose routing cost is less by 25% compared to previous solutions. In the general problem, Greedy routing and stream-selection lead to solutions that outperform previous solutions when we compare their $Gain()$ values.

   The heterogeneous problem appears in various practical situations. In this work, we focus on the distribution of video, which is a type of data that lends itself to hierarchical encoding. This way, applications like tele-education, tele-conferencing, and (near) video-on-demand could benefit from our algorithms. In addition, the notion of variable user-requirements appears in other multicast variations such as end-to-end constrained multicasting. Assuming constraints such as end-to-end delay bounds, different bounds create the need for different approaches to resource allocation as in the heterogeneous problem we examined. The algorithms developed here can be useful in other heterogeneous problems, such as real-time multicast communications in ATM networks [FK97].

4. *QoSMIC is a promising Internet protocol that is able to support QoS requirements.* QoSMIC is the first proposed scalable QoS-sensitive Internet protocol. Its routing combines the Greedy and Naive approaches in a flexible and dynamic framework. Supporting QoS requires increased control overhead compared to QoS-insensitive protocols, and this overhead was the the main problem of YAM, the previous QoS-sensitive protocol. However, QoSMIC introduces mechanisms that make the overhead controllable. A major innovation of QoSMIC is that it has reduced significantly the impact of pre-configuration decisions (e.g., *core* router), that is an unresolved problem of most previous protocols. A significant advantage of QoSMIC lies in its flexibility: it can adapt to the changing network conditions and respond to the requirements of different applications and users. In addition, when QoSMIC follows the Greedy routing scheme, it can take advantage of the higher routing efficiency (smaller tree cost) of Greedy, as we saw in section 6.4.

   The protocol has been presented in the Inter-Domain Multicast Routing (IDMR) working group of the 41st Internet Engineering Task Force (IETF) meeting [IET98]. The idea of incorporating QoS in the routing process was considered feasible and promising. A draft of our protocol [BFP98] has been submitted, which is the first step of the long process that may lead to an Internet standard[1].

7.2 Future Work

For the join-only problem, for less asymmetric graphs, we see a discrepancy between the lower and upper bounds for the Greedy algorithm. It is likely that the upper bound could be reduced. If we consider only one destination, it could be the case that the directed edges towards it are the higher cost edges (of each opposite pairs). In this case, even the optimal solution would have to include high cost edges, so the difference between the two solutions is not proportional to the asymmetry.

Rerouting. Rerouting can improve the performance of on-line problems, but it also has an associated cost. It is an interesting optimization problem to identify metrics for the tree degradation, and the criteria that would trigger the rerouting. In addition, it would be interesting to examine rerouting policies that would promise local and minimal disruption.

Regarding heterogeneous multicasting, a complementary aspect of this issue is the "pricing": associating resource use with a monetary price (which reflects on our notion of happiness). An effective pricing policy is essential for commercial network management.

The QoSMIC protocol establishes a basis for more research and implementation work. First, we would like to model in more detail the Internet environment; we would like to measure membership behavior and identify the topology of the network. Second, we want to fine-tune our protocol according to the properties and needs of Internet sessions. Third, we can implement our protocol and test it on the Internet. In these steps, feedback from the IETF community and interaction with interested industrial partners will be valuable. This will strengthen the protocol and advance it in two parallel but independent routes: towards becoming an Internet standard, and towards being implemented in commercial routers.

[1]During this procedure, a protocol matures by further research and feedback from the IETF community. A protocol can upgrade its status to a Request For Comments (RFC) and eventually to an Internet Standard by gaining the approval of the community.

Appendix A

Additional Simulations for Series I

In our simulations, we compare the performance of our algorithms on the two instances of real networks, and on a range of Random Euclidean graphs.

In these simulations, we make the following observations.

1. The simulations presented here indicate that the results we observe on the 32-MBone and 84-MBone graphs are also observed in a variety of other graphs.

2. The graph size does not seem to affect the relative performance significantly, which also agrees with previous experimental work [KP98].

3. The relative performance on the 84-MBone is considerably reduced compared to that on the 32-MBone graph (see figures 6.2, A.1). We can explain this, if we recall that the 84-MBone graph includes the 32-MBone graph plus 52 nodes that appear in trees. From a routing point of view, these 52 nodes do not introduce any interesting cases, but they increase the tree cost of both solutions by the same amount, which reduces the advantage of using Greedy over Naive. Recall that this observation confirms previous experimental work by other researchers [KP98].

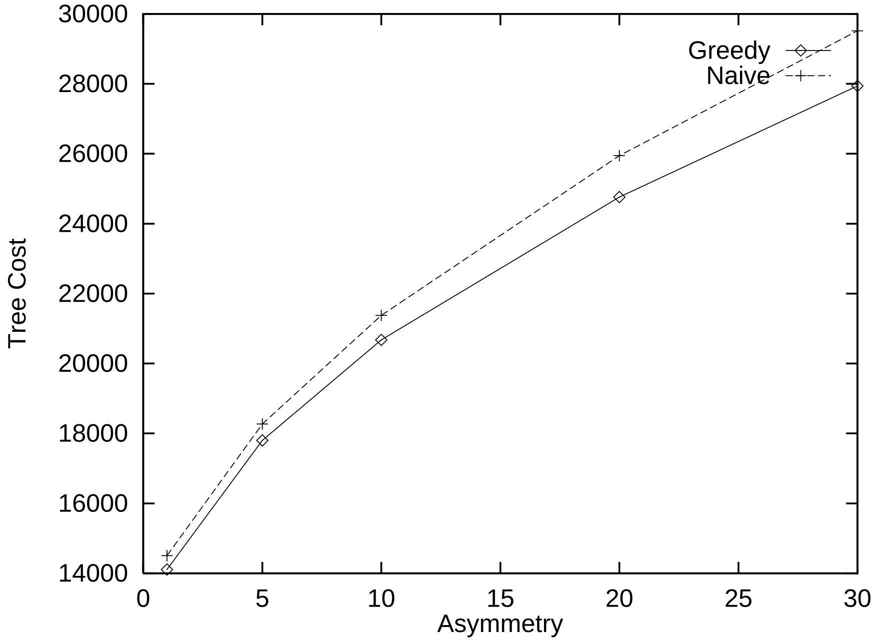


Figure A.1: Tree Cost versus Asymmetry in the 84-MBone map for the static problem with group-size 10.

A.1 The Static Problem

In this section we study the static problem.

Asymmetry. We plot the tree cost and end-to-end path length versus the asymmetry of 84-MBone graph in figures A.1, A.2 respectively. In figures A.3, A.4, we show the the tree cost and end-to-end path length versus the asymmetry ona 84-Euclidean graph. The two pairs of graph have comparable general trends.

Group Size. In figures A.5, and A.6, we show the the tree cost and end-to-end path length versus the group size on a 32-Euclidean. In figures A.3, A.4, we show the the tree cost and end-to-end path length versus the group size on a 84-Euclidean.

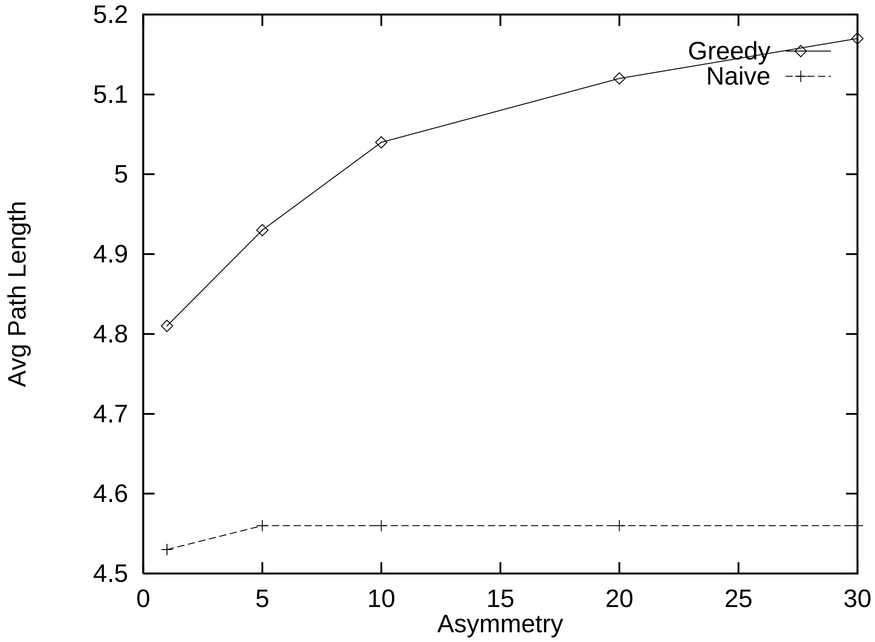


Figure A.2: Path Length versus Asymmetry on the 84-MBone map for the static problem with group-size 10.

