University of Alberta

An Active QoS Multicast Routing Protocol

by

Yuxi Li

©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Spring 2002

0-612-69729-0

University of Alberta

Library Release Form

Name of Author: Yuxi Li

Title of Thesis: An Active QoS Multicast Routing Protocol

Degree: Master of Science

Year this Degree Granted: 2002

Yuxi Li

Yuxi Li
Department of Computing Science
232 Athabasca Hall
University of Alberta
Edmonton, AB
Canada, T6G 2E8

Date: Jan, 2, 2002

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **An Active QoS Multicast Routing Protocol** submitted by Yuxi Li in partial fulfillment of the requirements for the degree of **Master of Science**.

[illegible]

Dr. Janelle Harms

[illegible]

Dr. Pawel Gburzynski

[illegible]

Dr. Witold A. Krzymien

Date: Dec 18, 2001

To my parents.

Abstract

QoS multicast routing plays a critical role in supporting multicast applications with QoS requirements. In this thesis, we present an active QoS multicast routing protocol using distributed agents(AQMR). AQMR introduces several new ideas for QoS multicast routing, which gives it the features of QoS awareness, scalability, responsiveness and efficiency. Agents store QoS information of the multicast tree in a scalable and flexible manner to facilitate member joins. AQMR employs an admission control scheme that enables it to better meet QoS requirements. It achieves low join overhead and low join latency. In addition, AQMR constructs a resource-efficient multicast tree. A performance study shows that AQMR can provide better QoS for receivers than QoSMIC.

Acknowledgements

Acknowledgments provide me a good chance to express my gratitude to so many people, although the gratitude may be inexpressible.

I am so grateful to my supervisor, Janelle Harms, for her invaluable guidance and support throughout the research project. I would like to say heartily she is so nice. I would like to thank the members of my examining committee, Dr. Pawel Gburzynski and Dr. Witold A. Krzymien for their precious time spent on reviewing this thesis. Thanks also go to professors and my fellow graduate students in our Communication Networks Research Group, e.g. Mike MacGregor, Ioanis Nikolaidis, Ehab Elmallah, Baochun Bai, Yanxia Jia, Chong Zhang, Kui Wu, Xudong Wu, Hongjun Zhang, Weiguang Shi, Xiaoqin Yuan, Juhua Shi, Xiang Wan, Qiang Ye, Long Li, Ioannis Batsiolas and more. I owe much to Baochun who has been working with me in the CITR project, discussing multicast, Active Networks, NS2, and other stuffs. So many people in our department, in our university, in the world who helped, and will help me, deserve my "thank you" from the bottom of my heart. One thing I could not omit—the fun I have with my friends when we play badminton. Also much fun to have parties!

I would especially like to thank my family for the encouragement they provided during the course of my studies.

Thanks to you all!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multicast is an efficient way to distribute data packets among group members. Instead of sending separate copies of a data packet to each of several recipients as a unicast protocol like TCP(the Internet Transmission Control Protocol) would do, a source in a multicast group sends out only one copy with the group address as the destination address. A multicast routing algorithm determines a multicast tree that connects the sources and receivers in a group according to a certain optimization goal. A multicast router decides where to forward the packet using the multicast routing table, replicating the packet only if necessary. As a consequence, a multicast transmission consumes network resources more efficiently than multiple TCP connections.

1.1 Motivation

Emerging multicast applications, such as teleconference and video distribution, have various requirements of quality of service(QoS) in terms of bandwidth, end-to-end delay, loss rate and so on[QU98]. Existing multicast routing protocols, such as CBT(Core Based Tree)[BA97] and PIM-SM(Protocol Independent Multicast-Sparse Mode)[EF98] may not provide the required QoS for a joining member. A recent study on MBone[AL00] indicates that "Poor Session Quality" is one of the problems that need to be tackled so that multicast can be widely deployed. A QoS multicast routing protocol is essential in supporting QoS requirements of applications. We need some mechanism to examine a member's join request before making an acceptance decision for the request, so that the multicast group can provide the QoS that the new member requires. The eligibility test requires certain QoS information of the multicast tree, which calls for a scalable and efficient approach to obtain the QoS information.

1.2 Design Objectives and Contribution

We design a QoS-aware multicast routing protocol in the context of Active Networks[CA99]. Active Networks can provide more flexibility to applications, by allowing programs to be executed by routers in the middle of the network. Furthermore, active routers enable quick deployment of routing protocols [ML01]. Our protocol can also be implemented in the current Internet, with limited support from routers.

We present our multicast routing protocol, Active QoS Multicast Routing Protocol(AQMR), in this thesis. In AQMR, *agents* are active nodes, which are strategically located in the multicast tree. An agent serves an area of the tree to gather QoS information from sources in its area, to exchange QoS information with other agents and to facilitate receivers' joins. After exchanging information with each other, the agents have a view of the entire multicast tree. A new router contacts the agent associated with it to join the tree. The QoS information from a receiver to an agent is also collected when the receiver requests to join the tree. AQMR is designed to achieve the following objectives.

- *QoS awareness.* By storing QoS information in agents, AQMR can conduct eligibility tests when a join request arrives. The architecture of AQMR can support various QoS metrics, including end-to-end delay, loss rate, and bandwidth. We are concerned about the QoS that receivers experience after being approved to join the multicast tree. We believe that how well a QoS routing scheme can support the required service is more important than how many receivers can be accepted to join the tree.

- *Scalability.* An agent represents an area of a multicast group. It gathers QoS information from sources in its area, and exchanges the information with other agents. A receiver sends the join request to its agent, which makes the join decision for the receiver. Agents, but not all the on-tree routers as in some previous work, need to store QoS information and to test the eligibility of join requests. We can choose agents according to the nature of the multicast group, so that we can make AQMR scalable for various groups in terms of QoS information gathering and storage and join decision making.

- *Responsiveness.* A receiver can expect a join ACK(or NACK) from the agent within a short period of time, since the receiver contacts the agent associated with it to join the multicast tree, and the agent makes the decision immediately.

- *Efficiency.* With agents as special points in constructing the tree, AQMR builds multicast trees with low cost, in terms of average delay and average hop number of the tree.

A performance study(Chapter 4) shows that AQMR can provide a better service than QoSMIC, a representative multicast routing protocol that was designed to support QoS requirements. AQMR can construct resource-efficient trees. It has low join overhead and low join latency. The performance study will confirm that we can achieve the design objectives.

1.3 Thesis Organization

The thesis is organized as follows. Previous work and background information are presented in the next chapter. Detailed discussions of the proposed QoS-aware multicast routing protocol AQMR come in Chapter 3. After that, a performance study is presented in Chapter 4. In Chapter 5, the conclusions and future work are discussed.

Chapter 2

Previous Work and Background

AQMR is a QoS-aware multicast routing protocol designed in the context of Active Networks. In the following, we classify the multicast routing schemes, and give a brief description of the representative related protocols. After that, we will introduce the platform of Active Networks, their characteristics and applications, and also Active Services.

2.1 Multicast Routing

Multicasting is the distribution of packets from a node to many recipients simultaneously. A multicast routing protocol builds a multicast tree that connects the sender(s) and the receiver(s). A multicast sender does not need to know where the receivers are. It uses the group address as the destination address, and the multicast-capable routers can forward the packets appropriately(possibly replicating them onto more than one outgoing links), so that the packets can finally reach the receivers. In this thesis we assume that a router is multicast-capable.

In this section, we will first discuss a multicast tree in terms of its structure, shared-tree or source-tree. Then we present schemes for a routing protocol to find on-tree routers. After that, we discuss protocols with respect to whether they support QoS or not, and introduce several protocols. For a comprehensive review on multicast routing protocols, refer to [WH00], [RA00], and [AL00-2].

2.1.1 Shared-tree versus Source-tree

Two types of multicast trees may be constructed, shared-tree and source-tree. A shared multicast tree is rooted at a center point, termed as the core in CBT(Core Based Tree)[BA97] or the RP(rendezvous point) in PIM-SM(Protocol Independent Multicast Sparse Mode). We call the center point as core here. In a shared-tree, all the sources use the core to transmit data packets to the group. A new member contacts the core to join the tree. In a source tree, the source is the root of the multicast tree for the source. That is, there will be a tree for each source in the group. In a shared-tree, a router needs only one entry for each group. However, the shared-tree may not be as efficient as a source tree since packets will be forwarded to the core. There may be traffic concentration problems at the core[RA00]. A source-tree may be more efficient than a share-tree. However, there is a routing entry for each source in a router, which may cause a scalability problem when the number of sources is large. In the following, we discuss two representative protocols, CBT and PIM-SM.

In CBT[BA97], the core is the center of the multicast group. When a new member wants to join the group, it sends a join request to the core following the reverse shortest path. Intermediate routers process the join request and set up transient routing state. When an on-tree router or the core receives the request, an acknowledgment message is sent back to the receiver on the reverse path the request followed. At each router, the incoming and outgoing interfaces of the join request packet are added to the routing table entry for this multicast group address. CBT builds a bi-directional tree, whose routing entry does not differentiate interfaces as incoming or outgoing, rather a data packet is forwarded to tree interfaces other than the incoming one. Consequently sometimes a data packet does not need to reach the core first before being forwarded to receivers.

PIM-SM[EF98] uses the center point called the rendezvous point(RP) to build a shared-tree by default. A new receiver contacts the RP of the group to

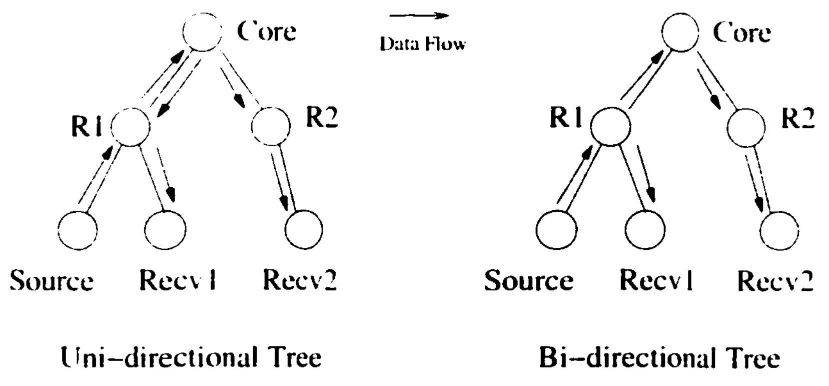


Figure 2.1: Uni-directional Tree vs. Bi-directional Tree

join the tree. The join request is forwarded to the RP using the reverse shortest path. Routers on the way update their routing tables. PIM-SM constructs a uni-directional tree routed at the RP. A source sends data packets to the RP first. The RP then forwards the packets to all the receivers. A router can switch from a shared-tree to a source-tree if it finds the transmission on the shared-tree is not satisfactory.

Figure 2.1 basically illustrates the difference between a uni-directional tree and a bi-directional tree. In the uni-directional tree, the route for a data packet from *Source* to *Recv*1 is *Source* $\rightarrow R1 \rightarrow$ *Core* $\rightarrow R1 \rightarrow$ *Recv*1. In contrast, the route in a bi-directional tree is *Source* $\rightarrow R1 \rightarrow$ *Recv*1.

2.1.2 Finding On-tree Router(s)

In a multicast routing scheme, the router directly adjacent to the new member needs to search for a path for the member to join the group. The searching schemes could be classified into three categories: distributed, centralized, and hybrid. In a distributed approach, routers make decisions based on local information. In a centralized approach, a special router(or routers) knows global tree information so that it can choose or help choose a path(or paths) to the new member. The new member's router sends the request to this special router. The hybrid approach is a combination of a distributed and a centralized methods.

YAM(Yet Another Multicast Routing Protocol)[CC98], QMRP(A QoS-aware Multicast Routing Protocol)[CX01] and QoSCBT(QoS Extension to CBT)[HT99] fall into the distributed category. Some schemes such as PIM-SM and CBT use reverse shortest path to connect to an on-tree router. They can be regarded as distributed approaches. DVMRP(Distance Vector Multicast Routing Protocol)[WP88] and PIM-DM(Protocol Independent Multicast Dense Mode)[DE99] use a "flood and prune" approach, where data packets will be periodically forwarded across the entire network, and a leaf router will prune the branch to the source if there is no member attached to it. DVMRP and PIM-DM can be regarded as distributed approaches. MOSPF(Multicast extension to OSPF)[MO94] uses Dijkstra's shortest path algorithm as the routing algorithm, which may be regarded as a centralized approach, although the information is gathered in a distributed way. Those algorithms that need global topology information such as the proposed protocols in [PZ98](BSMA) and in [KP93](KPP) are centralized approaches. In protocols like BSMA or KPP, they assume the algorithm knows the property of the graph that represents the topology. That is, they know information of each link in the topology. QoSMIC(Quality of Service sensitive Multicast Internet protoCol)[FB98] follows a hybrid approach. Our protocol, AQMR, can also be classified as a hybrid approach, which deploys distributed agents to store centralized multicast tree information to facilitate new members' joins. In AQMR, an agent has a complete view of the multicast tree. However, it knows only the cumulative metric(s) to sources and other agents. Table 2.2 shows the classification of multicast routing schemes according to how they find on-tree routers.

Distributed	Centralized	Hybrid
QMRP, QoSCBT, YAM, CBT, PIM-SM, PIM-DM, DVMRP,etc.	BSMA, KPP, MOSPF, etc.	QoSMIC, AQMR etc.

Table 2.1: Routing Scheme Classification(I)

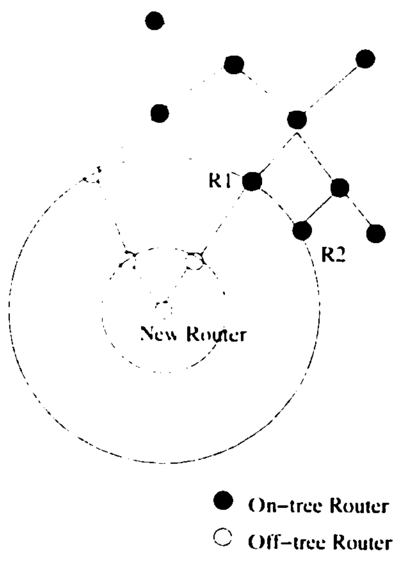


Figure 2.2: Expanding Ring Search

Expanding Ring Search

Without global topology knowledge, some schemes such as YAM and QoS-MIC deploy the approach of Expanding Ring Search(ERS) to locate on-tree router(s) to attempt to find a feasible path with good QoS. As shown in Figure 2.2, the new router sends out search packets via reverse path broadcast [CC97][DC90], with the TTL(Time To Live) increasing by 1 hop each time, until an on-tree router is found. That is, initially it will send out a search packet with $TTL = 1$. After the timer associated with the searching procedure expires, it will send out a search packet with $TTL = 2$, etc. An off-tree router will forward the search packet if the packet comes via the shortest path from itself to the new router that initiates the searching procedure, and if the TTL of the search packet is greater than 0. An on-tree router that receives the search message will notify the new router of its being on the tree, and stop forwarding the search packet. There may be more than one on-tree routers found. In the figure, routers $R1$ and $R2$ are found by the new router when $TTL = 2$. After locating on-tree routers, a routing scheme can choose one of the found on-tree routers to connect the new member to the multicast tree.

2.1.3 QoS Support

Multicast routing schemes can also be categorized as QoS-aware and QoS-oblivious. A QoS-aware scheme searches for a path that can meet the QoS requirement, or the path that is the best among the found paths with respect to the QoS requirement. A QoS-oblivious scheme does not take the QoS requirement into consideration. Most standardized or standard-track schemes are QoS-oblivious, such as MOSPF, DVMRP, PIM, CBT, etc. YAM, QoSMIC, QMRP, AQMR, and so on can be categorized as QoS-aware schemes. Table 2.2 shows the classification of multicast routing schemes according to whether they support QoS.

QoS-oblivious	QoS-aware
MOSPF, DVMRP, PIM-DM PIM-SM, CBT, etc.	QMRP, QoSCBT, YAM, QoSMIC, AQMR, etc.

Table 2.2: Routing scheme classification(II)

In the following, we give brief descriptions of some representative QoS-aware protocols including YAM, QOSMIC, QoSCBT, QMRP, etc.

YAM

YAM [CC97, CC98, CC99] uses the one-to-many join approach(ERS) to search for on-tree routers. On-tree routers reply to the new router; and during the process, the properties of the paths such as the delay are collected. The new router selects the best one to connect to the tree.

QoSMIC

In [FB98], Michalis Faloutsos, Anindo Banerjea and Rajesh Pankaj proposed QoSMIC(Quality of Service sensitive Multicast Internet protoCol). QoSMIC deploys *Local Search* and *Multicast Tree Search* together to find multiple candidate paths so that the new router can select the best one to connect to the tree. There are timers associated with *Local Search* and *Multicast Tree Search*, which determine how long the new router would wait for feedback before de-

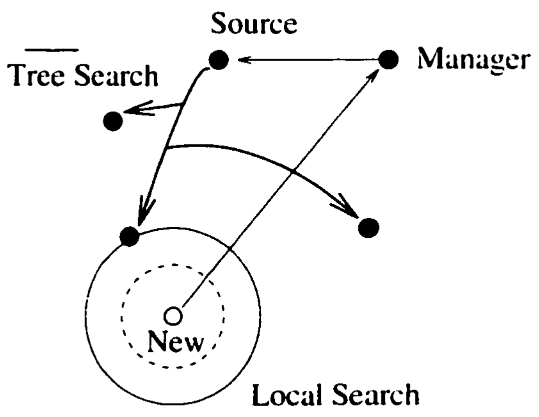


Figure 2.3: The Local Search and the Multicast Tree Search in QoSMIC[FB98]

ciding that the corresponding search has failed. QoSMIC terms a qualified on-tree router as a candidate.

Local Search uses the approach of ERS to locate candidates. *Local Search* may fail to locate an on-tree router, due to too small TTL or too short search timer. If a member fails to find a candidate during the stage of *Local Search*, it notifies the *manager*, a central administrative point in QoSMIC, to initiate the process of *Multicast Tree Search*. During the stage of *Multicast Tree Search*, candidates can be selected using *Centralized Selection* or *Distributed Selection*. In the former one, the *manager* has sufficient knowledge of the multicast group and can choose the proper candidates directly. This may be impractical since it needs global knowledge of the tree. In the latter case, three mechanisms are proposed to lower the protocol overhead, *Directivity*, *Local Minima*, and *Fractional Choice*. *Directivity* does not select distant on-tree routers. *Local Minima* selects locally optimal routers as candidates, given the absence of global knowledge. *Fractional Choice* chooses as candidates a representative fraction of all on-tree routers or those that satisfy the criteria of *Directivity* and *Local Minima*. Figure 2.3[FB98] illustrates the basic idea of *Local Search* and *Multicast Tree Search* in QoSMIC.

QoSMIC can collect QoS information from the found on-tree routers to the new router, however, it does not provide an approach to obtain dynamic information from a source(or the core) to the new router. Both QoSMIC and YAM use the best path to connect a receiver to the tree, which may not satisfy the QoS requirement of the request. In [YF99], more experiments are conducted to study QoSMIC.

QoSCBT

In QoSCBT[HT99] a QoS eligibility test is conducted at every router. If the request survives the test, it will be forwarded to the next router towards the core. Otherwise, the request is denied. The ratio of joining denial may be high since there is only one path examined which is derived from the best-effort unicast routing table. In [TH99], Hung-Ying Tyan, Chao-Ju Hou, and Bin Wang present a set of member join/leave and state maintenance procedures for core-based QoS multicast routing. They deploy approaches of *Local Search with Bidding* and *Sequential Random Search* to locate a path to an on-tree router. *Local Search with Bidding* is similar to the *Local Search* in QoSMIC, with the exception that in QoSCBT, a router that receives the search message needs to conduct an eligibility test. With *Sequential Random Search*, an off-tree router that receives a rejection message attempts to find a feasible path through another outgoing interface, rather than forwarding back the rejection message to the requesting router. The router will send back a rejection message once it has tried all the possible interfaces and failed to find a path. The two approaches can eliminate the problem in [HT99] where only one path is examined. In both studies, every router is able to conduct the eligibility test, so all of them need to store QoS information for the multicast tree.

QMRP

Shigang Chen, Klara Nahrstedt, and Yuval Shavitt proposed QMRP(A QoS-aware Multicast Routing Protocol) in [CN01]. In QMRP, a joining request follows the unicast routing path towards the core. If a router can not support the QoS requirement that the new member desires, the router asks the previous router to conduct *multiple path routing*, in which join request messages will be sent to interfaces other than the one defined by the unicast routing. The join latency of this approach may be high in the case that several *multiple path routing* searches have to be conducted. An example is, the join request will be forwarded following the router sequence of R_1, R_2, ... R_n. These routers could support the required QoS(thus all of them can survive the eligibility test), however only R_1 connects to some feasible path according to the QoS required, suppose this feasible path does not follow the reverse path that the request packet will travel. A NACK of the request will finally be sent back to R_1 after every other router has tried the *multiple path routing*, but all of them fail. In QMRP, every router should have the ability to check whether a request could be accepted. QMRP works for *non-additive* metrics such as bandwidth and buffer space, but does not support applications that require metrics such as end-to-end delay or loss rate.

