USING R-ACM METHOD IN QOS-BASED ROUTING

by

Mohammad-Reza Nassaji-Matin, B.Sc.

A thesis submitted to the

Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Information and Systems Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

September 2005

Your file Votre référence
ISBN: 0-494-10143-1
Our file Notre référence
ISBN: 0-494-10143-1

DEDICATION

To my beloved parents Ali and Shamsi, my lovely wife Shirin, and our yet to be born daughter Yalda.

ABSTRACT

Update policies are used in the context of the QoS-based routing to enhance the performance of the network by governing the frequency of sending Link State Advertisements.

In this work, we introduce new update policies based on the Rectangular Attribute Cardinality Map (R-ACM, US Patent No. 6,865,567) method, introduced by Oommen and Thiyagarajah. We call our update policies the *Traditional R-ACM* and the *Adaptive R-ACM update policies.* Using *MaRS* as the simulation tool, we analyze the performance of these methods along with the existing policies using different parameter sets.

It is shown that update policies can help in improving the performance of the network, yet they introduce a new deterministic type of inaccuracy in the path determination process. To offset that, we have optimized the sensitivity of our new adaptive update policy by incorporating it with a new R-CAM compatible tailor-made safety-based QoS-based routing.

ACKNOWLEDGMENTS

I would like to express my gratitude to my Professor, Dr. John Oommen, for being my guide during the term of this project, for his continuous support and encouragement.

I would also like to thank the Professors of the examining committee of my Thesis, Dr. Anil Somayaji, Dr. Babak Esfandiari, Dr. Michiel Smid, and Dr. Jean-Pierre Corriveau for their support and outstanding feedbacks that helped me in rectifying various errors in the early drafts of this project.

I would also like to thank my late father for his inspiration, my mother for her encouragement, and my wife for being helpful and patient during this time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: INTRODUCTION

1.1 General Description

1.1.1 Routing

In the internetworking context, r*outing* is the process of sending data from a source to a destination, usually in the form of data packets. In this process, network components called *Routers* take part in passing the packets of data on their way from the source to a destination by taking advantage of *Routing Algorithms*, which are in charge of initializing and maintaining *Routing Tables* used in the optimal path determination process [19].

1.1.1.1 Routing Algorithms and Routing Tables

In the process of routing, a path has to be first determined for any request to be sent from a source to a destination, or forwarded by the network components along the way. *Routing protocols* implement *routing algorithms* to initialize and maintain small databases called *routing tables to be used* in the path determination process, because calculating the paths upon the arrival of relentless requests is not cost efficient. Therefore routing algorithms use routing tables that keep a record of the best possible routes to all

of the destinations in a network, along with the costs associated with those routes using *routing metrics*. Well-known routing metrics are the available bandwidth, time delay, hop count, path cost, path load, reliability, and communication cost, and sometimes a combination of these is calculated as a "hybrid" metric [19].

1.1.1.2 QoS-Based Routing

There is a relatively new concept in networking (compared to *pure* routing) and that is the notion of Quality-of-Service (or in short QoS). QoS deals with measuring the performance of a system based on a predefined contract that defines the quality or quantity boundaries for a service between a service provider and the user applications. In this context, a set of constraints is defined which specifies the QoS type requirements of a request. The most important task of QoS-based routing is to find an optimal path for a new request that satisfies all the given constraints. In general, in the context of the QoS-based routing, almost all of the routing algorithms are designed to optimize the utilization of the network resources while searching for optimal paths. In other words they tend to incorporate the concerns of QoS into optimality concerns in routing. It is often difficult to design an optimal QoS-based routing algorithm due to the interaction between its multiple constraints while addressing the effect of the dynamics of the network on having up-to-date information [19][24].

1.1.1.3 Update Policies

In any network, in order to have effective QoS-based routing algorithms, routers need to frequently exchange the link state information in order to update their routing tables with the most up-to-date data. However, the frequency of the exchange of the

information for more efficient routing purposes has significant influence on the usage of the network resources. *Link state update policies* were introduced as the mechanism to efficiently generate and effectively advertise the link state update information (LSA). The link state update policy may be either periodic or trigger-based. In the case of the periodic update policy, each node in the topology periodically advertises information on its available link resources. For the case of the trigger-based policy, each node monitors the utilized link resources and triggers the LSA upon a "significant" change. In terms of exchanging link-state information, sending periodic update messages has a predictable network overhead, where as sending triggered update messages will result in more accurate link-state information [6][19].

Trigger-based update policies can be either threshold-based or class-based. Class-based update policies can further be classified into equal-class based and exponential-class based update policies [6].

In this work we will introduce two new trigger-based update policies utilizing the Rectangular Attribute Cardinality Map (R-ACM) algorithm, a patented technology (US Patent No. 6,865,567 issued on 8th of March 2005) [62], namely the Traditional R-ACM Trigger-Based update policy, and the Adaptive R-ACM Trigger-Based update policy.

1.1.1.4 QoS-Based Routing and Inaccuracy in Link-State Information

In any given network, inaccurate link-state information of its routers has a huge impact on the performance of its QoS-based routing algorithm. Hence, QoS-based routing algorithms must be incorporated with additional mechanisms so as to acquire the ability of offsetting the impacts of inaccurate information regarding the state of the links of the routers, resulting in better performance in their path determination process. The use of

different QoS-based routing methods has been suggested to minimize the effect of inaccurate state information. These methods include, but are not limited to, safety–based routing, randomized routing, multi–path routing and localized routing [5].

We will introduce a novel safety–based routing method in the Chapter 5, and we will incorporate safety-based routing applications into our adaptive R-ACM trigger-based update policy.

1.1.2 R-ACM versus Traditional Histograms

A *histogram* is a graphic representation of the frequency distribution of a continuous variable. Histograms are used in different applications like data mining, query result size estimation, to mention a couple. Traditional histogram methods are of the equi-width and equi-depth (also called equi-height) families [31][64].

A new histogram method was proposed in [59], [61] and [62], called the Attribute Cardinality Map (ACM) which then was awarded a US Patent whose details are No. 6,865,567 issued on 8th of March 2005. The Attribute Cardinality Maps (ACMs) can be used for query optimization as well as for other applications. The methods are based on the philosophies of numerical integration; therefore it is shown that these models have been analytically more accurate than the traditional histograms known earlier [59][61][62]. There are two types of ACMs, namely, the Rectangular ACM (R-ACM) and the Trapezoidal ACM (T-ACM). We will describe the R-ACM flavor of the ACM, in detail, in the next Chapters.

1.1.3 Using R-ACM in Update Policies

The R-ACM has been successfully used in query result size estimation as well as in some other applications. In this Thesis we demonstrate other applications for the R-ACM. In QoS-based routing, update messages are sent at a certain time according to an update policy. The update policies introduced so far are either periodic or trigger-based. Trigger-based update policies usually use the last advertised value as a basis to send or to not send update messages. In the R-ACM update policies we have taken another parameter into account, namely, the running mean of a link's bandwidth after the last advertise. We introduced two methods: the traditional R-ACM and the adaptive R-ACM update policies. We have performed a number of experiments using these two new update policies as well as the existing ones and the results of the experiments will be presented, compared and analyzed in the next Chapters.

1.1.4 R-ACM Based Update Policies and Safety-Based Routing

A number of methods have been proposed in the same context as of the QoS-based routing to deal with the inaccuracy of network state information including Safety-based routing. In Safety-based routing, at any given time, first the potential range that the actual value of the available bandwidth of a given link might belong to, is determined. This will be done based on the last advertised value for the available bandwidth for that link, and by also considering the way the link state update policy triggers the update messages. After the potential range is calculated, and by assuming that the probability distribution function of the actual values of available bandwidth within that range is

known (or can be estimated), we can then find paths which passes a higher probability for accepting new requests [5].

We have performed a number of experiments using the safety-based scheme as the routing method and the adaptive R-ACM as the update policy, and we will present the results obtained in this regard.

1.1.5 Simulation

To perform the experiments, we have to choose an appropriate simulation tool designed specifically for internetworking. We have studied the properties of a number of simulation tools and we have found that the *MaRS* system is the best choice. *MaRS* is a tool specifically used in QoS-based routing and has been used by other similar studies [1][2][3]. Indeed, we have tried the same test-bed used in similar studies [5][6] in order to minimize the inaccuracy of comparing the results.

1.2 Objectives of the Thesis

Oommen and Thiyagarajah presented in [59] and [62], a new histogram-like method for query size estimation, called the Attribute Cardinality Map (ACM). Two schemes were proposed in [59] for the ACM, namely the Rectangular ACM (R-ACM) and the Trapezoidal ACM (T-ACM). Query result size estimation is not the only application for which the ACM can be applied. It can be applied in enhancing the performance of many other applications.

In this work, we have focused on incorporating the application of the ACM method in a networking concept. We have studied different aspects of networking and we

have found that a flavor of the ACM can be used in QoS-based routing within a network. In QoS-based routing, routing tables must be updated by frequently advertising the link state information. Update policies are responsible for distribution of LSAs, but at the same time they should minimize the number of LSAs to reduce the protocol overhead cost without jeopardizing the overall performance of the QoS-based routing. We have found that the criteria for some of the existing trigger-based update policies are very similar to some of the histogram methods, and it is possible to incorporate a flavor of the ACM method into an update policy with the objective of achieving the same level of overall performance with less number of update messages flooding the network.

Several routing methods have been proposed in the literature in the same context as that of the QoS-based routing to deal with the lack of accuracy in the information regarding the state of the links in a network. One of these methods, which is pertinent, is safety based routing, and if it is incorporated with a compatible update policy, it has the potential of offsetting the impact of inaccuracies caused by the update policies of QoS-based routing, hence resulting in a better performance. The R-ACM based update policy methods can be therefore incorporated with safety based routing to minimize the inaccuracy of the information of the state of the network resources.

The objective of the Thesis has therefore been the following:

- Increasing the overall performance of the network by integrating ACM methods as new update policies in the context of QoS-based routing.
- Decreasing the protocol overhead cost in the context of QoS-based routing by reducing the number of update traffic messages that are broadcasted in the network without a loss in the packet acceptance rate.

1.3 Challenges

In this Thesis, we have confronted the following challenges:

- Studying different aspects and functionalities of networking in general and QoS-based routing, in particular, and finding novel applications for which the ACM can be used.
- Modifying the vanilla R-ACM algorithm so that it can be incorporated with an update policy to serve as a powerful trigger-based update policy.
- Modifying safety-based algorithms and introducing a new technique for link bandwidth range estimation to be compatible with the functionalities of an R-ACM trigger-based update policy.
- Finding the right simulation tool in order to implement the algorithms and compare the results.
- Learning how to use and modify the simulation tool in order to add increased functionality so that we can implement the new algorithm.

1.4 Content and Organization of the Thesis

This Thesis debuts with an overview of networking in Chapter 2. It introduces the concept of routing, routing tables and routing algorithms. Different types of routing algorithms have been discussed. It continues with an overview to Quality of Service (QoS) and QoS-based routing, and introduces update policies and also explains different kinds of update policies.

Chapter 3 deals with histograms, their definition. It explains different histogram methods and introduces the Attribute Cardinality Maps (ACMs) and the Rectangular ACMs (R-ACMs). The Chapter introduces *MaRS* as the simulation tool, which was chosen among different tools that have been studied for implementing the algorithms, and then the test-bed on which the algorithms have been implemented has been introduced.

In Chapter 4, we have explained how the R-ACM can be incorporated with QoS-based routing to yield a new trigger-based update policy. We explain how the algorithms are implemented, and present the results of simulation for different parameters with both tables and graphs. The Chapter also introduces the concept of clamp-down timers, and it gives the results of testing the system using different clamp-down timers.

Chapter 5 introduces different QoS-based routing methods that deal with the imprecision in the global state of the networks, including safety-based routing. Safety-based routing has been implemented along with the R-ACM trigger-based based update policy, and in this Chapter we present the results of simulations utilizing these algorithms.

Chapter 6 concludes the Thesis by presenting a summary of results, and outlining the main conclusions of this work. It also gives a perspective of how the study can be continued.

Chapter 2: QOS-BASED ROUTING

Routing is the process of sending data from a source to a destination, usually in the form of packets. In this process, as mentioned earlier, network components called *Routers* will take part in passing the packets of data on their way from the source to a destination by taking advantage of *Routing Algorithms*, which are in charge of initializing and maintaining *Routing Tables* used in the optimal path determination process [19].

QoS as is defined in [24] is "A set of service requirements to be met by a network while transporting a flow". This concept deals with measuring the performance of a system based on a predefined contract that defines the quality or quantity boundaries for a service between a service provider and user applications [19][24]. A flow in the notion of QoS-based routing is then defined in [24] as "A packet stream from a source to a destination with an associated QoS", which adds the QoS concerns to a regular flow of data in routing, but this time in the context of QoS-based routing. And finally QoS-based routing is also defined in [24] as "A routing mechanism under which paths for flows are determined based on some knowledge of resource availability in the network as well as the QoS requirement of flows", which incorporates the notion of QoS into the path determination process of a traditional routing method, thus introducing a new mechanism.

2.1 Routing Process, Algorithms and Tables

In the process of routing, a path has to be first determined for any request to be sent from a source to a destination or forwarded by the network components along the way. *Routing protocols* implement *routing algorithms* to initialize and maintain small databases called *routing tables to be used* in the path determination process, because calculating the paths upon arrival of an unending stream of requests is not cost efficient. Therefore routing algorithms use routing tables that keep a record of the best possible routes to all of the destinations in a network, along with the costs associated with those routes using *routing metrics*. Well known routing metrics are the available bandwidth, time delay, hop count, path cost, path load, reliability, and communication cost. Sometimes a combination of the latter metrics is calculated as a "hybrid" measure [19].

In each routing table, based on the policies of the routing algorithm, some information is stored. This information, which is used in the path determination process, usually represents the paths. To represent a path, the data concerning the source, destination and next-hop associations of a path along with their associated costs are maintained in the routing table. For calculating the cost of a path, several parameters can be considered. The available bandwidth, number of hops and time delay are examples of such parameters. When a new request comes, a router will check its routing table for the destination of the request, and will forward the request to the next router found in its routing table as the next-hop associated with the optimal path to that destination. In the path determination process, at any given time, there might exist more than one optimal path from a source to a destination, and thus, at the time of selecting a path, routing algorithms should be well-designed to make the best choice. Sometimes routing

algorithms based on their design goals, might store more than one optimal path in their routing tables in order to use them for traffic multiplexing which can result in better throughput and more reliability by load sharing. Packet switching is another functionality of routing, which is relatively straightforward and is not our focus in this Thesis [19].

2.2 Dynamic Networks and Routing Update Messages

In reality, networks are very dynamic, which means that the network topology will change with time. This usually happens when a network component (or one of its links) goes down or comes up from an up or a down state respectively. Therefore there should be mechanisms in place to ensure that once a change in the network topology happens, all network components get notified and update their routing tables with the up-to-date information about the paths, their costs and next-hop associations. Routers achieve this through the exchange of the *routing update messages* or *link-state advertisements.* A routing update message is usually sent by a *Distance Vector Algorithm, and* contains the complete new routing table of its sender or of only new portions of it. On the other hand a link-state advertisement message sent by a *Link-state routing algorithm* contains the state of all of the links of its sender or of only those that have changed. Once new information is received routers update their routing tables and start to build a new picture of the network topology and the state of its links in their local databases [6][19][67].

In the context of QoS-based routing, *update policies* are mechanisms to govern the frequency of sending the routing update messages and link-state advertisement messages [6][19]. Therefore, we first describe the notion of the QoS-based routing in

more details, and then we elaborate update policies, which are the main focus of the work done in this Thesis.

2.3 Quality-of-Service Based Routing

Quality of Service is usually defined as a set of performance agreements, visualized in a contract, in terms of the quantity or quality, between a service provider and a user [19]. QoS in the context of internetworking is defined in RFC 2386 is "A set of service requirements to be met by a network while transporting a flow". As its definition explicitly describes, this concept deals with measuring the performance of a system based on a set of service requirements while providing service for the user applications. A flow in the same context (based on the definition of QoS-based routing) is further defined in the same RFC as "A packet stream from a source to a destination with an associated QoS", which adds some associated QoS criteria to a data flow in the routing process, but this time in the context of QoS-based routing.

In the RFC 2386 then we can find the definition for QoS-based routing as "A routing mechanism under which paths for flows are determined based on some knowledge of resource availability in the network as well as the QoS requirement of flows", this incorporates the notion of QoS into the path determination process of a traditional routing method, thus introducing a new mechanism that tries to both optimally address the QoS concerns based on a predefined agreement, and the routing concerns and criteria. From the descriptions of the RFC 2386, it can be concluded that a set of constraints are to be defined and agreed upon between a service provider and a user, which will then form the QoS type of requirements necessary for a connection request.

When a path is to be found for a request, the availability of the network resources will be taken into account to satisfy the QoS-based routing goals.