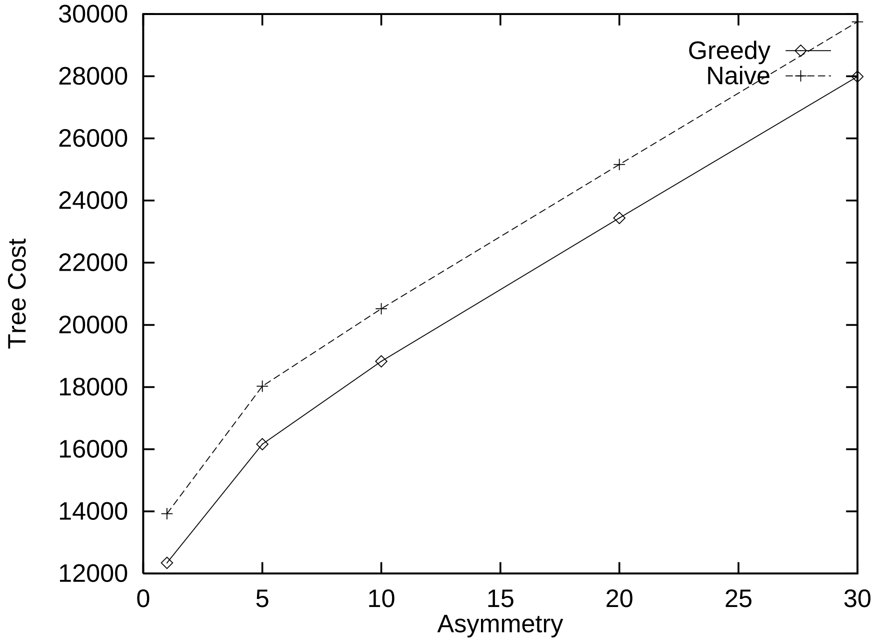


Figure A.3: Tree Cost versus Asymmetry in the 84-Euclidean map for the static problem with group-size 10.

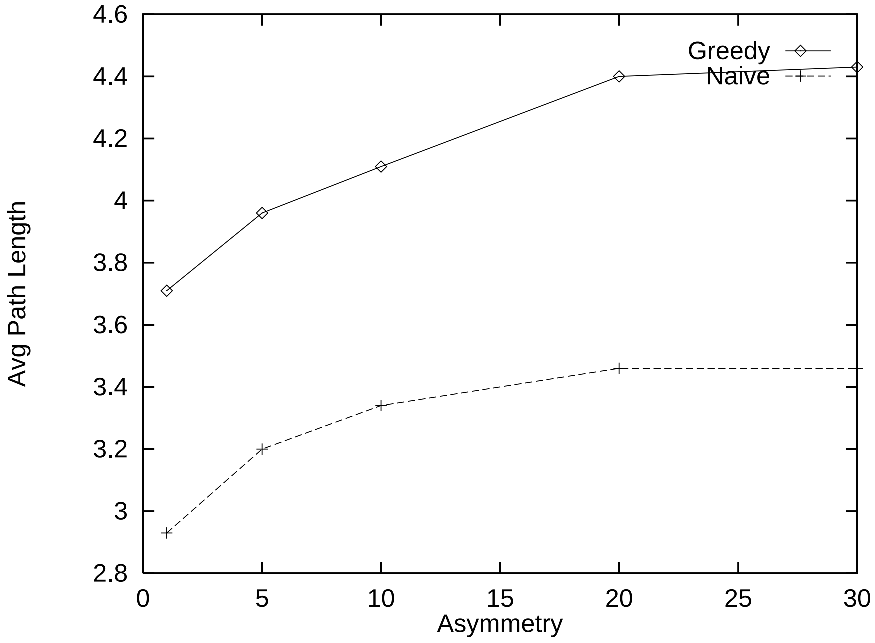


Figure A.4: Path Length versus Asymmetry on the 84-Euclidean map for the static problem with group-size 10.

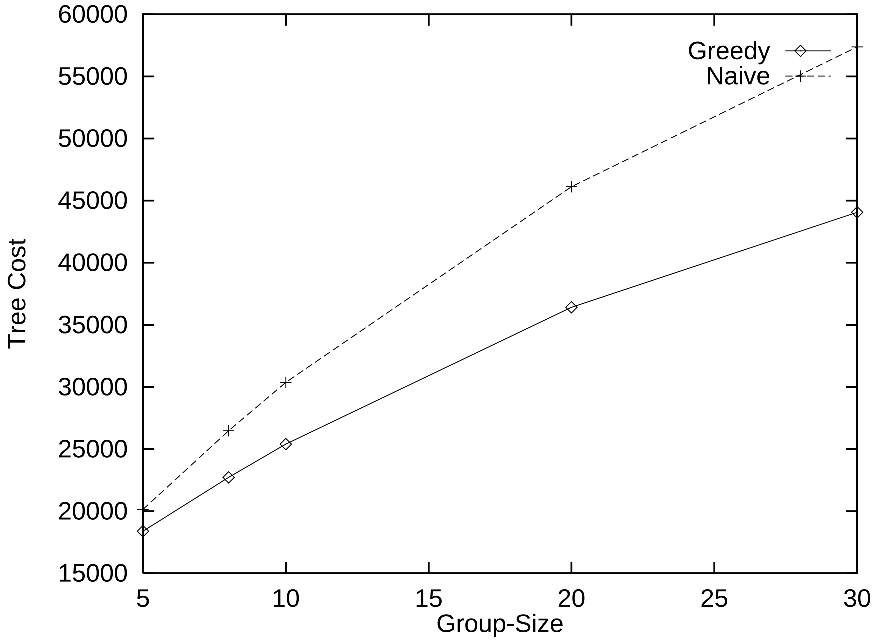


Figure A.5: Tree Cost versus Group-Size on the 32-Euclidean map for the static problem with group-size 8.

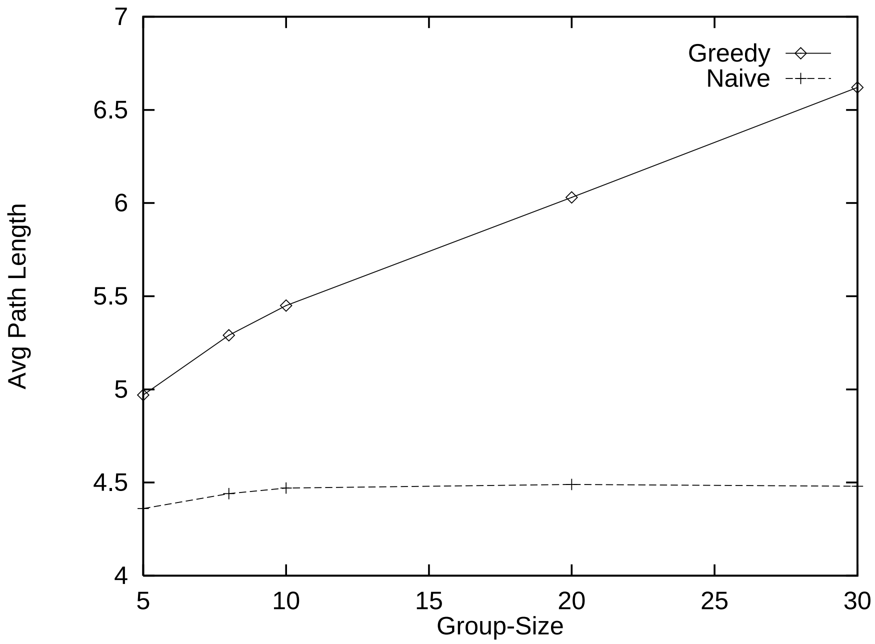


Figure A.6: Path Length versus Group-Size on the 32-Euclidean map for the static problem with group-size 8.

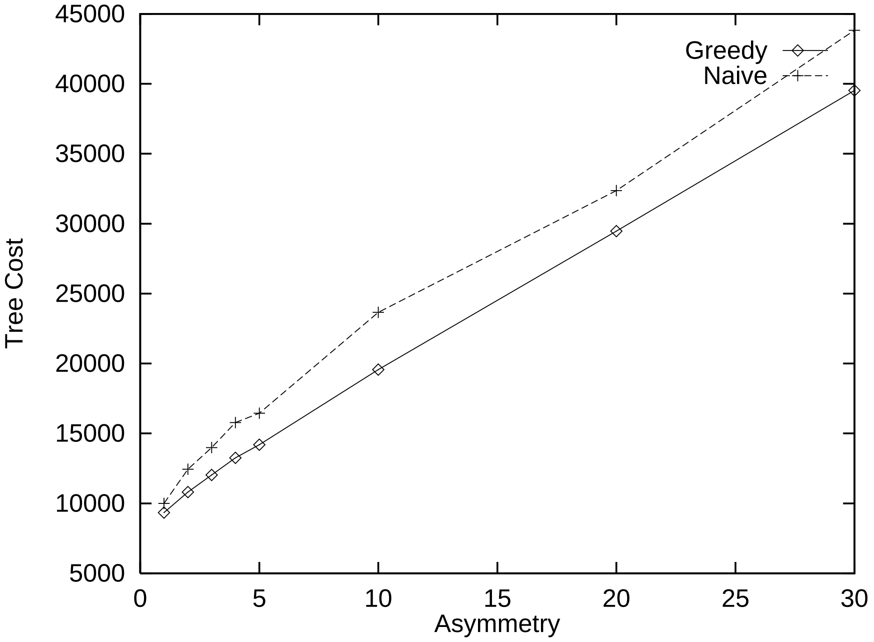


Figure A.7: Tree Cost versus Asymmetry in the 32-MBone map for the join-only problem with group size 8.