In [CS00], Shigang Chen et al. claim that the protocol S-QMRP supports additive metrics such as end-to-end delay. In S-QMRP, a joining member starts with a single path routing as in CBT to connect to the tree. The join request packet accumulates the delay of the path it traverses. When it meets an on-tree router, the router can check whether the request can be accepted, using the accumulated delay and the delay information from the root to it. If the on-tree router can not accept the request, the request will be forwarded to the root. The root then uses *Grow* messages to search from multiple paths for a feasible path to the receiver. During the *Grow* process, routers that forward the *Grow* message can learn the delay from the root to it. S-QMRP still needs all routers to store such information. In addition, the delay information may not be up-to-date[1], and the receiver must wait for a long time until the path is found.

BSMA and KPP

There are also algorithms that assume some node(the source) knows global knowledge of the network(such as the topology and QoS metrics on links) and

[1]AQMR has mechanisms for agents to update QoS information periodically and on a triggered basis, so the information stored at agents is almost up-to-date.

try to find an optimal multicast tree for the source and the receivers. We only briefly discuss two papers of this direction. In [PZ98], the algorithm BSMA allows arbitrary real-valued destination delay bounds by employing an iterative approach to construct the multicast tree. It can accommodate asymmetry of links. The algorithm KPP in [KP93] deals with only constant integer-valued destination delay bounds, and assumes the link costs and delays are symmetric.

2.2 Active Networks

In this section, we briefly discuss two representative Active Networks platforms, ANTS[WG98] and PLAN[HK98], as well as a competing research direction, Active Services[AM98]. We also present some research work that applies the concept of Active Networks to applications such as reliable multicast and multicast congestion control.

Active Networks[CA99] enables routers to employ new services dynamically at run time, so that new protocols can expect fast deployment. It can provide more flexibility to applications by allowing programs to be executed by routers in the middle of the network. Active routers enable quick deployment of routing protocols [ML01]. At the same time, it attempts to guarantee the network security and to avoid compromising network performance such as delay. A. Campbell et al. in [CA99] review a number of Active Networks projects according to programmable network characteristics: networking technology, level of programability, programmable communications abstraction and architectural domain.

ANTS [WG98] is an Active Networks approach for dynamically building and deploying network protocols, based on techniques of mobile code, demand loading and caching. In the architecture of ANTS, a capsule, a replacement for a packet, contains a program and is processed at active nodes with restricted access to node resources. An active node supports primitives such as environment access, capsule manipulation, control operations and node storage to express forwarding routines. Examples of programming with ANTS for mobile IP and multicast are presented in [WG98] to illustrate the ease of building new protocols.

PLAN [HK98] enables programs to form packets of a programmable network by replacing the currently used packet header. PLAN code can call service routines residing at a node to implement required functionalities. PLANet [HM99] is an active network architecture implemented with PLAN.

In [AM98], the Active Services framework is proposed to address the difficulties of deployment of Active Networks technology. The Active Services framework supports application level programability while keeping compatible with current Internet infrastructure so that it can be deployed incrementally.

Active Networks have the potential to become the network computing platform in the future. In [ML01], N. F. Maxemchuk and S. Low discuss applications such as QoS routing, mobility, and receiver-controlled routing as good candidates for Active Networks deployment. Previous work such as ARM(Active Reliable Multicast)[LG98], AERM(Active Error Recovery For Reliable Multicast)[BH01-1] and AHCCM(Active Hierarchical Congestion Control for Large-scale Multicast)[BH01-2] have shown that some applications, e.g. reliable multicast and multicast congestion control, can benefit from introducing some functions within the network, based on the architecture of Active Networks. ARM[LG98] deploys active routers to suppress duplicate NACKs, to perform best-effort cache of recent packets, and to deliver retransmitted packets to receivers experiencing data loss. AERM[BH01-1] improves previous active error recovery schemes(including ARM) by using soft-state storage in active routers. In AHCCM[BH01-2], a multicast flow can share the bandwidth adjustably when competing with TCP flows. It can also achieve inter-receiver fairness, scalability and responsiveness [BH01-2].

2.3 Summary

Multicast is an efficient way to distribute data packets among group members. A multicast routing protocol builds the distribution multicast tree among group members. A multicast tree can be a shared tree or a source tree. Multicast routing schemes can be classified into distributed, centralized and hybrid schemes in terms of how to find a path for a new member to join the tree. They can also be categorized as QoS-oblivious and QoS-aware schemes, depending on whether they support QoS or not. There are challenges to provide QoS in multicasting, such as how to gather QoS information to make proper decision for join requests thus to provide a satisfactory QoS support and how to lower the protocol overhead for joining the tree. Active Networks will potentially be a promising computation platform that will allow flexible deployment of multicast protocols. We discussed the implementation and application of Active Networks.

Chapter 3

Active QoS Multicast Routing Protocol(AQMR)

In this chapter, we describe our QoS multicast routing protocol in detail. We give an overview of our protocol first. Then we discuss agent selection and path searching. In the section of admission control, we describe mechanisms for QoS information gathering and eligibility tests for join requests. After that, we present the group dynamics of our protocol, which includes data packet flow, receiver join and leave, routing state maintenance, and source join.

3.1 Protocol Overview

In this section, we give an overview of our active QoS multicast routing protocol using distributed agents(AQMR). In AQMR, a source or a receiver is associated with an agent close to it. A source sends QoS information collection packets periodically to the agent associated with it. Agents exchange source QoS information among themselves periodically and also on a triggered basis to obtain a complete view of the multicast tree. After gathering QoS information of the tree, an agent has sufficient information to make admission decisions when new members request to join the group. A new member contacts the agent associated with it to join the tree. Using distributed agents, we aim at satisfying QoS requirements of receivers and expediting the QoS joining decision. We also attempt to make QoS information gathering and storage efficient and scalable.

There are four types of routers in AQMR: agent, source, receiver, and on-tree router. Agents are responsible for gathering QoS information of the multicast tree and for making join decisions for new members. They also play a critical role in path setup. Sources are the points where data packets originate. Sources also need to periodically send QoS information collection packets to agents. Receivers initiate join requests, and are the recipients of data packets. On-tree routers are routers that are on the multicast tree. They simply forward data packets. These router types are summarized in Table 3.1.

Router Type	Function
Agent	Gathering QoS information of the multicast tree. Conducting eligibility test to accept or reject join requests. Being involved in path setup.
Source	Originating data packets. Sending QoS information collection packets to an agent.
Receiver	Requesting to join the multicast tree. Receiving data packets.
On-tree Router	Forwarding data packets to receivers. They are part of the data distribution tree.

Table 3.1: Router Types

When a new router wants to join the multicast tree, it contacts its agent. The agent has already gathered QoS information of the multicast tree, so that it can conduct an eligibility test for the new request as soon as the request arrives. If the request is approved, the agent sends an acknowledgment message back to the new router. At the same time, the agent may also need to set up paths to other agents and then to sources(details of this are discussed later in Section 3.5). After the receiver receives the acknowledgment, it joins the tree, so that routers along the path from the receiver to its agent change their states to forward packets to the new receiver. If this agent can not accept the request based on the QoS information, it sends a NACK to the new router.

As shown in Figure 3.1, each agent services an area of the multicast group. That is, a source in its area sends QoS information collection packets to it; and the receivers in its area contact it directly to join the multicast tree. In

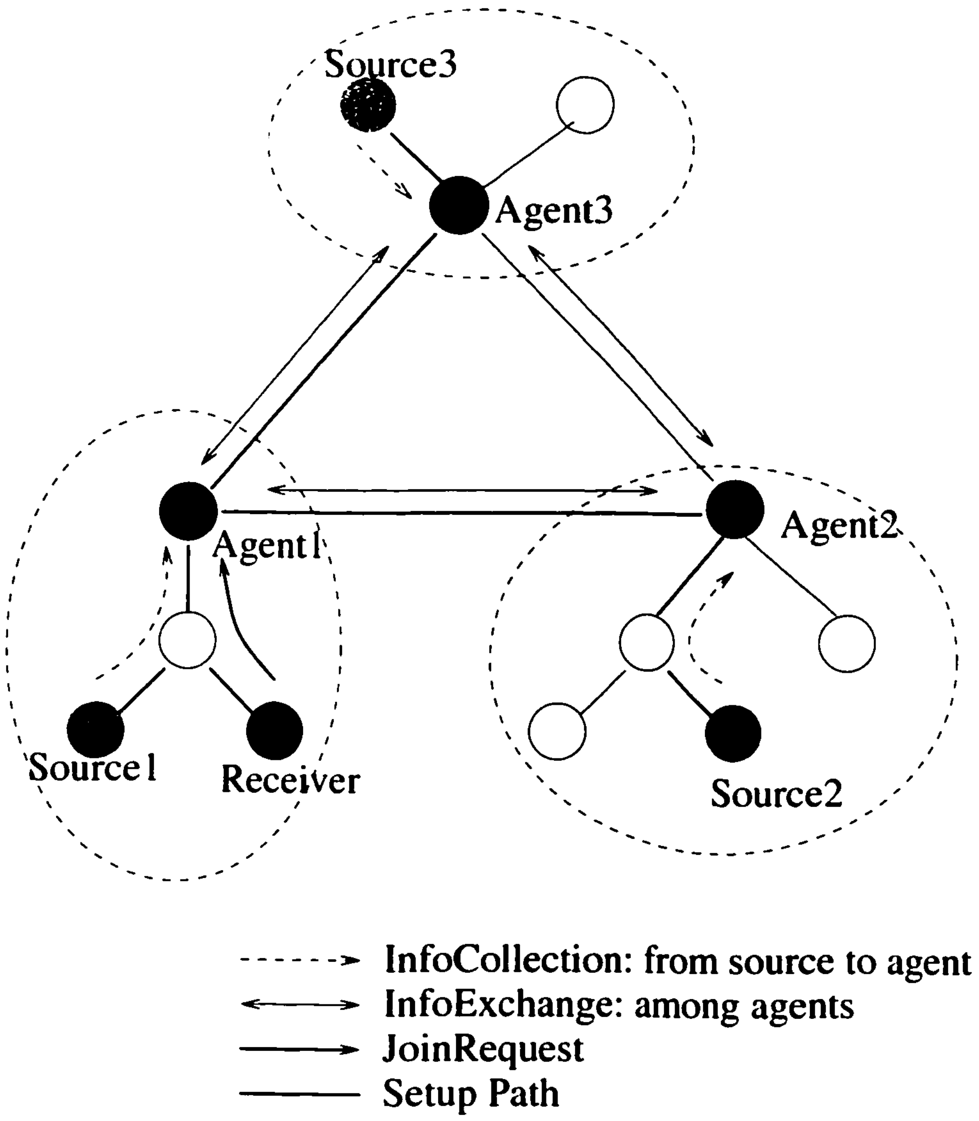


Figure 3.1: Protocol Overview

Figure 3.1, there are three agents, *Agent1*, *Agent2* and *Agent3*, and three areas associated with them. Dashed lines show the flow of information collection packets. *Source1* sends information collection packets to *Agent1*, so that *Agent1* knows QoS information to *Source1*. Similarly, *Source2* sends such packets to *Agent2*, and *Source3* to *Agent3*. These agents exchange QoS information with each other, as shown in the bi-directional arrow-ed lines, so that they can compute the QoS metrics to all the sources. When the *Receiver* wants to join the multicast tree, it sends a join request with its QoS requirement to its agent, *Agent1*. *Agent1* makes the join decision based on the information it stores locally.

In the following, we present AQMR in detail. First we discuss how to choose agents. We then present a method to find paths between two nodes that can approximately achieve the optimization goal. After that, we discuss the mechanisms in AQMR to conduct admission control, which include approaches for gathering QoS information for a multicast tree, description of parameters of intervals and triggers, and the mechanism of the eligibility test. Group dynamics are illuminated next. We explain how to set up paths after a receiver is acknowledged, how to leave a group for a receiver, how to maintain the router state, and how to join a group for a source.

3.2 Agent Selection

Strategically placed agents can make QoS information gathering efficient, accelerate join request response, lower tree cost and lower protocol overhead. The Internet[TA96] can be viewed as a collection of subnetworks that are connected together. A subnetwork is an Autonomous System(AS), which includes a backbone, and several areas that are attached to the backbone. A backbone and an attached area can also be called a transit-AS(transit-domain) and a stub-AS(stub-domain) respectively[JJ00][CD97]. In [JJ00], Sugih Jamin et al. present heuristics for Internet instrument placement to collect distance information without having global topology knowledge. We could use a mechanism

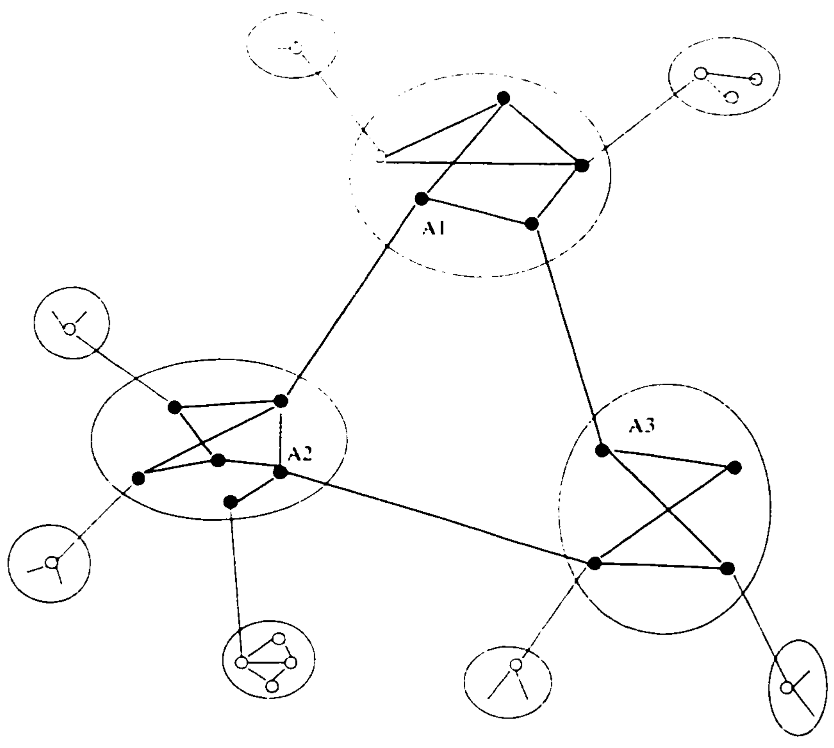


Figure 3.2: An example of Agent Selection

similar to Transit-AS[JJ00] to place agents on transit-ASes. That is, agents are on ISP(Internet Service Provider) backbones. We may also choose agents in a stub AS, if the stub AS is a big one, or any transit AS is too far away. To choose agents on both transit and stub ASes is a combination of the approaches of Transit-AS and Stub-AS in [JJ00]. An example of choosing agents is shown in Figure 3.2, in which A1, A2, A3 are selected as agents in three transit-ASes. In Section 4.5.2, we study the effect of the number of agents manually chosen from a given topology on the protocol performance. Automatic agent selection is left as future work.

For the special case of a small group, we may only require one agent. In this case, AQMR works similarly to the PIM-SM shared-tree. The difference is that in AQMR an agent knows the condition of the multicast tree, so that it can make more appropriate admission decisions for join requests.

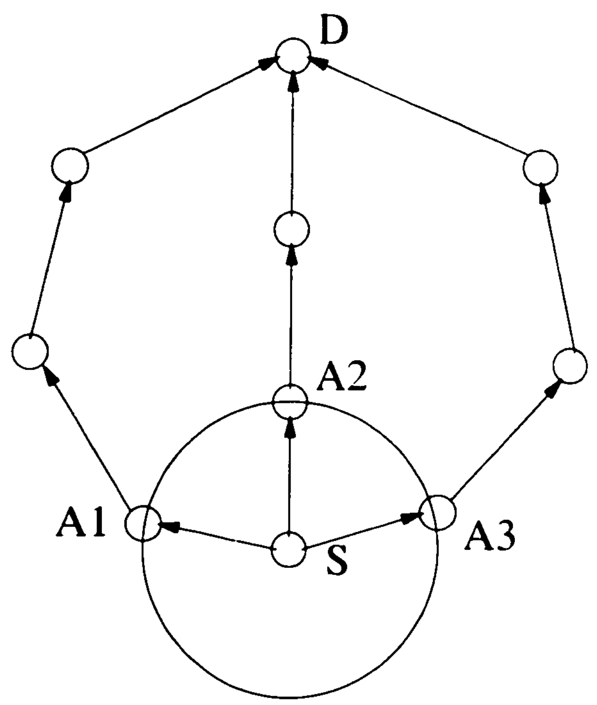


Figure 3.3: Path Searching

3.3 Path Searching

In QoS routing, the path derived using the unicast routing protocol may not be the satisfactory one in terms of the QoS required. In AQMR, it is desirable to find paths between a source and an agent, between a receiver and an agent, and between two agents that have the best QoS among all the feasible paths between the two entities respectively. It may be costly to find the optimal path between two nodes, therefore we consider a heuristic approach to try to find a good path with relatively low overhead.

We assume a node S wants to find a path to node D. Node S sends search packets to all its neighbors, A_1, A_2, ..., A_n. After these neighbors of S get the search packets, they send the packets to node D using the underlying unicast routing protocol. During the transmission of the searching packets, the QoS metrics of interest on the traversed paths are measured, so that the node D can choose the best one among all these paths. This is a one-hop ERS(Expanding

Ring Search, see Section 2.1.3). We call this scheme $1-ERS$ search. Figure 3.3 illustrates the scheme of $1-ERS$ search. We can use a $2-ERS$ search to get a better path, in which a search packet have the TTL field of 2. A node forwards the search packet to node D using the unicast routing protocol if and only if the search packet arrives via the shortest path from the node to the node S, and the TTL is 0. When the packet comes from the reverse path to node S, and its $TTL \geq 1$, the node forwards the packet on. Otherwise, the packet will be simply discarded. Similarly there could be a $3-ERS$ search, etc. Clearly $2-$ or $3-ERS$ search may find better paths, but it increases the overhead of searching for a path between two nodes, compared to $1-ERS$. To reduce the overhead, we may ask the router to record whether there has been a search packet traversing it, and, if so, simply discard those searching packets that come later.

3.4 Admission Control

In this section, we discuss the problem of deciding whether to accept or reject a join request, that is, admission control. We illustrate how to gather QoS information of the multicast tree such as path delay and how to conduct an eligibility test when an agent receives a join request.

3.4.1 QoS Information

We discuss how to gather QoS information with respect to: QoS information collection and exchange, QoS information acquisition, and the completeness of QoS information. In Table 3.2, we present the message types that are used in QoS information gathering, *InfoCollection* and *InfoExchange*. The rest of message types in Table 3.2 will be used in Section 3.5 that discusses the group dynamics.

QoS Information Collection and Exchange

QoS information is important since routing decisions are based on it. One way to gather QoS information is to design specific control messages. We

Message Type	Function
InfoCollection	A source sends *InfoCollection* packets to its agent for it to collect QoS information from the source.
InfoExchange	Agents exchange QoS information, meanwhile, QoS information between two agents is also collected.
JoinRequest	A receiver requests to join the tree. QoS information between a receiver and its agent is also collected.
JoinACK	An agent informs the receiver that its join request has been approved.
JoinNACK	A request is rejected.
PathSetUp	To set up tree paths.
KeepAlive	To refresh soft-state routing table.
Prune	For pruning unnecessary links on the tree.

Table 3.2: Protocol Message

can obtain QoS information from periodic information gathering messages. A triggered approach can be used to capture significant changes in the tree, such as available network resource changes due to background traffic changes. We keep track of only significant changes to reduce the overhead of the protocol. We can also derive QoS information from data packets.

In AQMR, we design a periodic mechanism combined with a trigger-based approach for QoS information gathering, which can update QoS information in a timely manner and catch significant changes. In this way, we can set the periodic intervals used in AQMR relatively large, and, with the help of triggers, obtain accurate QoS information and keep the overhead low. Specifically, in AQMR, a source periodically sends *InfoCollection* packets to its agent. The agent updates its source information after receiving the *InfoCollection* packets. If the agent detects a significant change of QoS information, it re-calculates the path, and informs other agents of the updated information. An agent also periodically computes paths and exchanges information with other agents. An agent can aggregate information from all the sources that send information to it, and send only one copy of the aggregate information to other agents to lower the protocol overhead. After exchanging QoS information with the other agents, and computing paths, each agent will have a complete view of the multicast tree. That is, each agent has the knowledge of the QoS of paths to all the sources, and the agents the sources are associated with respectively.