2.3.1 QoS-Based Routing versus Best-Effort Routing

In general, from the concepts described within RFC 2386 and RFC1633, we can deduce that QoS-based routing and best-effort routing are defined in two relatively different contexts and try to address different concerns and achieve relatively different goals. Best-effort routing which is also known as IP routing, only deals with a single criterion in the path determination process of the routing algorithm, namely that of implementing and using the Shortest Path First (SPF) strategy as the main tool for finding paths. OSPF and RIP are among IP routing protocols that follow this convention. In contrast, in QoS-based routing, the algorithm will always take into account fulfilling more than a single objective. The objectives to be addressed usually are the desired requirements of the network resources as well as the optimality of the path in terms of the costs based on the defined metrics. In the context of QoS-based routing, the most important objectives are defined in RFC 2386 as the "dynamic determination of feasible paths", "optimization of resource usage", and "graceful performance degradation" [24]. Later in this work, we will elaborate on how we will be addressing the above objectives in the routing algorithm that we will be proposing.

2.3.2 QoS-Based Routing versus Other QoS Routing Techniques

At this juncture, it is worth mentioning that QoS-based routing is only one of the techniques that addresses the QoS concerns in the context of routing, and should not be mistaken to be the only one. In subsequent Sections and Chapters, we will discuss various

issues regarding QoS-based routing that we will be focusing on in detail. But before getting to this, we emphasize that QoS in general, or QoS in the context of the routing, in particular, are not the focus of our contribution in this Thesis. Rather, it is the QoS-based *routing technique* which is our main area of focus, and thus we will not be discussing other techniques that potentially can or will address QoS issues in the context of routing.

To address the QoS concerns in the context of the routing, there are several other techniques other than the QoS-based routing algorithms proposed in the literature, that rather try to address the routing problem as well as the QoS requirements. Some of these techniques are "Traffic Engineering" [RFC 2702], "Admission Control" [RFC 2753], "Resource Reservation, RSVP [RFCs 2205, 2208]", "Integrated Services or IntServe" [RFCs 1633, 2212, 2215], "Differentiated Services or DiffServ" [RFCs 2474, 2475], and "MultiProtocol Label Switching or MPLS" [RFCs 3031, 3032]. Although these techniques are related to QoS in the context of routing, they are not within focus of this Thesis. Hence the reader is encouraged to refer to the above references for more information regarding these other techniques.

2.3.3 SLA, and its Impact on QoS-Based Routing

QoS-based routing has to follow a set of constraints defined and documented in the Service Level Agreement (SLA) between the service provider and the user. The SLA usually deals with concerns like the user's required available bandwidth, the network link resources, the end-to-end path utilization criteria, the level of the network nodes' resource consumption, the delay and latency, and the packet loss probability. In the path determination process of a QoS-based routing protocol, when the routing algorithm is searching for optimal paths, it has to consider the SLA requirements too [6][24].

It is often very difficult to design a QoS-based routing algorithm, because of the interaction between its multiple constraints as well as the fact that it has to follow the set of SLA and at the same time to not introduce more overhead for the routing protocol.

2.3.4 Costs of QoS-Based Routing

As a consequence of the above, the trade off in designing a good QoS-based routing algorithm with acceptable QoS performance will be between the computational costs of the algorithm related to frequency and the complexity of the path determination process, and protocol overhead introduced by the algorithm caused by the update messages flooding the network and the content of those update messages. In the path determination process of QoS-based routing, the criteria to consider a link on a path are hop count, width of the path, and so on, as long as they fall within the set of constraints and the outcome is an optimal path. In the same context, the frequency to compute paths and what should trigger that computation is also very important. For example, should potential paths be computed periodically, whenever a request arrives, or when a new update message is received? In the context of QoS-based routing, for each destination, a set of paths with equal costs can be considered, and whenever a new request arrives, a path can be randomly selected among them. This can theoretically offset the inaccuracy in network state information and adds to the robustness of the QoS-based routing algorithm. However the computational cost related to the path determination process can be offset by using faster processors and larger memories, but the protocol overhead cost introduced by the flood of update messages in the network needs more cost sensitive update policies that without jeopardizing their performance, produce less protocol overhead cost [6].

Now that we are familiar with the notion of the QoS-based routing, and we have learned the importance of the computational cost of the path determination process in this context as well as the protocol overhead cost caused by the update messages flooding the network, let us focus on the mechanisms that govern the triggering of the update messages and the potential means to reduce the impact of the protocol overhead on the performance of the QoS-based routing algorithms by learning more about update policies themselves.

2.3.5 Update Policies

In QoS-based routing, to deal with the rapid changes in the state of the routers and their links in a dynamic network, mechanisms called update policies are defined to govern the rules of when, how, and how often the global state of the network maintained in the routing tables should be updated through the exchange of update messages between network routers [6][73].

Update policies will either send periodical update messages to maintain the global state of the network, or will send them based on a trigger. In the trigger-based policies if the absolute change in the state of a link (such as its available bandwidth) surpasses the threshold, or forces the state of the link to belong to a new class, the router will trigger the process of sending the updates. In the trigger-based update policies, routers continue monitoring their links, and as soon as they detect that the current value of any of their links has significantly changed from its previously advertised value, they will send an update message to all other routers in the network. What triggers an update message can be the events of surpassing a predefined threshold or moving from one class of a link state to another one. Therefore, trigger-based update policies are classified as threshold-

based and class-based update policies. Class-based update policies are further divided into two forms of equal-class based and exponential-class based update policies. [6][73]

While studying trigger-based update policies, and after focusing on their characteristics, we have found that we can have a contribution in terms of introducing new trigger-based update policies that can address some of the concerns associated with the existing methods. We will thus discuss the above update policies in more detail in the following Sections, and try to build a good case for our purpose.

2.3.5.1 Periodic Updates

In periodic update policies, routers send update messages based on a predefined timer setting, i.e. whenever a new time interval is perceived, a new update message will be sent to notify other routers about the state of one's links such as their available bandwidth. In these type of update policies, also known as timer-based policies, the most important factor is the time setting which determines the sensitivity of the update policy, i.e. policies with larger time intervals achieve less number of update messages flooding the network, which implies that the policy is also less sensitive toward the changes in the global state of the network [6].

2.3.5.2 Trigger-Based Updates

In this type of policy, trigger points are defined in the form of a threshold or class boundaries. Whenever a router detects significant change in the state of at least one of its links, and that change is bigger than the threshold value or it shifts the state of its links to pass a class boundary, it triggers link state advertisements (LSAs) in an effort to notify other routers about the current state of its links. Apostolopoulos, Guerin, Kamat and

Tripathi classified the trigger-based update policies into threshold-based update policies and class-based update policies (equal and exponential) [6].

In the following Sections we will describe threshold-based update policies and class-based update policies in details. From now on without loss of generality we assume that the available bandwidth of a link is the representation of the link's state.

2.3.5.2.1. Threshold-Based Updates

Apostolopoulos, Guerin, Kamat and Tripathi defined a threshold-based update policy as a trigger-based update policy. A constant threshold value (*th*) is predefined by the policy designer, which is the heart of the threshold-based update policy. Each router continues monitoring its links and whenever the relative absolute difference between the current available bandwidth of one of its links and the last advertised value of that link's available bandwidth is bigger than the value of the threshold, an update message will be sent [6].

Therefore, if we assume that bw^o_i represents the value of the last advertised available bandwidth for the link *i*, and bw^c_i represents the current value of the available bandwidth for that link, an update will be triggered when the following condition is satisfied [6]:

$$|(bw^o_i - bw^c_i)| / bw^o_i > th.$$

The authors of this policy [6] claim that this policy will only provide more detailed information when it is working in the low available bandwidth ranges, since the relative absolute difference won't trigger timely updates in the upper ranges of the available bandwidth, rendering it to be less accurate [6].

Looking closely into the logic of this policy, and the claims made by its authors in greater detail raises a couple of questions. Why is the threshold value predefined constant, which is not modified adaptively? How can it be modified to be able to cope with the changes in the state of a link in both the upper and lower available bandwidth ranges? Can a new threshold-based update policy be introduced that addresses these issues? Can a new threshold-based update policy be introduced that relatively achieves the same level of performance in terms of the packet acceptance rate with less protocol overhead cost (number of update messages flooding the network)? In this Thesis we will try to address these issues by introducing new update policies.

2.3.5.2.2. Equal-Class Based Updates

An equal-class based update policy is based on the idea of dividing the available bandwidth range of links into equal size classes, and as soon as a router detects that in one or more of its links, the current available bandwidth belongs to a different class that the previously advertised bandwidth value belonged to, it sends an update message. In other words an update is sent whenever a class boundary is passed due to changes in the state of at least one link. In this policy, a constant value B is predefined and is used to partition the range of the available bandwidth of links into equal size classes: *[0, B), [B, 2B), [2B, 3B)*, ..., etc. Since the class sizes are the same, it is clear that this policy will have the same level of precision in dealing with the changes and prompting new update messages for different classes of available bandwidth [6].

When studying this update policy, the reader will query why we should have equal size classes, which will result in the same sensitivity for all the available bandwidth ranges. And finally, he will observe that by flip-flopping around a boundary range, the

update policy might trigger many redundant update messages for changing around one single class boundary.

2.3.5.2.3. Exponential-Class Based Updates

Apostolopoulos, *et al* defined exponential-class based update policies to address some of the issues regarding classes of the same size in the equal-class based update policy. To address the problem of having equal size classes in the previous update policy, they suggested having unequal size classes. To create these unequal size classes, they characterized the exponential-class based update policy using constant values B and a quantity, f ($f > 1$). They used these two constants to define new classes, (as mentioned with unequal sizes) that grow geometrically by the factor f as: $[0, B), [B, (f+1)B), [(f+1)B, (f^2+f+1)B), \ldots$, etc. In this policy the triggering mechanism remains the same as the equal-class update policy. By looking at the classes, the way that they are divided and the fact that they grow geometrically, one can see that, since this policy has more and smaller classes in the lower ranges of the available bandwidth, it will perform with more precision when available bandwidth decreases. But unfortunately in the higher available bandwidth it is not able to produce accurate results and send timely update messages [6].

2.3.6 Potential for a New Update Policy

After looking closely into the details of these two class-based update policies, we observed that they have similarities with histograms and histogram-like techniques.

The equi-depth histogram technique is actually very similar to the equal-class based method, in which the new available bandwidth of a link is considered as the parameter instead of the *frequency* of the parameters in the estimation methods.

The R-ACM is another histogram-like technique proven to be more accurate than its traditional rivals, namely the equi-depth and the equi-width [59][61][62]. How this could be achieved was unsolved. Therefore this gave us the motivation to introduce new update policies equipped with a better range classification technique based on the R-ACM, which could potentially address the issues of class-based update policies, and produce better results. At the same time, since the R-ACM method takes advantage of a threshold value τ in its partitioning mechanism, and as an update policy crossing that threshold will trigger update messages, we believe that transforming the R_ACM into new update policies will actually introduce policies that fundamentally belong to the category of the threshold-based update policies. In general we can define the main objective of our new update policies to reduce the number of update messages flooding the network, hence decreasing the protocol overhead cost of the QoS-based routing, while relatively achieving the same level of packet acceptance rate. This will be discussed in details in the next Chapter.

2.3.7 Clamp-Down Timers

In general, if update policies are not designed properly, or due to strange circumstances in the network such as a link, router or segment of a network going up and down repeatedly, it might flood the network with redundant update messages, hence increasing the protocol overhead cost. Therefore in the QoS-based routing context, to reduce the protocol overhead cost and to offset the impacts of the triggering update policies, *the notion of clamp-down timer*s are introduced. Clamp-down timers will enforce a minimum time interval between two consecutive update messages, which will control the number of update messages per second. It is clear that we might as well

reduce the protocol overhead cost by optimizing the sensitivity of the triggering update policies. To optimize the sensitivity of the triggering update policies the designer can try to adjust the parameters of this policy. This can be done by computing the optimum feasible values for the constants ***B*** and ***f*** for class-based update policies to adjust the size of classes or to tune the threshold value ***(th)*** for the threshold-based update policy [6].

We mentioned that we wanted to introduce new update policies based on an analytically accurate mathematical algorithm. As was mentioned, the latter was the R-ACM. We shall now describe the R-ACM, which was first introduced in literature as a new histogram-like method, accompanied by a brief description of histograms in the next Chapter. This will follow the "vanilla" R-ACM as a state of the art singleton histogram, and then we show its strength by an example of its proposed applications, hopefully without forfeiting the focus. It will then be followed in Chapter 4 by an introduction of our two new trigger-based update policies based on the R-ACM algorithm.

Chapter 3: R-ACM, A NEW HISTOGRAM-LIKE METHOD

Class-based update policies, as described in the previous Chapter, are very similar to histogram techniques. Specifically, the range classification in the equal-class based update policy is almost identical with the equi-depth histogram technique. In the equal-class based update policy the new available bandwidth of a link is considered as the parameter instead of the *frequency* of the parameters in the estimation method of an equi-depth histogram technique.

The R-ACM method (protected under the US Patent No. 6,865,567 issued on 8th of March 2005), was introduced as a histogram-like method with precision and accuracy advantages over its traditional rivals, namely the equi-width and equi-depth histogram techniques [59][62]. Oommen and Thiyagarajah first proposed that R-ACM to be used in query result size estimation [61]. To compete with the existing class-based update policies that take advantage of the traditional histograms like the equi-depth, as well as the threshold-based update policy, we will introduce the application of the R-ACM so that it is extended to update policies in the context of QoS-based routing. It is worth mentioning that even though we will be using the partitioning rule of R-ACM in new update policies, this new application of R-ACM will actually introduce new update policies which can be perceived as being threshold-based.

The experiments performed used the *MaRS* system, and the test-bed on which the experiments have been done is also explained in this Chapter.

3.1 Traditional Histograms and Attribute Cardinality Maps

When we want to depict the frequency distribution of data attributes, we will display the value ranges of the attributes along with the frequency of the values in a graphical representation form called a histogram. For this, we first divide the attribute values into a predefined number of buckets. We then count the number of the values, as the *total frequency,* which falls within the buckets. Finally we depict these quantities in a histogram form using vertical and horizontal axes and buckets showing the attribute value ranges and their frequencies respectively. The Equi-depth and Equi-width histogram techniques are examples of traditional histograms [31][64].

Here we introduce a new histogram-like method, namely the Attribute Cardinality Map (ACM), which can be used for query optimization as well as other applications. The method is based on the philosophies of numerical integration, and it has thus been shown analytically that these models are more accurate than the traditional above-mentioned histograms [61]. There are two types of ACMs, namely, the Rectangular ACM (R-ACM) and the Trapezoidal ACM (T-ACM). The R-ACM method is explained in more detail in Section 3.1.1.

3.1.1 Rectangular Attribute Cardinality Map

The Rectangular Cardinality Map (R-ACM) of a given attribute is a one-dimensional integer array of the number of tuples in subdivisions based on the value

ranges of a relation corresponding to that attribute. The R-ACM is, if correctly understood a modified form of a histogram. But unlike the equi-width and the equi-depth histograms, the R-ACM has a variable sector width as well as a varying number of tuples in each sector. In order to minimize the estimation error of tuples in the subdivisions, we have to come up with a rule to compute sector widths to form subdivisions of the R-ACM [59][61][62].

Depending on the number of attributes, the R-ACM can be either uni-dimensional or multidimensional. Since we use only one-dimensional R-ACM in this work, the definition of the one-dimensional R-ACM and its range partitioning rule as given in [61]:

One dimensional Rectangular ACM Definition: "Let $V = \{ v_i: 1 \leq i \leq |V| \}$, where $v_i < v_j$ when $i < j$, be the set of values of an attribute X in a relation R. Let the value set V be subdivided into a number of sector widths according to the range partitioning rule described below. Then the Rectangular ACM of attribute X is an integer array in which the j^{th} index maps the number of tuples in the j^{th} value range of the set V for all j, $1 < j \leq s$."

Range Partitioning Rule: "Given a desired tolerance value τ for the R-ACM, the sector widths, l_j, $1 \leq j \leq s$, of the R-ACM should be chosen such that for any attribute value X_i, its frequency x_i does not differ from the running mean of the frequency of the sector by more than the tolerance value τ, where the running mean is the mean of the frequency values examined so far in the current sector."

It is worth mentioning that when it comes to apply the range partitioning rule of the R-ACM, we have several options of where to start partitioning the sectors on a given set. Partitioning can be done by visiting the values of the set from left to right, right to

left, or somewhere randomly in the middle and move to the sides. These options do not however change the characteristics of the R-ACM.

For example consider a set of values corresponding to the grades of a group of students. We can partition the whole range of the grades {0-100} into 20 categories {0-4, 5-9, 10-14, ...}. The number of students with the grade of each group is depicted in the frequency set A:

A = {5, 7, 4, 3, 6, 3, 12, 10, 15, 17, 14, 11, 10, 15, 20, 12, 10, 8, 2, 2}.

With the tolerance value τ, set to 3, the set A is partitioned into 8 subsets A1 to A8 as follows:

A1 = {5, 7, 4, 3, 6, 3},

A2 = {12, 10},

A3 = {15,17,14},

A4 = {11, 10},

A5 = {15},

A6 = {20},

A7 = {12, 10, 8},

A8 = {2, 2}.

Figure 3-1 displays the same example, where (a) shows the partitioning according to equi-width histogram method, and (b) shows the same data partitioned by the R-ACM.