A.2 The Join-only Problem

In this section we study the join-only problem.

Asymmetry. We plot the tree cost and end-to-end path length versus the asymmetry for our 32-MBone graph in figures A.7, and A.8 respectively. We can compare this pair of plots with the similar pair of plots for the 32-Euclidean graph in figures A.9, and A.10. We can see that the asymmetry has comparable effect in the two topologies.

Group Size. We can also compare the effect of the group size in the two graphs: 32-MBone and 32-Euclidean. We plot the tree cost and end-to-end path length versus the asymmetry for our 32-MBone graph in figures A.11, and A.12 respectively, and for the 32-Euclidean graph in figures in figures A.13, and A.14.

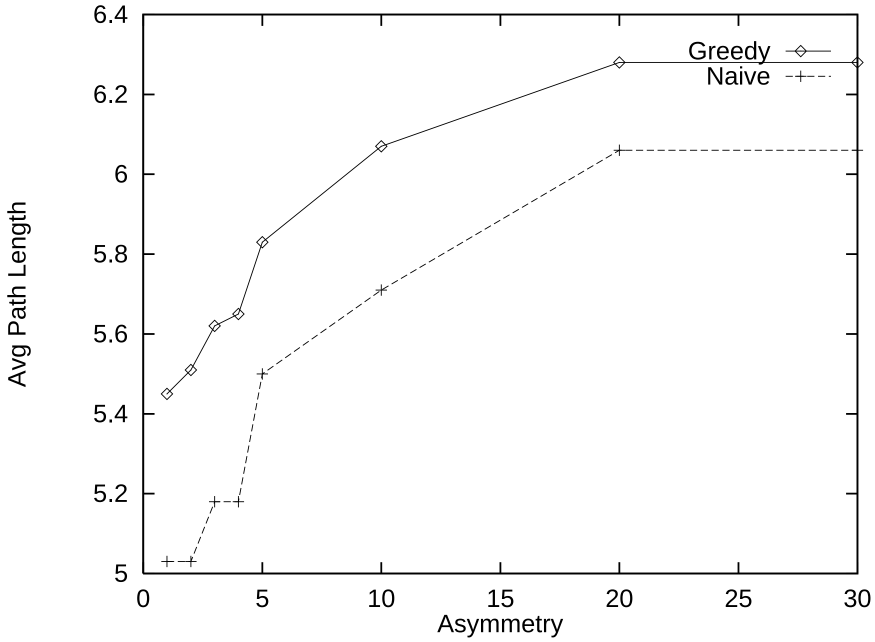


Figure A.8: Path Length versus Asymmetry on the 32-MBone map for the join-only problem with group size 8.

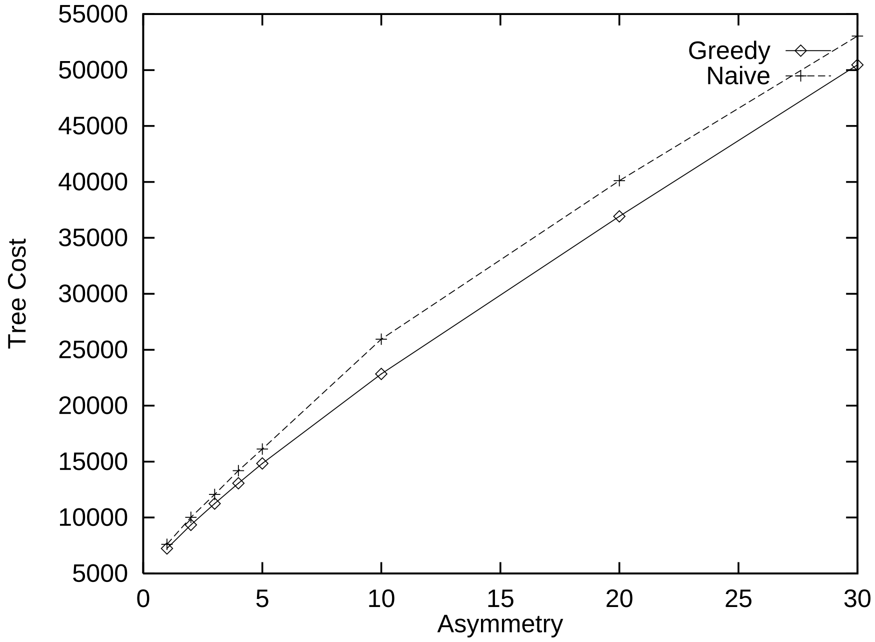


Figure A.9: Tree Cost versus Asymmetry on the 32-Euclidean map for the join-only problem with group size 8.

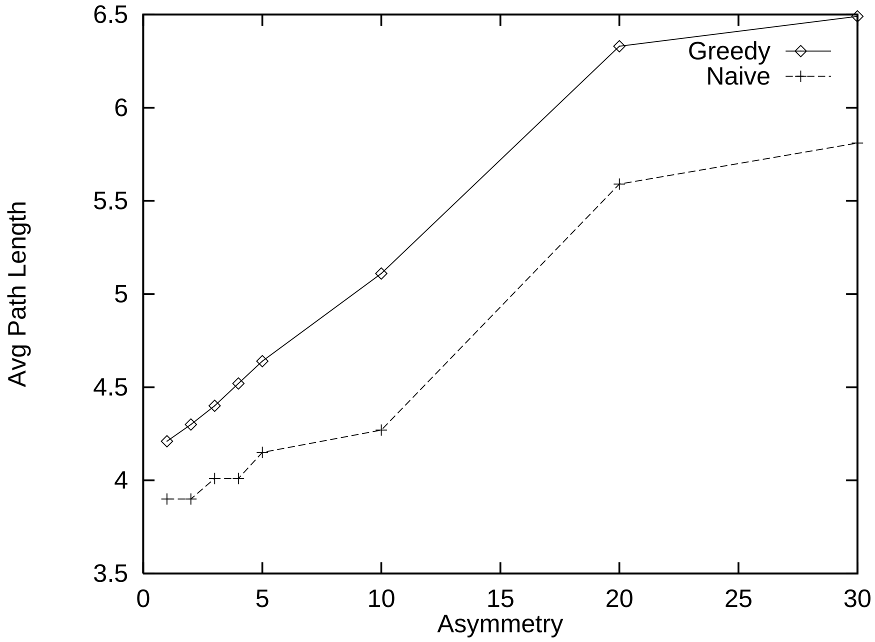


Figure A.10: Path Length versus Asymmetry on the 32-Euclidean map for the join-only problem with group size 8.

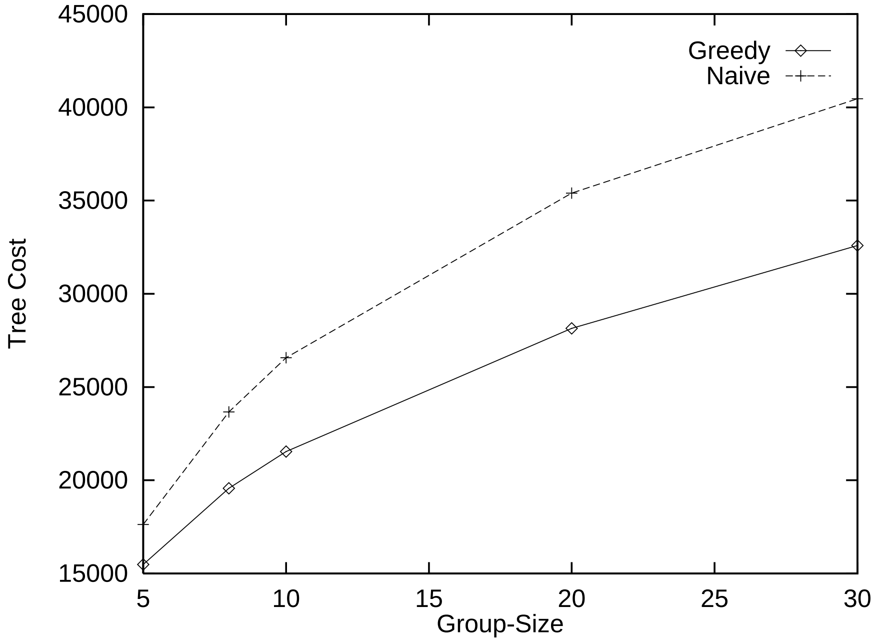


Figure A.11: Tree Cost versus group Size in the 32-MBone map for the join-only problem with asymmetry 10.