QoS Information Acquisition

QoS metrics may be obtained by accessing the information stored at routers, or by a measurement approach. This is metric-dependent. For example, the delay of a path may be measured using two time-stamps at the sender of the measurement message and at the recipient. In this case, only the sender and the recipient are involved in the delay measurement. The minimal available bandwidth of a path may be obtained by computing the minimal one of all the links on the path, providing bandwidth information is available at routers. In this case, the routers in between need to provide the bandwidth information. The packet loss rate of a path can be measured by dealing with the sequence number and acknowledgment number of TCP packets as proposed in [SA99], or by using the end-to-end multicast traffic as measurement probes as presented in [CD99].

QoS Information Completeness

We make the QoS information complete(e.g., for end-to-end delay, we have the delays from sources to a receiver) by using the following procedures. The QoS information between a source and its agent is collected when the source sends the *InfoCollection* packet to the agent. Information between agents is collected when the two agents exchange information with *InfoExchange* packets. The QoS information between an agent and a receiver is collected when a new receiver requests to join the multicast tree.

3.4.2 Path Computation

There is some work discussing how to compute paths. [GO97] discusses issues such as when(on-demand or pre-computation) and how to compute paths. An on-demand approach can generate more accurate results but it is computationally expensive: while a pre-computation approach is cost effective but needs triggering mechanisms to know when to compute paths, and each computation is costly[GO97]. [AG98] suggests a technique that combines pre-computation and triggering policies with large timers to produce an efficient and cost effective solution. Apostolopoulos et al. in [AG98] also discussed update policies to catch significant changes.

In AQMR, path computation is conducted periodically, and is also triggered by significant changes. An agent can compute paths to sources, once QoS information between sources and agents and between each pair of agents is available. For each source, the agent calculates the cumulative QoS metric from the source to itself, then it stores the path, the QoS metric and the source's corresponding agent. With such multicast tree QoS information, an agent can make admission decisions for join requests.

3.4.3 Intervals and Triggers

In this section, we discuss parameters of intervals and triggers for both QoS information gathering and path computation.

QoS Information Gathering

We design a periodic mechanism combined with a trigger-based approach for QoS information gathering as described in the above section. We need the following parameters, *SrcInterval*, *ExchangeInterval*, and *AgentTriggerThreshold*, as shown in Table 3.3. The intervals are used to decide the timer interval between two information gathering packets. *SrcInterval* decides how often a source sends an *InfoCollection* packet to its neighboring agent. *ExchangeInterval* tells each agent when to send *InfoExchange* packets to other agents. The trigger threshold is used to catch significant changes in QoS information and to lower protocol overhead. After an *InfoCollection* packet from a source arrives at an agent, the agent determines whether to trigger an *InfoExchange* message to other agents using the *AgentTriggerthreshold*.

If we denote the change of the QoS metric as M_{change}, the original value

Parameter	Description
SrcInterval	Determining how often a source sends an *Info-Collection* packet to its agent.
ExchangeInterval	Deciding how often each agent sends *InfoExchange* packets to other agents.
AgentTriggerThreshold	Determining whether an agent triggers an *InfoExchange* message to send to other agents.

Table 3.3: Parameters for Information Gathering

as $M_{original}$, the agent will send out *InfoExchange* packets to other agents to inform them of the current network condition, if

$$M_{change} \geq AgentTriggerthreshold * M_{original}.$$

When an *InfoExchange* packet is triggered, the corresponding timer will be rescheduled to avoid unnecessary periodic information gathering packets. In this way, we can guarantee that there will always be a packet sent within the corresponding time period and keep the QoS information up-to-date.

Path Computation

As discussed in Section 3.4.1, paths between an agent and the sources are computed periodically as well as triggered by significant changes in QoS information. For path computation, we have two parameters, *ComputeInterval* and *ComputeTriggerThreshold*, as shown in Table 3.4. That is, the multicast tree paths are computed periodically using the *ComputeInterval* to decide the intervals. However, path computation can also be triggered when an agent detects a significant change in QoS information.

Parameter	Description
ComputeInterval	Determining how often an agent computes the paths.
ComputeTriggerThreshold	Deciding whether an agent triggers path computation.

Table 3.4: Parameters for Path Computation

If we denote the change of the QoS metric to a source as M_{change}, the original value as $M_{original}$, the agent will compute paths to sources if

$$M_{change} \geq ComputeTriggerThreshold * M_{original}.$$

3.4.4 Eligibility Test

An agent makes join request decisions based on the current QoS information and a protocol parameter, *Admission Factor*, denoted as α, to make the protocol flexible for different types of applications, and to reduce the impact of information inaccuracy.

We discuss three types of QoS metrics, namely, additive, multiplicative, and concave metrics [HT99]. For the following discussion, a path between u and v is denoted as $P_{u,v} = t_0t_1...t_n$, where $t_i \in V$(the set of routers), $t_0 = u, t_n = v$, and $i \in \{0, 1, ...n\}$. We denote the link metric on link ab as $m_{a,b}$, and the cumulative metric on path $P_{u,v}$ as $M_{u,v}$.

We say the cumulative path metric $M_{u,v}$ is additive if

$$M_{u,v} = \sum_{i=0}^{n-1} m_{t_i,t_{i+1}}.$$

We say the cumulative path metric $M_{u,v}$ is multiplicative if

$$M_{u,v} = \prod_{i=0}^{n-1}(m_{t_i,t_{i+1}}).$$

We say the cumulative path metric $M_{u,v}$ is concave if

$$M_{u,v} = min\{m_{t_0,t_1}, ..., m_{t_{n-1},t_n}\}.$$

The end-to-end delay of a path is an additive metric, which is the sum of the delays a packet experiences on all the links in the path. Loss rate of a path can be dealt with as a multiplicative metric, with additional handling as shown later. The available bandwidth on a path is a concave metric, which measures the minimum available bandwidth of all the links in the path. In the following subsections discussing QoS metrics, we first describe information gathering and computation at agent a_0. Then we present the eligibility test for a join request at a_0. For QoS information acquisition of a link or a path, see Section 3.4.1.

We denote the set of sources as S, the set of agents as A, and the routers as V. For each $s \in S$, there is an agent $a \in A$ that is associated with s. The path between s and a is $P_{s,a}$. The path between a and $a_0 \in A$ is P_{a,a_0}. In the case s is associated with agent a_0, $a = a_0$, we get P_{a_0,a_0}, which is actually a point. We set the cumulative metric on path P_{a_0,a_0}, $M_{a_0,a_0} = 0$, if the metric is delay or loss rate; $M_{a_0,a_0} = max$, if the metric is bandwidth, where max is a maximum value. The requesting receiver is denoted as r, and the path from the agent a_0 to r is $P_{a_0,r}$.

Additive Metrics

Delay is used as a representative of additive metrics. We denote the delay on link ab as $d_{a,b}$, and the cumulative metric on $P_{u,v}$ as $D_{u,v}$. We have,

$$D_{u,v} = \sum_{i=0}^{n-1} d_{t_i,t_{i+1}}.$$

Thus the QoS information for source s stored at agent a_0 is,

$$D_{s,a_0} = D_{s,a} + D_{a,a_0},$$

where source s is associated with agent a.

The agent a_0 calculates the delay information of the tree, D_{tree,a_0}, that is, from all the sources to it, as

$$D_{tree,a_0} = max\{D_{s',a_0}\}, s' \in S.$$

We can also calculate the QoS metric between a_0 and the receiver r, $D_{a_0,r}$, when the receiver requests to join the group. A request will be accepted if its delay requirement is larger than the product of the *Admission Factor*, α, and the current delay the tree can support. That is, if

$$D_{tree,a_0} + D_{a_0,r} \leq \alpha * D_{Req},$$

where D_{Req} is the delay requirement of a receiver, the request will be approved.

Multiplicative Metrics

Loss rate is used as a representative of multiplicative metrics. We denote the loss rate on link ab as $l_{a,b}$, and the cumulative metric on $P_{u,v}$ as $L_{u,v}$. Consequently the probability that there is no loss on link ab is $1 - l_{a,b}$. And the probability that there is no loss on the path $P_{u,v}$ is $1 - L_{u,v}$. We have,

$$L_{u,v} = 1 - \prod_{i=0}^{n-1}(1 - l_{t_i,t_{i+1}}).$$

Thus the QoS info for source s stored at agent a_0 is,

$$L_{s,a_0} = 1 - (1 - L_{s,a}) * (1 - L_{a,a_0}),$$

where source s is associated with agent a.

The agent a_0 calculates the loss rate information of the tree, L_{tree,a_0} as,

$$L_{tree,a_0} = max\{L_{s',a_0}\}, s' \in S.$$

We can also calculate the loss rate between a_0 and r, $L_{a_0,r}$. If the following formula holds,

$$1 - (1 - L_{tree,a_0}) * (1 - L_{a_0,r}) \leq \alpha * L_{Req},$$

where L_{Req} is the loss rate requirement of a receiver, the agent a_0 accepts the request of the receiver r.

Concave Metrics

Available Bandwidth is a representative of concave metrics. We denote the available bandwidth on link ab as $b_{a,b}$, and the minimal available bandwidth of all links on $P_{u,v}$ as $B_{u,v}$.

$$B_{u,v} = min\{b_{t_0,t_1}, ..., b_{t_{n-1},t_n}\}.$$

Thus the QoS info for source s stored at agent a_0 is.

$$B_{s,a_0} = min\{B_{s,a}, B_{a,a_0}\}.$$

where source s is associated with agent a.

The agent a_0 calculates the available bandwidth information of the tree. B_{tree,a_0} as

$$B_{tree,a_0} = min\{B_{s',a_0}\}, s' \in S.$$

The minimal available bandwidth between a_0 and r, $B_{a_0,r}$ can be obtained too. If

$$min\{B_{tree,a_0}, B_{a_0,r}\} \geq \alpha * B_{Req},$$

where B_{Req} is the bandwidth requirement of a receiver, the agent a_0 accepts the request of the receiver r.

We can see that for metrics such as delay and loss rate, the larger the α is, the more aggressive the protocol behaves, since the protocol with a larger α will accept more join requests. For metrics such as available bandwidth, the larger the α is, the less aggressive the protocol behaves.

3.5 Group Dynamics

In this section, we discuss the group dynamics of AQMR. We describe how data packets flow in AQMR. After that, we present a simple join scheme. Then we present an improved scheme called Agent-based Multicast SubGroups. After that we discuss receiver leave, routing state maintenance, source join, and possible extensions. Relevant protocol message types and their functions are presented in Table 3.2.

3.5.1 Data Packet Flow

The multicast tree needs to distribute data packets from a source to all the receivers currently on the tree. In AQMR, basically a data packet is delivered

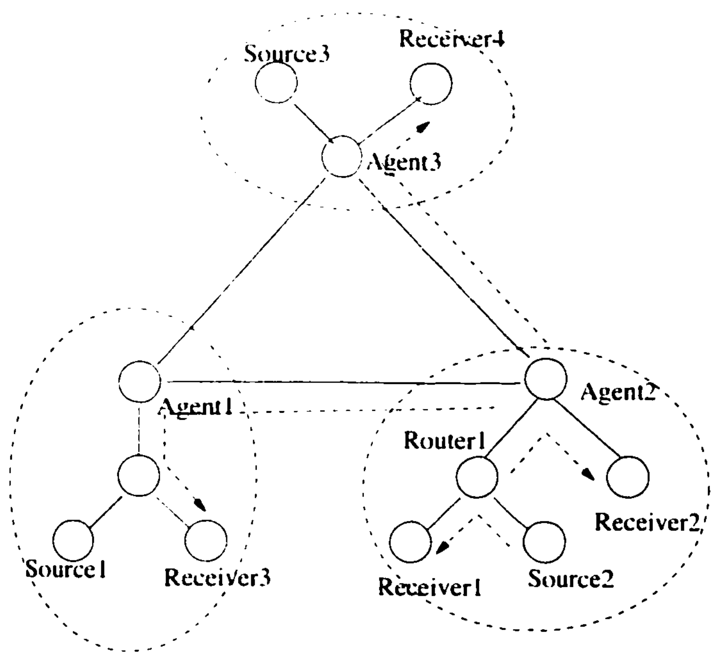


Figure 3.4: Packet Transmission

to the agent associated with the source first. The agent forwards the packet to other agents that are on the tree. Then those agents transmit the packet to the receivers in their areas respectively. To shorten transmission latency, a mechanism similar to the bi-directional tree in CBT[BA97] is deployed in the area in which the source resides, so that a packet will follow a short-cut route to receivers in the same area the source resides in, rather than necessarily following the route through the agent first. Figure 3.4 illustrates the idea. A packet is generated by *Source2*. *Router1* forwards the packet to *Receiver1*. *Router1* also duplicates the packet and forwards it to *Agent2*. When *Agent2* receives the packet, it makes three copies of the packet. One copy is sent to *Receiver2*; one copy to *Agent1* and the third copy to *Agent3*. Finally *Agent1* forwards the packet to *Receiver3*, and *Agent3* to *Receiver4*.

An agent needs to know where to forward data packets from different sources, which packets are from sources in its area and which are not. For example, in Figure 3.4, when *Agent2* receives a data packet from *Source2*, it needs to forward the packet to *Receiver2*, *Agent1*, and *Agent3*. However, when

Agent2 receives data packets from sources not in its area, e.g. *Source1*, it should forward the data packet to *Receiver1* and *Receiver2*, but not to *Agent1* and *Agent2*. Such capability of agents can be achieved when the routing protocol sets up the tree. That is, when the protocol updates the routing table entries.

3.5.2 Receiver Join

A simple join mechanism can be done as follows. An improved scheme will be presented in the next section.

When a join request arrives at an agent, the agent conducts the eligibility test. The agent sends a *JoinACK* message to the receiver if the join request is approved. Otherwise the agent replies to the receiver with a *JoinNACK*. The paths between a receiver and an agent, between a source and an agent and between two agents are obtained using the $1 - ERS$ search(see Section 3.3).

The agent will set up paths at the same time it approves a join request if the agent is not on the tree to some source. This could happen if the agent is serving the first join request, or it has torn down branches to other agents(see Section 3.5.4). The agent checks the path information for setting up paths to sources. For each source, it sends the setup packet towards the agent(denoted as a') which is in charge of the source in question, if a' is a different agent. The setup packet stops proceeding further when it meets an on-tree router, or it will reach a' finally. a' sets up the path to the source if necessary. If a' is the agent itself, it sets up the path to the source directly if needed. An agent does not need to set up paths to sources if the paths have already been set up.

When the receiver receives the *JoinACK* from the agent, it will set up the part of the path from the agent to itself. The routers on the way to the agent update their status to forward data packets. The setup process may terminate when an on-tree router is met. Otherwise the agent will receive the setup packet and terminate the setup process. In Figure 3.5, dash-dotted lines show the path setup routes for general cases. The *Receiver* sets up the path from itself to *Agent1*. *Agent1* sets up the path to *Source1*. It also sets up paths to

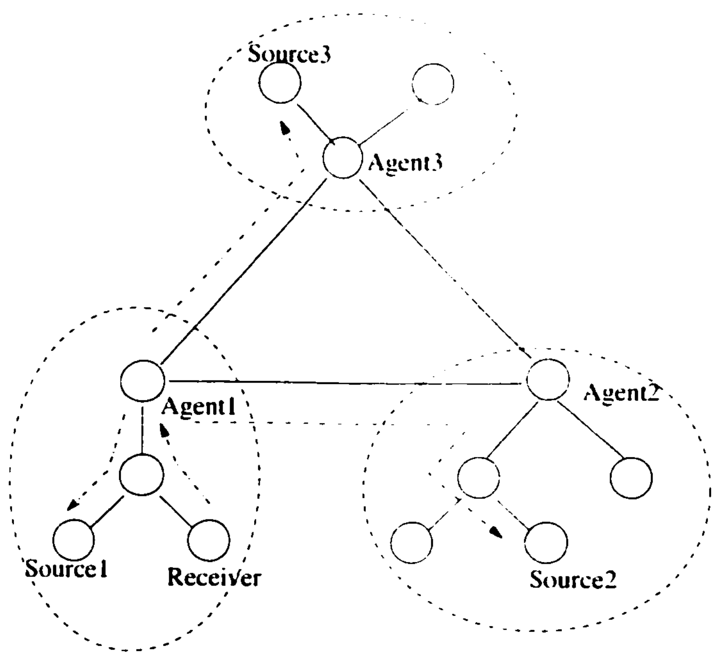


Figure 3.5: Path Setup

Agent2 and *Agent3*. *Agent2* and *Agent3* set up paths to *Source2* and *Source3* respectively.

From the above setup procedure, we can see that, for each source, the protocol constructs a uni-directional tree rooted at the source. That is, there are no loops in the tree for a certain source. In the case of multiple sources, there is also no loop, since their trees do not interfere with each other.

Currently, the QoS information from an agent to a joining receiver is collected when the request packet travels to the agent. The agent makes its decision assuming the QoS information of both directions is approximately the same. To make it more accurate, the agent may instead send back an ACK with QoS information, and temporarily set up the path for the new receiver. When the ACK reaches the receiver, it knows exactly whether its requirement is satisfied or not and stays in the group or tears down the temporary path accordingly. Furthermore, the agent can send ACK packets using multiple paths so that the receiver can choose the best one to join the group.

3.5.3 Agent-based Multicast SubGroups

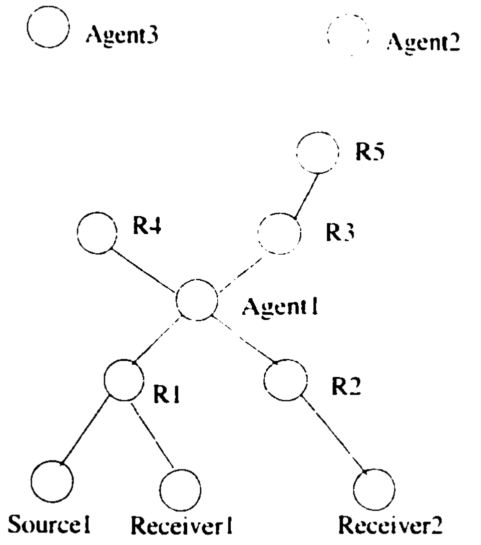
To construct a multicast distribution tree, a straightforward approach is like that described in the above section. The routing protocol builds a multicast tree similar to a source-based tree for each source, with agents as special points in the tree. However, this may incur a scalability problem when the number of sources increases, since each router on the multicast tree needs to record routing information for all the sources in order to forward their replicated data packets. One of the objectives of core-based tree solutions is to tackle such scalability problems. That is, in a core-based multicast tree, each group needs only one multicast address, so that a router on the tree has only one routing entry for a multicast group. On the other hand, the core-based tree solution may construct trees with low efficiency, since traffic follows a route to the core first, then to the receivers, which may not be a route of good quality from a source to a receiver. Bi-directional tree approaches do not entirely solve this problem, although with a bi-directional approach, some receivers do not need a path that passes the core to receive data packets. In the following we describe a routing scheme that can achieve both efficient paths and low routing state storage.

In this routing scheme, we take advantage of the capability of the agents. In AQMR, data packets from a source reach the agent in its area before being forwarded to agents in other areas. For an agent(a_1), and the receivers in its area, data packets from a source(its agent is a_2) in another area are relayed by its corresponding agent(a_2) to the agent(a_1), then forwarded to the receivers by the agent(a_1). The example in Section 3.5.1 illustrates the scheme. We construct multiple multicast subgroups, rather than only one multicast group to achieve efficient resource consumption. In this scheme, each agent is a central point in its area, so that a multicast subgroup is built using the agent as the core. A bi-directional tree is built so that a packet does not need to reach the agent first before it is sent to a receiver in the same area, if there is a short-cut between the source and the receiver. In Figure 3.4, *Source2* can send the data packets to *Receiver1* through the dashed line. In addition, each agent needs to record outgoing interfaces to other agents.

When a source generates a data packet, it assigns the destination address as the subgroup address, $Addr_1$ (different subgroups of different agents have different subgroup addresses). The packet is forwarded by the routers until it reaches the agent. The agent forwards the packet to branches other than the one from which it comes using the same destination address, $Addr_1$, to forward the packet to receivers in the same area. The agent also forwards the data packet to branches leading to other agents. At this time, it assigns the destination address using a specific address reserved for this multicast group(note: not subgroup), $Addr_0$. Routers between a pair of agents can recognize $Addr_0$, since that is the multicast group address. Thus the packet can be sent to other agents. When an agent receives a packet with destination $Addr_0$, it knows it is from sources in other areas, therefore it forwards the packet according to the routing entry for $Addr_0$. Generally it forwards the packets(from sources in other areas) only to branches leading to receivers in its areas, but not to those of other agents. However, if the agent is on the way between two agents, it can still forward the packets to the interfaces according to the routing table.

In Figure 3.6, we illustrate how the scheme of Agent-based Multicast Sub-Groups deals with packet forwarding. *Agent2* and *Agent3* need to pass *Agent1* to reach each other. The routing table for address $Addr_1$ is bi-directional; while the routing table for address $Addr_0$ is uni-directional. In addition, an agent needs to record the interfaces that lead to the local area, and those that lead to other agents, so that it can make the appropriate handling of the destination addresses. For instance, *Agent1* knows that interfaces R_1 and R_2 are local, such that when a packet comes from R_3, it can forward the packet to R_1 and R_2 after changing the destination address to $Addr_1$.