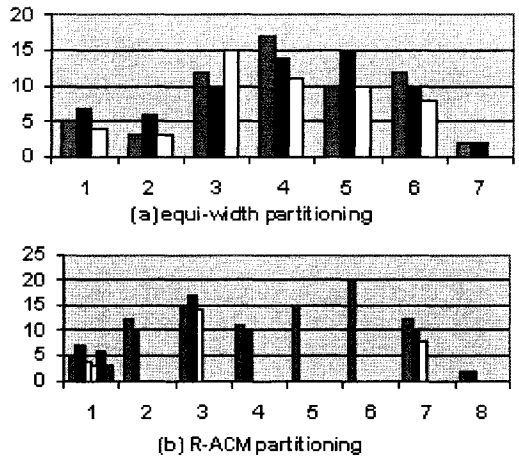


Figure 3-1: Partitioning According to (a) the Equi-width and (b) the R-ACM Methods.

3.1.1.1 The R-ACM Algorithm

The following algorithm explains how the R-ACM works. It uses the above Range Partitioning Rule and partitions the range of values of the attribute X into s sectors with variable widths [61].

Algorithm: Generate_R-ACM

Input: tolerance τ, frequency distribution of X as A[0 ...L-1]

Output: the R-ACM

begin

initialize_ACM;

current_mean := A[1];

ACM[j] := A[1];

j := 0;

for i:=1 **to** L-1 **do**

```
        if abs (A[i]-current_mean) < τ
            ACM[j] := ACM[j] + A[i];
            current_mean := (current_mean*i + A[i]) / (i+1);  //running mean
        else begin
            l_j := i-1;    //set the sector width
            j ++;          //move to next sector
            current_mean := A[i];
            ACM[j] := A[i];
        end;
    end;
end Algorithm Generate_R-ACM;
```

In the above algorithm, the inputs are the tolerance value τ and the actual frequency distribution of the attribute X. It is assumed that the frequency distribution is available in an integer array A, which has a total of L entries for each of the L distinct values of X. For simplicity, it is assumed that the attribute values are ordered integers from *0* to *L-1*. The output of the algorithm is the R-ACM for the given attribute value set [61].

Assuming that we already have the frequency distribution of X in array A, the running time of the algorithm Generate_R-ACM is O(L) where L is the number of distinct attribute values.

As was mentioned, the tolerance value τ, is an input to the above algorithm. Determining an "optimal" value for the value τ of an R-ACM is still an open question.

The use of adaptive techniques (involving possibility or learning automata) has to be investigated to solve this problem.

Later, in this Thesis, we shall use the above algorithm as an update policy in a QoS-based routing algorithm. We will then observe that the adaptivity of τ in this algorithm is a necessity.

3.1.1.2 The Rationale for Using R-ACM versus Traditional Methods

We shall now argue a case presenting the advantages of using the partitioning rule of R-ACM, in the context of QoS-based routing, by elaborating on the precision of the R-ACM in comparison with other traditional histogram methods. To achieve this, we will first focus on the power of the R-ACM partitioning rule over the equi-width and equi-depth histogram methods.

Hence without loss of generality, let us consider an arbitrary continuous frequency function *f(x)*. Consider the histogram partitioning of *f(x)* using the traditional equi-width, equi-depth and the R-ACM methods as shown in Figure 3-2 [61] [1].

[1] All the figures in this Section are directly derived from [61] with the author's permission.

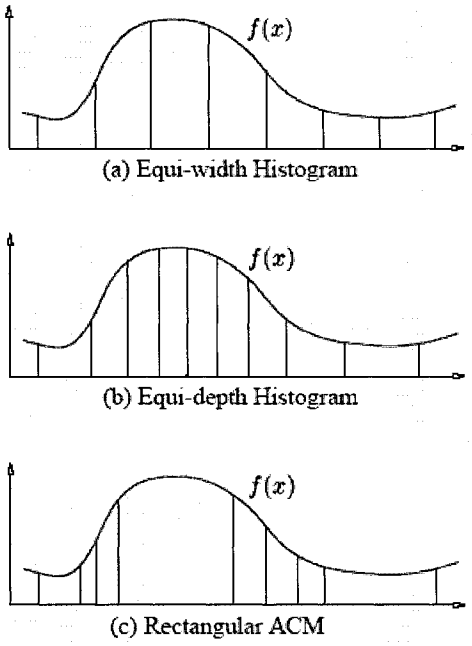


Figure 3-2: R-ACM and Traditional Histograms.

In (a), because of the nature of the equi-width histogram technique, without following the changes in the frequency of the attribute values, the sector widths remain the same and this definitely results in inaccuracy of the estimations. In the equi-depth case (b), the area of each histogram sector is the same, but the sector widths do not offset the steep changes in the frequency of the attribute values, leading to a similar problem [61].

In the R-ACM method, as can be seen in Figure 3-2, the sector widths decrease in response to sudden changes in the frequency of the attribute values, or it is achieved by an increase in the number of sectors corresponding to that steep change. Therefore the authors of [61] claimed that if we follow the R-ACM partitioning rule, we will make sure that within each sector, the actual frequencies of all the attribute values remain relatively

close to the average frequency of that sector. Hence it can be claimed that the R-ACM is a more precise and accurate class-based histogram method.

In general, the variance of an arbitrary random variable X_k is given as $Var(X_k) = E\left[(x_k - \mu_k)^2\right]$. Since, in each sector of the R-ACM, we force the difference between the frequency of a given value and the running mean of the frequencies to be less than the tolerance τ, we are actually making sure that the variance of the values falls within an acceptable range. Since the variance of R-ACM is proportional to the variances of its sectors, it is clear that if the variance of each sector is minimized, this will result in an optimal value for the variance of the R-ACM [61].

We mentioned earlier that in the equal-class based update policy the new available bandwidth of a link can be considered as the parameter instead of the *frequency* of the values in the estimation method of an equi-depth histogram technique. Therefore, for the purpose of sending new updates, by using the partitioning rule of R-ACM in a new threshold-based update policy we will make sure that triggering update messages based on the sudden changes of the available bandwidths, will produce better results in comparison with the existing class-based methods that follow traditional classification techniques. In addition, taking advantage of the flexible tolerance value τ, will provide us the means to "tune" our R-ACM based update policies to perform as a persistent competitive threshold-based update policy for the existing threshold-based ones.

The issue of how τ is chosen is, in itself, an interesting matter and is still an open field for research, and it involves issues related to optimizing the R-ACM. This is discussed in detail in [80].

3.1.1.3 Using R-ACM in the Context of QoS-Based Routing

As mentioned earlier, in QoS-based routing, update messages are broadcasted in the network so that all the routers can use up-to-date information in their path determination process. In this work, the R-ACM is used in the context of an update policy. The details of how it is used and the algorithm itself are explained in the next Chapter. We have also experimented an adaptive version of the R-ACM model in our update policy, which is also described in details in the next Chapter. A number of experiments have been performed and the results will be shown to validate the comparison. In our work, we have tried to follow the same experimental models and test-beds that other researchers have been used in similar research endeavors.

In the next Section, we introduce some of the simulation tools that we could potentially use, and will explain why the *MaRS* system was chosen as our simulation study tool. We will also introduce the test-bed on which our simulations were performed.

3.2 Simulation Tool

Performing research experiments in a real network environment is costly and almost impossible. Therefore we have opted to use simulation methods in our Thesis to "mimic" a real network and to simulate its QoS-based routing algorithm and the associated update policy. By enhancing the capabilities and functionalities of an existing simulation tool, we can implement our new update policies along with the existing ones, and then after some simulation runs, by gathering and analyzing the data, we will be able to evaluate our new policies and compare their performance with the existing methods.

There are different simulation tools specifically designed for networking problems such as *NS-2, Ptolemy, Opnet*, and *MaRS* [11][12][37][44][65][66].

MaRS (Maryland Routing Simulator) is a very simple event driven routing simulator that can be used for simulating a link state routing algorithm, in general, and a QoS-based routing algorithm, in particular. It follows a discrete-event simulation approach, (not a process emulation approach), based on the claim of its designers that a discrete-event simulation approach is more flexible in modeling such systems. In *MaRS*, similar to other discrete-event simulation tools, sequences of timely ordered events and several different sets of state variables with predefined initial values are defined. Those events are triggered in their order, and, of course, following a timely fashion. This will cause the values of the state variables to be updated [1][2][3].

Since *MaRS* has been specifically designed for simulating QoS-based routing, and it has been used in simulating and testing several update policies in the context of the QoS-based routing by other researchers, we have decided to use it in our experiments in this research work. Of course, based on the requirements of our new policies, we will have to add more state variables and events to the *MaRS* software.

In the next Section, based on the reference materials [1], [2] and [3] we briefly explain the current structure and the main components of *MaRS*. We leave the explanation regarding the new state variables and events that we will be adding to implement our new update policies for the next Chapter, where we will also discuss our new update policies and their requirements.

3.2.1 *MaRS*

Every simulation prototype created using *MaRS*, has to contain a *simulation engine*, a user interface, and a *set of components*. The simulation engine is required to manage the event list, to invoke the simulation loops running the events and to initialize all the variables. The *set of components* is used to model the network being investigated. When a user starts a simulation run on *MaRS* using the user interface, the start command will run the simulation engine to initialize all the variables and to invoke the basic simulator loop. Within a single step of the basic loop being executed, the routines of one event from the event list are executed in order, following their predefined times of execution, unless they are physically or manually interrupted by the inputs from the keyboard or mouse. Each routine's execution can cause new events to be added to the event list. The results of the iterations of the loop and the execution of the events are gathered and processed by internal processes and the graphical user display is updated with the latest information. The simulation will stop only when there are no more events in the event list, or if the user prompts the "Stop" command [3].

In *MaRS*, there are eight predefined components. The user can use multiples of them to model the topology of the system that is to be simulated. The user can also add more components if it is required. Each component has its own set of variables and events. The user can add more events to the set of events, and variables to data structures of the components. When the user starts each simulation run, routines of components will be executed in order and following their time stamps, which will in turn instantiate the components [3].

*Node, link, routing and link-cost function, workload (*of type *source* or *sink), and performance monitor* components are used to model physical nodes, links, routing protocols, application and transport protocols, and the performance measurement units of a real network [3].

In next Chapter, where we will discuss the details of our new update policies, their corresponding routing algorithms and their implementation details in terms of simulation requirements, we will briefly describe the new variables and tables that we have introduced to some of the above components so as to yield a greater flexibility for the purpose of this Thesis.

3.3 Test-Bed

The test-beds used for evaluating the performance of QoS-based routing algorithms in different studies are not very different [6][45]. Almost all of them effectively use the same test-bed and therefore the comparison becomes simpler and more realistic. We shall follow the same approach here. Besides this, we have also aligned our traffic model with the traffic model proposed in [6] and [45] for simplicity, and to enable the comparison between the performance of the existing policies and the newly introduced ones.

The network topology that is chosen for the simulation is shown in Figure 3-3. It is a typical large ISP network.

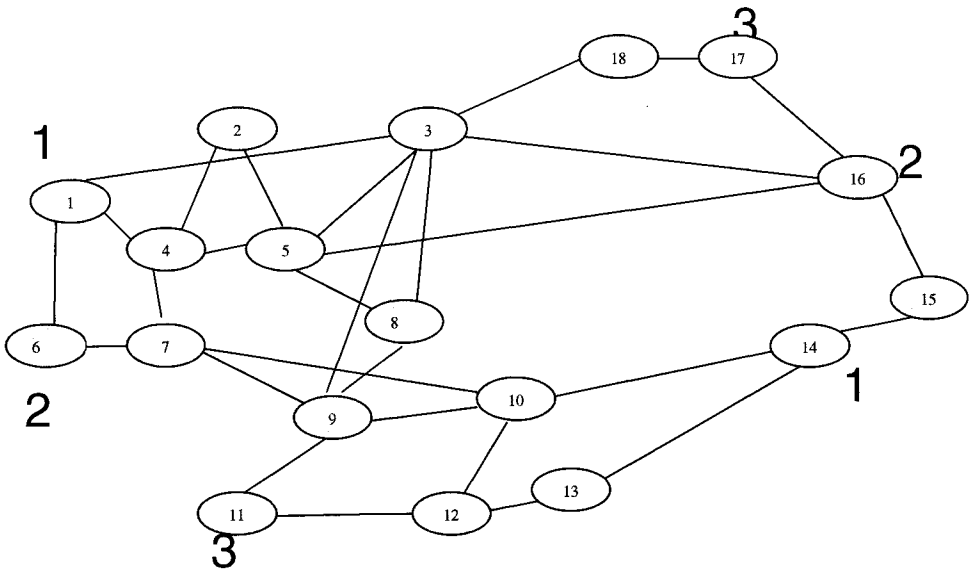


Figure 3-3: The Network Topology Used for Simulation, Derived from [6][45]

To model our traffic based on the proposed traffic model of [6] and [45], the requests will come from three specific sources, as hot spot sources/nodes, of which at any time only one of them is active and produces the main source of non-uniform traffic known as foreground traffic with the mean request duration of 3 minutes. There are three predefined destinations for foreground traffic, as hot spot sinks, each associated with only one of the hot spot sources. The specified nodes in the left side of the Figure 3-3 are source nodes and the marked nodes on the right side are the sinks. In our experiments, we always assume that none of the nodes will ever fail.

Every request comes with a random bandwidth requirement that falls within a predefined range. The request arrivals follow a Poisson distribution with parameter λ = 0.32 for foreground traffic and λ = 0.01 for background traffic. The request duration times are random variables following an exponential distribution function with the mean

of 3 minutes. The requested bandwidths are random variables following a uniform distribution function that falls within a range with lower bound of L=64 Kbits/sec, and upper bound of U=5Mbits/sec. The links connecting the nodes have an initial available bandwidth of 50 Mbits/sec and without loss of generality we assume that none of the links will ever fail.

As proposed in [6], we ignore the details of data traffic of the flows, such as the packet type and structure, packet loss recovery mechanisms, link and node queue sizes, and so on; and the emphasis will only be on the flow arrivals and departures. When a request arrives in a node, a bandwidth is reserved for it on a link that falls in the path to the destination. When a request's duration time has expired, the reserved links will be freed. The amounts of the available bandwidth on links that are reserved or freed are computed accordingly.

It is obvious that the traffic load on our simulated system depends on the mean of the three following random variables: the arrival rate of the requests, the duration of the flows, and the bandwidth requirements of the request. In our experiments, based on the mean of the above variables, the average load will reach to150 Mbits/sec.

Chapter 4: USING R-ACM IN UPDATE POLICIES

In the previous Chapters, we described a variety of existing update policy methods for QoS-based routing algorithms. Among them, well-known update policies, namely the "threshold-based", "equal-class based" and "exponential-class based" methods were described. In this Chapter we will build our case for introducing new update policies followed by their introduction and the results of the simulation runs to test our new policies as well as the existing policies.

4.1 Rational

The rational for introducing better update policies with less protocol overhead cost was discussed in the previous Chapter. We learned that the computational cost related to the path determination process of a QoS-based routing method can be offset by using today's super fast processors, and the large and relatively cheap memories. However, the protocol overhead cost introduced by the flood of the update messages in the network needs more cost sensitive update policies. Thus, without jeopardizing their performance in terms of packet loss, they need to produce less protocol overhead cost in terms of a lesser number of update messages flooding the network.

As was observed and explained in Chapter 2, there are several other issues that potentially can also be addressed by introducing new update policies. We argued that the

equi-depth histogram technique is actually very similar to the equal-class based update policy, in which the new available bandwidth of a link is considered as the parameter instead of the *frequency* of the parameters in the estimation methods. Therefore, we can use a provenly more accurate histogram-like method instead of the equi-depth histogram technique. But this is not the only enhancement to which we can contribute; there are more issues that can be addressed such as those involving an adaptive threshold value that learns from the changes in the global state of a network and which fluctuates with the flows of the available bandwidths of the links. Additionally, we propose an analytically proven superior partitioning mechanism to monitor the changes in the link bandwidth and uses a threshold value to capture sudden changes, which creates update policies that can address both issues at the same time. Another objective is to introduce new update policies that can produce better performance in terms of the overall number of update messages flooding the network with an acceptable marginal loss in the packet acceptance rate or even with keeping that intact. These considerations have motivated us to introduce new update policies that use the partitioning rule of the R-ACM method, which could potentially address the issues of class-based update policies, and produce better results. At the same time, since the R-ACM takes advantage of a threshold value τ in its partitioning mechanism, we believe that transforming it into new update policies introduces policies that fall into the category of the threshold-based update policies. We believe that addressing all these issues by introducing new update policies capable of reducing the protocol overhead cost related to the number of update messages flooding the network, can serve as a fundamental contribution of our Thesis.

In the next Section, we will see how the R-ACM can be applied to update policies in routing algorithms.