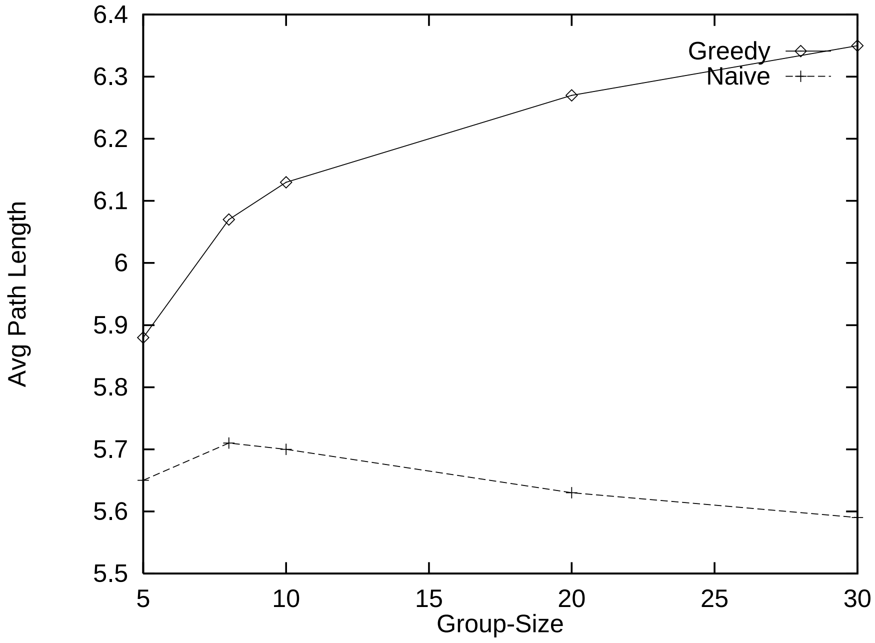


Figure A.12: Path Length versus group Size on the 32-MBone map for the join-only problem with asymmetry 10.

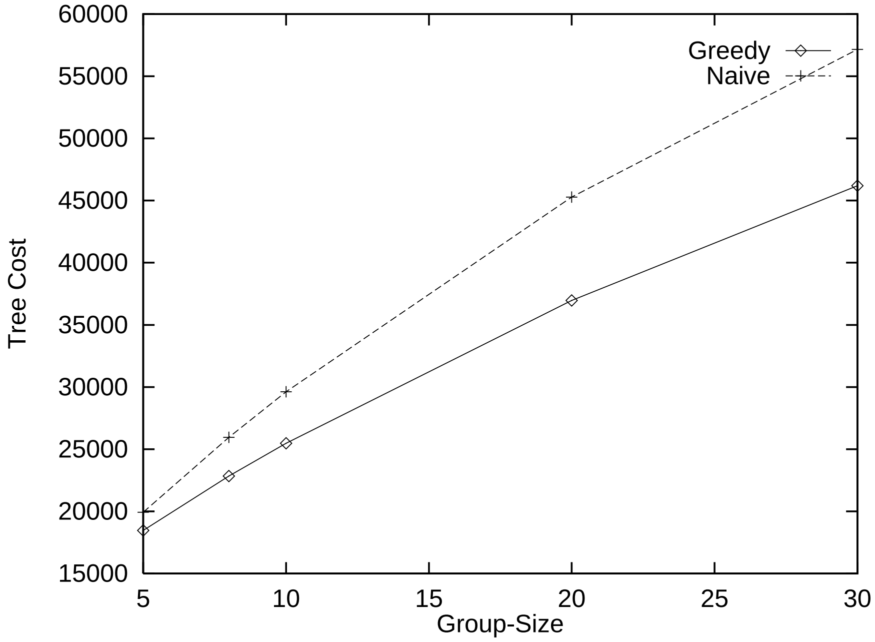


Figure A.13: Tree Cost versus groupsize on the 32-Euclidean map for the join-only problem with asymmetry 10.

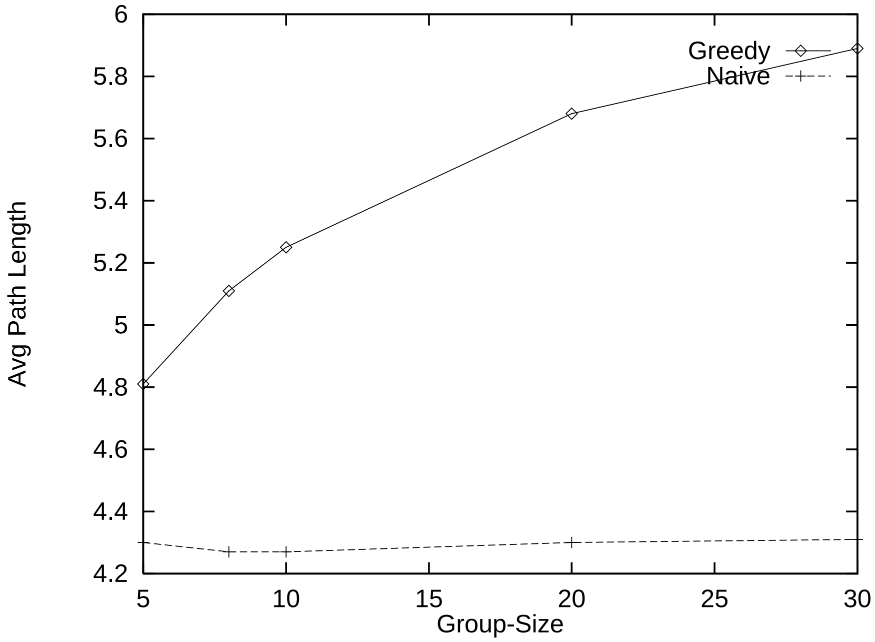


Figure A.14: Path Length versus groupsize on the 32-Euclidean map for the join-only problem with asymmetry 10.

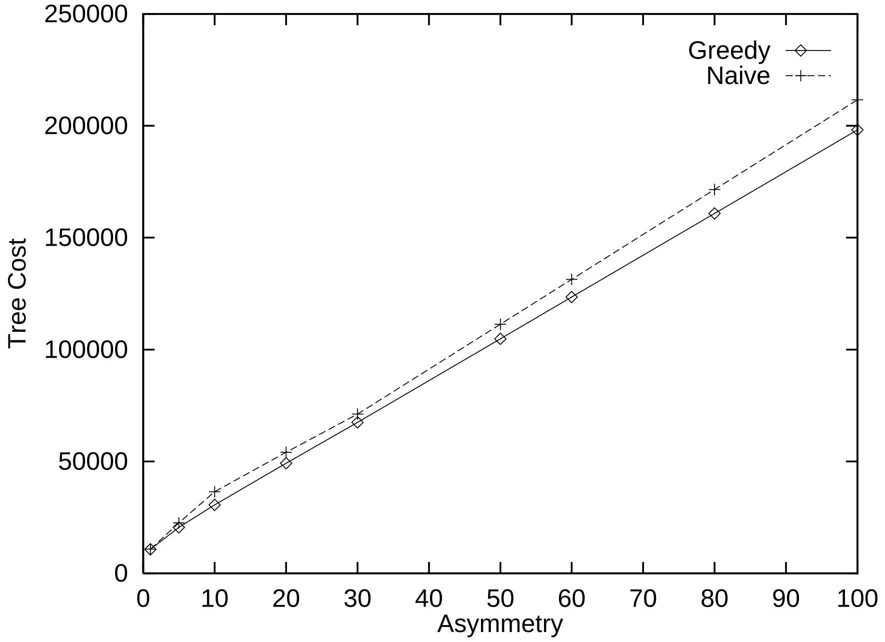


Figure A.15: Tree Cost versus Asymmetry on the 32-Euclidean map for the join-leave problem with group-size 8.

A.3 The Join-Leave Problem

In this section we study the join-leave problem. We vary a number of parameters and examine the sensitivity of our results to a number factors that we list in section 6.1.

Asymmetry. We plot the tree cost and end-to-end path length versus the asymmetry for a 32-Euclideangraph in figures A.15, and A.16 respectively. We can compare this pair of plots with the similar pair of plots for the 32-MBONE graph in figures 6.12, and 6.13.

We also plot the tree cost and end-to-end path length versus the asymmetry for the 84-MBone graph in figures A.17, and A.18 respectively, and compare it with the plots on 84-Euclidean graph in figures A.19, and A.20. We can see that the asymmetry has comparable effect in the two topologies.

Group Size. We plot the tree cost and end-to-end path length versus the asymmetry for our 84-MBone graph in figures A.21, and A.22 respectively. The plot shows similar general trends as the pair of figures for the 32-MBone graph in figures 6.14, and 6.15.

Requests-Time. We plot the tree cost and end-to-end path length versus the number of requests or time in figures A.23, and A.24 respectively. We can compare these two plots with figures 6.16, and 6.17 respectively.

Graph Size. We conduct an experiment increasing the size and a, and and keeping the average degree equal to 4.3 and paramater $a = 0.25$. We also made the group size equal to one fourth of the graph under consideration, because this would make a more fair comparison. In figures A.25 and A.26, we plot the tree cost and end-to-end path length versus the size of the graph respectively.