With strategically located agents, efficient consumption of network resources is achievable. The scheme can be regarded as a distributed multiple core approach. It can construct trees similar to source-tree solutions in terms



Routing Table for Addr1(bi-directional)

Router	Interfaces
R1	Source1, Agent1, Receiver1
R2	Agent1, Receiver2
Agent1	R1, R2, R3*, R4*

Routing Table for Addr0(uni-directional)

Router	iif	oif
Agent1	R3	R1*, R2*, R4
Agent1	R4	R1*, R2*, R3
R3	Agent1	R5
R3	R5	Agent1

* Denotes approriate address change needed

Figure 3.6: A Routing Table(Scheme of Agent-based Multicast SubGroups)

of hop count of multicast tree paths. Meanwhile, it does not need to maintain per-source routing state. A router needs to maintain a routing table for the agent-based subgroup the router resides in, or a per-group routing table for the specific subgroup address $Addr_0$. The scheme may be criticized for using too many addresses for the multicast group(the number of agents plus one specific address). However, as proposed in Simple Multicast[PL99], a multicast address like (C, M), the combination of a node address(C) and an original multicast address(M), can eliminate such problems. In addition, this scheme may be helpful to source joins, since with the specific subgroup address $Addr_0$, receivers in areas other than the one a source resides in need not be aware of the source's subgroup address.

The scheme of Agent-based multicast subgroups also constructs loop free trees. In each area centered with an agent, the subgroup with address $Addr_1$ is a bi-directional tree. All the agents form another subgroup with address $Addr_0$. In this subgroup, each agent has uni-directional branches leading to other agents. The agents relay packets from sources not in its area from subgroup $Addr_0$ to subgroup $Addr_1$(which is the subgroup centered with the agent). An agent isolates and coordinates the multicast subgroup in its area with the subgroup addressed $Addr_0$ and subgroups in other areas.

3.5.4 Receiver Leave

When a receiver wants to leave the group, it sends a *Prune* packet to upstream routers. A router on the way to the agent will forward the leave request if it has become a leaf node. The agent may choose to tear down the tree branches to other agents when it receives the leave request, after checking out that no receiver is currently in the area. It may do nothing after receiving a leave request. In the latter case, the agent keeps paths to sources in other areas, so that it does not need to set up the paths when a new receiver comes.

3.5.5 Routing State Maintenance

AQMR uses a soft-state approach to maintain routing state at routers. A receiver needs to send *KeepAlive* packets periodically to keep the routing state alive in the routers between its agent and itself. An agent intercepts *KeepAlive* packets it receives. It sends *KeepAlive* packets periodically on branches leading to other agents, if there are receivers in its area. It may choose not to send *KeepAlive* packets when there is no receiver in its area. In this case, when a new receiver is accepted, the agent needs to set up paths to other agents and finally sources in other areas in order to receive data packets from the sources in other areas. Alternatively it may send *KeepAlive* packets to keep the paths for later joining receivers, even if there are no receivers in its area currently. However, there should be an additional mechanism to tear down the routing table entries when the whole group ends communication.

3.5.6 Source Join

The current design assumes all sources are on the tree in the beginning so that there is no source joining during the communication. We present a brief discussion about issues concerning the source join and a possible solution.

When a new source wants to join the multicast group, we assume it can locate an agent to associate with it. It needs to pass a certain eligibility test to guarantee that other multicast members' QoS requirements will not be violated. Additionally, it may be necessary that receivers be informed of the coming of a new source.

To facilitate this, each agent should store the tightest QoS requirement of receivers in a multicast group. For example, for end-to-end delay, it records the lowest delay requirement. It needs to compare the requirements of both receivers in its area and also in other areas. Therefore, a mechanism needs to be designed to exchange such information among agents. Since receivers may come and go, agents need to keep track of such changes. It may have to record each receiver's requirement to make an accurate calculation. With the tightest information, and the QoS information from the new source to its agent collected with the join request packet, the agent associated with the source can make decisions whether to accept the new source.

The issue of source join needs more investigation. We leave it as future research work.

3.6 Summary

In AQMR, the QoS information is complete and almost up-to-date. It addresses the issue of "partial path information" in QoSMIC and YAM. Each agent stores path information to sources so that a member can join the group after passing the eligibility test at the agent and expects satisfactory QoS support. The architecture of AQMR allows it to work well with QoS metrics including non-additive ones such as end-to-end delay and loss rate, which is not supported by QMRP.

The design of AQMR aims at the following objectives.

- *QoS awareness*. A source sends *InfoCollection* packets to the agent associated with it so that the agent can gather QoS information from sources to it. Agents exchange QoS information with other agents, during which the QoS information on the path between two agents is also collected. By storing QoS information, agents can conduct eligibility tests to accept or reject join requests. AQMR can support not only non-additive QoS metrics such as bandwidth, but also additive and multiplicative metrics such as end-to-end delay and loss rate.

- *Scalability.* In a multicast group, we can choose several agents to collect QoS information and to store the information in a scalable manner to facilitate receiver joins. It is more scalable than the approach in which every router needs to store QoS information.
- *Responsiveness.* A receiver can receive a *JoinACK*(or *JoinNACK*) from the agent shortly after submitting the join request, since the agent is close to the receiver, and the agent makes the decision immediately based on the multicast tree information it stores.
- *Efficiency.* With agents as special points in constructing the tree, AQMR attempts to build resource-efficient multicast trees, in terms of the average delay and average hop number of the tree. The multicast tree AQMR builds is expected to be better than a shared tree and to be similar to a source-based tree in terms of the average path delay and average path length of the tree paths. However, the routers need to record at most two entries, rather than per-source entries for a multicast group. Data packets need to go through fixed agent points on the tree.

In the next chapter, we will present the performance study of AQMR, and show that the design objectives are achieved.

Chapter 4

Performance Evaluation

We evaluate the performance of AQMR by simulation. We implement AQMR in Network Simulator 2(NS2)[NS], using the PANAMA package[PA] to simulate an Active Networks platform. First, we study the characteristics of AQMR to determine protocol parameters. Then we compare AQMR with QoSMIC in several aspects, including the ratio of delayed packets, the ratio of join requests being approved, tree cost, join latency, and join overhead. We concentrate on the QoS metric of end-to-end delay in the following performance study.

4.1 Evaluation Methodology

To conduct a performance evaluation study, we can use the techniques of analytical modeling, simulation or measurement [JA91]. To evaluate performance of a routing protocol, measurement is possible only if routers have implemented the protocol. In addition, the routers should be distributed over a large area, rather than in a lab, or even on a campus. Analytical modeling needs to simplify the scenarios, and to make many assumptions such that it can not always guarantee the accuracy of the results.

We will use simulation to evaluate the performance of our routing protocol. Simulation is suitable for studying a system that is not available. During the design phase of a routing protocol, we can use simulation to predict its performance, and to compare several alternatives of the protocol and with other previously proposed protocols in a variety of scenarios.

In the following, we first briefly introduce NS2 and the PANAMA package. Then we discuss the techniques to verify and validate the simulation module we designed. In our experiments, the performance results are obtained when the system is running at steady state. Furthermore, we run each simulation 8 times with different seeds and average the results. 95% confidence intervals are calculated.

4.1.1 Simulation Platform

NS2[NS] is a discrete event-driven simulator. It implements link-layer technologies, unicast and multicast routing algorithms, transport protocols, reservation and integrated services, application-level protocols as well as scheduling and queue management algorithms. The multi-protocol nature of NS2 makes it easy for researchers to concentrate on implementing the new protocol on top of an existing infrastructure and to compare performance results across research work. NS2 also provides topology generators and traffic generators.

PANAMA stands for Protocols for Active Networking with Adaptive Multicast, a cooperative research and development effort between TASC and the University of Massachusetts. The PANAMA package simulates the Active Networks technology in NS2. It is an extension to NS2. The package implements basic active functionality of active agents[1] such as initialization and disabling of an active agent. After an agent is initialized, it can intercept an active packet passing the node it resides in, do the Active Networks functions as defined, and forward the packet on to the next node.

We use the technique in the PANAMA package to add the functionalities of Active Networks to NS2. We adds modules in NS2 to implement the entities in AQMR including agent, router, source, and receiver.

[1]In NS2, an agent constructs and consumes network-layer packets. Also it can be protocol entities at various layers.

4.1.2 Simulation Model Verification and Validation

We use the following techniques[JA91] to verify and validate the simulation modules that we added to NS2.

- *Top-Down Modular Design.* Modularity and top-down techniques were used to develop, debug and maintain the simulator program. We use the object oriented programming language(C++) to design classes for implementation of agent, router, source, and receiver. TCL is used to script a network topology and scenarios to evaluate the performance of AQMR.

- *Antibugging.* We check the total number of packets at sources and at the receivers to check whether there is something wrong in between. Dropped packets were considered when computing the number of packets.

- *Simplified Cases Test.* Simplified cases were tested, such as if bandwidth is sufficient for all the sources and background traffic, then there should be no or very few packets lost if the simulator works well.

- *Continuity Test.* With slight change of parameters, the simulation results should change only slightly. To test the continuity of the simulator, several simulations were conducted with various but slightly different source rates or background CBR rates. The performance metrics(see Section 4.2) such as the *Delay Ratio* changed only slightly.

- *Intermediate Result Display.* Some intermediate results were displayed during the execution of the simulator in the debugging stage. In this way, we tracked the ongoing status of the simulation. However, this is limited since the simulation is long and too many printed values become confusing.

- *Seed Independence.* We use independent seeds for the random number generator. Simulations with the same parameters but independent seeds have similar outcomes.

4.2 Performance Metrics

We will study AQMR and compare it with QoSMIC in terms of the following performance metrics.

- *Delayed Ratio.* Delayed ratio measures the ratio of data packets that reach receivers beyond their delay requirements. It measures the QoS perceived by receivers. We denote the number of delayed packets of all the receivers as *Delayed#*, the number of lost packets of all the receivers as *Lost#*(loss is very small in this experiment), and the number of packets received by all the receivers as *Received#*. The Delayed Ratio is calculated as.

$$Delayed\ Ratio = \frac{Delayed\#}{Lost\# + Received\#}$$

- *Success Ratio. Success Ratio* measures the ratio of join requests being approved. In our protocol, a receiver will be accepted or rejected for all sources by its agent, which makes admission decisions for its receivers. In QoSMIC, in the case of the source-tree request, a receiver initiates one request for each source, thus it may be accepted by some sources, but be rejected by other sources. We divide the number of requests being approved by the total number of requests as *Success Ratio* in both protocols. Such measurement favors QoSMIC, since in QoSMIC, a receiver can join part of all the sources, whereas in AQMR, a receiver needs to join all the sources, or to join none of the sources. We may study the issue of "partial join" in the future. It should be noted that a high *Success Ratio* may not be good in a QoS routing scheme. The *Success Ratio* is calculated as.

$$Success\ Ratio = \frac{\text{Number of Requests Approved}}{\text{Total Number of Requests}}$$

- *Path Length* and *Path Delay. Path Length* and *Path Delay* are respectively the average path hop number and the average path delay of all the data packets received by all the receivers. These two metrics measure multicast tree cost.

$$Path\ Length = \frac{\sum \text{Hops of Each Data Packet}}{\text{Number of Received Packets}}$$

$$Path\ Delay = \frac{\sum \text{Delay of Each Data Packet}}{\text{Number of Received Packets}}$$

- *Join Latency*. *Join Latency* measures the time interval between the time a receiver initiates a join request and the time it receives a *JoinACK* or a *JoinNACK* from the agent.

- *Join Overhead*. *Join Overhead* refers to the hop count for messages involved in processing a join request. In AQMR, overhead messages include those for join requests and for join ACKs and NACKs. In QoSMIC, message types of BID_REQ, BID, M_JOIN, BID_JOIN[FB98] fall into this category.

- *Info Overhead*. *Info Overhead* is the protocol overhead for gathering QoS information among sources and agents. It counts the total hops traversed by the messages of *InfoCollection* and *InfoExchange*.

4.3 Experiment Setup

The topology used in the study is generated by GT-ITM[CD97] using the Transit-Stub model. It is composed of transit and stub domains that are connected together to resemble the structure of the Internet. There are 176 nodes. We set all link bandwidth to 1.544Mb. We choose 4, 9 or 11 agents manually using a combination of the Stub-AS and the Transit-AS scheme discussed in Section 3.2. Six sources and 76 receivers are chosen from the nodes in stub domains. A source or a receiver is associated with an agent manually, if the agent is in the same stub domain, or the agent is on the transit domain and is close to the stub domain in which the source or the receiver resides. Sources generate CBR multicast traffic throughout the simulations. We randomize the receivers' joining time from the start time of a simulation to 9/10 of the simulation end time, and the durations of receiver's staying in the group are randomly chosen from 500 to 1000 seconds. To make our implementation simple, we find paths between a source and an agent, between a receiver and an agent, and between two agents beforehand using the scheme of $1 - ERS$ search(refer to Section 3.3). Paths do not change during simulations.

Five bi-directional background traffic flows are placed between 5 pairs of sources and their agents. Each background flow alternates between busy and idle, of which the duration times are calculated using two parameters, *cbr_busy* and *cbr_idle* respectively. During the busy stage, each background traffic flow is a CBR flow. We set *cbr_busy* to 29.0 seconds and *cbr_idle* to 7.0 seconds. Using such an on-off traffic model, we expect to obtain a certain level of background traffic with some fluctuation.

We use two parameters, *InfoInterval* and *TriggerThreshold* for the intervals and triggers in the simulations. We set the protocol parameters related to these two respectively as shown in Table 4.1. We set *SrcInterval* equal to 1/3 of *InfoInterval*, and *ExchangeInterval* and *ComputeInterval* equal to *InfoInterval*. *AgentTriggerThreshold* and *ComputeTriggerThreshold* are set equal to *TriggerThreshold*. In Section 4.5.1, we study the effect of *InfoInterval* and *TriggerThreshold* on the protocol performance when the relevant parameters are set in the above way. The effect of setting these parameters in other ways is left as future work.

Parameter	Parameter Setting
SrcInterval	SrcInterval = InfoInterval/3
ExchangeInterval	ExchangeInterval = InfoInterval
AgentTriggerThreshold	AgentTriggerThreshold = TriggerThreshold
ComputeInterval	ComputeInterval = InfoInterval
ComputeTriggerThreshold	ComputeTriggerThreshold = TriggerThreshold

Table 4.1: Experiment Parameters for Intervals and Triggers

Table 4.2 displays the parameter setting that applies to all the experiments.

Parameter	Value
Bandwidth	1.544Mb
Simulation Run Time	9000 seconds
Receiver Join Time	Chosen in [0, 8100] seconds
Receiver Stay Duration	Chosen in [500, 1000] seconds
cbr_busy	29.0 seconds
cbr_idle	7.0 seconds

Table 4.2: Experiment Parameters(General)

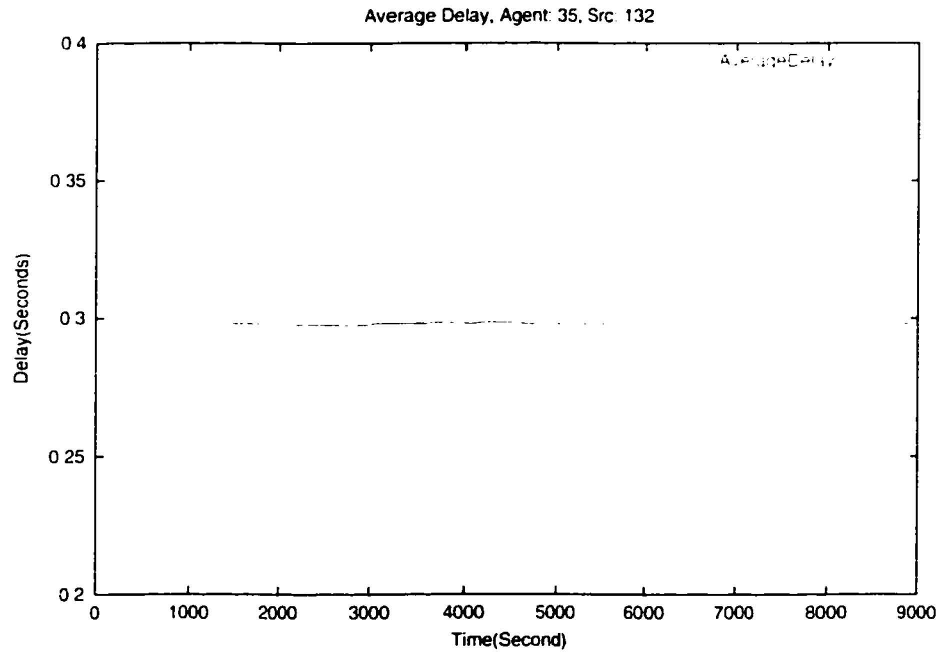


Figure 4.1: Average Delay versus Time(Transient State Removal)

4.4 Transient State Removal

We study the performance of the protocol only after the simulator has entered the steady state. We measure the data packet delays at the agents, and average the results from the beginning. We plot the result for a source(Node 132) and an agents(Node 35) in Figure 4.1. In our experiments, we start collecting experiment data after the time of 1500 second, and run the simulations for 9000 seconds.

4.5 Characteristics Study on AQMR

In this section, we study the characteristics of our protocol AQMR. We first study how intervals and triggers for QoS information gathering affect the protocol performance. Then we study the effect of the number of agents on the performance of AQMR. After that, we study changes of performance caused by varying the Admission Factor(α). These experiments give us feedback in how to set some of the parameters in the AQMR protocol.

4.5.1 Characteristics Study on Intervals and Triggers

In this section, we study how the protocol performance is affected by the two parameters, *InfoInterval* and *TriggerThreshold*. We run 8 simulations for each of the following combinations of *InfoInterval* and *TriggerThreshold*, and plot the figures of the averaged results. 9 agents are chosen for this study.

We investigate the effect of triggers on the performance. We choose a small trigger threshold(0.001) and a very large one(100.0, with which AQMR does not actually trigger messages). We set the *InfoInterval* to 250, 500, 1000, 1500, 2000 or 3000 seconds. The parameter settings are shown in Table 4.3.

Parameter	Value
Source Rate(CBR)	200Kb/s
Background CBR rate	400 seconds
Number of agents	9
Admission Factor(α)	1.1
InfoInterval	250, 500, 1000, 1500, 2000 or 3000 seconds
TriggerThreshold	0.001, or 100.00(equivalent to no trigger)
Delay Requirement	0.45 seconds

Table 4.3: Experiment Parameters(Study on *InfoInterval*& *TriggerThreshold*)

In this set of experiments, we look at the effect of having a trigger mechanism with *Admission Factor*, $\alpha = 1.1$. Figure 4.2 shows the *Success Ratio* versus *InfoInterval*. Figure 4.3 shows the ratio of delayed data packet versus *InfoInterval*. Figure 4.2 and Figure 4.3 indicate respectively that with a trigger mechanism (*TriggerThreshold* = 0.001), the protocol has a lower *Success Ratio* and a lower *Delayed Ratio* than without a trigger mechanism(*TriggerThreshold* = 100.0). The two figures confirm that the trigger mechanism helps AQMR achieve more accurate QoS information, so that there are less data packets delayed. The quite flat curves indicate the *InfoInterval* does not have much effect on the results.

Next we consider the case where $\alpha = 1.0$. Figure 4.4 shows the *Success Ratio* versus *InfoInterval*. Figure 4.5 shows the ratio of delayed data packet versus *InfoInterval*. Figure 4.4 and Figure 4.5 further confirm the earlier result, with the difference that the two curves in each figure diverge from each other much more, after the point of $InfoInterval = 1000$ second. That is, if no trigger mechanism is used, a larger *InfoInterval* will create more delayed packets; however, with the trigger mechanism to control the performance, the *InfoInterval* is not so reactive. Figure 4.6 shows protocol overhead for QoS information gathering versus *InfoInterval*. It indicates that AQMR has higher protocol overhead for QoS information gathering with a trigger mechanism than that of without, which means that our information gathering scheme is responding to the changes in network traffic, and triggering *InfoExchange* messages to update QoS information stored at agents. It also shows that as the *InfoInterval* decreases, the information overhead decreases dramatically at first, and then becomes much flatter. We can say that with the trigger mechanism, AQMR can set *InfoInterval* relative large to achieve good performance.

The above results show that AQMR works well at *InfoInterval* = 1500 seconds, and *TriggerThreshold* = 0.001. We choose this setting of parameters in later experiments.

4.5.2 Characteristics Study on Number of Agents

We also investigate how many agents(on a given topology) can provide good performance. The parameter setting for the experiment is shown in Table 4.4.