4.2 R-ACM Based Update Policy

As explained earlier, in QoS-based routing, the nodes of a network need the information of other nodes in order to keep their routing tables updated so that they can forward the incoming packets along the best possible routes towards the destinations. For this purpose, they have to send and receive update messages through the network on a regular basis. It is clear that sending too many updates or flooding the network with redundant update messages will increase the protocol overhead cost, and will thus reduce the QoS-based routing performance. On the other hand, by reducing the sensitivity of the update policies and regulating the number of update messages sent to be as few as possible has the effect that the routing tables are not updated properly or in a finely manner and timely updated, and this can cause the packets to be dropped due to not having up-to-date information in the routing tables of routers. Therefore, in designing optimal update policies we need to consider all the aspects, and yet put less emphasis on reducing the computational costs of the policies in order to create a good and sensitive-enough algorithm so as to have the best performance in terms of number of dropped packets, as well as the number of update messages flooding the network. The net effect would be to have less protocol overhead cost while satisfying more requests. In this work, we have tried to incorporate the R-ACM, a patented technology protected under the US Patent No. 6,865,567 issued on 8th of March 2005 (explained in the previous Chapter), into an update policy. With a few changes, we show that we can use it as the base for an

update policy. In the present work, we have tried to use two versions of the R-ACM, which we have called the *Traditional R-ACM based* update policy, and the *Adaptive R-ACM based* update policy respectively. We will discuss each of these algorithms in the following Sections.

4.2.1 Traditional R-ACM Based Update Policy

For the traditional R-ACM based update policy, we opted to use an algorithm similar to the one we had in the previous Chapter, while appropriately modifying it for an update policy. The resulting algorithm is defined as below, and is followed by an explanation:

Algorithm Traditional_R-ACM_Update

```
Input: available_bandwidth, i, current_mean, τ
Output: broadcast_flag, i, current_mean
Begin
   broadcast_flag := 0
   If (abs(current_mean - available_bandwidth < τ)
   Begin
        current_mean := (current_mean * i + available_bandwidth) / (i+1)
        i := i + 1
        broadcast_flag := 0
   End
   Else
   Begin
        i := 1
        current_mean := available_bandwidth
```

```
            broadcast_flag := 1
        End
    End
End Algorithm Traditional_R-ACM_Update
```

For each link between two nodes, the *current_mean* is calculated separately. The *available_bandwidth* is the remaining bandwidth of that link, and initially this is set to be the original bandwidth of the link. The variable i denotes the number of requests that have been served on this link from the start of this class.

When a new request comes, the variable i will be incremented to update the number of requests served by this link from the beginning of this class, and the amount of bandwidth it requires is subtracted from the value of the available bandwidth of that link. Therefore the *available_bandwidth* in the above algorithm is already updated. The value of the *current_mean* parameter is the mean of all bandwidths on that link since the start of this class. For any given link, as long as the difference between the last value of the *current_mean* and the latest current available bandwidth is less than τ, the class remains unchanged and no update is triggered. As soon as the difference becomes greater than τ, the value of the *broadcast_flag* is set and an update message is sent.

Example #1: Consider a link with a set of available bandwidths on that link after the arrival of random requests. This set, for this example, is taken directly from our simulation, described presently. The available bandwidths of one link for a specified amount of time are scanned, and this set is given by the set A given below:

A={51[2], 50, 46, 46, 42, 37, 33, 33, 30, 25, 20, 25, 30, 27, 26, 25, 24, 23, 21, 19, 15, 17, 18, 16, 20, 19, 15, 13, 13, 11, 8, 7, 4, 3, 6, 1, 2}

With the tolerance value τ, set to 10Mbps, the range will be partitioned into three sets A1, A2 and A3 as follows:

A1 = {51, 50, 46, 46, 42}

A2 = {37, 33, 33, 30, 25}

A3 = {20, 25, 30, 27, 26, 25, 24, 23, 21, 19, 15, 17, 18, 16, 20, 19, 15, 13, 13, 11, 8, 7, 4, 3, 6, 1, 2}

Whenever a transition from one set to the other occurs, the update policy forces an update message to be sent. Figure 4-1 depicts the same example. Part (a) shows how the set is partitioned using the equi-depth histograms, and part (b) shows how the values are partitioned by the traditional R-ACM. Note that, in these figures, an update message is sent at the border values of each class.

From this example, we can see how a suitable update policy can reduce the number of update messages in the network. In the case of partitioning the values of available bandwidth with a policy similar to the equi-depth (observe that the equi-depth method is actually analogous to an equal-class-based policy), we will have 10 bins, whereas in case of using the R-ACM, we will have only 3 bins implying that only 3 update messages will have to be broadcasted instead of 10.

---

[2] The values are in Million of bits per seconds.

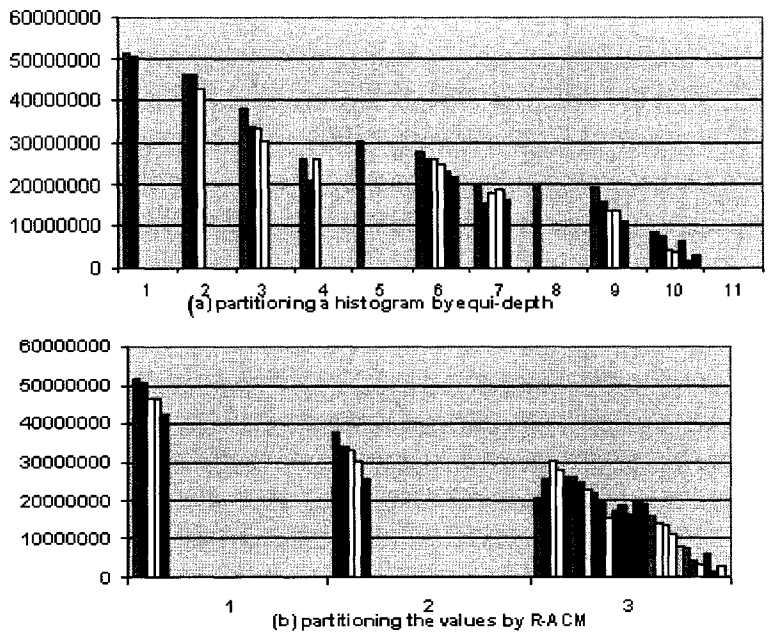


Figure 4-1: Partitioning a Histogram by the Equi-Depth and the R-ACM Algorithms.

As we will see in the next Chapter, using the R-ACM algorithm in general reduces the number of update messages, when compared to equal-class based methods. Furthermore, adaptively choosing the value of τ can be useful, as we shall see presently. Some unanswered questions are how the value of τ is to be picked, and how this value could impact the results. Another pertinent question is that of determining if we can have an adaptive value for τ based on the flow of the data. We shall discuss all these aspects in the next Section.

4.2.2 Adaptive R-ACM Based Update Policy

The nature of the problem here requires that if we offset the decrease in the available bandwidth of a link, the number of update messages has to increase. Therefore, it is better if our update policy increases the number of updates as the available

bandwidth on a link is reduced. This is clearly the rationale for threshold-based and exponential-class-based update policies. It is therefore reasonable to modify the traditional R-ACM based update policy, so that it produces more messages as the bandwidth decreases. As we can see from Figure 4-1, the traditional R-ACM based update policy does not change the number of update messages in the upper and lower bands of the available bandwidth, and this is a consequence of using a constant τ. This has motivated us to consider an adaptive R-ACM based update policy, which changes the value of τ according to the available bandwidth of the link. In lower bands, it uses a smaller τ and in the upper bands, it uses larger ones. Therefore, as we can see in Figure 4-2 derived from another simulation run, the number of updates increases when the available bandwidth decreases, resulting in more accuracy in routing tables of other nodes.

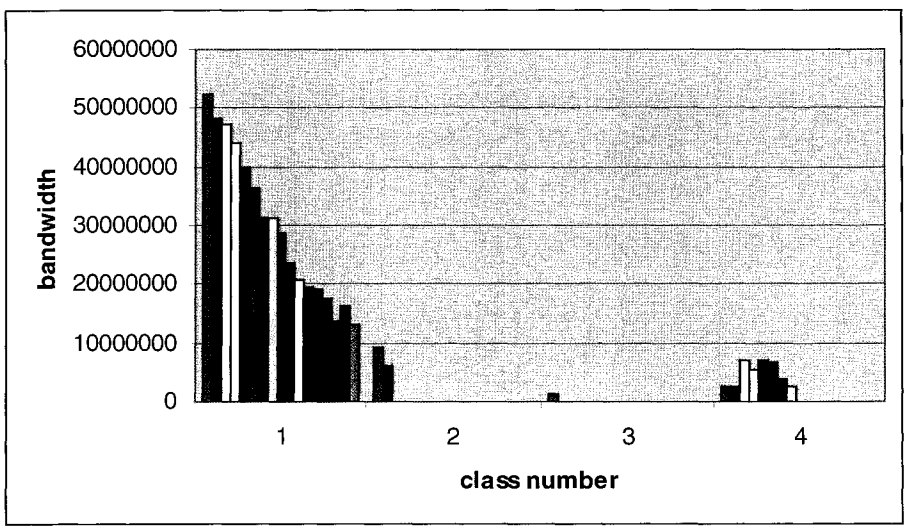


Figure 4-2: Update Messages for Different Values of the Bandwidth for the Adaptive R-ACM Based Update Policy

The algorithm for the adaptive R-ACM based update policy is given below:

```
Algorithm Adaptive_R-ACM_Update
  Input: available_bandwidth, i, current_mean, adaptivity_factor
  Output: broadcast_flag, i, current_mean
  Begin
    broadcast_flag := 0
    τ := current_mean * adaptivity_factor
    If (abs(current_mean - available_bandwidth) < τ)
    Begin
      current_mean := (current_mean * i + available_bandwidth) / (i+1)
      i := i + 1
      broadcast_flag = 0
    End
    Else
    Begin
      i := 1
      current_mean := available_bandwidth
      broadcast_flag := 1
    End
  End
End Algorithm Adaptive_R-ACM_Update
```

As we can see from the above algorithm, in this case, the value of τ is changed according to the *current_mean* and an *adaptivity_factor* that can be set to different values. This change makes the algorithm more effective and further reduces the number of update messages flooding the network as we shall see later in this Chapter.

In the above algorithm, where we check the *if* condition, if we replace τ with the product of the *current_mean* and *adaptivity_factor,* and we move *current_mean* to the left side of the expression as the "divisor" of the division, we will obtain an expression that can be perceived to be of the same flavor of Threshold-based update policy. In that case we could have the following *if* statement in our algorithm:

If (abs((current_mean - available_bandwidth) / *current_mean)* < *adaptivity_factor*).

This is obviously just another form of expressing the *if* condition, but to be consistent with the expression of the *if* condition of the R-ACM algorithm, we have opted to introduce the algorithm of the new Adaptive R-ACM based update policy as was presented in the above algorithm.

4.3 Simulation

We shall now present some simulation results, which demonstrate the properties of both of these algorithms. In the previous Chapter, we explained that the network simulation tool *MaRS* is more suitable for our needs, since it is specifically applicable for QoS-based routing, as was observed in similar studies.

4.4 Test-Bed

As explained earlier, we have used the same test-bed in our experiments as has been used in other studies. The schema for such a network in *MaRS* is shown in Figure 4-3. We mentioned that we would be using *MaRS* as our simulation tool, but to satisfy our needs we had to modify parts of it and introduce new variables, tables, and functions. We gratefully credit the core of the related material used concerning *MaRS* in our Thesis to its authors and the reference material found in [1], [2] and [3].

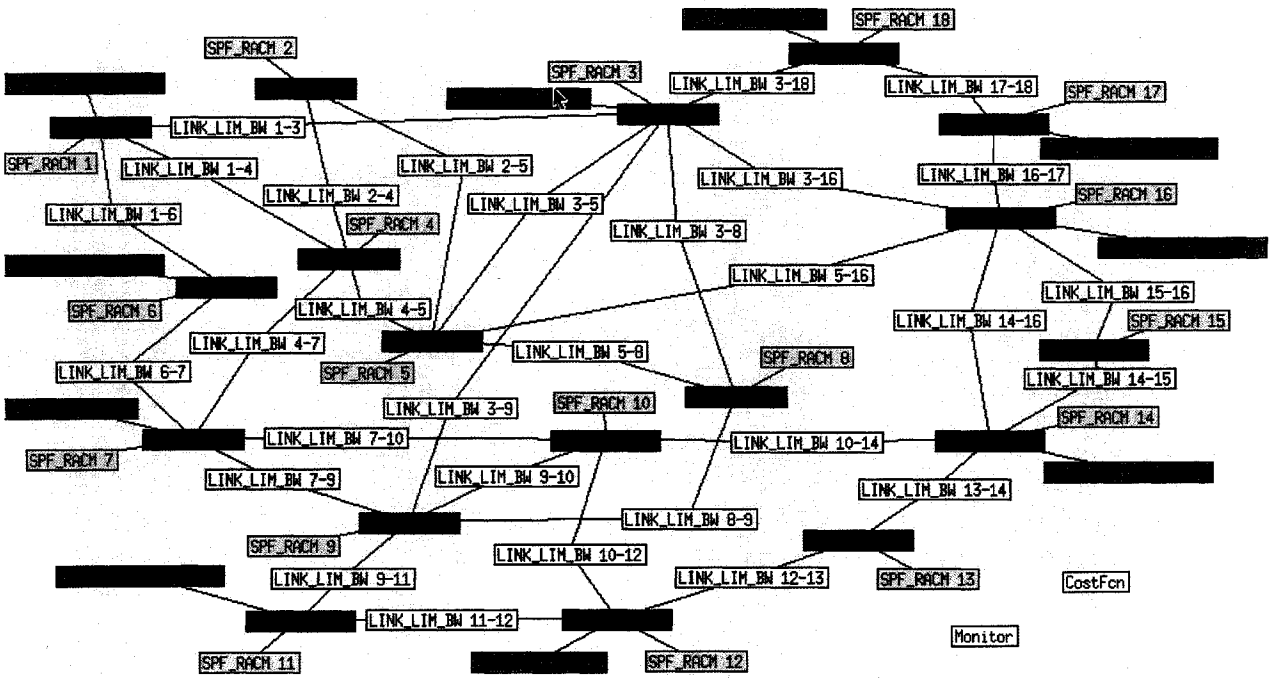


Figure 4-3: The Scheme of the Network Simulated in *MaRS*

As we can see, the green boxes are the sources for generating the packets. We have three sources as hot-spot nodes – known as REQBWTRFC_SRC1-14, REQBWTRFC_SRC 6-16 and REQBWTRFC_SRC 11-17 – which are also called foreground sources and they actually generate the main stream of packets with a higher

rate. We have also two background sources – known as BGTRFC_SRC 7-18 and BGTRFC_SRC 12-3. Each source has a corresponding sink, which is also shown as in a green box. The foreground sources have BGTRC_SINK sinks and the background sources have REQBWTRFC_SINK sinks. The foreground and background sources generate packets following a Poisson distribution with $\lambda = 0.32$ and $\lambda = 0.01$ respectively, which results in an overall traffic load of 150Mbits/sec. Each packet has a requested bandwidth, which is uniformly distributed between 64Kbits/sec and 5Mbits/sec. As was explained in Section 3.2.1, we have added some new attributes to some of the components of *MaRS* in order to implement the simulation corresponding to our new methods. The new attributes that we added to the existing source components of *MaRS* are as follows:

- Trfc Type: This attribute determines if the source is producing foreground or background traffic. In case of foreground traffic, this value is set to 1, and in case of background traffic, this value is set to 2.
- Source Start Time (secs): As explained before, the sources produce the packets sequentially. This attribute specifies the start time of producing packets for each source. The value is set in terms of seconds.
- Lambda: This value is the rate of producing packets for each source, which follows a Poisson distribution function.
- Minimum Req. Bandwidth: This is the minimum size of the request to be produced, which is set to 64 Kbits/sec in our simulation.
- Maximum Req. Bandwidth: This is the maximum size of the request to be produced, which is set to 5 Mbits/sec.

Every source or sink is connected to a node component, and there are 18 nodes in the network. In this diagram, cyan boxes depict nodes. No new attributes were needed for the node component. Node components are connected using link components, which are shown in blue boxes. Each link has the maximum capacity of 50Mbits/sec, and in order to be consistent in the various runs of the simulation, this value remains constant and does not change. The attributes for the link component are not changed either, only the link bandwidth parameter is set to 50Mbits/sec. In our simulation, the nodes are called NODE_RACM X, where X is a different number for each node, and the links are called LINK_LIM_BW X-Y, where X and Y indicates the two numbers of the two nodes which this link connects. Additionally, each node has a corresponding routing component – known as SPF_RACM X, where X shows the node number that it is connected to. There are four tables defined within each SPF component, used for the purpose of routing. The routing algorithm used here is the widest-shortest path scheme explained later in this Chapter. Figure 4-4 displays different parameters of the SPF component, and the tables it maintains for each node.