Average Node Degree. In figures A.27 and A.28, we plot the tree cost and end-to-end path length versus the average node degree. In doing this plot we vary both the parameters a, b. It is interesting to note that this plot looks like an interpolation of the plots where we varied each of the parameters (see figs. 6.8, and 6.10). Recall that those figures are for the join-only problem, so

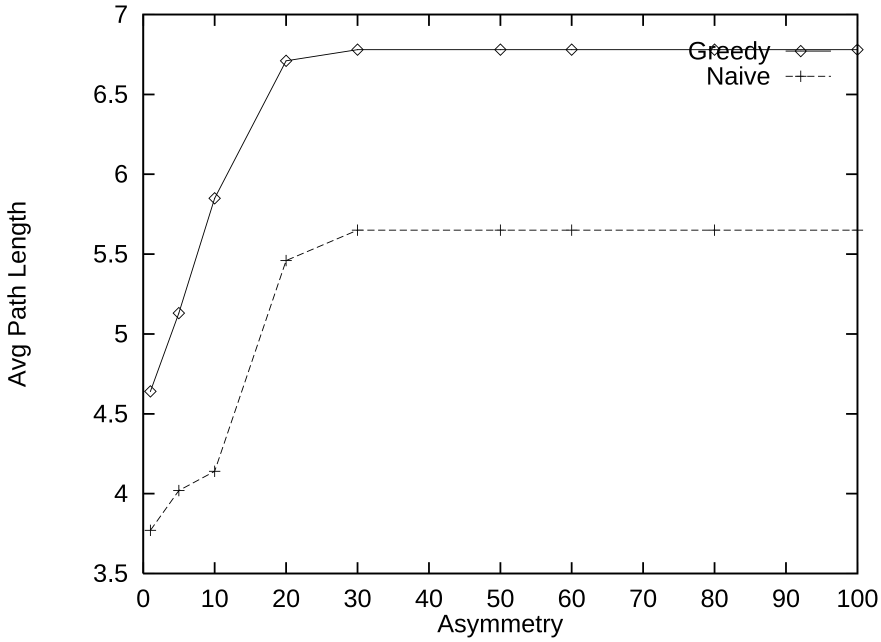


Figure A.16: Path Length versus Asymmetry on the 32-Euclidean map for the join-leave problem with group-size 8.

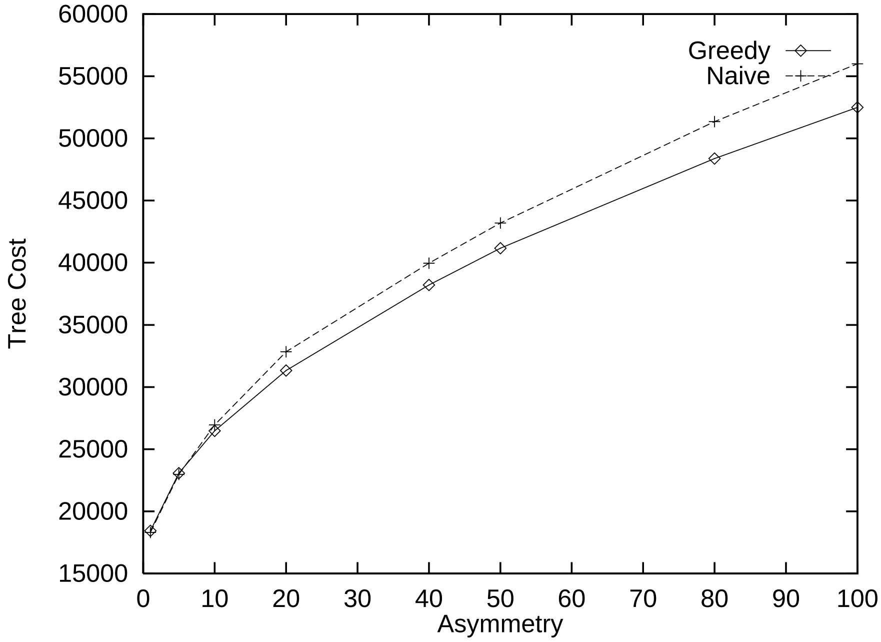


Figure A.17: Tree Cost versus Asymmetry in the 84-MBone map for the join-leave problem with group-size 10.

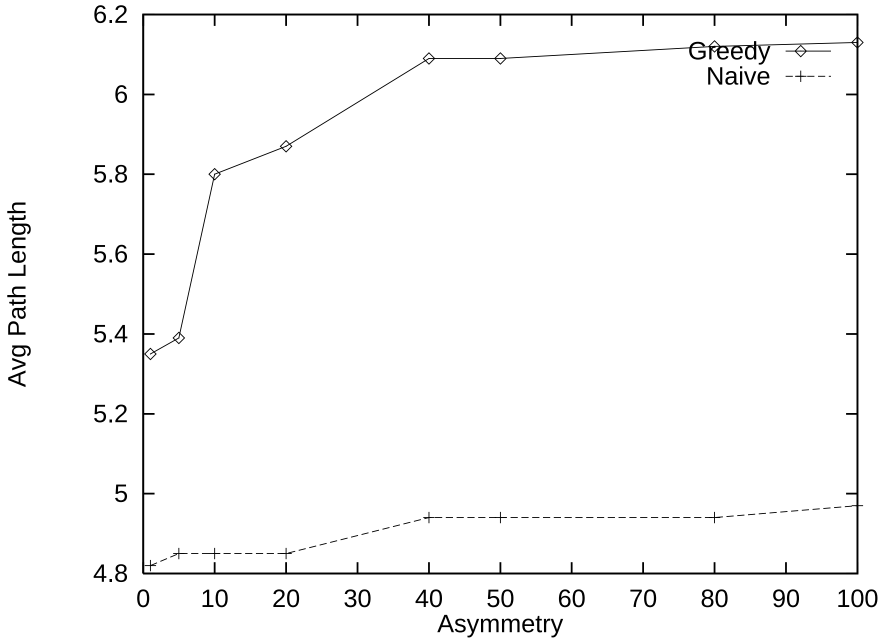


Figure A.18: Path Length versus Asymmetry on the 84-MBone map for the join-leave problem with group-size 10.

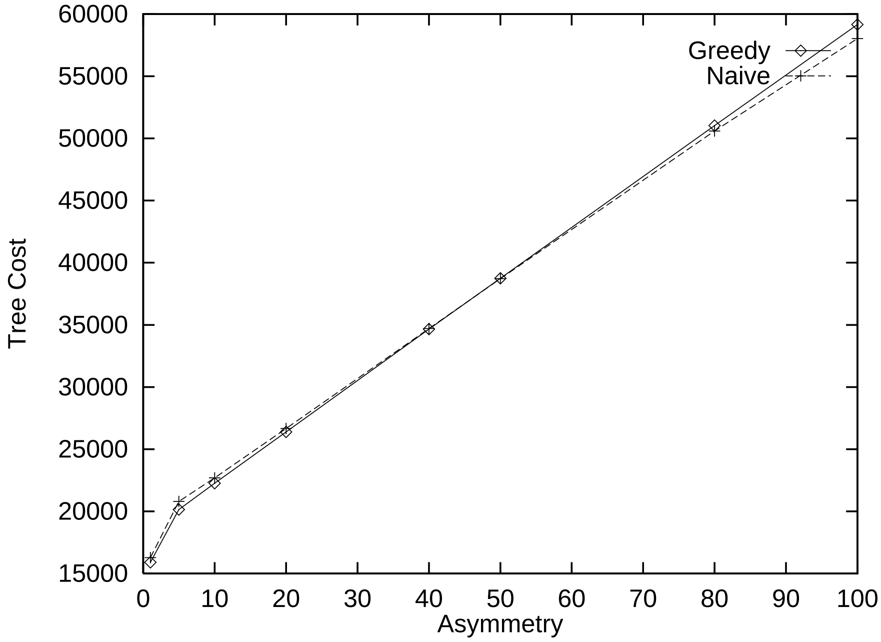


Figure A.19: Tree Cost versus Asymmetry on the 84-Euclidean map for the join-leave problem with group-size 10.

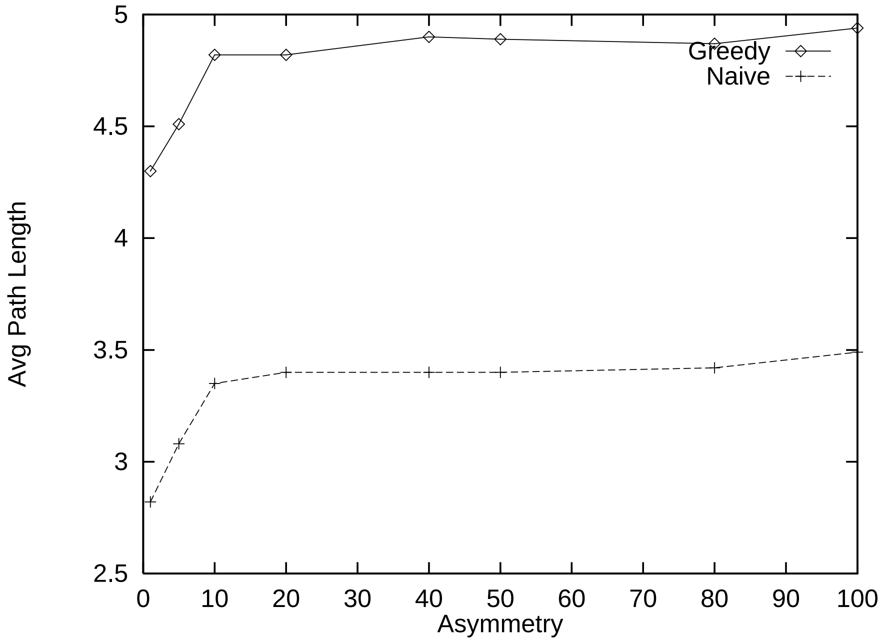


Figure A.20: Path Length versus Asymmetry on the 84-Euclidean map for the join-leave problem with group-size 10.