Table 4.5 shows the performance metrics of AQMR with 4, 9, and 11 agents, with the 95% confidence intervals in parentheses. It shows that AQMR with 9 agents has the lowest *Delayed Ratio*. The *Success Ratio* in the case

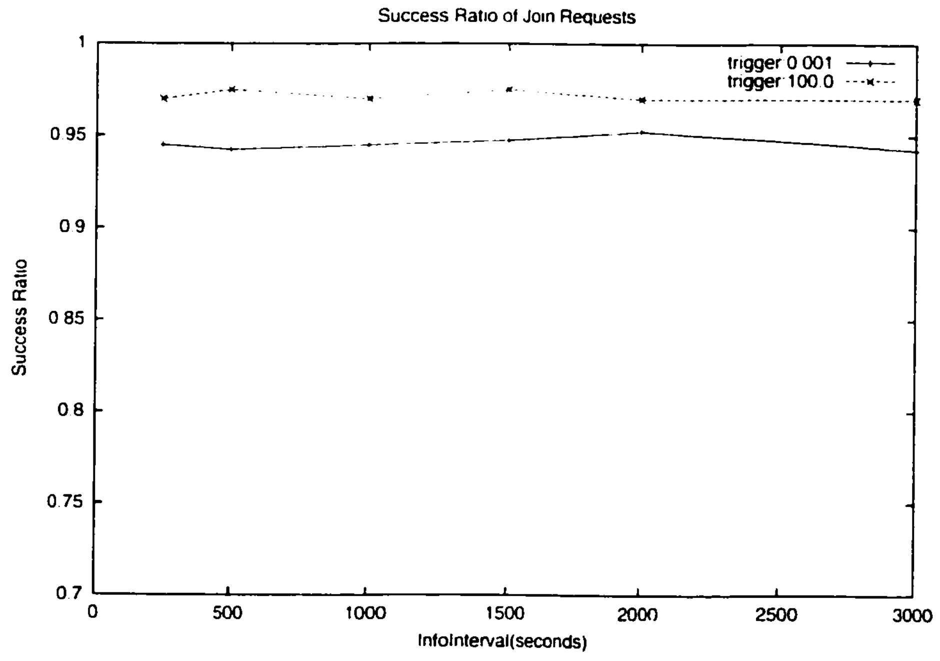


Figure 4.2: Join Request Success Ratio vs. *InfoInterval*(Parameter Study)

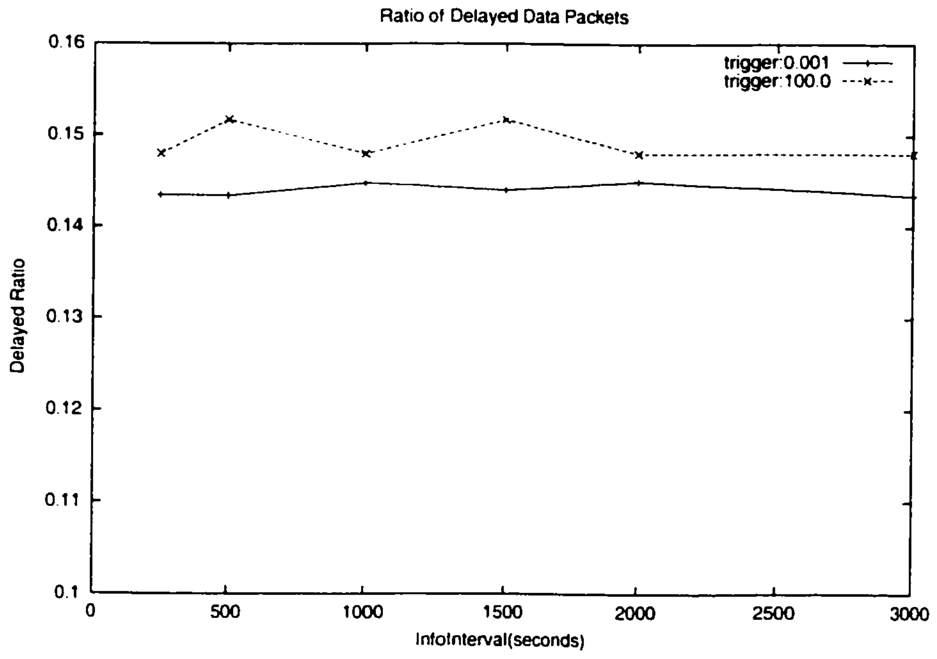


Figure 4.3: Ratio of Delayed Packets vs. *InfoInterval*(Parameter Study)

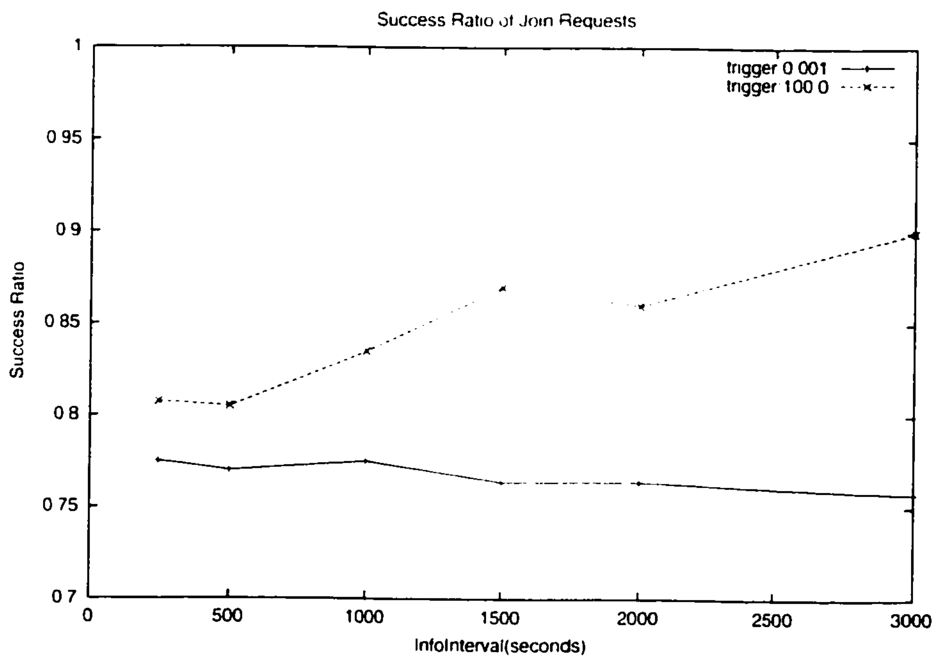


Figure 4.4: Join Request Success Ratio vs. *InfoInterval*(Parameter Study, $\alpha = 1.0$)

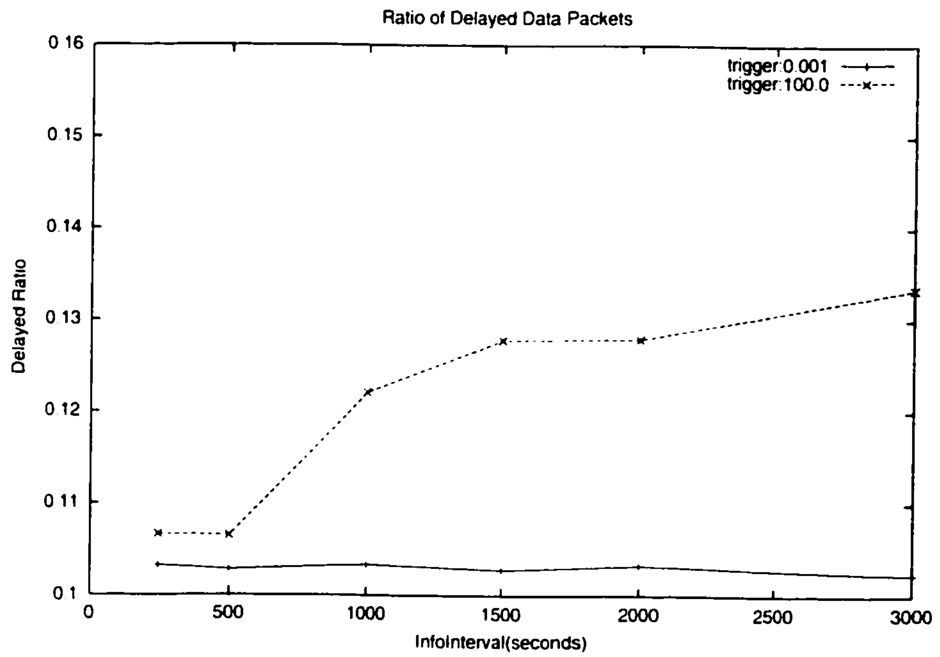


Figure 4.5: Ratio of Delayed Packets vs. *InfoInterval*(Parameter Study, $\alpha = 1.0$)

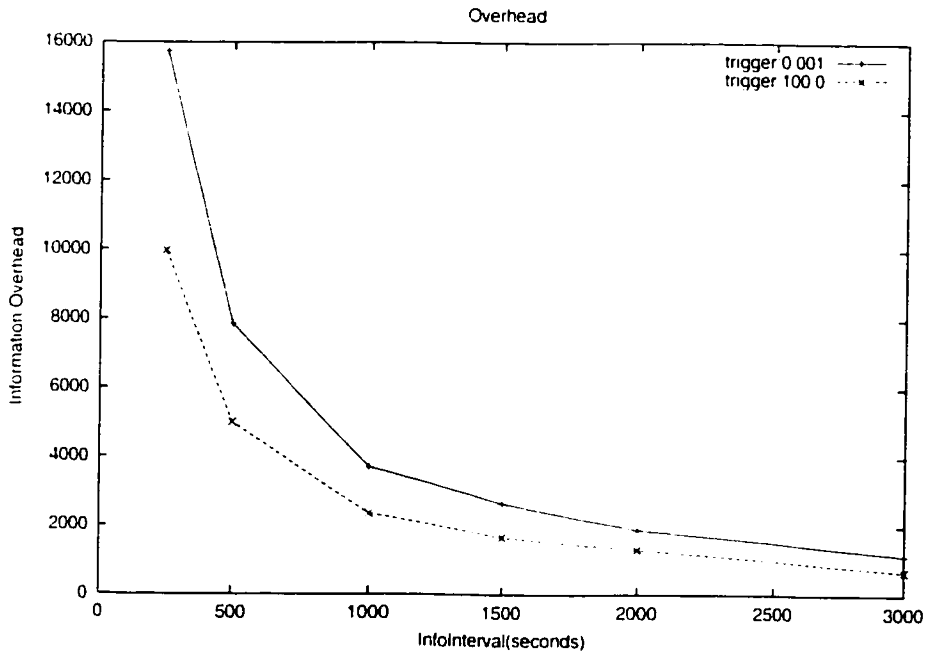


Figure 4.6: Information Overhead vs. *InfoInterval*(Parameter Study)

of 9 agents is statistically the same as that of the case of 4 agents, which is greater than that of the case of 11 agents. AQMR has very similar or overlapped performance of *PathDelay* for placing different numbers of agents, when considering the confidence intervals. For the other metrics, *Join Latency*, *Join Overhead* and *InfoOverhead*, AQMR with 9 agents performs well between AQMR with 4 and 11 agents.

In AQMR, a packet is sent to a receiver via a bi-directional tree rooted at the agent, if the source and the receiver are in the same area; otherwise, the

Parameter	Value
Source Rate(CBR)	200Kb/s
Background CBR rate	400 seconds
Number of agents	4, 9, or 11
Admission Factor(α)	1.1
InfoInterval	1500 seconds
TriggerThreshold	0.001
Delay Requirement	0.45 seconds

Table 4.4: Experiment Parameters(Study on Number of Agents)

Performance Metric	Result with Confidence Interval		
	4 agents	9 agents	11 agents
Delayed Ratio	0.1706(0.0037)	0.1441(0.0027)	0.1522(0.0033)
Success Ratio	0.946(0.0051)	0.946(0.0051)	0.856(0.0132)
Info Overhead	762.5(1.7)	2280.4(1.7)	3400.8(1.7)
Path Length	8.2938(0.016)	8.1090(0.031)	7.8800(0.020)
Path Delay	0.3684(0.0018)	0.3626(0.0018)	0.3616(0.0018)
Join Latency	0.2241(0.0033)	0.1850(0.0032)	0.1561(0.0010)
Join Overhead	7.00(0.0000)	5.690(0.0000)	5.000(0.0000)

Table 4.5: Protocol Performance With Different Number of Agents

packet will be relayed by two agents(those associated with the source and the receiver) before arriving at the receiver. When there are more agents on the topology, on average they are closer to receivers in their areas. In addition, more agents may also make a source and a receiver in different areas closer. This can account for the reduced *Delayed Ratio* and *Path Delay*. The result of *Info Overhead* indicates that the messages for exchanging QoS information among agents constitute the major part of the *Info Overhead*, so that when the number of agents increases, the *Info Overhead* increases a lot. The reason for the results of *Join Latency* and *Join Overhead* is that the more agents there are on the topology, the closer the agents are to the receivers in their areas on average.

4.5.3 Characteristics Study on *Admission Factor*

In this section, we study how the *Admission Factor* affects the protocol performance. Table 4.6 shows the parameter setting for this study.

Parameter	Value
Source Rate(CBR)	200Kb/s
Number of agents	9
Admission Factor(α)	0.9, 1.0, 1.1, or 1.2
InfoInterval	1500 seconds
TriggerThreshold	0.001
Background CBR rate	250, 300, 325, 350, 400 Kb/s
Delay Requirement	0.45 seconds

Table 4.6: Experiment Parameters(Study on Admission Factor)

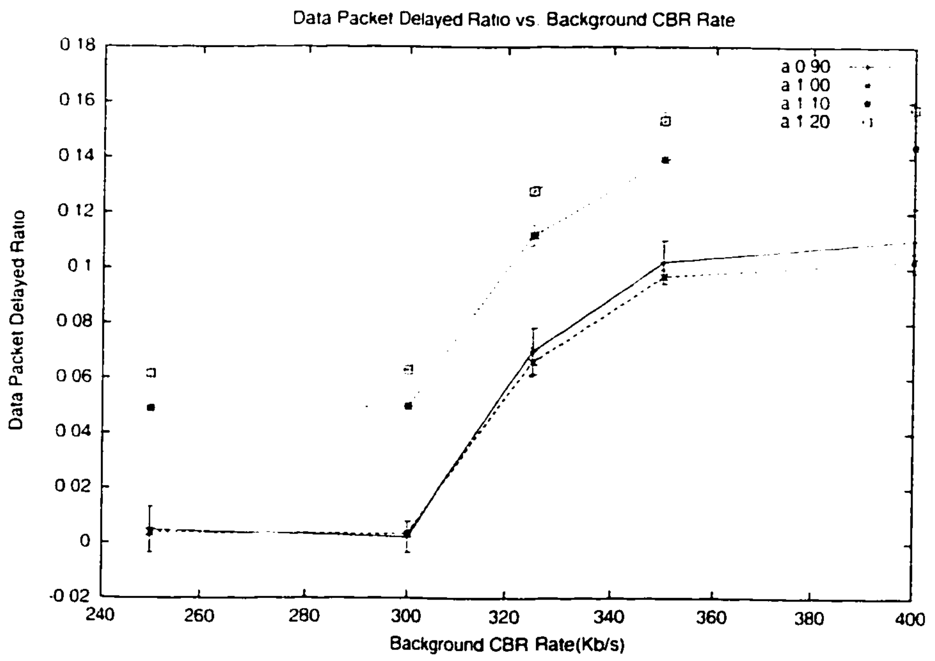


Figure 4.7: Percentage of delayed packets vs. Background CBR Rate(α study)

Figure 4.7 shows the *Delayed Ratio* of AQMR with *Admission Factor* α = 0.9, 1.0, 1.1 and 1.2 versus background CBR rate. For the same CBR rate, the greater α is, the more packets are delayed. The greater α is, the more join requests are approved(refer to Section 3.4.4 on the Eligibility Test), so that there are more chances that join requests with tight requirements are approved, which results in more packets arriving as delayed packets. For the same α, the *Delay Ratio* keeps relatively constant first(CBR:250 to CBR:300), then it increases(CBR:300 to CBR:350) significantly, but after some point(CBR:350), it increases only slightly. The reason can be seen in Figure 4.10, where the *Path Delay* keeps relatively constant first, then it increases significantly, but after some point, it increases only slightly. The changes of *Path Delay* affect the changes of *Delayed Ratio*.

Figure 4.8 shows the trend of *Success Ratio* with the changes of α versus background CBR rate. For the same CBR rate, the *Success Ratio* increases as α increases, since the acceptance criteria becomes less stringent. It changes only very slightly as background CBR rate increases, for α = 0.9, 1.0 or 1.1,

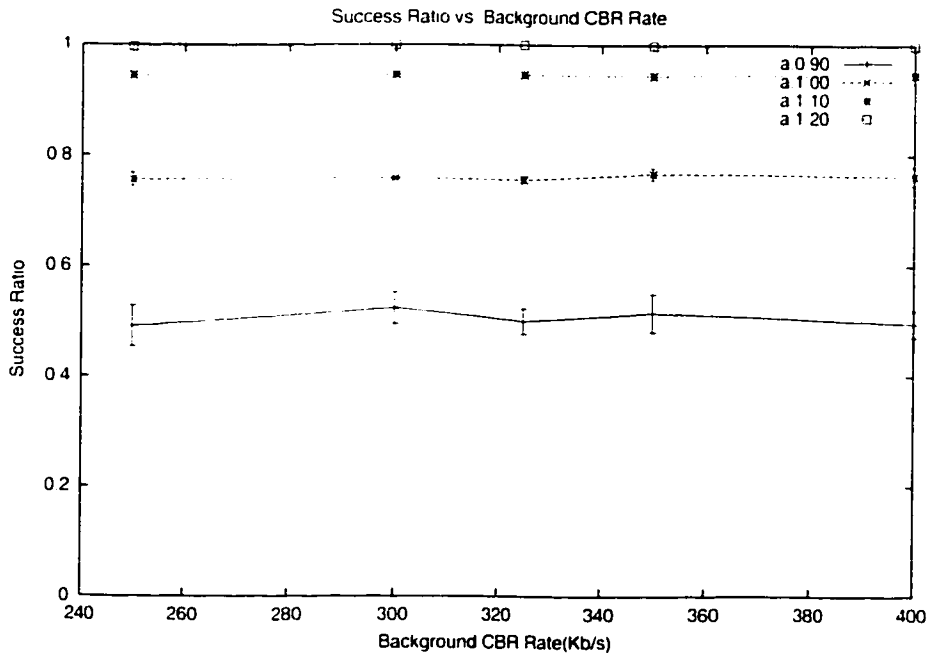


Figure 4.8: Join Success Ratio vs. Background CBR Rate(α study)

and all requests are approved for $\alpha = 1.2$. This means that the background CBR does not change QoS information enough to affect the eligibility tests, especially for the case of $\alpha = 1.2$. Figure 4.10 shows that the path delay is much less than 0.40 seconds, which is less than the join requirements (0.45 seconds). This may account for the flat curve of the *Success Ratio* versus background CBR rate with the same α. We conducted experiments with very high background CBR rate. The results are shown in Figure 4.9. It indicates that the *Success Ratio* could be very low when the background CBR rate is very high[2] .

Figure 4.10 shows the trend of *Path Delay* with the changes of α versus background CBR rate. The *Path Delay* has a similar trend as *Delayed Ratio*, with small changes. For a certain background CBR rate, the *Path Delay* increases as α increases, since with a greater α, there are more requests being approved, which results in more traffic on some paths.

[2]When the background CBR rate is very high, both AQMR and QoSMIC have very high *Delayed Ratio*, which means that both protocols do not perform normally. Therefore we will not conduct further experiments in such scenarios.

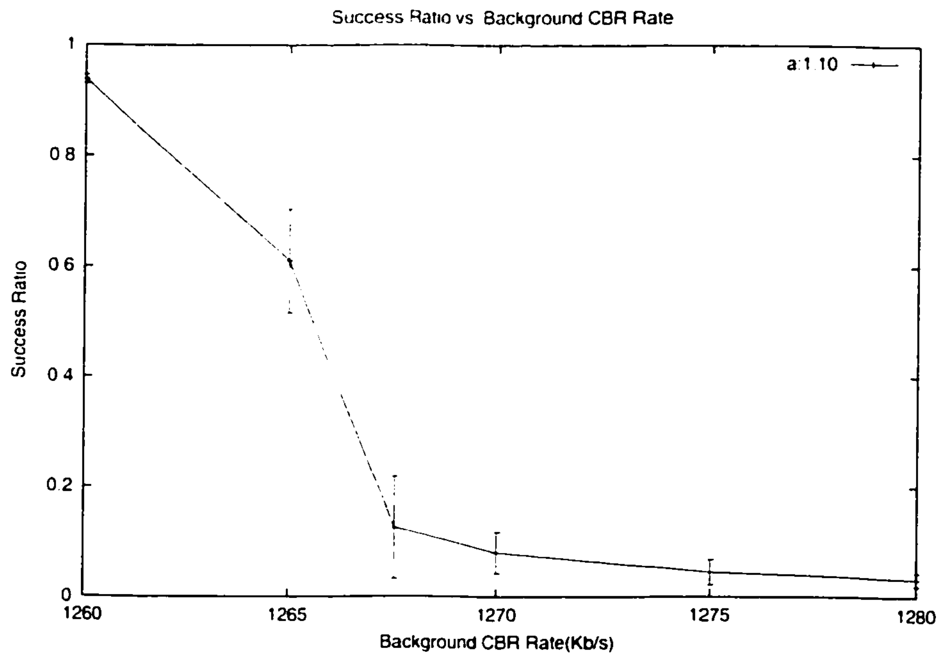


Figure 4.9: Join Success Ratio vs. Background CBR Rate(High CBR Rate)

Experiment results show that, as the background CBR rate increases, the *Path Length* keeps almost constant(around 8 hops). For a certain background CBR rate, the *Path Length* increases slightly as α increases, since with a greater α, join requests from farther receivers have more chances of being approved. Experiment results also show that the *Join Latency* is approximately 0.18 seconds with different *Admission Factors* and different background CBR rates. The *Join Overhead* keeps constant(5.690 seconds). The reason is that we use one network topology, the same set of agents, receivers and sources for the simulations.