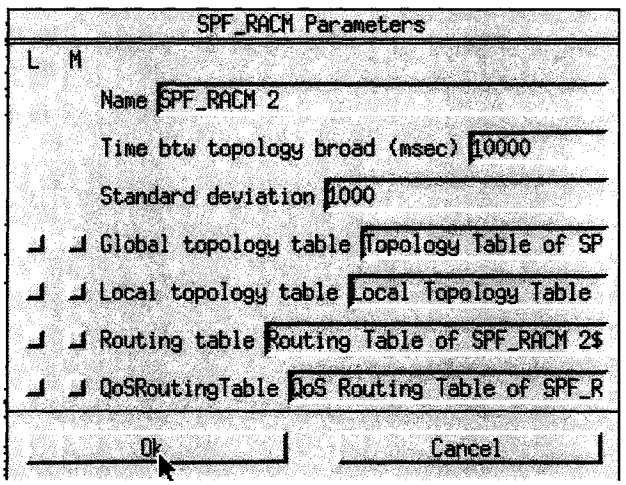


Figure 4-4: The SPF Component Parameters Used in the Simulation

The first three tables are used in the same way as in the default version of *MaRS*. Only the QoS-based routing table is introduced as a new table to record the results of widest shortest path algorithm in our QoS-based routing. The global topology table deals with the global state of the network and monitors the network information. The local topology table contains the information regarding the state of the links of each node. The routing table has the information of the SPF routing algorithm, and hence it is only good for the best-effort routing. Additionally, since it does not consider the width of the paths, it is not directly used in our QoS-based routing. The last table of this component is the QoS-based routing table, which specifically maintains the results of the QoS-based widest-shortest path routing algorithm, as explained later in this Chapter.

There are also two other components in the figure, *Monitor* and *Costfcn*. The statistical data such as the number of produced packets, number of dropped packets, number of update messages and many other useful pieces of information, which can be used for different studies, are maintained in the *Monitor* component. This component by default has many parameters, which are not of any use in our simulation. We have only used a few of them in addition to the new parameters that we have added to observe and evaluate the performance of the system. The new parameters that we have added are as follows:

- Packets Produced: This shows the number of packets produced during the effective period of simulation by all sources.
- Packets Dropped: This records the number of packets dropped either due to lack of bandwidth or for any other reason.

- Packet Drop Rate: This parameter shows the rate of dropped packets.
- Packet Accepted Rate: This attribute is the rate of packets accepted.
- Routing Pkts per Sec: This parameter is the rate of sending routing packets per second.

All of the above parameters are added to the existing version of *MaRS* in order to satisfy the requirements of our simulation. It has to be noted that the "Packets Dropped", "Packet Drop Rate" and "Packet Accepted Rate" parameters are computed only in the effective time of simulation, which starts after producing 30,000 packets in the network. That means that the first 30,000 packets are ignored in calculating the drop rate and the accepted rate of the packets, which was to reduce the impact of the transient phase of the simulation till we reached a steady state configuration. The two parameters "Packets Accepted Rate" and "Routing Pkts per sec" are the two parameters used for evaluating and comparing the results of the different runs of the simulation.

The role of the *Costfcn* component in *MaRS* was to deal with the cost of the links. The default version of this component has four functions for calculating the costs of the links. We do not use any of them in our simulation. Rather, we have added another function, which uses the available bandwidth on the link in the cost function calculations.

When we run the simulation, the source components associated with the nodes will generate requests, following their Poisson distribution function. The requests are generated by sources one at a time in a sequential order of the hot spot nodes for foreground traffic and at a lower rate for background traffic. The path to destinations are already precompiled and stored in the QoS-based routing table by the SPF components. The requests require a certain amount of available bandwidths, and based on the

requirements and the data from the QoS-based routing table, the requests are forwarded along the optimal paths to the destinations. Links are supposed to be reserved for each request following an Exponential distribution function with a mean of 3 minutes. This is determined when the requests are generated by the sources. Based on the rate of the arrival of the requests, the rate of the requested times and the available bandwidth of the links, the average load we considered was intended to achieve is 150 Mbits/sec. If we calculate the number of packets that safely reach the destinations and divide that number by the total number of produced packets after the transient phase, we can measure the performance of our QoS-based routing method. In this Chapter we study the result of using the traditional R-ACM based update policy and the adaptive R-ACM based update policy as update policies, and we will compare the results. First, we introduce the routing algorithm that was chosen for this simulation.

4.5 Routing Algorithms and Path Selection Criteria

Several routing algorithms have been introduced in the literature of routing in general or in QoS-based routing in particular. In designing a good QoS-based routing algorithm, the designers usually try to limit the use of the network resources and tend to balance that usage to be in line with the predefined set of QoS constraints and the service level agreement (SLA) between the service provider and the user application, yet at the same time they try to optimize and balance the network load. In this regard, the path selection process can be revised not only to find paths with the lowest number of hops possible in an effort to minimize using the network resources but also with considering the bandwidth requirements to be reserved on the links based on the SLA. As was

mentioned, another criterion to be considered in the path selection process is the current load of paths, to which the process can be optimized by selecting the least loaded path if several paths with the minimum bandwidth guarantees exist, which causes the network load to be balanced from perspective of the usage of its links perspectives. In this context, there are some path selection algorithms including the widest shortest path [7], the shortest widest path [85], shortest distance path [16] and dynamic alternative path algorithms [45] all of which try to find a feasible trade-off between the criteria mentioned above, sometimes by giving one criterion more weight at the time of path selection, depending on the SLA and design requirements of the QoS-based routing algorithm [7][16][45][85].

It is worth observing that these algorithms try to find the optimal path for a request. They, however, can not guarantee that the path that they find is actually the best path for a request at any given time based on an individual criterion, because they have to consider and address several criteria at the same time, sometimes by giving them different weights in the path determination process. It may also be possible that given algorithms select a path that is not even feasible. This may happen either because of old or inaccurate routing information, or because of a sudden change in the value of available bandwidth of the links that are being considered for a request. This could be due to the service being provided for the previous requests and for the costs incurred just before this new request is being finalized and established. It is also theoretically possible that one of these algorithms does not find a feasible path even though one such path might exist; this can happen due to the interaction between the two criteria of the algorithms that limits the process of path determination to follow a set of constraints [7][16][24][45][85].

In this Thesis, since the emphasis is on the bandwidth guarantees for requests, the intention is that our new update policies enhance the performance of our QoS-based routing algorithm by monitoring the links and keeping the routing tables up-to-date. This is done whenever a significant change in the available bandwidth of a link happens. The good performance of the widest shortest path in addressing such concerns, coupled with the fact that similar studies have chosen the same scheme, we have opted to use it and will implement the widest shortest path as the "standard" QoS-based routing algorithm. A version of the widest shortest path algorithm was first proposed in [7] based on the BellmanFord algorithm. We will describe the widest shortest path algorithm based on the BellmanFord algorithm and the way we will be implementing it as our QoS-based routing algorithm in details in the next Section.

4.5.1 Widest-Shortest Path

A widest-shortest path algorithm based on the Bellman-Ford shortest path strategy is proposed in RFC 2676. In this algorithm, first all the feasible paths from each source to each of the possible destinations are found. Then, from the list of those paths, the path that has the least number of hops is marked, but if among the marked paths, the algorithm can find more than one path with the minimum number of possible hops, the algorithm will only choose the path that has the maximum available bandwidth [7]. In this RFC it is claimed that there is not a huge difference when it concerns the computational complexity of the widest shortest path algorithm and a standard shortest path algorithm.

In our simulation of QoS-based routing, we have used this algorithm as was proposed in RFC 2676 [7] from which source the reader can find the details of the pseudo code. Our test-bed network topology on which the algorithm operates is a directed graph,

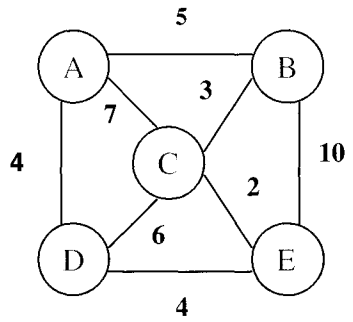
and therefore each link between two nodes will be considered differently in the computation based on the direction of the path. We have associated the available bandwidth of a link as the metric used in our computations for all the links in the graph. In our algorithm, for different number of hops up to a maximum number of possible hop counts in the network, we try to find a feasible path from a source to every destination that holds the maximum available bandwidth. Obviously the available bandwidth of a path consisting of several links cannot be more than the value of the available bandwidth of the link that has the minimum available bandwidth. This minimum available bandwidth is considered to be the cost of a path and its current available bandwidth. To find the widest shortest path with a maximum of h hops form a source to each destination, following the BF algorithm, at the h-th iteration, we select the path with the maximum available bandwidth, or the path with the optimal cost, among the paths of at most h hops. Clearly, here the available bandwidth is the primary criterion and the number of hops is the second optimization criterion. Assuming that for each node there are k destinations, it is obvious that the above algorithm will have to run k times, each time to find the optimal paths with maximum available bandwidths for different number of hops for a specific destination [7].

The QoS-based routing table, *QoSRoutingTable,* which we introduced in *MaRS* and depicted in Figure 4-4, will record the results of executing the widest shortest path algorithm runs. The structure of this table following RFC 2676 is similar to a $K * H$ matrix with two fields at each entry of the matrix. It has K rows and H columns, K for the number of destinations and H representing the maximum possible number of links (hops) on the path to a destination. At the h^{th} iteration, at row k and column h, two values are

recorded as the entry (k,h). One of them is denoted by *bw* representing the maximum available bandwidth on the path connecting the source to destination node *k*. This path is the optimal path and can have up to *h* hops at the most. The other one is referred to by *neighbor,* which shows that if this path is selected to forward a request to destination *k,* the next node on that path that the request will be forwarded to is the node *neighbor* [7].

The way the above algorithm works can be clarified by the use of the following simple example:

Consider the network in Figure 4-5 and its corresponding QoS-based Routing Table for the node A. We assume that a request arrives in A that is destined for D with a requested bandwidth of 6 Mbits/Sec. Based on A's routing table, the algorithm picks C as the next hop that can satisfy the requested bandwidth with 2 hops, considering that from A to D there was another 1 hop path but could not satisfy the requested bandwidth. Obviously, the same scenario presents itself at every node in more complicated networks.



Destination	BW (1 h path) - neighbor	BW (2 h path) - neighbor	BW (3 h path) - neighbor	BW (4 h path) - neighbor
A	0	0	0	0
B	5 - B	3 - C	4 - D	4 – C
C	7 - C	4 - D	2 - D	4 – B
E	∞	5 - B	4 - C	3 - B

Figure 4-5: A Simple Network, Showing Link Bandwidths in Mbits/Sec, and QoS-based Routing for Node A

4.6 Experiments

We performed our experiments on the test-bed explained earlier. We ran simulations up to a point that generated approximately 100,000 packets, even though the effect of the first 30,000 packets was ignored in order to bypass the effect of the transient phase and to thus yield a more accurate result.

Initially, no clamp-down timer was applied to the experiments. The use of the clamp-down timer for some experiments will be shown later.

It is worth mentioning that the two important criteria that we considered as the key performance indicators for the purpose of comparing the performance of different

update policies are "the number of update messages flooding the network", which defines the protocol overhead cost, and "the packet acceptance rate", which specifies the ratio of the number of the packets that safely reached the destinations over the total number of packets produced after the transient phase. These two criteria are independent from each other and the ideal update policy is the one that achieves a higher level of packet acceptance rate with a lesser number of update messages. However, the problem of combining these two criteria and coming up with one key performance indicator that is constructed as a weighted product of these two indicators is yet to be solved and is still an open case study for research. In addition, the weighted product of the above indicators has to follow some sort of an appropriate weighting system based on the requirements of the QoS-based routing, which adds to the complexity of the problem and can not be solved by a random weighting system or by simply giving the two criteria the same weight. Therefore we have opted to present the results of our experiments without combing the two performance indicators, but by submitting them "as is".

4.6.1 Results for the Traditional R-ACM Based Update Policy

Earlier, we explained the algorithm for the traditional R-ACM based update policy. We now present the results obtained by setting the parameter τ to different values in Table 4-1.

The following attribute types are listed in the tables that report the results:

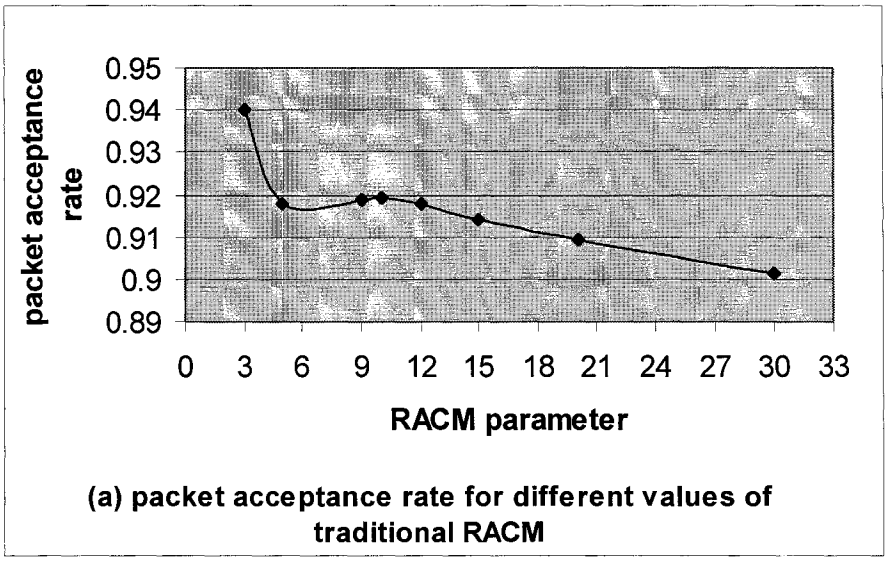
- **Produced Packets:** This is the total number of packets produced during the simulation.

- **Effective Packets:** The first 30,000 packets are not included in calculating the rates. This number indicates the actual number of packets in evaluating the performance of a simulation run.
- **Dropped Packets:** These are the number of packets dropped from the effective packets.
- **Drop Rate:** The drop rate is equal to the number of dropped packets divided by the number of effective packets.
- **Accepted Rate:** This quantity is the number of effective packets that reach the destination divided by the number of effective packets.
- **Update msgs/sec:** This is the number of update messages that flooded the network for the entire experiment.

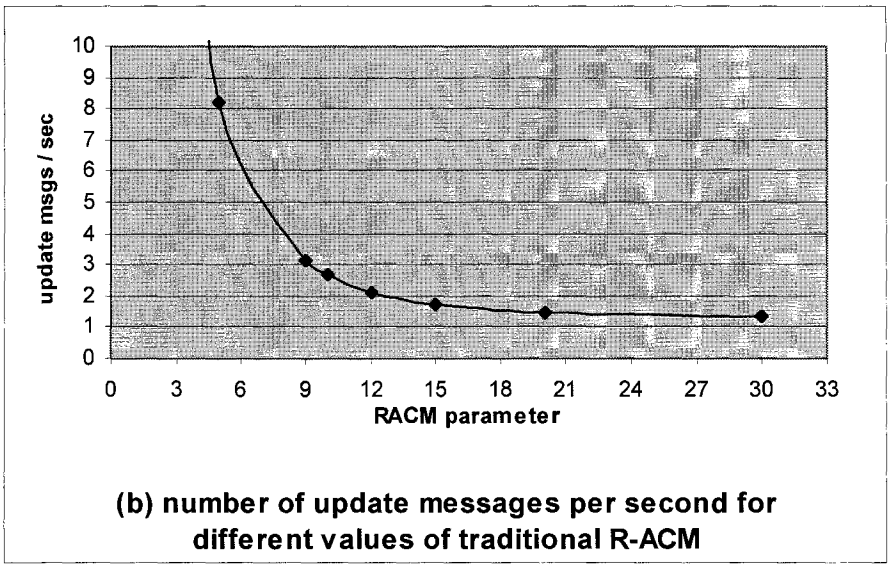
Policy (R-ACM)	Produced Packets	Effective Packets	Dropped Packets	Drop Rate	Accepted Rate	Update msgs/sec.
3M	107,654	77,654	4,657	0.06	0.94	21.18
5M	100,017	70,017	5,749	0.08	0.92	8.22
9M	101,111	71,111	5,768	0.08	0.92	3.15
10M	119,318	89,318	7,190	0.08	0.92	2.69
12M	104,670	74,670	6,145	0.08	0.92	2.14
15M	100,037	70,037	6,019	0.09	0.91	1.73
20M	106,454	76,454	6,917	0.09	0.91	1.46
30M	107,482	77,482	7,633	0.10	0.90	1.32

Table 4-1: The Results for Different Values of τ in the Traditional R-ACM Based Update Policy

As we can see from Table 4-1, where the experiments for approximately 100,000 packets were performed, we recorded only the effect of the last 70,000 packets. Figure 4-6 displays how changing τ affects the packet acceptance rate and the number of update messages sent in our simulations.



(a) packet acceptance rate for different values of traditional RACM



(b) number of update messages per second for different values of traditional R-ACM

Figure 4-6: The Results for Using Different Values of τ for the Traditional R-ACM Based Update Policy

From these figures, we see that the packet acceptance rate declines as the value of τ increases. This is obvious since the number of update messages decreases as the value of τ increases. For example when τ is set to 3, we can achieve the acceptance rate of 94.00% considering the fact that 21.17 update messages per second will have to be flooding the network, whereas when τ is set to 15, with only 1.73 update messages per second we will have 91.40% acceptance rate. Furthermore, if we set τ to be 30 we will have 1.31 update messages per second required to get the acceptance rate of 90.14%. This is the reason why we have considered an improved version of the traditional R-ACM based update policy, which changes the value of τ adaptively.