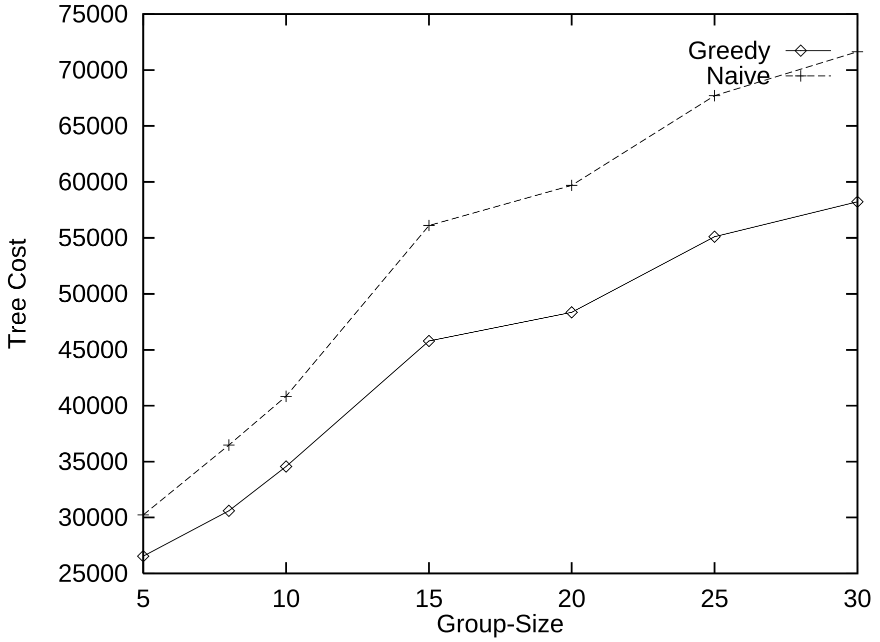


Figure A.21: Tree Cost versus Group-Size on the 32-Euclidean map for the join-leave problem with asymmetry 10.

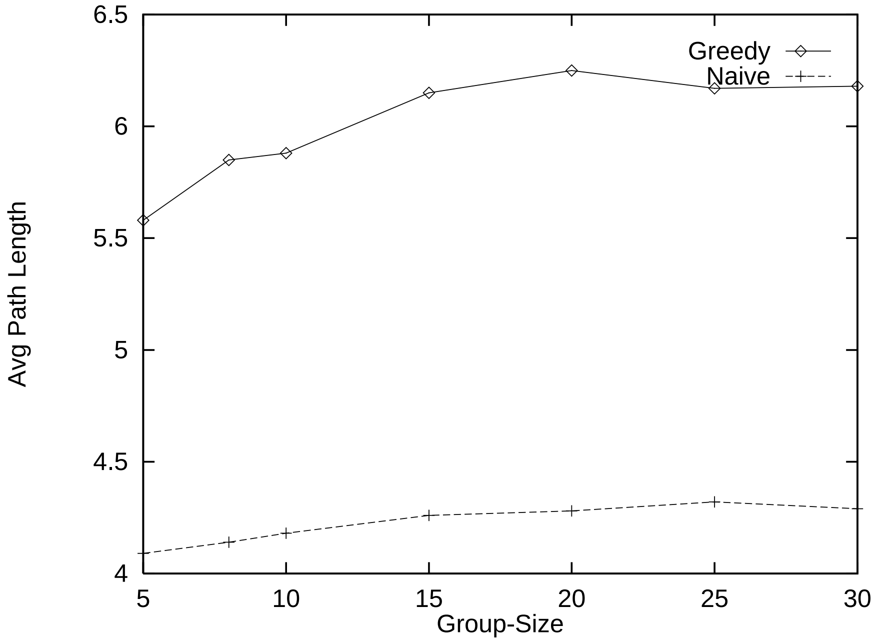


Figure A.22: Path Length versus Group-Size on the 32-Euclidean map for the join-leave problem with asymmetry 10.

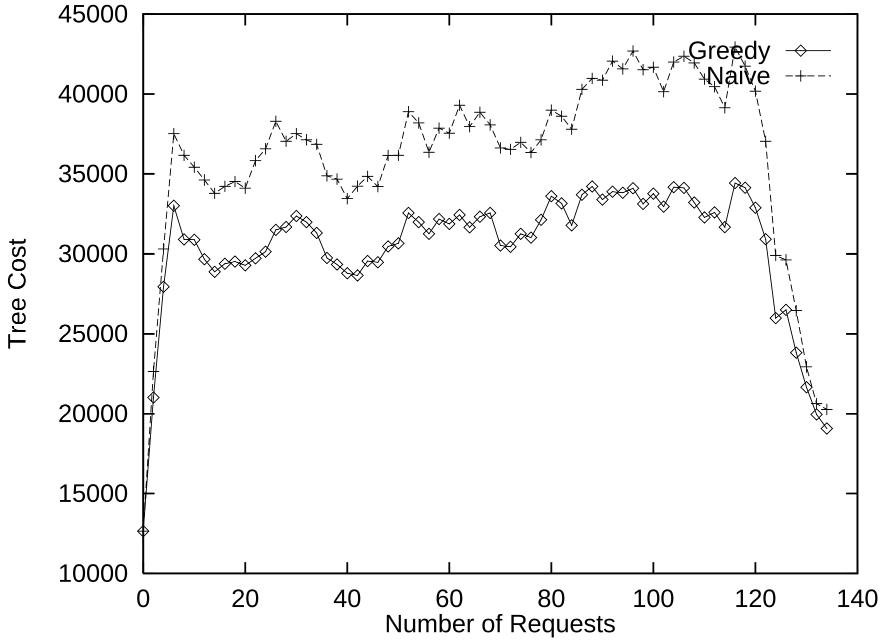


Figure A.23: Tree Cost versus Group-Size on the 32-Euclidean map for the join-leave problem with asymmetry 10 and group-size 8.

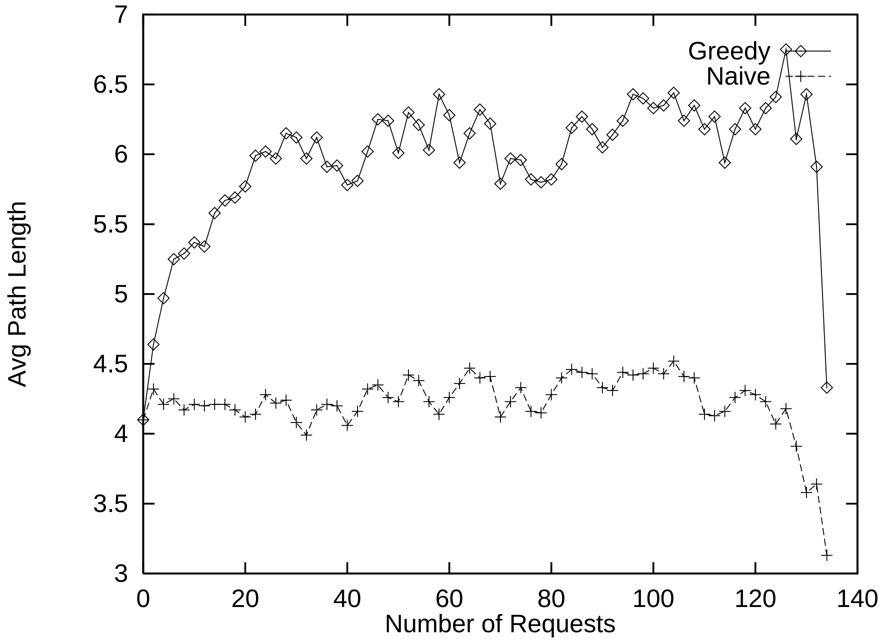


Figure A.24: Path Length versus Group-Size on the 32-Euclidean map for the join-leave problem with asymmetry 10 and group size 8.

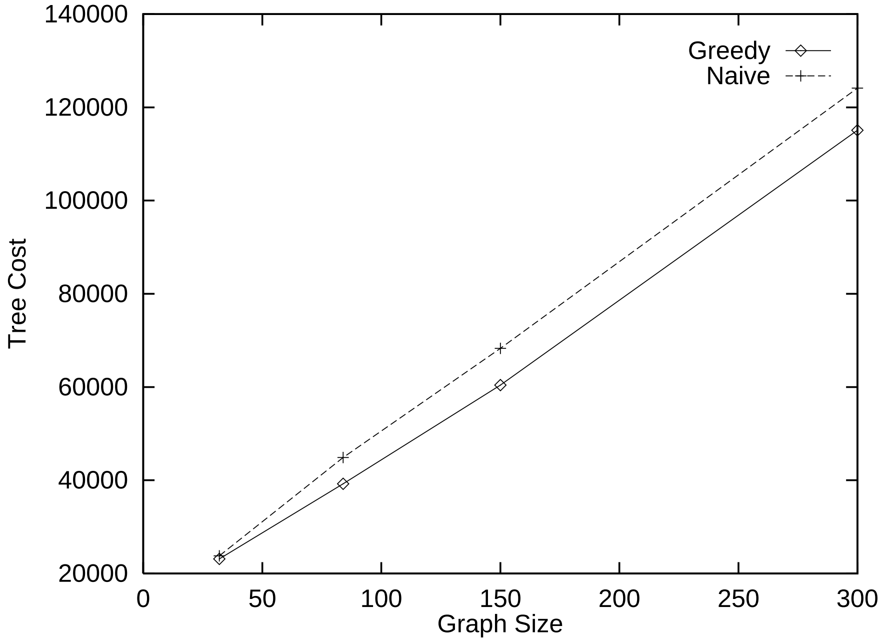


Figure A.25: Tree Cost versus Graph Size for the join-leave problem with asymmetry 10, average node degree and group size equal to one fourth of the nodes of the graph.