When choosing α, we can mainly consider the tradeoffs between *Delayed Ratio* and *Success Ratio*. The other metrics such as *Join Latency* may also be taken into consideration depending on the requirements of the application. For $\alpha = 1.2$, we can achieve 100% *Success Ratio*; however, it has high *Delayed Ratio*. For $\alpha = 0.9$ or $\alpha = 1.0$, the *Delayed Ratio* is low even in the worst cases; however, the *Success Ratio* is low. We may choose $\alpha = 1.1$ or $\alpha = 1.0$, depending on which performance metrics we favor.

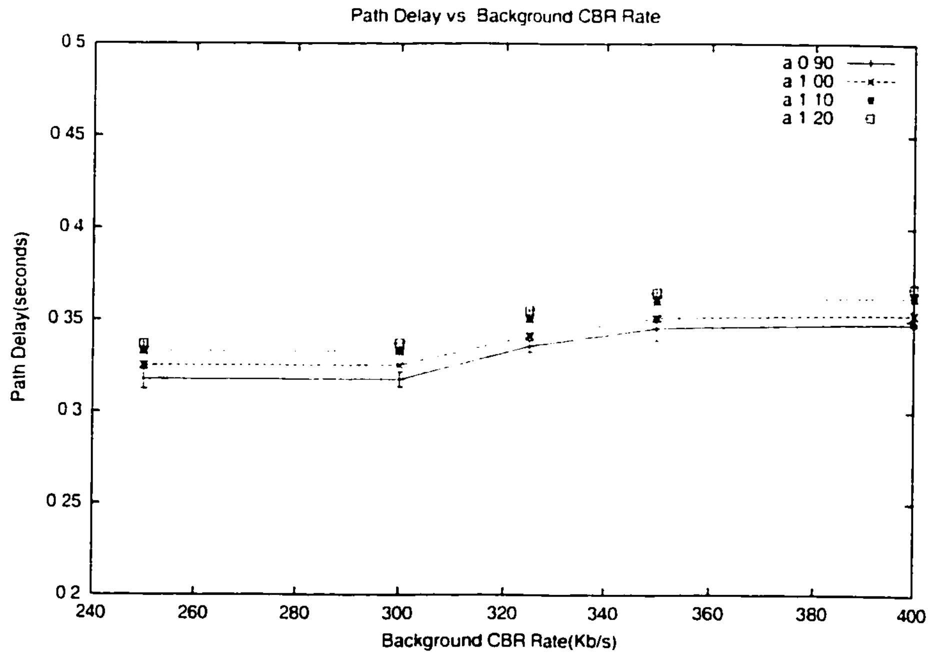


Figure 4.10: Average Path Delay vs. Background CBR Rate(α study)

The experiment results also indicate that it may be useful to choose α according to the background CBR rate. In our studied case, we can choose $\alpha = 1.1$, or even $\alpha = 1.2$ when background CBR rate is less than or equal to 300Kb/s in order to obtain high *Success Ratio*. When CBR rate is greater than 300Kb/s, we can choose $\alpha = 1.0$. In this way, we can achieve good performance with respect to both *Delayed Ratio* and *Success Ratio*.

Summary

Parameter settings affect the protocol performance. Generally speaking, we choose parameters to strike a balance among several metrics to achieve our favored performance. When choosing the intervals and thresholds for information gathering, we attempt to choose the setting that provides AQMR with low *Delayed Ratio* and low *Info Overhead*. We choose the number of agents and the Admission Factor(α) according to the goal of lowering *Delayed Ratio*, tree cost, *Join Latency* and *Join Overhead*, and/or the goal of lowering the protocol overhead for information gathering. We also attempt to approve a high ratio of join requests.

In the following comparison study with QoSMIC, we set the following parameters based on the characteristics study in the previous sections. We set *InfoInterval* = 1500 seconds, *TriggerThreshold* = 0.001; α = 1.1(which gives the protocol less than 15% delayed packets and more than 94% success ratio), and choose 9 agents(which gives the protocol good overall performance).

4.6 Implementation of QoSMIC

QoSMIC [FB98] uses *Local Search* and *Multicast Tree Search* to locate multiple on-tree routers. QoSMIC terms a chosen on-tree router as a candidate. The new router selects the best paths from the candidates to connect to the tree. The *Multicast Tree Search* can use a certain combination of techniques including *Directivity*, *Local Minima*, and *Fractional Choice* to lower the protocol overhead. In the following, we first discuss the switch-over problem in QoSMIC and then provide our approach to implement QoSMIC to make the performance comparison. We also discuss how to set the parameters of searching timers in QoSMIC.

4.6.1 Switch-over Problem in QoSMIC

QoSMIC[FB98] mentions a mechanism for switching from a shared tree to a source tree, which is similar to that in PIM-SM. However, due to its join mechanism, *Local Search* and *Multicast Tree Search*, to find the best path to connect to the tree, it may be problematic to implement such a switch-over mechanism for QoSMIC.

PIM-SM

PIM-SM uses the shortest path from the new member to the RP(Rendezvous Point, or core) to connect the new member to the tree. When a router has joined some source, say S, and has received packets from S, it will S-prune the shared tree towards the RP of the group. That is, this router, and possibly some routers on the way to the RP, will not receive S-packets from the shared-tree. Such an S-prune is required since the shared-tree serves not only for packets from S. When a new receiver wants to join the shared-tree, it may need to initiate the router state update procedure, so that it can receive data from S again. Therefore a control packet needs to be sent towards the RP.

In PIM-SM, such a switch-over and router state update are feasible, since all such control packets are forwarded along the reverse shortest path(from the unicast routing table) to the RP. For a detailed description, please refer to PIM-SM[EF98], Section 2.4.

QoSMIC

A new member in QoSMIC uses the best path it finds rather than the reverse shortest path to the core to connect to the shared tree. The shared-tree is a bi-directional tree, whose routing entry does not differentiate interfaces as incoming or outgoing, so that a router can not tell which interface leads to the core. In QoSMIC, after a router receives an S-prune packet, it will not forward shared-tree packets from S to affected interfaces. This S-prune packet will be forwarded somewhere upstream if necessary. When a new receiver comes, this router may need to update the router state of itself and possibly some upstream routers, so that it can receive packets from all the sources, similar to that in PIM-SM. A router can not always make proper necessary router state updates towards the core, since QoSMIC does not use the reverse shortest path to construct the tree as PIM-SM does. An example is provided below to illustrate the possible problem in updating router state in QoSMIC.

In Figure 4.11, *Receiver1* joins the shared-tree first, so *Router1* is on the shared-tree. When *Receiver2* wants to join, it gets the best path of *Router1* $\rightarrow$ *Router2* $\rightarrow$ *Router3* $\rightarrow$ *Router4* so that it joins the shared-tree via *Router1*. Thus *Router2*, *Router3* and *Router4* are on the shared-tree. Now, *Receiver2* switches from shared-tree to source-tree for *Source1*. After the source-tree for *Source1* works, *Receiver2* sends S-prune packets for *Source1* to the core. Now it knows the upstream router(s) since there are still *Source1* packets coming

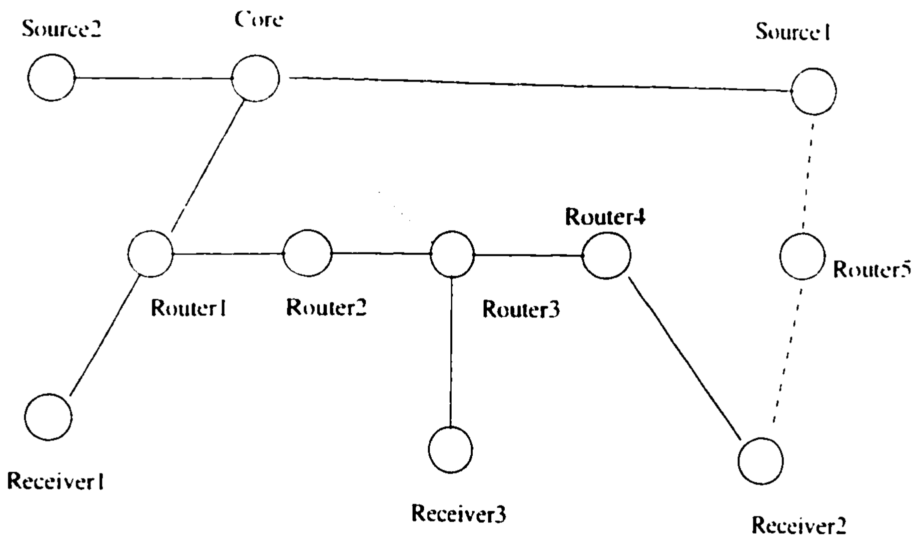


Figure 4.11: An Example of Switch-over Problem in QoSMIC

from the shared-tree. As a result, *Router4*, *Router3* and *Router2* will not forward packets from *Source1* to *Receiver2*. Note, *Receiver2* can not prune the shared-tree since it needs the shared-tree to forward packets from *Source2*. Finally, *Receiver3* wants to join the shared-tree. The best path happens to be *Router3*(only a node) for it to join. *Router3* knows it does not forward packets from *Source1*, so it needs to initiate the router state upstate procedure. Given the shortest path is the dotted line between itself and the core, *Router3* can not figure out that the adjacent upstream router for the shared-tree is *Router2*, unless it records such path information, or it relies on the data flow on the shared-tree to learn this. Otherwise, every on-tree router may need to record the adjacent upstream router for the shared tree. Or a router relies on the data flow on the shared-tree, which could not guarantee the timeliness of updating the routing state. QoSMIC does not provide a mechanism for a router in the shared-tree to learn its adjacent upstream router.

4.6.2 QoSMIC_Source and QoSMIC_Shared

To circumvent the switch-over problem, we implement two versions of QoS-MIC, pure source-tree QoSMIC (QoSMIC_Source) and pure shared-tree QoS-

MIC (QoSMIC_Shared). In the pure source-tree version, all receivers join the source trees; while in the pure shared-tree version, all receivers join the shared tree. That is, there is no switch-over from a shared-tree to a source-tree. We can envision that QoSMIC_Source can provide better QoS to receivers than QoSMIC_Shared, and also better than a hybrid version in which a receiver can be on both shared-tree and source-trees. Therefore, we can say AQMR outperforms QOSMIC if AQMR performs better or as well as QoSMIC_Source. QoSMIC_Source needs more routing entries than QoSMIC_Shared, however, it does not affect the performance results. We implement *Directivity* and *Local Minima* for *Distributed Selection* in the stage of *Multicast Tree Search* in QoSMIC [FB98].

4.6.3 Timers for Searching On-tree Routers

In QoSMIC, there are two timers associated with candidate searching, the one for *Local Search* and the one for *Multicast Tree Search*. Those two timers can affect the result of searching for candidates. We set the *Local Search* timer to either the average delay or the maximum delay of all the links in the topology, and the *Multicast Tree Search* timer to 20 times the *Local Search* timer for this study. Average delay in this case is 2 * (the transmission delay on one link + the average propagation delay). Note that we assume link bandwidth is the same for all the links. Maximum delay is 2 * (the transmission delay on one link + the maximum propagation delay). For simplicity, we call the two cases the maximum case and the average case. In the following comparisons, timers are set using the maximum delay, if not explicitly stated.

4.7 Comparison with QoSMIC

In this section, we present two sets of experiment results. In Set I, we vary the background CBR rate(from 250, 300, 325, 350, to 400 Kb/s) while keeping the source rate constant(200 Kb/s). In Set II, conversely, we vary source rates while keeping background CBR rate constant. In both of these two sets of experiments, we set receiver delay requirements to 4.5 seconds.

From Section 4.7.1 to Section 4.7.5, we show the experiment results of Set I. Results of Set II experiments will be shown in Section 4.7.6. We show only the maximum case for QoSMIC if the average case is very similar to the maximum case.

Table 4.7 shows the parameter settings for the Set I comparison study with QoSMIC.

Parameter	Value
Number of Agents	9
Admission Factor	1.1
InfoInterval	1500 seconds
TriggerThreshold	0.001
Source Rate(CBR)	200Kb/s
Background CBR rate	250, 300, 325, 350, 400 Kb/s
Delay Requirement	0.45 seconds

Table 4.7: Experiment Parameters(Comparison with QoSMIC: Set I)

4.7.1 Delayed Ratio

In Figure 4.12, we plot the ratio of delayed packets for QoSMIC_Shared, QoSMIC_Source, and AQMR with *Admission Factor* of 1.1 versus background CBR rate. As the background CBR rate increases, the three curves have similar trends, keeping constant first, then increasing slightly. This indicates that increasing background CBR rate can increase the ratio of delayed packet in QoSMIC and AQMR. The graph shows that AQMR has fewer delayed data packets than QoSMIC_Shared, and QoSMIC_Source, which provides the best QoS for the scheme of QoSMIC. In AQMR, an agent conducts the eligibility test based on the multicast tree information, so that it knows if the delay requirement will be satisfied or not with a certain level of confidence. QoSMIC chooses the best on-tree router for a receiver to join the multicast tree. The dynamic information is collected from an on-tree router, but not from the core(or sources) to the receiver. Therefore, it can not guarantee that the best path can provide satisfactory QoS to the receiver with such partial path

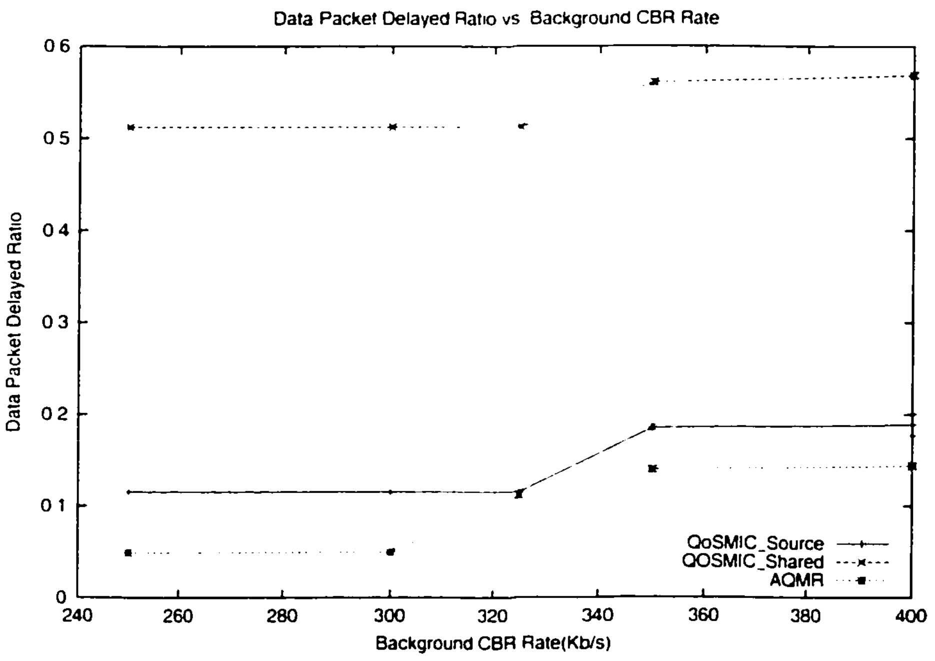


Figure 4.12: Delayed Packets Ratio vs. Background CBR Rate

information. AQMR provides better service to receivers than QoSMIC, since it has the mechanism of admission control of join requests, while QoSMIC does not have an equivalent mechanism.

Note that the *Delayed Ratio* depends on the delay requirements of receivers. When receivers have requirements of very low end-to-end delays, QoSMIC will perform even worse; while AQMR will deny many of the requests since the network could not accommodate them. When receivers have loose requirements of delay, both QoSMIC and AQMR will behave well in terms of *Delayed Ratio*, since most of the packets will arrive in time.

4.7.2 Success Ratio

In Figure 4.13, we plot the *Success Ratio* for QoSMIC_Shared, QoSMIC_Source, and AQMR with *Admission Factor* of 1.1 versus background CBR rate.

Figure 4.13 shows that in QoSMIC_Shared, all join requests are approved. In QoSMIC_Source, a small number of requests are rejected when the background CBR is beyond some value(350Kb/s). AQMR has more than 94%

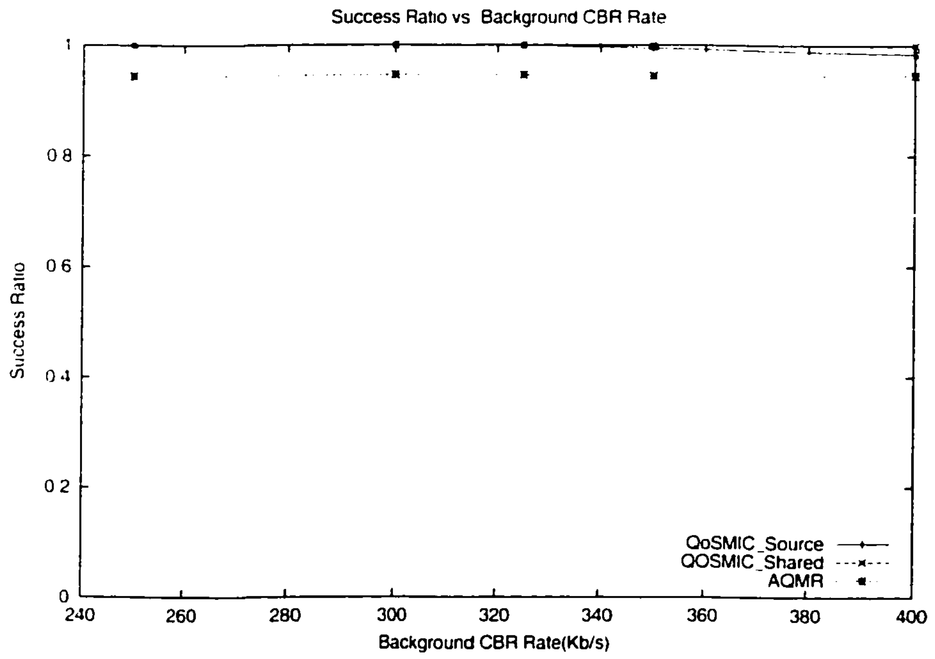


Figure 4.13: Join Success Ratio vs. Background CBR Rate

of requests being approved. Figure 4.12 and Figure 4.13 show that AQMR outperforms QoSMIC with respect to *Delayed Ratio*, at the cost of less than 6% *Success Ratio*. In AQMR, agents conduct eligibility tests for join requests. Those that fail to pass the test will be denied. QoSMIC chooses the best candidate for a receiver to connect to the tree, which may not satisfy the requirement of a receiver. The denial of join request in QoSMIC occurs when receivers fail to locate on-tree routers if timers are not long enough. In addition, the "partial join" in QoSMIC as discussed in Section 4.2 does have impact on this comparison, such that AQMR may have a lower *Success Ratio*, since an agent accepts or rejects a receiver's request for all the sources. AQMR can provide satisfactory service in terms of both *Delayed Ratio* and *Success Ratio*. We believe that how well a QoS routing scheme can support the required service is more important than the percentage of receivers it can accept to join the tree.

4.7.3 Join Latency

In Figure 4.14, we plot the *Join Latency* for QoSMIC_Shared, QoSMIC_Source, and AQMR with *Admission Factor* of 1.1 versus background CBR rate, using both maximum and average delay for timer settings in QoSMIC. Figure 4.14 shows that *Join Latency* does not change much for AQMR and QoSMIC_Shared, as the background CBR rate increases. For QoSMIC_Source, *Join Latency* increases as the background CBR rate increases, after some point(325Kb/s).

The figure shows that AQMR has a lower *Join Latency* than QoSMIC. In AQMR, a receiver can expect that it can receive a reply from its agent shortly after requesting to join the tree, since the agent is close to it, and the agent makes the decision as soon as it receives the join request. In QoSMIC, lucky receivers can get the reply quickly if it has routers not far away(two hops away as suggested by QoSMIC) on the tree. Otherwise, it needs to wait for the candidate(s) being selected by the *Multicast Tree Search*, which may take a long time. With a shorter timer, QoSMIC can either locate an on-tree router in less time using *Local Search*, or it can finish the phase of *Local Search* and resort to *Multicast Tree Search* more quickly. This accounts for the average case outperforming the maximum case as shown in the figure.