4.6.2 Results for Adaptive R-ACM Based Update Policy

The adaptive R-ACM based update policy is a modified version of the traditional R-ACM based update policy, with the additional facet that it adaptively changes the value of τ. The latter varies with the slow or rapid changes in the available bandwidth of a link, as explained in detail earlier. From the results we observe (see Table 4-2) a big improvement in the acceptance rate of packets, without flooding the network with a large number of update messages, compared to the results that we had in the previous Section for the traditional R-ACM based update policy.

Policy (Adaptive)	Produced Packets	Effective Packets	Dropped Packets	Drop Rate	Accepted Rate	Update msgs/sec.
0.20	121,475	91,475	1,770	0.02	0.98	21.09
0.30	100,017	70,017	1,632	0.02	0.98	14.76
0.40	102,073	72,073	2,214	0.03	0.97	11.34
0.50	103,978	73,978	3,092	0.04	0.96	8.27
0.55	100,487	70,487	3,450	0.05	0.95	7.43
0.60	100,016	70,016	3,821	0.05	0.95	6.53
0.70	100,021	70,021	4,755	0.07	0.93	4.94
0.80	100,902	70,902	5,656	0.08	0.92	3.72
0.90	116,659	86,659	7,531	0.09	0.91	2.75

Table 4-2: The Results of Using the Adaptive R-ACM Update Policy with Different Values of τ

We can see the results with different values of the adaptivity factor for the adaptive R-ACM based update policy in Figure 4-7. Figure 4-7 (a) shows the packet acceptance rate versus the adaptivity factor used in the algorithm, and Figure 4-7 (b) shows the number of update messages versus the adaptivity factor. We can see from the Table 4-2 that when adaptivity factor is set to 0.20, we can achieve the acceptance rate of 98.06% considering the fact that 21.09 update messages per second will have to be sent to other nodes, hence flooding the network, where as when adaptivity factor is set to 0.50, with only 8.27 update messages per second we will have 95.82% acceptance rate. And if we set adaptivity factor to be 0.90 we will even have less number of update messages (2.75) per second required to get the acceptance rate of 91.30%.Therefore it is now clear

that we can achieve a better performance by using the adaptive R-ACM based update policy compared to the traditional R-ACM based update policy. It can also be seen that the results are actually very close to the threshold-based update policy. We will compare the results achieved from simulation runs of different update policies in the next Section.

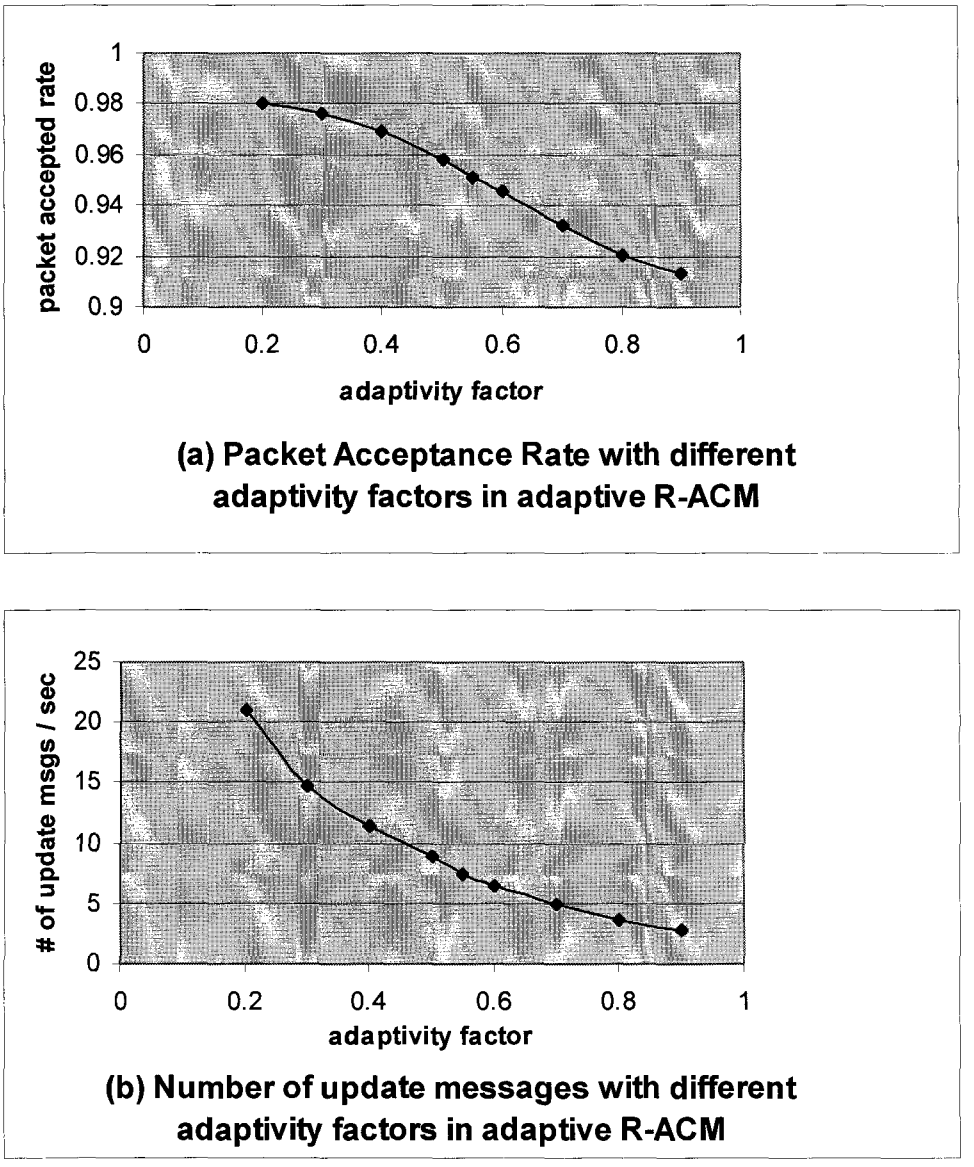


Figure 4-7: Different Adaptivity Factors for the Adaptive R-ACM Based Update Policy

4.6.3 Comparison of the Results

In order to be able to determine which policy will work to yield superior acceptance rates and lower update messages sent, we ran simulations on the same test-bed using different update policies. These methods include the equal-class-based, exponential-class-based and threshold-based update policies. In each case, we used different parameters for each method, and compared them all.

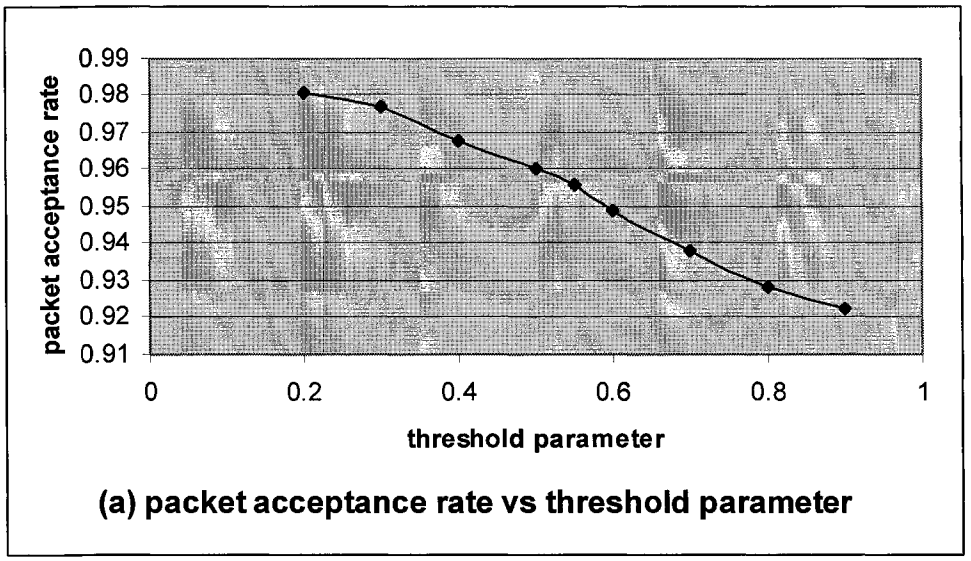
Table 4-3 shows the results of these experiments. It has to be noted that in the equal-class-based method, we have used two parameters. In the first case the parameter is equal to the maximum requested bandwidth, which is 5Mbits/sec. We refer to this method as the Equal-100%. In the second experiment, we have set the parameter to 10Mbits/sec, which we refer to as Equal-200%.

Policy	Produced Packets	Effective Packets	Dropped Packets	Drop Rate	Accepted Rate	Update msgs/sec.
Equal-100%	100,058	70,058	4,855	0.07	0.93	24.11
Equal-200%	107,461	77,461	8,454	0.11	0.89	11.26
Exp 100%/2	100,016	70,016	5,137	0.07	0.93	9.98
Exp 200%/2	128,339	98,339	10,626	0.11	0.89	6.62
Threshold 0.20	100,010	70,010	1,345	0.02	0.98	21.67
Threshold 0.30	100,850	70,850	1,612	0.02	0.98	15.59
Threshold 0.40	100,253	70,253	2,246	0.03	0.97	12.14
Threshold 0.50	100,010	70,010	2,793	0.04	0.96	9.07
Threshold 0.55	100,022	70,022	3,088	0.04	0.96	8.03
Threshold 0.60	103,162	73,162	3,719	0.05	0.95	7.10
Threshold 0.70	100,015	70,015	4,344	0.06	0.94	5.31
Threshold 0.80	111,735	81,735	5,828	0.07	0.93	3.80
Threshold 0.90	100,145	70,145	5,450	0.08	0.92	2.47

Table 4-3: Performance of Different Update Policies

We have used the same parameters for the exponential-class based scheme, except that it has a second parameter set to the value 2 in both cases. Comparing Table 4-1 and Table 4-3 we see that the traditional R-ACM based update policy outperforms the equal-class-based method, since it has a better packet acceptance ratio with a lesser number of update messages. But the tables and graphs show that the threshold-based and the adaptive-R-ACM based update policies outperform the other methods.

For the threshold-based method, a larger number of experiments were performed. The reason for this is that the results of this method are comparable to those obtained from the adaptive-R-ACM based update policy. Figure 4-8 shows the result of using different parameters for the threshold-based update policy.



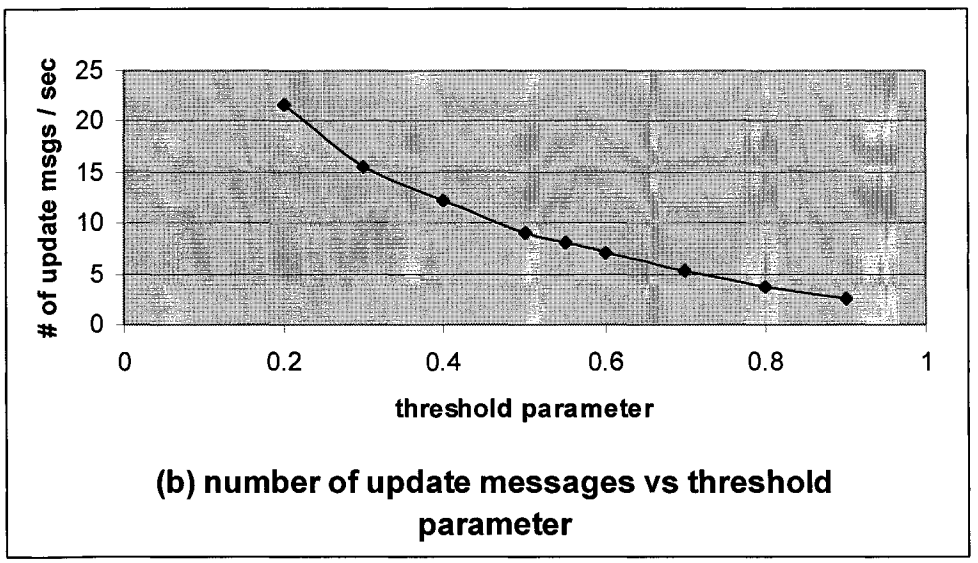


Figure 4-8: Results Demonstrating the Effect of Using Different Parameters for the Threshold-Based Update Policy.

In Figure 4-9, we have shown the packet acceptance rate *versus* the number of update messages by using the traditional R-ACM based update policy, the adaptive R-ACM based update policy, the threshold based, equal-class based and exponential class based update policies with different parameters.

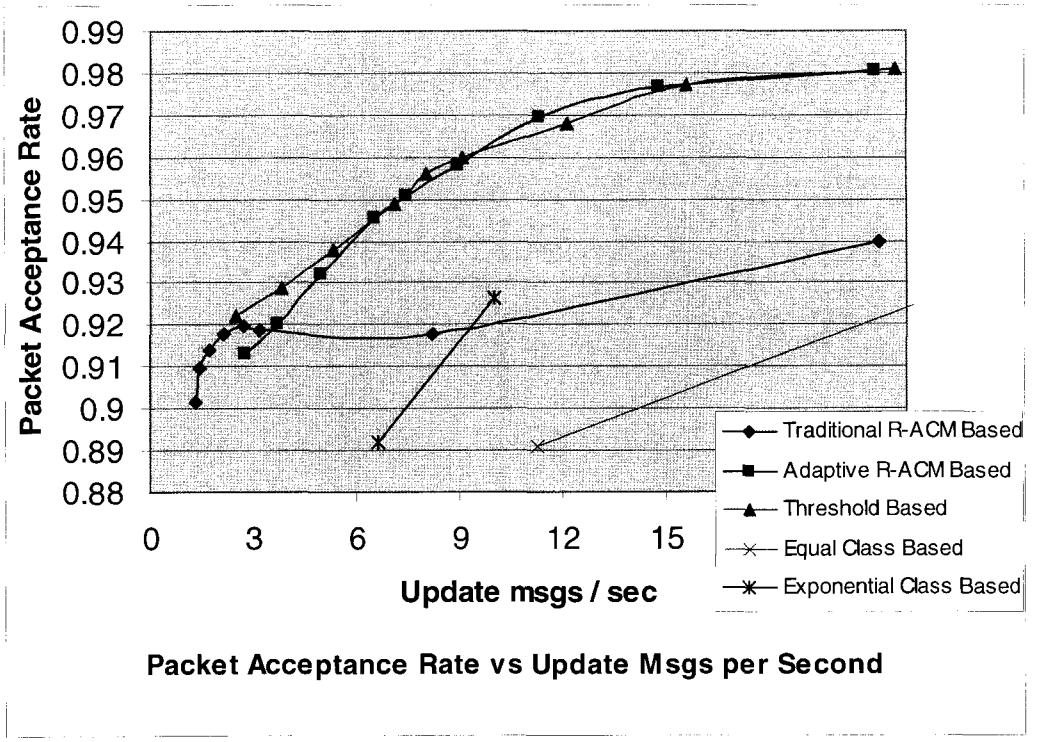


Figure 4-9: Comparing Different Update Policies

In this figure we have set the value of τ to 30M[3], 20M, 15M, 12M, 10M, 9M, 5M and 3M for the traditional R-ACM based update policy. For the other two methods, we have the adaptivity factor as well as the threshold parameter set to 0.20, 0.30, 0.40, 0.50, 0.55, 0.60, 0.70, 0.80 and 0.90. It is obvious that as the parameters of the adaptive R-ACM and the threshold-based policies decrease, (or in the case of the traditional R-ACM, as τ increases), the numbers of update messages sent, decrease. This results in an inferior performance. In order to have a low-cost environment[4], the traditional R-ACM based update policy can be a good choice, but for a higher performance it is not suitable anymore. The adaptive R-ACM based and threshold-based methods are very close in

medium-cost and high-cost environments. For medium-cost environments, the threshold-based policies perform slightly better. As opposed to this, for high-cost environments, the adaptive-R-ACM based update policy has slightly fewer number of update messages for the same performance. From this figure, we can also see the results of the equal-class based update policy with parameters set to 100% and 200%. The number of update messages in this case is rather high and the performance gain obtained does not warrant using it. The results of the experiments for the exponential-class based update policy are also depicted with the parameters 100%/2 and 200%/2. As we can see the number of update messages reduces in this case, but the performance is still low.

In all the above experiments, we did not utilize the advantages of incorporating clamp-down timers. We introduce these timers in the next sub-Section, and demonstrate how different update policies can yield different results with the added use of clamp-down timers.

4.7 Clamp-Down Timers

Clamp-down timers can be used to set a minimum time elapse requirement between two consecutive update messages. In the previous experiments, we assumed that triggering policies were strictly following their design routines and were not equipped and controlled by clamp-down timers. If there is no limit for the time between two consecutive update messages, there is no way of further controlling the volume of the network traffic related to the flood of update messages. This will result in decreasing the

---

[3] M stands for Millions.
[4] The concept of low-cost, medium-cost and high-cost was introduced in [6].

bandwidth utilization of the links of the network, and hence the only control over the protocol overhead cost in the context of the QoS-based routing will remain adjusting the sensitivity of the update policy. However, in practice, especially during the transient time of network traffic where a lot of update messages might be exchanged, we might have to consider enforcing a minimum time elapse between two consecutive update messages, i.e. using clamp-down timers. Therefore clamp-down timers can be used to further enhance the performance of the QoS-based routing in terms of optimizing the protocol overhead cost, and increasing the bandwidth utilization of the links of the network by making sure that updates are sent following a timer. There is a trade-off in using clamp-down timers, which should be looked at closely before being applied to a QoS-based routing algorithm. It is obvious that using clamp-down timers will help us in reducing the number of update messages flooding the network, but that also means less exchange of information between routers and inaccurate and less up-to-date data in routing tables. The latter could result in a loss in the performance of the QoS-based routing method in terms of the packet acceptance ratio, or even a decrease in the bandwidth utilization of the links of the network due to the imprecise data, even though the latter was one of the motivations to use clamp-down timers in the first place [6].