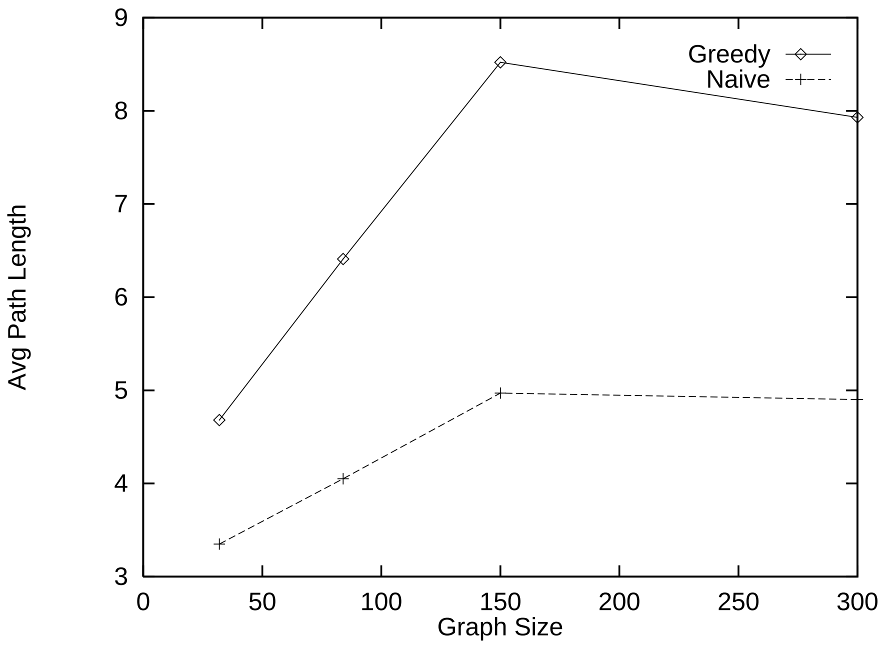


Figure A.26: Path Length versus Graph Size for the join-leave problem with asymmetry 10 and group size equal to one fourth of the nodes of the graph.

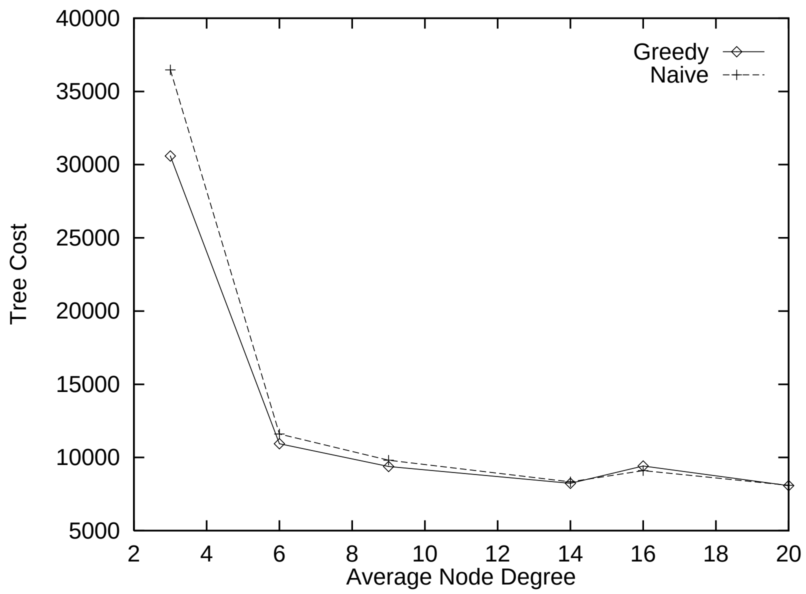


Figure A.27: Tree Cost versus average node degree in the 32-Euclidean map for the join-leave problem with asymmetry 10.

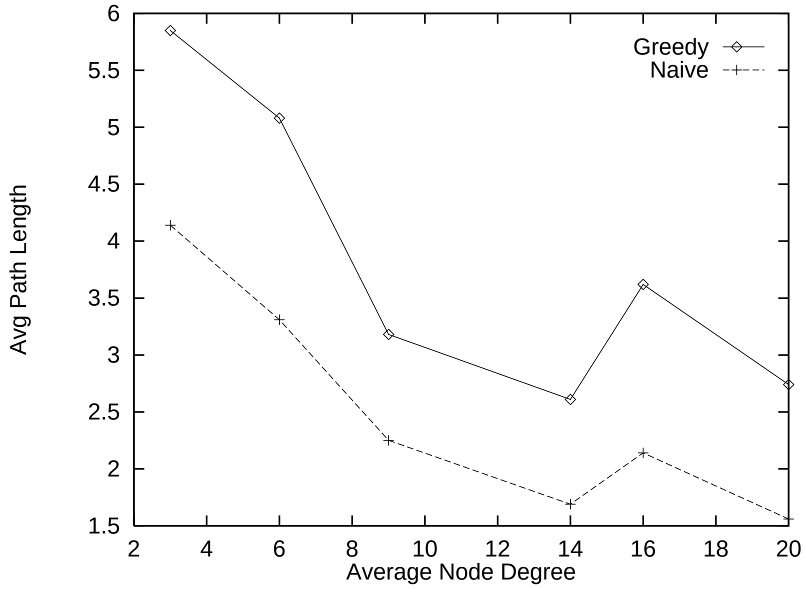
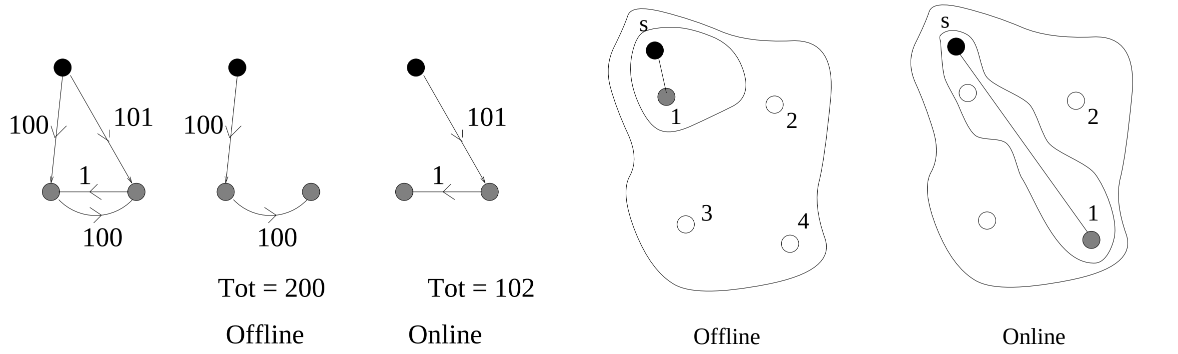


Figure A.28: Path Length versus average node degree in the 32-Euclidean map for the join-leave problem with asymmetry 10.

we expect that the relative difference was higher there than in the join-leave problem.

Appendix B

Why the On-line Greedy Solution is Able to Exploit Asymmetry



(a) Neighbors can benefit from on-line joins. (b) In on-line, neighborhoods are spread widely.

Figure B.1: On-line can be better than offline.

In figure 6.31 of section 6.4, as the asymmetry increases the relative performance of the on-line Greedy (QoSMIC) improves compared to off-line Greedy. Actually, in some cases the on-line Greedyperforms better than the off-line Greedy, and, although the difference is small ($\sim 2\%$), this is a counter intuitive result that we explain as follows. The off-line Greedy approach works very well in an undirected network, finding a minimum spanning tree. However, in an asymmetric case it can do considerably worse (fig. B.1(a)). The minimization procedure can be seen as a "game" where we want to find the low-cost directed edges that are good for the solution. The online version chooses the destinations randomly as opposed to choosing the one closest to the current distribution tree. This randomicity leads to a "widely scattered" initial tree (fig. B.1(b)). Subsequent joins are more likely to find the low-cost links, and on-line wins the game.

Bibliography

[AA95] B. Awerbuch and Y. Azar. Competitive multicast routing. *Wireless Networks*, 1:107–114, 1995.

[AA97] K. Almeroth and M. Ammar. Collection and modeling of the join/leave behavior of multicast group members in the mbone. *IEEE Communications*, June 1997.

[Awe96] B. Awerbuch. *Online Selective Multicast and Maximal Dense Trees: A survey*, 1996. Available from the author's homepage: http://www.cs.jhu.edu/ baruch/.

[Bal97] A. Ballardie. Core Based Trees (CBT version 2) multicast routing. IETF RFC 2201, 1997.