4.7.4 Protocol Overhead

To reduce the *Join Overhead* of QoSMIC, we introduce a new version of QoSMIC called QoSMIC_Manager. In QoSMIC_Manager, the *manager* assumes itself to be a candidate and notifies the receiver directly in the shared-tree, rather than initiating a *Multicast Tree Search*. We expect that in QoSMIC_Manager, the *Join Overhead* may be lower than that in QoSMIC_Shared, since the manager does not attempt to find other on-tree routers, which may cause many message hops.

In Figure 4.15, we plot the *Join Overhead* for QoSMIC_Shared, QoSMIC_Source, QoSMIC_Manager, and AQMR with *Admission Factor* of 1.1

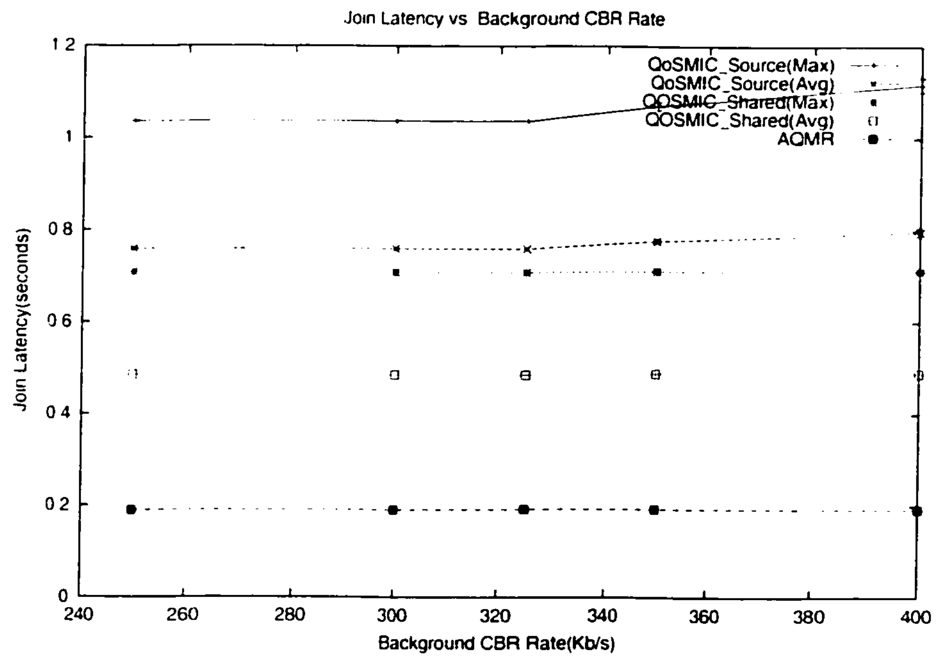


Figure 4.14: Join Latency vs. Background CBR Rate

versus background CBR rate, using the maximum delay setting in the QoS-MIC protocol. In Figure 4.16, we plot the same experiment results, using average delay for the timer setting in QoSMIC.

Figure 4.15 and Figure 4.16 show that AQMR outperforms QoSMIC. Figure 4.15 shows that in AQMR, the *Join Overhead* is nearly constant(around 6 hops) on average. This is expected since the receivers contact the agents to join the tree. With strategically located agents, it needs only a few hops before a receiver knows the join decision from the agent associated with it. QoSMIC needs more hops for a receiver to locate an on-tree router and receive feedback, for both QoSMIC_Source and QoSMIC_Shared. In QoSMIC, every time a receiver wants to join the tree, it needs to conduct a *Local Search* first. If this fails, it has to rely on the *Multicast Tree Search*, which may incur many hops before a candidate has been selected. The process of *Multicast Tree Search* may not terminate even after a good candidate appears due to its distributed nature. Even in the case of QoSMIC_Manager, the *Join Overhead* is more than 11 hops. The reason is that on average the *manager* in QoSMIC

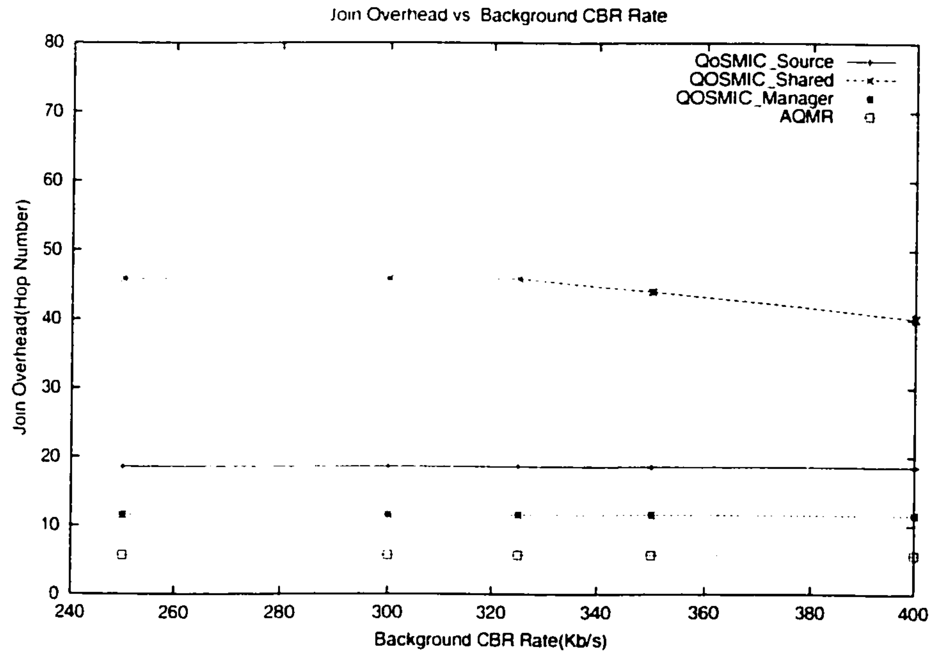


Figure 4.15: Join Overhead vs. Background CBR Rate

is located farther away from receivers than the agents are in AQMR. The two figures also show that QoSMIC works better in the maximum case than in the average case, since the longer timer gives it more chances to locate candidates in the phase of *Local Search*, thus less chances to initiate the *Multicast Tree Search*.

On the other hand, AQMR has the protocol overhead for QoS information gathering. Table 4.8 shows the extra overhead(message hops) that AQMR has for all information gathering in each simulation run, and their corresponding resource consumption. In each run, we have more than 5,000,000 data packets and each packet has an average of around 8 hops from a source to a receiver(see Section 4.7.5). Roughly, the information overhead costs, e.g. for the case of background CBR rate is 250Kb/s, $2151.4/5,000,000 = 0.043\%$ of the network resources, which is insignificant. Moreover, if we mainly use data packets for QoS gathering, with the assistance of some control packets, we can achieve much lower overhead.

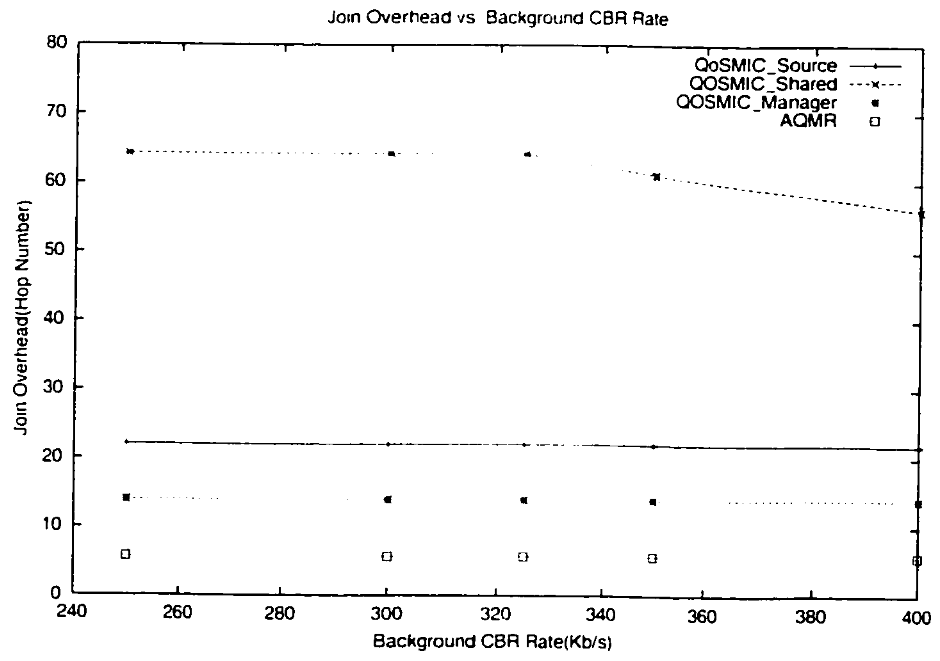


Figure 4.16: Join Overhead vs. Background CBR Rate(Average Case)

Background CBR Rate(KB/s)	250.0	300.0	3250.0	350.0	400.0
Information Overhead	2151.4	2138.3	2225.0	2226.5	2280.4
Resource Consumption	0.043%	0.043%	0.045%	0.045%	0.046%

Table 4.8: Information Gathering Overhead(Varying CBR Rate)

4.7.5 Tree Cost

We investigate the tree cost by two metrics. *Path Length* and *Path Delay*. In Figure 4.17, we plot the *Path Length* for QoSMIC_Shared, QoSMIC_Source, and AQMR with *Admission Factor* of 1.1 versus background CBR rate. In Figure 4.18, we plot the *Path Delay* for QoSMIC_Shared, QoSMIC_Source, and AQMR with *Admission Factor* of 1.1 versus background CBR rate.

Figure 4.17 shows that the *Path Length* keeps constant when the background traffic increases, which is expected, since the routes remain the same. Figure 4.18 shows that the *Path Delay* keeps constant for a while, then increases slightly, as the background CBR traffic increases. The reason is that as the background CBR rate increases, there is more traffic on some links, such that some packets take a longer time to reach receivers. As shown in

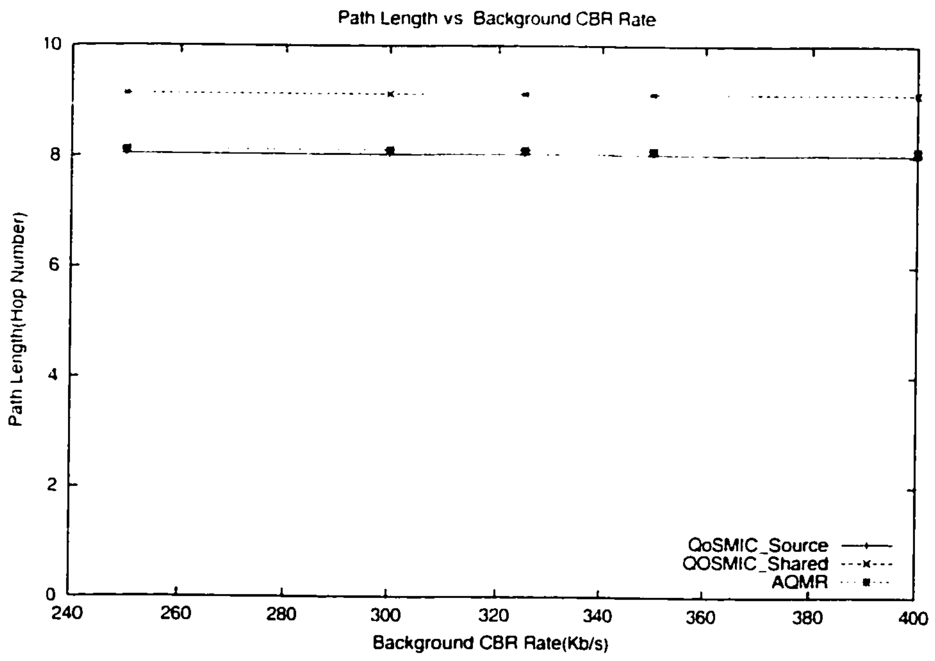


Figure 4.17: Average Path Length vs. Background CBR Rate

Figure 4.17. AQMR requires about 8 hops on average for a data packet to reach receivers, which is similar to QoSMIC_Source. QoSMIC_Shared needs around 9 hops. AQMR and QoSMIC_Source need about 0.34 seconds for a data packet to travel over the multicast tree from a source to a receiver, which is lower than what QoSMIC_Shared needs, more than 0.4 seconds, as shown in Figure 4.18. Therefore, AQMR constructs more efficient trees than QoSMIC.

4.7.6 Experiment Results: Set II

In this set of experiments, we vary the data rates of sources, from 50, 100, 150 to 200 Kb/s, while keeping the background CBR rate constant(325 Kb/s). Parameter settings are shown in Table 4.9. We set the largest data rate to 200 Kb/s, which can make the link bandwidth consumption almost to its capacity. There may be 6 data flows and 1 background flow on a link, which consume 6*200Kb/s + 325Kb/s = 1.525Mb/s: while the link bandwidth is 1.544Mb/s.

In Figure 4.19, we plot the ratio of delayed packets for QoSMIC_Shared, QoSMIC_Source, and AQMR with *Admission Factor* of 1.1 versus source data

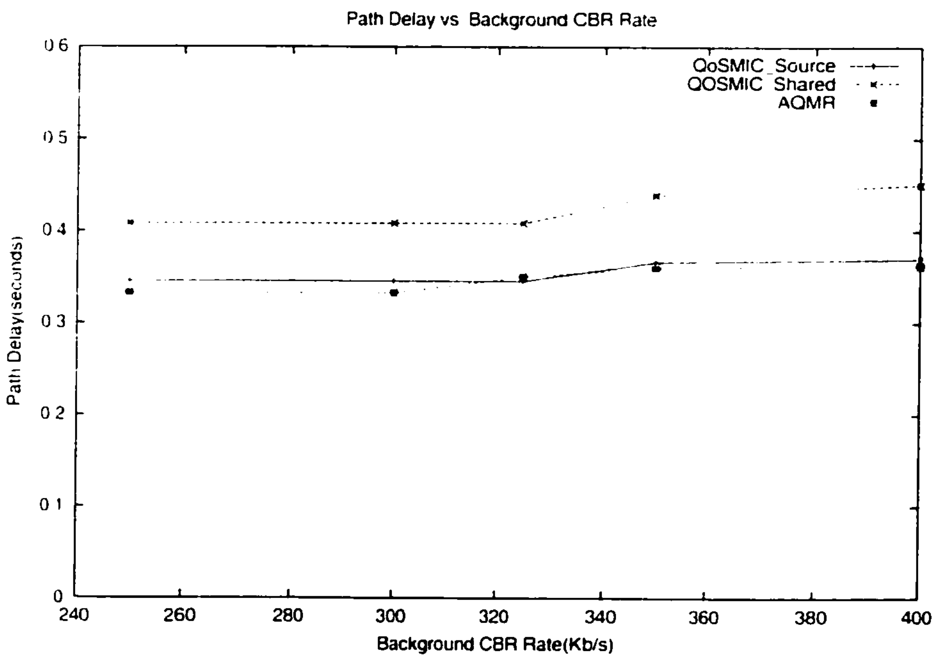


Figure 4.18: Average Path Delay vs. Background CBR Rate

Parameter	Value
Number of Agents	9
Admission Factor	1.1
InfoInterval	1500 seconds
TriggerThreshold	0.001
Source Rate(CBR)	50, 100, 150, 200 Kb/s
Background CBR rate	325 Kb/s
Delay Requirement	0.45 seconds

Table 4.9: Experiment Parameters(Comparison with QoSMIC: Set II)

rate. Figure 4.19 shows that with the increase of source data rate, the ratio of delayed packets tends to increase slightly for QoSMIC_Shared, QoSMIC_Source, and AQMR. AQMR performs better than QoSMIC_Source(at data rate of 200Kb/s, they are very close), and much better than QoSMIC_Shared.

In Figure 4.20, we plot the *Success Ratio* for QoSMIC_Shared, QoSMIC_Source, and AQMR with *Admission Factor* of 1.1 versus source data rate. Figure 4.20 shows that QoSMIC_Shared and QoSMIC_Source accept all requests. AQMR approves almost all requests(98.3%) at the data rates of 50.0, 100.0 and 150.0 Kb/s. At the data rates of 200 Kb/s, it rejects 3.4% join requests. AQMR

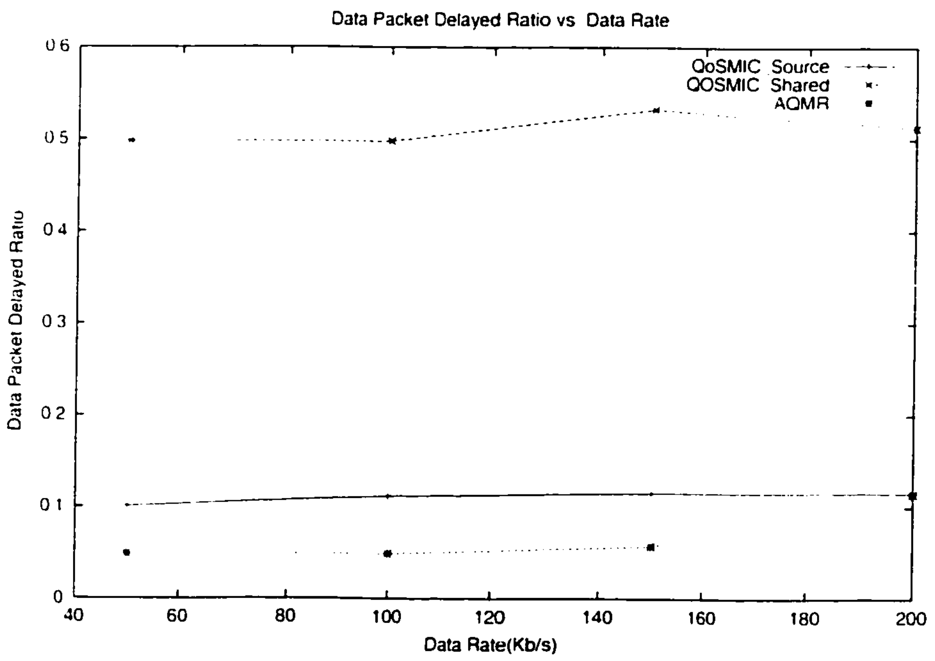


Figure 4.19: Delayed Packets Ratio vs. Source Data Rate

constructs a more efficient tree than QoSMIC(as shown in Figure 4.22). Furthermore, the admission control mechanism allows AQMR to experience fewer delayed packets and therefore meet QoS objectives better than QoSMIC.

In Table 4.10, we show the result of the *Join Latency* for QoSMIC_Shared, QoSMIC_Source, and AQMR with *Admission Factor* of 1.1 versus source data rate, using both maximum and average delay for the timer setting in QoSMIC. Table 4.10 indicates that in AQMR, receivers can get the join decision faster than in QoSMIC. The table also shows that QoSMIC works better in the average case than in the maximum case.

Data Rate(KB/s)	50.0	100.0	150.0	200.0
AQMR	0.1865	0.1870	0.1881	0.1865
QoSMIC_Source(Max)	1.0247	1.0284	1.0331	1.0370
QoSMIC_Source(Avg)	0.7473	0.7512	0.7558	0.7600
QoSMIC_Shared(Max)	0.7040	0.7057	0.7104	0.7087
QoSMIC_Shared(Avg)	0.4807	0.4827	0.4870	0.4857

Table 4.10: Comparison: Join Latency

In Table 4.11, we show the result of the *Join Overhead* for QoSMIC_Shared,

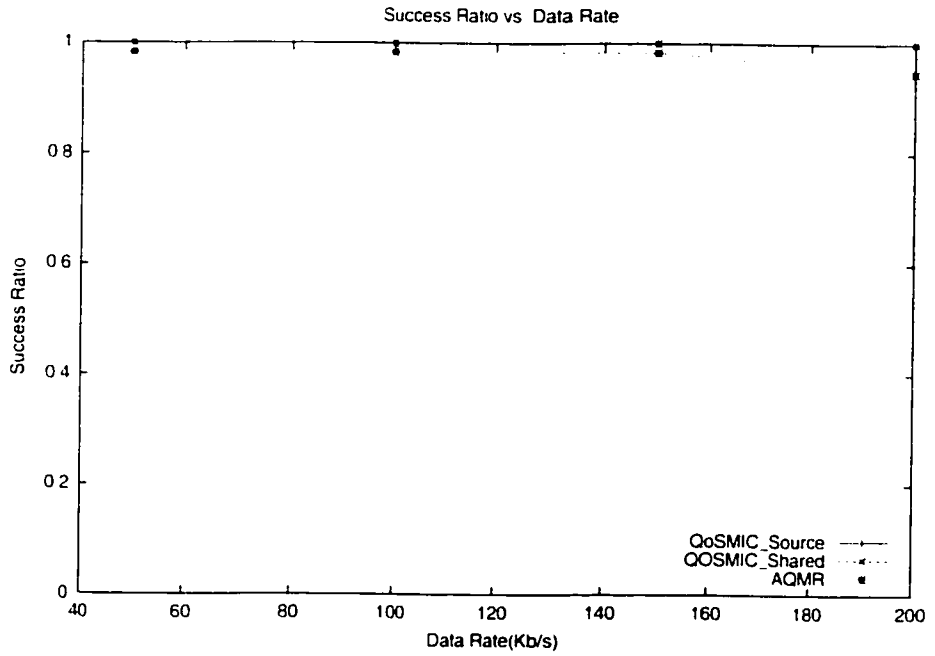


Figure 4.20: Join Success Ratio vs. Source Data Rate

QoSMIC_Source, QoSMIC_Manager, and AQMR with *Admission Factor* of 1.1 at source data rate of 200Kb/s, for both the maximum and average cases for QoSMIC. The protocol overhead changes slightly for different data rates for each of the algorithms in the table. Table 4.11 indicates that AQMR outperforms QoSMIC. It also shows that QoSMIC works better in the maximum case than in the average case for QoSMIC_Source, QoSMIC_Shared and QoSMIC_Manager. Table 4.12 shows the protocol overhead for QoS information gathering packets over the entire simulation and their corresponding network resource consumption(see Section 4.7.4).