In this Section, we have performed some experiments by using different values for clamp-down timers for both the traditional R-ACM based update policy, and for the adaptive R-ACM based update policy and a comparison of the results will follow. These experiments were performed to monitor the impacts of clamp-down timers on the performance of our new update policies.

The impacts of the clamp-down timers on the threshold-based and class-based update policies were studied by other researchers through similar experiments. Those experiments show similar trend as of reducing the performance of the QoS-based routing in terms of the packet acceptance rate where the value of the clamp-down timer increases, yet improving its performance in terms of a lesser number of update messages flooding the network. One might think that using clamp-down timers coupled with class-based update policies might enhance the performance of the QoS-based routing. This might come form the idea that, using clamp-down timers might decrease the oscillatory behavior of the class-based update policies when the frequent changes of available bandwidth happens around a class boundary. But the studies performed by other researchers show that indeed a hysteresis mechanism will help in this regard, where this mechanism will postpone triggering a new update until the available bandwidth of a link is less than the middle value of the new class, and not when the boundary of the previous class is crossed [6].

We first report the results of the traditional R-ACM based update policy with different values of the clamp-down timer. This test has been done using only a single parameter for τ, which is 5M. The results are shown in Table 4-4. This table has one column more than the other tables, which records the clamp-down timer. The value indicates the minimum time (in milli-seconds) imposed between the consequent update messages.

We can see from the Table 4-4 when τ is set to 5M, if no clamp-down timer is set, the acceptance rate of 91.78% can be achieved with 8.21 update messages per second sent to other nodes, where as if the clamp-down timer is set to 20,000 msec, the

acceptance rate of 88.39% will be achieved with only 5.22 update messages per second to be sent. And for the same value of τ, sending only 2.98 update messages per second will be required to achieve 91.18% acceptance rate if the clamp-down timer is set to 80,000 msec.

Policy (R-ACM)	Clamp-Down (msec)	Produced Packets	Effective Packets	Dropped Packets	Drop Rate	Accepted Rate	Update msgs/sec.
5M	0	100,017	70,017	5,749	0.08	0.92	8.22
5M	20,000	129,388	99,388	11,535	0.12	0.88	5.22
5M	40,000	105,618	75,618	7,151	0.09	0.91	4.19
5M	60,000	100,156	70,156	6,293	0.09	0.91	3.49
5M	80,000	124,430	94,430	8,324	0.09	0.91	2.99

Table 4-4: Traditional R-ACM Based Update Policy, for τ = 5M and Different Values for Clamp-Down Timer

We also report the result of tests with different values of clamp-down timers used in conjunction with the adaptive R-ACM based update policy where the adaptivity factor is set to 0.60. The results of these experiments are shown in Table 4-5.

From the Table 4-5 we can see that when adaptivity factor is set to 0.60, we can achieve the acceptance rate of 94.54% with 6.52 update messages per second required to be sent to other nodes if no clamp-down timer is set. As opposed to this, when the clamp-down timer is set to 40000 msec, with only 2.28 update messages per second we will

have 92.13% acceptance rate. And for the same value of the adaptivity factor, if we set the clamp-down timer to 80,000 msec, we will need to send an even less number of update messages (1.83) per second to obtain the acceptance rate of 92.02%.

Policy (Adaptive R-ACM)	Clamp-Down (msec)	Produced Packets	Effective Packets	Dropped Packets	Drop Rate	Accepted Rate	Update msgs /sec.
0.60	0	100,016	70,016	3,821	0.05	0.95	6.53
0.60	20,000	112,258	82,258	6,522	0.08	0.92	2.90
0.60	40,000	117,175	87,175	6,881	0.08	0.92	2.29
0.60	60,000	100,067	70,067	5,512	0.08	0.92	2.00
0.60	80,000	100,360	70,360	5,613	0.08	0.92	1.83

Table 4-5: Adaptive R-ACM Based Update Policy, for Adaptivity factor = 0.60 and for Different Values of Clamp-Down Timer

Figure 4-10 displays the packet acceptance ratio for both the traditional and the adaptive R-ACM based update policies. In the case of the traditional R-ACM based update policy, for clamp-down timer was set to 20,000 milli-seconds. Although the ratio declines suddenly for larger values it shows a better performance. In the case of the adaptive R-ACM based update policy, we see a very small difference in the performance, when the value of the clamp-down timer increases from 20,000 milli-seconds. But the characteristics are similar to the traditional R-ACM based update policy, except that there

is a loss of performance regarding the packet acceptance ratio when compared to the previous results where no clamp-down timers were used.

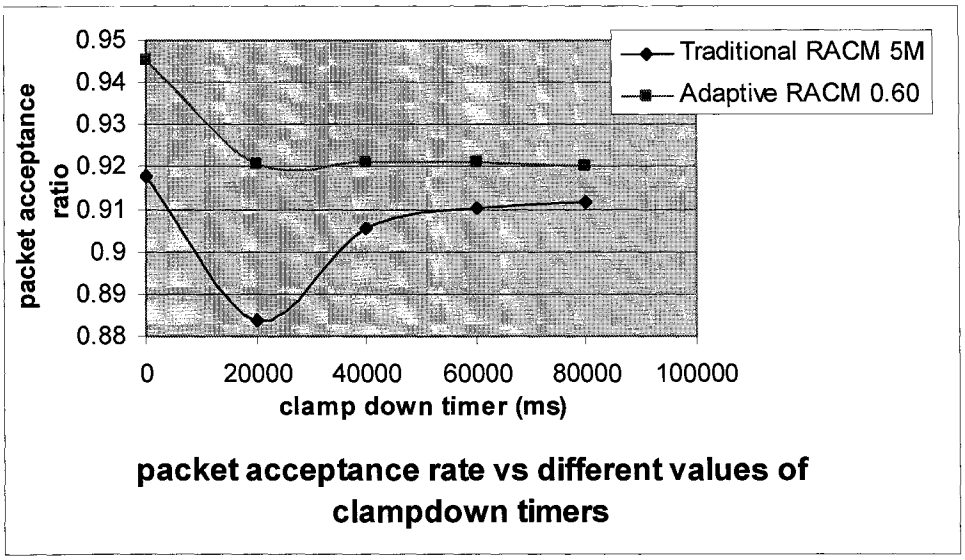


Figure 4-10: The Packet Acceptance Rate for Different Values of the Clamp-Down Timer

Figure 4-11 shows the number of update messages sent as the clamp-down timer value increases. It is obvious that these numbers decrease for the larger values of clamp-down timers. However, we also see that for values of the clamp-down timer greater than 20,000 milli-seconds, we do not see a significant change in the number of update messages being sent.

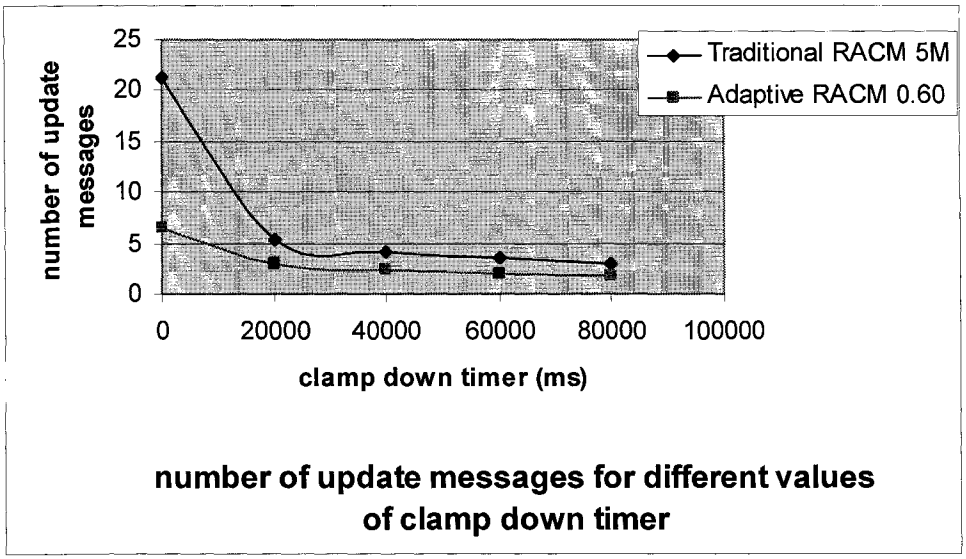


Figure 4-11: The Number of Update Messages for Different Values of the Clamp-Down Timer

In conclusion, if the cost of update messages and their effect on overloading the network traffic, known as protocol overhead cost in the context of the QoS-based routing, is considerable and we want to have less update messages, and if it is efficient to adjust the sensitivity of the update policy, we can use clamp-down timers. This will reduce the traffic overhead cost by a slight loss in the packet acceptance performance, and potential decrease in the bandwidth utilization of the links of the network.

The update policies introduce a deterministic type of inaccuracy which we will address in the next Chapter where we will be describing safety-based routing as a mechanism to offset that inaccuracy [5]. After that we will introduce a tailor-made safety-based routing algorithm compatible to our newly introduced update policies, and finally we will present the results of incorporating them together.

Chapter 5: R-ACM AND INACCURACIES CAUSED BY UPDATE POLICIES

In previous Chapters, we introduced link state update policies as mechanisms to govern the contents of the update messages and enforce the frequency of their transmission. These policies are used to help routers take advantage of the relative up-to-date data in their path determination process. In reality, particularly in today's dynamic large networks, maintaining accurate information of the state of the networks is very complex and difficult. There are so many factors involved that have impact on the accuracy of the state of the networks, including propagation delay on links, frequency limitations of sending link state update messages due to concerns regarding routing protocol overhead costs, the robustness and performance of the link state update policy, and aggregated performance impacts of small sub-networks on the performance of adjacent networks [5][6][57][87][88].

One or more of the above reasons can cause the information of the state of a network to become inaccurate. But since these reasons are different in nature, they can cause different type of impacts, and hence different type of inaccuracy. For example, the nature of what causes the propagation delay on links or the nature of what stimulates the frequency of triggering link state update messages could be *random.* But there is another type of inaccuracy, the so-called deterministic type of inaccuracy. This type of inaccuracy

can be caused by the link state update policies that depend on not very up-to-date information. In reality, it can be correctly claimed that the lack of precision and accuracy is actually a combination of the deterministic lack of accuracy and the random imprecision. To overcome such inaccuracies and offset their impacts on the performance of the network in the path determination process of routing, it has been suggested that routers do not solely depend on the last advertised values regarding the resources of the links. To achieve that, for a given flow, routers might be able to compute the actual values of the resources of the links or a range containing those values, and not depend on the last advertised values. Otherwise they need to have timely up-to-date information which has to be considered in their computations, obtaining which is not an easy task and not always feasible [5][87][88].

In this Chapter we present methods that deal with offsetting the impacts of such inaccuracies.

5.1 QoS-Based Routing Methods Dealing with Inaccuracies

In previous Chapters, we mentioned that QoS-based routing stores and maintains the information regarding the state of the network using a distance vector algorithm [47] or a link state algorithm [57]. We described that in a link state routing algorithm, whenever a network component detects a significant change in the state of at least one of its links, it informs all other components in the network by sending link state update messages regarding the changes following the trigger mechanism of the *link state update policies* [6].

QoS-based routing algorithms require accurate and up-to-date information regarding the state of the links of a network. This will help them to achieve a high level of performance, which is proportional to the level of accuracy of the information they store in their routing tables. But here the question is whether something can be done to offset the inaccuracy of the information without jeopardizing the performance, and the answer is affirmative. In the case of inaccurate link state information, a QoS-based routing algorithm can utilize some special mechanisms designed to potentially rectify the inaccuracy in the link state information. Thus, routing algorithms can now take advantage of these new techniques to gauge the potential range of the actual values of the state of the links that they want to consider in their path determination process [5].

In practice, safety–based routing [5], randomized routing [5], multi–path routing [16] and localized routing [58] were introduced as the ideal mechanisms to deal with the inaccuracies (mentioned above) in the context of the QoS-based routing.

Safety–based routing was introduced to deal with the deterministic type of inaccuracy, which was caused by the infrequent flooding of the link state update messages triggered by an update policy. The idea is that between two consecutive update messages there is a good probability that the value of the available bandwidth of links might be different from their previously broadcasted values, and that is the cause of inaccurate information to be used in the path determination process. Upon arrival of a request and before the new link state update arrives, safety-based routing uses the potential range of the values of the state of the links that it wants to consider in its path determination process, and finds the path with the highest probability which satisfies the requirements of that request [5].

Randomized routing as its name suggests, deals with the random type of inaccurate information. The idea is that, instead of storing one path for any given destination as the optimal path, the routing algorithm should store a set of feasible paths. The assumption here is that the cost of these paths were calculated based on the so-called possible inaccurate information. Then for a given request, the routing algorithm should randomly select a path from that set. In this way, the selected path may not actually be the optimal path according to the stale information, although it has the potential to compensate for the impact of the inaccurate information of the state of the network [5].

In multi–path routing, the consideration is that the routing algorithm will simultaneously choose a set of feasible paths instead of a single optimal path. Therefore, by multiplexing forwarding the request over several paths, it has the potential of reducing the impact of the lack of accuracy of the information of the state of the network [16].

Localized routing was introduced with the idea that, it might be the best if the routers use the information that they have in their local routing tables when it comes to the path determination process. Based on that, it is claimed that localized routing has the great potential of totally eliminating the impact of the inaccurate information in the state of the links of the network [58].

In the next Section, we explain the safety-based routing in more details. The reason for choosing this, among the techniques mentioned above is that we have concluded that if we are going to enhance our newly introduced R-ACM based update policies, we can only profit by choosing to implement a tailor-made compatible safety-based routing that can take advantage of the characteristics of R-ACM method. Therefore we will implement safety-based routing along with the threshold-based update policy as

well as our adaptive R-ACM based update policy, which were introduced and described in the previous Chapter, and we will compare the results to conclude our analysis in the end.

5.1.1 Safety-Based Routing

In a network utilizing a triggering-based update policy implemented by its QoS-based routing protocol, the conditions upon which new link state updates are triggered, significantly affect the volume of the update traffic. In other words, how often a network component is stimulated to flood the network with information regarding significant changes in the state of its links has a great impact on the network's update traffic volume. We can use clamp-down timers to better control the number of update messages being sent, which is achieved by imposing a minimum time elapse between consecutive update messages triggered by each network node. In this regard, we can also adjust the parameters of the triggering update policy so that the volume of the update traffic can be better controlled. It is obvious that when we reduce the sensitivity of the triggering policy, we are actually minimizing the accuracy of the routers' available information. But at the same time, it is highly desirable to operate with reduced update traffic loads resulting in less protocol overhead cost. Therefore we shall try to design a QoS-based routing algorithm that can maintain its level of performance when it faces a situation where the sensitivity of its triggering update policy has been reduced so as to decrease the protocol overhead cost, which in its own turn has led to a less accurate information available for routers in the network [5][6][29].

The above trade off can be achieved using "safety-based" routing. Safety-based routing was introduced for the first time in [5], [7] and [29] along with the introduction of

a new concept called link safety. The rational behind the introduction of the safety–based routing algorithms was that it could potentially offset the impacts of the deterministic type of inaccuracy caused by the dependency of the path determination process on the last advertised values of the state of the links and before the new states are received through new update messages triggered by the threshold-based and the class-based update policies. In other words, it is possible that between two consecutive update messages some links might have new available bandwidths due to the service they have offered to new requests without having to transit new update messages, or the update messages have been sent due to the significant changes in the available bandwidth of some of the links, but those updates have not yet arrived at some of the nodes, and that is the cause of inaccurate information which might be used in the path determination process. It is suggested that safety-based routing should not be applied when the clamp-down timers are used [5][29].

From now on we assume that b_{adv} denotes the available bandwidth value associated with a given link representing its last stored value in the QoS routing tables based on the last broadcast of the state of that link. For all triggering policies sensitive to changes of a link bandwidth, upon arrival of a request and before the new link state update arrives, given the value of b_{adv} of a link that is to be considered in the path determination process, if the update policy's triggering process and the value of its parameters are known, safety-based routing can potentially infer a unique range that shall include the real value of the bandwidth of that link between the two update messages [5][29].

Safety-based routing not only requires the above information to compute the desired bandwidth range for a given link, but also needs the information regarding the probability distribution function that can represent where the actual available bandwidth values at any time will fall within that range. The safety of a link in this context is defined as the probability value within the range of [0, 1] representing the chance that the real available bandwidth of a link can handle the bandwidth requirements of a request. This value can be estimated using the distribution function and the knowledge of the required bandwidth of a request. Based on the safety values for the links, in the path determination process, the algorithm will be able to find the path with the best value of probability to satisfy those requirements and handle the request gracefully [5][29].