[BCFG$^+$97] T. Billhartz, J. B. Cain, E. Farey-Goudreau, D. Fieg, and S. G. Batsell. Performance and resource cost comparisons for CBT and PIM multicast routing protocols. *IEEE Journal of Selected Areas in Communications*, 15(3):304–315, April 1997.

[BFP98] A. Banerjea, M. Faloutsos, and R. Pankaj. Designing QoSMIC: a QoS Multicast Internet protoCol. Internet Draft: draft-banerjea-qosmic-00.[ps,txt], available from ftp://ftp.ietf.org/internet-drafts/, 1998.

[BKJ83] K. Bharath-Kumar and J.M. Jaffe. Routing to multiple destinations in computer networks. *IEEE Trans. on Communications*, 31:343–351, 1983.

[Bor97] A. Borodin. Personal communication. 1997.

[BV95] F. Bauer and Anujan Varma. Degree-constrained multicasting in point-to-point networks. *Proc. IEEE INFOCOM*, pages 369– 376, 1995.

[BV97] F. Bauer and A. Varma. ARIES: A rearrangable inexpensive edge-based on-line Steiner algorithm. *IEEE Journal of Selected Areas in Communications*, 15(13):382–397, April 1997.

[Cas94] S. Casner. Major MBONE routers and links. Available from ftp.isi.edu:mbone/mbone-topology.ps, 1994.

[CC97a] K. Carlberg and J. Crowcroft. Building shared trees using a one-to-many joining mechanism. *ACM Computer Communication Review*, pages 5–11, January 1997.

[CC97b] K. Carlberg and J. Crowcroft. Yet another multicast (YAM) routing protocol: Specification version 1. Unpublished manuscript, 1997.

[DC90] S. E. Deering and D. R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8:85–110, May 1990.

[DDC97] C. Diot, W. Dabbous, and J. Crowcroft. Multipoint communications: A survey of protocols, functions, and mechanisms. *IEEE Journal of Selected Areas in Communications*, 15(13):277–290, April 1997.

[Dee89] S. Deering. Host extensions for IP multicasting. IETF RFC 1112, available from ftp://ftp.ietf.org/internet-drafts/, 1989.

[Dij59] E.W. Dijkstra. A note on two problems in connection with graphs. *Numer. Math.*, 1:269–271, 1959.

[DL93] M. Doar and I. Leslie. How bad is naive multicast routing? *Proc. IEEE INFOCOM*, pages 82–89, 1993.

[DW72] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.

[EFH+97] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, F. Sharma, and L. Wei. Protocol independent multicast-sparse mode (PIM-SM): Protocol specification. IETF RFC 2117, available from ftp://ftp.ietf.org/internet-drafts/, 1997.

[EHKT97] D. Estrin, M. Handley, S. Kumar, and D. Thaler. The Multicast Address Set Claim protocol. Internet-Draft:draft-ietf-idmr-masc-00.txt, available from ftp://ftp.ietf.org/internet-drafts/, 1997.

[Eri94] H. Eriksson. MBone: The multicast backbone. *ACM Communications*, 37(8):54–60, 1994.

[Fen97] W. Fenner. Domain wide multicast group membership reports. Distributed in the IDMR mailing list. To appear as a Working Draft, 1997.

[FK97] M. Faloutsos and I. Katzela. Multicast routing algorithms for real-time dynamic applications on ATM networks. *IEEE 9th Workshop on Local and Metropolitan Area Networks (LANMAN'98)*, Banff, Alberta, Canada, May, 1997.

[FPS96] M. Faloutsos, R. Pankaj, and K. C. Sevcik. Multicasting in directed graphs. *18th Biennial Symposium on Communications, June 2-5*, pages 29–32, 1996.

[FPS97a] M. Faloutsos, R. Pankaj, and K. C. Sevcik. Bounds for the on-line multicast problem in directed graphs. *Proceedings of 4th International Colloquium on Structural Information and Communication Complexity (SIROCCO '97), Monte Verita', Ascona, Switzerland July 24-26*, pages 81–98, 1997.

[FPS97b] M. Faloutsos, R. Pankaj, and K. C. Sevcik. Multicast routing with heterogeneous quality. *The Fourth IEEE Workshop on the Architecture and Implementation of High Performance Communication Systems (HPCS'97), June*, 1997.

[GHP98] A. Goel, M. R. Henzinger, and S. Plotkin. Online throughput-competitive algorithm for multicast routing and admission control. *ACM/SIAM Proc. of Symposium on Discrete Algorithms (SODA)*, (to appear), 1998.

[GKR95] R. Gawlick, C. Kalmanec, and K.G. Ramakrishnan. On-line routing for permanent virtual circuits. *Proc. IEEE INFOCOM*, pages 278–288, 1995.

[Han97] M. Handley. An examination of the MBone performance. Unpublished manuscript, 1997.

[HJ95] M. Handley and V. Jacobson. SDP: Session description protocol. Internet Draft. Work in progress., 1995.

[IET98] Internet Engineering Task Force IETF. 41st IETF meeting. Los Angeles, March 30 - April 1, 1998.

[IW91] M. Imase and B.M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4:369–384, 1991.

[KK95] J. Kadirire and G. Knight. Comparison of dynamic multicast routing algorithms for wide-area packet switched (ATM) networks. *Proc. IEEE INFOCOM*, pages 212–219, 1995.

[KP98] S. Keshav and S. Paul. Centralised multicasting. unpublished manuscript, 1998.

[KPP93] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transaction on Networking*, pages 286–292, June 1993.

[Max97] N. F. Maxemchuck. Video distribution on multicast networks. *IEEE Journal of Selected Areas in Communications*, 15(13):357–372, April 1997.

[MJV96] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. *ACM SIGCOMM*, pages 117–130, August 1996.

[MS97] T. Maufer and C. Semeria. Introduction to IP multicast routing. Internet-Draft: draft-ietf-mboned-intro-multicast-02.txt, available from ftp://ftp.ietf.org/internet-drafts/, 1997.

[PGLA97] M. Parsa and J. J. Garcia-Luna-Aceves. A protocol for scalable loop-free multicast routing. *IEEE Journal of Selected Areas in Communications*, 15(13):316–331, April 1997.

[Plo95] S. Plotkin. Competitive routing in ATM networks. *IEEE Journal on Selected Areas in Communications*, pages 1128–1136, August 1995. Special issue on Advances in the Fundamentals of Networking. (Invited paper).

[Ram96] S. Ramanathan. An algorithm for multicast tree generation in networks with asymmetric links. *Proc. IEEE INFOCOM '96*, pages 337–344, 1996.

[SGLA97] C. Sheilds and J. J. Garcia-Luna-Aceves. The Ordered Core Based Tree protocol. *Proc. IEEE INFOCOM*, Kobe, Japan, April 7-11 1997.

[SGLA98] C. Sheilds and J. J. Garcia-Luna-Aceves. The HIP protocol for hierachical multicast routing. *Proc. ACM Symposium on Principles of Distributed Computing(PODC)*, Puerto Vallarta, Mexico, June 28-July 2 1998.

[Sha92] N. Shacham. Multipoint communication by hierarchically encoded data. *Proc. IEEE INFOCOM*, pages 2107–2114, 1992.

[SM94] N. Shacham and James S. Meditch. An algorithm for optimal multicast of multimedia streams. *Proc. IEEE INFOCOM*, pages 856–864, 1994.

[SRV97] H. F. Salama, D. S. Reeves, and Y. Viniotis. Evaluation of multicast routing algorithms for real-time communication on high-speed networks. *IEEE Journal of Selected Areas in Communications*, 15(13):332–345, April 1997.

[TEM97] D. Thaler, D. Estrin, and D. Meyer. Grand Unified Multicast (GUM): Protocol specification. Internet-Draft: draft-ietf-idmr-gum-00.txt, available from ftp://ftp.ietf.org/internet-drafts/, 1997.

[Tha98] D. Thaler. Personal communication. 1998.

[TM80] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Japonica*, 24(6):573–577, 1980.

[Wax88] B. M. Waxman. Routing of multipoint connections. *IEEE Journal of Selected Areas in Communications*, pages 1617–1622, 1988.

[Wax93] B. M. Waxman. Performance evaluation of multipoint routing algorithms. *Proc. IEEE INFOCOM*, pages 980–986, 1993.

[WE94] L. Wei and D. Estrin. The trade-offs of multicast trees and algorithms. *International Conference on Computer Communications and Networks*, 1994.

[Won84] R. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984.

[WY95] J. Westbrook and D. Yan. Greedy algorithms for the on-line Steiner tree and generalized Steiner problems. *Mathematical Systems Theory*, 28:451–468, 1995.

[ZCD97] E. W. Zegura, K. L. Calvert, and M. J. Donahoo. A quantitative comparison of graph-based models for internetworks. *IEEE/ACM Transactions on Networking*, 5(6):770–783, December 1997. http://www.cc.gatech.edu/projects/gtitm/.

[ZDE+93] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource reservation protocol. *IEEE Network*, September 1993.

[ZES97] D. Zappala, D. Estrin, and S. Shenker. Alternate path routing and pinning for interdomain multicast routing. Technical Report USC CS TR 97-655, U. of South California, 1997.