In Figure 4.21, we plot the *Path Length* for QoSMIC_Shared, QoSMIC_Source, and AQMR with *Admission Factor* of 1.1 versus source data rate. In Figure 4.22, we plot the *Path Delay* for QoSMIC_Shared, QoSMIC_Source, and AQMR with *Admission Factor* of 1.1 versus source data rate. The two figures show that AQMR performs similarly as QoSMIC_Source, but better than QoSMIC_Shared. Therefore AQMR outperforms QoSMIC in terms of the tree cost.

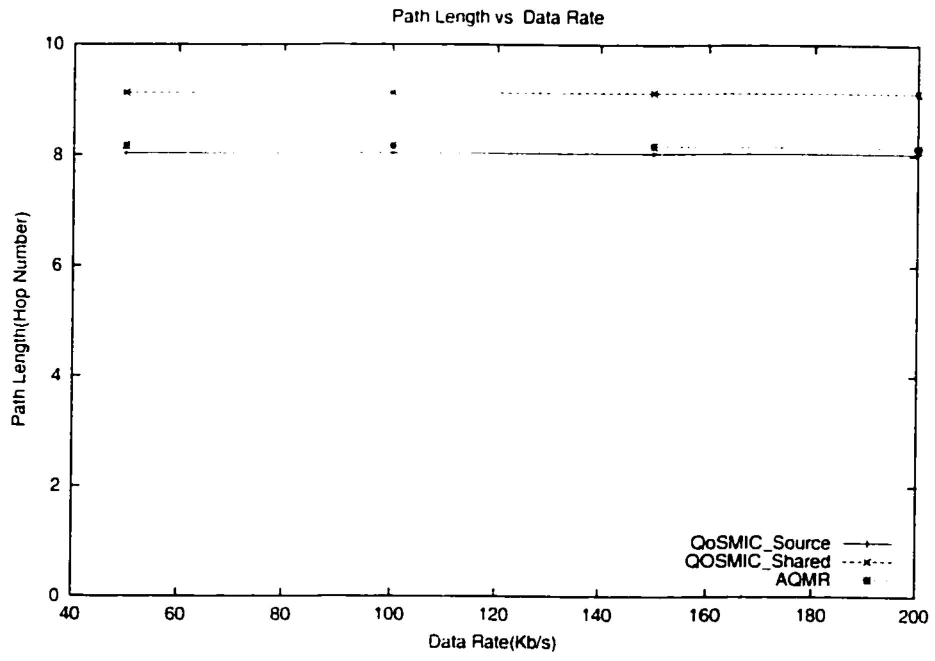


Figure 4.21: Average Path Length vs. Source Data Rate

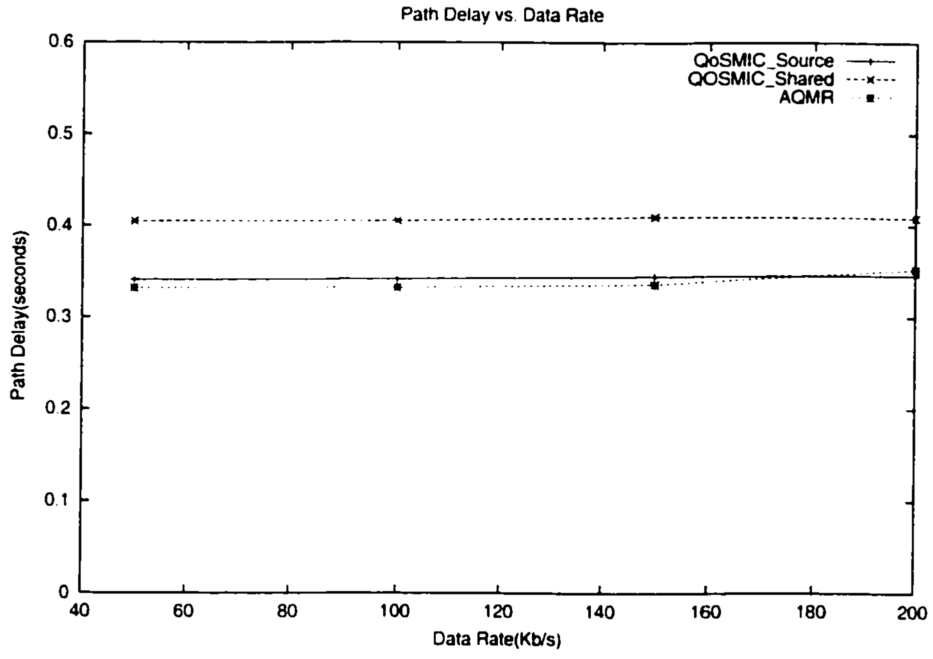


Figure 4.22: Average Path Delay vs. Source Data Rate

Algorithm	Join Overhead
AQMR	5.690
QoSMIC_Source(Max)	18.5690
QoSMIC_Source(Avg)	22.0517
QoSMIC_Shared(Max)	45.8793
QoSMIC_Shared(Avg)	64.2241
QoSMIC_Manager(Max)	11.5517
QoSMIC_Manager(Avg)	13.9655

Table 4.11: Comparison: Join Overhead

Data Rate(KB/s)	50.0	100.0	150.0	200.0
Information Overhead	2544.0	2544.0	2544.0	2225.0
Resource Consumption	0.051%	0.051%	0.051%	0.045%

Table 4.12: Information Gathering Overhead(Varying Data Rate)

4.8 Summary

In this chapter, we conduct performance study of the characteristics of AQMR and the comparison with QoSMIC.

The characteristics study of AQMR shows that parameter settings affect the protocol performance. AQMR can provide satisfactory QoS to applications, by tuning the admission control(the Eligibility Test and the *Admission Factor*, α) according to the application requirement and the background traffic.

We also compare AQMR with QoSMIC(QoSMIC_Source and QoSMIC_Shared). As the background CBR rate increases, the *Delayed Ratio* and *Path Delay* keep constant first, then increase slightly; the other performance metrics keep constant or change slightly. By varying the source data rate, we can obtain similar results. We summarize the comparison results in Table 4.13.

The above comparison shows that AQMR works well in the studied cases. AQMR can accept more than 94% of the join requests. At the same time, there are less than 15% delayed packets, which is better than QoSMIC_Source, and much lower than QoSMIC_Shared(more than 50%). The mechanism of admission control in AQMR accounts for AQMR outperforming QoSMIC in terms of *Delayed Ratio*. AQMR can construct multicast trees as efficiently as

Performance Metric	Comparison with AQMR	
	QoSMIC_Source	QoSMIC_Shared
Delayed Ratio	higher	much higher
Success Ratio	about 5% more	about 5% more
Path Length	similar	higher
Path Delay	similar	higher
Join Latency	much higher	higher
JoinOverhead	higher	much higher

Table 4.13: Summary of Comparison Results

QoSMIC_Source, and much better than QoSMIC_Shared, with respect to *Path Delay* and *Path Length*. Moreover, the *Join Latancy* and *Join Overhead* are low, due to the introduction of agents. Similar results are also obtained by setting receiver requirements randomly between 4.0 seconds and 5.0 seconds. Therefore, the experiment results show that AQMR outperforms QoSMIC.

On the other hand, AQMR has the extra overhead for QoS information gathering. It does not consume much network resources as calculated. Furthermore, it can be reduced dramatically if using data packets to carry QoS information. In addition, AQMR has more configuration work than QoSMIC. In AQMR, sources and receivers need to know who are their agents, and an agent needs to know where the other agents are to exchange information and to set up paths. In QoSMIC, receivers need to know only the location of the manager.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

In this thesis, we design the Active QoS Multicast Routing Protocol(AQMR) using distributed agents. With strategically located agents, AQMR can gather QoS information and make join decisions in a scalable manner. It supports various QoS metrics, including end-to-end delay, loss rate, and bandwidth. AQMR conducts eligibility tests responsively for join requests based on the QoS information stored at agents locally. It constructs resource-efficient multicast trees, with low join overhead and low join latency.

We discuss the implementation and applications of Active Networks, which will potentially be a promising computation platform. A multicast tree can be a shared-tree and/or a source-tree. Multicast routing schemes can be classified into distributed, centralized and hybrid schemes. They can also be categorized as QoS-unaware and QoS-aware schemes. There are challenges to provide QoS in multicast, such as how to meet QoS requirements of receivers, how to gather QoS information for a protocol to make proper decisions for join requests and how to lower the protocol overhead for joining a multicast tree.

In AQMR, agents are strategically placed active nodes in the multicast tree. It deploys periodical mechanisms combined with trigger-based approaches for QoS information gathering, and for path computation. A source sends QoS information collection packets periodically to its agent. Agents exchange source QoS information among themselves to obtain a complete view of the multicast tree so that they can compute paths to all the sources. A new member contacts the agent associated with it to join the tree. The agent determines whether to accept the request or not. AQMR is designed to be QoS-aware, scalable, efficient and responsive. To be QoS-aware, it has mechanisms to gather QoS information and to make join decisions. Its architecture supports additive, multiplicative and concave QoS metrics. The scalability, responsiveness and efficiency are provided by strategically selected agents.

After describing the performance methodology and implementation issues, we study the characteristics of AQMR in terms of intervals and triggers, number of agents, and *Admission Factor*. We then compare AQMR with QoS-MIC with respect to the performance metrics of *Delayed Ratio*, *Success Ratio*, *Path Length*, *Path Delay*, *Join Latency*, *Join Overhead* and *Info Overhead*. The performance study shows that AQMR can provide better QoS than QoS-MIC and can construct resource-efficient trees, with low *join overhead* and low *join latency*. The performance study confirms that the design objectives are achieved. Although AQMR has the extra overhead for QoS information gathering, it does not consume much network resources, and can be reduced dramatically by using data packets to carry QoS information.

5.2 Future Work

The current version of AQMR can be refined and extended in the following directions.

- The parameters of AQMR need further investigation. The protocol parameters for intervals and triggers are set in a way that they are dependent. We will study the effect of setting these parameters to different, independent values on the performance of the protocol.

- We use a single topology generated by GT-ITM to conduct the performance study. It would be more convincing to evaluate the performance of a protocol on various topologies. We may choose topologies from big

ISPs or Mbone. We may also use the power-law[FF99] to create topologies resembling the Internet.

- We have made suggestions to choose agents in a multicast group. However, we need to come up with approaches that select strategically located agents automatically, and compare their effects on the protocol performance. We will attempt to design heuristic schemes to choose agents, since generally we lack global topology information. Also of interest is to change the number of agents based on the change in membership.

- In the current version of AQMR, only agents store multicast tree QoS information, therefore only agents can make join request decisions. We can choose some routers between an agent and leaf nodes of the tree to store QoS information, so that they can help agents make join request decisions. That is, such routers can intercept join requests addressed to agents. The join latency can be shortened. However we need to study the tradeoff since there will be more routers storing QoS information.

- If a join request fails to pass the eligibility test, AQMR rejects the request. There may be a chance for the receiver to join the tree through other agents, since it is hard to place agents on optimal points of the multicast tree. Such "cooperative join" may be beneficial since it may increase the successful ratio of join requests, however, it also increases protocol complexity and protocol overhead. The tradeoff needs further investigation.

- Measuring QoS metrics causes protocol overhead. Currently we use explicit control messages to gather QoS information. We can study the approach that mainly uses data packets to carry such information. At the same time, assistance from control messages is also needed since some sources may send data at low rates, which may make the QoS information stored at agents out-of-date. By using data packets and control messages together for QoS information, we can achieve lower protocol overhead and up-to-date information.

- Active Networks technologies cause processing overhead, which has not been studied in this thesis. We may extend our work to an Active Networks platform such as PLAN to evaluate the processing cost and the tradeoff with protocol gains.

Bibliography

[AL00] K. Almeroth, "A Long-Term Analysis of Growth and Usage Patterns in the Multicast Backbone(MBone)", *Proceedings of IEEE INFOCOM'00*, March 2000.

[AL00-2] K. Almeroth, "The Evolution of Multicast: From the MBone to Interdomain Multicast to Internet2 Deployment", *IEEE Network*, 14(1):10-20, January/February 2000.

[AG98] G.Apostolopoulos, R. Guerin, S. Kamat, S. Tripathi, "Quality of Service Based Routing: A Performance Perspective", *Proceedings of ACM SIGCOMM'98*, pages 17-28, September 1998.

[AM98] E. Amir, S. McCanne, R. Katz, "An Active Service Framework and its Application to Real-time Multimedia Transcoding", *Proceedings ACM SIGCOMM'98*, pages 178-189, September 1998.

[BA97] A. Ballardie, "Core Based Trees (CBT) Multicast Routing Architecture", RFC 2201, September 1997.

[BH01-1] Baochun Bai, Janelle Harms, and Yuxi Li, "Active Hierarchical Congestion Control for Large-scale Multicast" *Submitted for review*.

[BH01-2] Baochun Bai, Janelle Harms, and Yuxi Li, "Active Error Recovery For Reliable Multicast" *Proceedings of IEEE ICCCN'01*, 2001.

[CA99] Anddrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente, and Daniel Villela, "A Survey of Programmable Networks", *ACM SIGCOMM Computer Communication Review*, 29(2):7-23, April 1999.

[CC'97] K. Carlberg, J. Crowcroft. "Building shared trees using a one-to-many joining mechanism". *ACM Communication Review*, pages 5-11, January 1997.

[CC'98] K. Carlberg, J. Crowcroft. "Quality of Multicast Service(QoMS) by Yet Another Multicast (YAM) Routing Protocol". Proceedings of HIPPARCH'98, June 1998.

[CC'99] K. Carlberg, J. Crowcroft. "Examining the Construction of Shared Trees Using Different Metrics". *Proceedings of IEEE RTAS'99 Workshop*, June 1999.

[CD99] R. Caceres, N.G. Duffield, J. Horowitz, D. Towsley. "Multicast-based inference of network-internal loss characteristics". *IEEE Transactions on Information Theory*, 45(7):2462-2480, November 1999.

[CS00] Shigang Chen and Yuval Shavitt. "A Scalable Distributed QoS Multicast Routing Protocol". *DIMACS Technical Report: 2000-18*, August 2000.

[CD97] K. Calvert, M. Doar and E. Zegura. "Modeling Internet Topology", *IEEE Communication Magazine*, 35(6):160-163, June 1997.

[CN01] Shigang Chen, Klara Nahrstedt, Yuval Shavitt, "A QoS-aware Multicast routing protocol", *IEEE Journal on Selected Areas in Communications(Special Issue on Internet QoS)*, 18(12):2580-2592, December 2000.

[DC'90] Stephen Deering and David Cheriton "Multicast routing in datagram internetworks and extended LANs" *ACM SIGCOMM'88*, pages 55-64, August 1988.

[DE89] S. Deering, "Host Extensions for IP Multicasting", *RFC1112*, 1989.

[DE99] S. Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ahmed Helmy, David Meyer, Liming Wei, "Protocol Independent Multicast Version 2 Dense Mode Specification", *Internet draft*, 1999.

[EF98] D. Estrin, et al., "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification", *RFC 2362*, June 1998.

[ER94] Hans Eriksso, "MBONE: the multicast backbone " *Communications of the ACM*, 37(8):54-60, August 1994.

[FB98] Michalis Faloutsos, Anindo Banerjea, Rajesh Pankaj, "QoSMIC: Quality of Service sensitive Multicast Internet protoCol", *Proceedings of ACM SIGCOMM'98*, pages 144-153, September 1998.

[FF99] Michalis Faloutsos, Petros Faloutsos, Christos Faloutsos, "On Power-Law Relationships of the Internet Topology", *Proceedings of ACM SIGCOMM'99*, pages 251-262, September 1999.

[GO97] R. Guerin, Ariel Orda, and D. Williams, "QoS Routing Mechanisms and OSPF Extensions", *Proceedings of IEEE Globecom'97*, November 1997.

[HK98] M. Hicks, P. Kakkar, J. Moore, C. Gunter, and S. Nettles, "PLAN: A Packet Language for Active Networks" *Proceedings of the International Conference on Functional Programming (ICFP) '98*, pages 86-93, September, 1998.

[HM99] M. Hicks, J. Moore, D. Alexander, C. Gunter, and S. Nettles, "PLANet: An Active Internetwork" *Proceedings IEEE INFOCOM'99*, pages 1124-1133, March 1999.

[HT99] J. Hou, H. Tyan, B. Wang, Y. Chen, "QoS Extension to CBT", *Internet Draft*, March 1999.

[JA91] Raj Jain, "The art of computer systems performance analysis : techniques for experimental design, measurement, simulation, and modeling", *John Wiley and Sons, Inc.*, New York, 1991.

[JJ00] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, L. Zhang, "On the Placement of Internet Instrumentation", *Proceedings of IEEE INFOCOM'00*, pages 26-30, March 2000.

[KP93] Vachaspathi P. Kompella, Joseph C. Pasquale and George C. Polyzos. "Multicast routing for multimedia communication". *IEEE/ACM Transactions on Networking*, 1(3):286-292, Jun. 1993.

[LH01] Yuxi Li, Janelle Harms, and Baochun Bai. "An Active QoS Multicast Routing Protocol" *Submitted for review*.

[LG98] L. Lehman, S. Garland and D. Tennenhouse. "Active Reliable Multicast" *Proceedings of IEEE INFOCOM'98*, April 1998.

[ML01] N. Maxemchuk, S. Low, "Active Routing", *IEEE Journal on Selected Areas in Communications*, 19(3):552-565, March 2001.

[MO94] J. Moy, "Multicast extension to OSPF", *RFC 1584*, 1994.

[PL99] R. Perlman, C. Lee, A. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, J. Thoo, M. Green, "Simple Multicast: A Design for Simple, Low-Overhead Multicast", *Internet Draft*, October 1999.

[PZ98] Mehrdad Parsa, Qing Zhu and J. J. Garcia-Luna-Aceves, "An iterative algorithm for delay-constrained minimum-cost multicasting", *IEEE/ACM Transactions on Networking*, 6(4):461-474, August 1998

[QU98] B. Quinn, "IP Multicast applications: Challenges and solutions", *Internet Draft, work in progress*, November 1998.

[RA00] M. Ramalho, "Intra- And Inter-Domain Multicast Routing Protocols: A Survey and Taxonomy", *IEEE Communications Surveys and Tutorials*, 3(1):2-25, January-March 2000.

[SA99] Stefan Savage, "Sting: a TCP-based Network Measurement Tool", *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, pages 71-79, October 1999.

[TA96] Andrew S. Tanenbaum, "Computer Networks(Third Edition)", *Prentice Hall, Inc.*, 1996.

[TH99] H. Tyan. C. Hou. B. Wang. "A Framework for Provisioning of Temporal QoS in Core-Based Multicast Routing". *IEEE 20th Real-Time System Symposium(RTSS'99)*. pages 168-178. December 1999.

[WG98] D. Wetherall. J. Guttag. D. Tennenhouse. "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols". *IEEE OPENARCH*. pages 117-129. April 1998.

[WH00] B. Wang. C. Hou. "Multicast Routing and Its QoS Extension: Problems. Algorithms. and Protocols". *IEEE Network*. 14(1):22-36. January/February 2000.

[WP88] D. Waitzman. C. Partridge. S. Deering, "Distance Vector Multicast Routing Protocol" *RFC 1075*. 1988.

[YF99] Shuqian Yan. Michalis Faloutsos, Anindo Banerjea. "QoS-Sensitive Routing for Real-time Multicasting" *Real-Time Internet Applications Workshop (RTAW'99)*, Vancouver. June 1999.

[ZD93] L. Zhang. S. Deering, D. Estrin, S. Shenker, and D. Zappala. "RSVP: A New Resource ReSerVation Protocol", *IEEE Networks*. 7(5):8-18. September 1993.

[NS] http://www.isi.edu/nsnam/ns/

[PA] PANAMA, http://www.tascnets.com/panama/