The range $[b_l, b_u]$ describes the potential available bandwidth range of a link. If the safety of a link based on the above definition follows a uniform probability distribution and if the requested bandwidth b_r is in the range $[b_l, b_u]$, the safety of the link can be computed as $(b_u-b_r) / (b_u-b_l)$. Using this safety calculation formula, when $b_r < b_l$, the safety of the link will be computed as unity. This it true, since the bandwidth requirements of the request can be satisfied easily when the lowest value for the real available bandwidth is even bigger than the requested bandwidth. But if the requested bandwidth is even higher than the upper bound of the available bandwidth of a link, (meaning that $b_r > b_u$), the safety of that link will be 0, and that link will certainly not be able to satisfy the bandwidth requirements of the request. In the path determination process when the safety of all links for a path is determined, the safety of the path can be calculated by multiplying the safeties of those links in that path [5][29].

When a new request with specific bandwidth requirement arrives, based on the QoS routing table, it will be forwarded to a specific link. After the distribution function is computed, the algorithm can calculate the likelihood that the required bandwidth is actually available on that specific link. At the last stage of the path determination process, when a path is to be selected, the calculated probability values will then be used with the idea of considering paths with the best safety values or in other words with the maximum probability. This will result in accounting for the inaccuracies mentioned earlier which were produced as a result of the objective of the update policy having less update traffic [5][29].

If we set the hold–down timer value to 0, the potential range of the available bandwidth value of a link for the threshold-based and class-based update policies can be determined by taking into account the way their triggering processes work.

Upon arrival of a new request, in the threshold-based update policy with a predefined parameter, before the next link state update message for this link arrives, we can conclude that the relative change in the bandwidth value has not exceeded the threshold value *th*. Therefore the lower and upper boundaries of the range that the actual bandwidth of any given link belongs to are $(1\text{-}th)b_{adv}$ and $(1+th)b_{adv}$ respectively [5].

For the class-based update policy, the link state update message basically advertises crossing the boundaries of a class of available bandwidths, therefore based on the last broadcasted values for the links, before the arrival of a new update message, we can conclude that the potential range of the bandwidth of a link is actually the class that the previous advertised bandwidth belonged to [5].

In reality, the probability distribution of the potential range is not always a uniform distribution; therefore a measurement-based mechanism was proposed in [5] to find the actual distribution by monitoring the network. Computing the probability distribution function of the potential range is a very complex problem because of the impact of the factors like the network topology, the packet arrival distribution and its parameters, bandwidth request distribution and its parameters, link bandwidth distribution, traffic load preferences and their patterns, and also the QoS-based routing requirements itself [5].

In addition, to ensure that we can maintain accurate information for all network nodes, because of the nature of the measurement-based requirements, we will need to exchange a lot of messages, which is exactly in contradiction to the safety-based routing goals in the first place [5]. We believe that an ACM-flavored scheme could be used to enhance the classification of the ranges in finding the probability distribution of the potential range in the measurement-based approach to reduce the number of routing messages required. However, even though its enhancement can affect our experiments, in its nature it is different from our focus and interest, and therefore the details are left open for future research. In general, due to the complexity of implementation of the measurement-based mechanism, we decided not to implement it in our Thesis, and therefore we will not describe it in more detail. We encourage the reader to refer to [5] for more information. Indeed, that is why, as was explained before, a simpler approach was suggested by other researchers in [29] and reiterated in [5], which assumes that all the network link bandwidths follow a uniform distribution within the bandwidth ranges.

In this Chapter, for the first time we incorporate the notion of the safety-based routing with the previously introduced adaptive R-ACM based triggering update policy. To achieve this we assume that within all the ranges, the distribution function of available bandwidth of its links is uniform. Using this assumption, we will compare the results of its simulation runs with the results of the simulation runs of the safety-based routing incorporated with the threshold-based triggering update policy.

5.1.2 Safety-Based Routing Algorithms

There are two known safety-based routing algorithms introduced as *safest-shortest* and *shortest-safest* routing algorithms. These algorithms were introduced and defined in response to the requirements of the safety-based routing algorithms and its new defined metrics. The safety of a path or so-called *path safety*, considering that all the safeties of the links on that path are known and independent from each other, was inspired from path metrics in other routing algorithms. It was defined as a new metric, tailor-made for safety-based routing algorithms [5].

A path with the least number of links is known as the shortest path, and the safest path is a path with the largest value of safety among all other paths. Therefore, the safest-shortest routing algorithm is defined to choose the path with the largest safety value among all the paths between a given source and destination with the minimum number of hop counts. It is clear that choosing the safest path among a set of shortest paths cannot guarantee that the chosen path is actually the safest among all the feasible paths, because of the focus is on the length of the paths in the first place, rather than on the safety of the paths. In contrast, the shortest-safest routing algorithm is defined to choose the shortest path from the list of paths with the highest safety value, and this guarantees that the

chosen path is actually one of the safest paths, since the first criterion was to choose a set of the safest paths and then, among them to choose the path that has the minimal number of links [5].

There is one more issue to be addressed, and that is how links with known safeties are even considered to be in the path determination process. There are two pruning policies introduced to address this issue. Apostolopoulos, *et al* in [5] suggested that link safety information could be used in link pruning policies that deal with the way links are accepted or rejected when searching for a feasible path. They specified two families of pruning policies with a single mechanism, namely, one which prunes a link only if its safety value, s, is less than a certain prescribed value. The first family of pruning policies then was defined so as to prune all links with safety less than unity ($s < 1$). In other words this conservative pruning policy ignores all the links that are not 100% safe, or, in this family of pruning policies, only links that have enough bandwidth that can guarantee to satisfy the requested bandwidth are not pruned. In contrast, the second family of pruning policies was defined as an aggressive pruning strategy, which prunes all the links with safety 0 ($s = 0$). In fact, in the second family of pruning policies, we only prune links that have no way of satisfying the request [5].

In our experiments, we implemented our safety-based routing algorithm incorporated with the R-ACM based as well as the threshold-based update policies using both of these two pruning policies.

5.1.3 R-ACM Safety-Based Routing Implementation Proposal

In this Section, we describe existing methods proposed by other researchers for implementing safety-based routing and then introduce our new R-ACM safety-based

routing method. As we discussed before, using a measurement-based approach would enhance the accuracy of the safety-based routing algorithm, but that comes at a big cost. In addition to its complexity in terms of implementation, in measurement-based approach, to ensure that we can maintain accurate information for all network nodes, because of the nature of its requirements, we will need to exchange a lot of messages, which is exactly in contradiction to the safety-based routing goals in the first place. To overcome this hurdle, a simpler and more practical approach suggested by other researchers, as in [29], is to assume that all the network link bandwidths follow a uniform distribution within the bandwidth ranges.

In the current work, experiments have been performed for threshold-based update policy as well as for the adaptive R-ACM update policy. We will present and compare the results in detail in the next Section. We did not perform any experiment in this part for equal-class based and exponential-class based update policies since even in a non safety-based approach they did not produce competitive and satisfying results when compared with the threshold-based and the adaptive R-ACM update policies.

5.1.3.1 Safety-Based Routing for the Adaptive R-ACM Based Update Policy

The Adaptive R-ACM Based Update Policy, introduced earlier, is another triggering policy sensitive to changes of link bandwidths.

At any specific time, for all the links which are supposed to be considered in the path determination process, by considering several factors, safety-based routing computes a potential range that their actual bandwidths belongs to. One factor is the sensitivity of the triggering policies related to changes of a link bandwidth. Another factor is the value of b_{adv} that denotes the last advertised value of the available bandwidth. Safety-based

routing then computes those potential ranges upon arrival of a request and before the new link state update arrives. In its computations, safety-based routing takes the value of b_{adv} into account, and as well, it follows the update policy's triggering process and uses the value of its parameters in the calculations [5].

After computing the range $[b_l, b_u]$, the minimum value b_l, and the maximum value b_u (the bounds of the range) can be used to estimate the probability value (the so-called safety of a link) that the bandwidth requirements of a request is indeed available on this link. As described earlier, this safety value can be easily computed, if the probability distribution of the bandwidth of a link within that range is known [5].

Our focus here is to find the values of the bounds of the range b_l and b_u mentioned above for the adaptive R-ACM based update policy. Without loss of generality and for simplicity we assume that the actual available bandwidth of all the links follows a uniform probability distribution. Hence, since the requested bandwidth b_r is in the range $[b_l, b_u]$, the safety of the link can be computed as $(b_u-b_r) / (b_u-b_l)$. We know that in an adaptive R-ACM based update policy an update message will be sent as soon as the following condition is satisfied:

$$|current\_mean - b_{av}| > \tau \quad , \quad \text{where } \tau = adaptivity\_factor * current\_mean.$$

As described earlier *current_mean* is the mean of all available bandwidths on that link since the last update has been sent. Also, b_{av} is the available bandwidth on the link, and *adaptivity_factor* is the parameter introduced to adjust the update triggering sensitivity of the adaptive R-ACM based update policy. Now, in order to apply the value range estimation concept of safety-based routing in our adaptive R-ACM based update policy, we propose the use of the following formulae to compute the values for b_l and b_u:

b_l = *current_mean (1 - adaptivity_factor)*

b_u = *current_mean (1 + adaptivity_factor)*

Unless advertised for an explicit request by other routers, the exact value for *current_mean* of available bandwidth of a given router's link at any specific time is unknown to other routers. Since it is our intention to reduce the number of update messages or any of these purposes in network traffic, instead of introducing a new pair of request and update messages to exchange the *current_mean* mentioned above, we shall now derive the estimation for the *current_mean* of available bandwidth of a given link.

From a straightforward computation, it is easy to see that the estimation for the *current_mean* of available bandwidth of a given link is:

$current\_mean = b_{adv} - ((avg\_req\_size * \lambda * t) / 2)$,

where b_{adv} is the last advertised bandwidth value of the link, *avg_req_size* is the average value of the bandwidth requests, λ is the rate of incoming requests on the link, and t is the time interval between that link's last bandwidth advertised and the current time stamp.

In our experiments, without loss of generality, we consider *avg_req_size* to be equal to the average of the two request size bounds of *64Kbits/sec* and *5Mbits/sec* which is *2.65 Mbits/sec*, and we assume the value *0.32* for the Poisson parameter λ.

5.1.4 Results

In this Section we discuss the results of applying the safety-based routing along with the adaptive-R-ACM based and the threshold-based update policies. We have used the same test-bed with the exact simulation details used in the previous Chapter. We have

performed the tests for *safety* < *1* and *safety* = *0* pruning policies as was explained above. We performed the experiments for 30,000 to 100,000 packets, and without enforcing any clamp-down timer. As described before, the safest-shortest algorithm does not produce satisfying results since choosing the safest path among a set of shortest paths can not guarantee that the chosen path is actually the safest among all the feasible paths. In contrast, the shortest-safest routing algorithm guarantees that the chosen path is actually one of the safest paths, since the first criterion was to choose a set of the safest paths and then among them to choose the path with the minimum number of hops; therefore in the current work, we have only focused on the shortest-safest algorithm.

5.1.4.1 Threshold-Based Update Policy, Pruning *s<1*

The results of the experiments for threshold-based update policy with different threshold parameter values applied with a safety-based algorithm with a pruning policy using *s<1* are shown in Table 5-1.

Policy (Threshold)	Produced Packets	Effective Packets	Dropped Packets	Drop Rate	Accepted Rate	Update msgs/sec.
Th = 0.20	100,339	70,339	1,239	0.02	0.98	21.32
Th = 0.30	110,066	80,066	1,484	0.02	0.98	14.39
Th = 0.40	100,755	70,755	1,564	0.02	0.98	9.74
Th = 0.50	116,651	86,651	2,279	0.03	0.97	6.17
Th = 0.60	100,333	70,333	2,355	0.03	0.97	3.97
Th = 0.70	100,222	70,222	3,896	0.06	0.94	2.79
Th = 0.80	100,317	70,317	7,677	0.11	0.89	1.91

Table 5-1: Results of the Threshold-Based Update Policy, Pruning Policy *Safety* < *1*.

As we can see from the results, the threshold-based update policy has a better packet acceptance rate with relatively higher number of update messages when the threshold parameter is lower, and its performance is even better than when it was implemented with widest shortest path algorithm. For example, in this case, the packet acceptance rate for threshold set to 0.20 is 98.23%, whereas it was 98.06% in case of using widest shortest path algorithm. But as we can see, the packet acceptance ratio of performance declines as the threshold value gets larger even though the number of update messages is lower.

5.1.4.2 Threshold-Based Update Policy, Pruning *s=0*

In this part, the threshold-based update policy is simulated using safety-based routing algorithm with a pruning policy ignoring only links with $s = 0$. The results from the simulation experiments are shown in Table 5-2.

Policy (Threshold)	Produced Packets	Effective Packets	Dropped Packets	Drop Rate	Accepted Rate	Update msgs/sec.
Th = 0.20	100,643	70,643	1,216	0.02	0.98	21.48
Th = 0.30	100,026	70,026	1,280	0.02	0.98	14.64
Th = 0.40	114,716	84,716	1,656	0.02	0.98	9.85
Th = 0.50	136,700	106,700	2,327	0.02	0.98	6.32
Th = 0.60	118,055	88,055	2,308	0.03	0.97	4.26
Th = 0.70	107,325	77,325	2,951	0.04	0.96	3.01
Th = 0.80	109,316	79,316	6,717	0.08	0.92	2.54

Table 5-2: Results of the Threshold-Based Update Policy, Pruning Policy *Safety = 0*.

As we can see, the packet acceptance ratio for smaller threshold values is higher with the overhead of more update messages. For the larger values of threshold the performance decreases for the packet acceptance ratio. For example, when the threshold value is set to 0.20, 98.27% of the packets reach the destination, whereas when the threshold is set to 0.80, only 91.53% of the packets reach the destination successfully. In general, when compared, the performance of this part is relatively better than the performance of the last part where the pruning policy used was $s < 1$.

5.1.4.3 Adaptive-RACM Based Update Policy, Pruning $s<1$

The results of the experiments using the adaptive R-ACM based update policy and safety-based algorithm with a pruning policy of $s < 1$ is shown in Table 5-3.

Policy (Adaptive R-ACM)	Produced Packets	Effective Packets	Dropped Packets	Drop Rate	Accepted Rate	Update msgs/sec.
A_f[5] = 0.20	100,018	70,018	1,533	0.02	0.98	19.48
A_f= 0.30	100,219	70,219	1,884	0.03	0.97	12.22
A_f= 0.40	99,911	69,711	2,434	0.04	0.96	6.98
A_f= 0.50	100,316	70,316	3,515	0.05	0.95	4.05
A_f= 0.60	100,009	70,009	5,418	0.08	0.92	2.76
A_f= 0.70	101,799	71,799	7,898	0.11	0.89	2.31
A_f= 0.80	102,151	72,151	11,760	0.16	0.84	2.16

Table 5-3: Adaptive R-ACM Based Update Policy, Pruning Policy *Safety* < *1*.

As can be seen, the packet acceptance ratio is less than of the threshold-based update policy with safety-based, yet the number of update messages being sent is also less when compared to that of threshold-based method with safety-based algorithm. For example, with the adaptivity factor set to 0.20, using the adaptive R-ACM based update policy, we have the packet acceptance ratio of 97.81%, whereas the same experiment

[5] This value has been called the adaptivity factor in the current work.

with threshold-based update policy, with the threshold set to 0.20, 98.23% of the packets reach to destination, noting the fact that the number of update messages is reduced from 21.31 in the latter case to 19.47 in the former case, which is remarkable. Therefore when less number of update messages is preferred, the adaptive R-ACM update policy with safety-based algorithm tends to produce better results.

5.1.4.4 Adaptive-RACM Update Policy, Pruning *s=0*

In Table 5-4 the result of the experiments with pruning policy of *s=0 are* shown.

(Adaptive R-ACM)	Produced Packets	Effective Packets	Dropped Packets	Drop Rate	Accepted Rate	Update msgs/sec.
A_f = 0.20	110,396	80,396	1,695	0.02	0.98	19.83
A_f = 0.30	100,003	70,003	1,584	0.02	0.98	12.25
A_f = 0.40	100,029	70,029	1,865	0.03	0.97	7.22
A_f = 0.50	102,584	72,584	2,368	0.03	0.97	4.37
A_f = 0.60	100,335	70,335	3,855	0.06	0.94	3.11
A_f = 0.70	123,630	93,630	7,795	0.08	0.92	2.65
A_f = 0.80	100,019	70,019	9,496	0.14	0.86	2.66

Table 5-4: Adaptive R-ACM Based Update Policy, Pruning Policy *Safety = 0*

In this set of experiments, the update policy used was once again the adaptive R-ACM based update policy applied with its compatible safety-based scheme with the pruning policy of *s=0*. We can see that the packet acceptance rate is less than that of the

threshold-based update policy with safety-based policy, noting the fact that the number of update messages is also reduced. This is indeed desirable as it would achieve the goal of having less network traffic overhead, leading to less protocol overhead cost.

5.1.4.5 Comparison

Figure 5-1 displays the packet acceptance rate of the threshold-based and the adaptive R-ACM based update policies with safety-based algorithm with pruning policies $s < 1$ and $s = 0$. The results for the threshold-based update policy with widest shortest path have been included to have a better view for comparison.

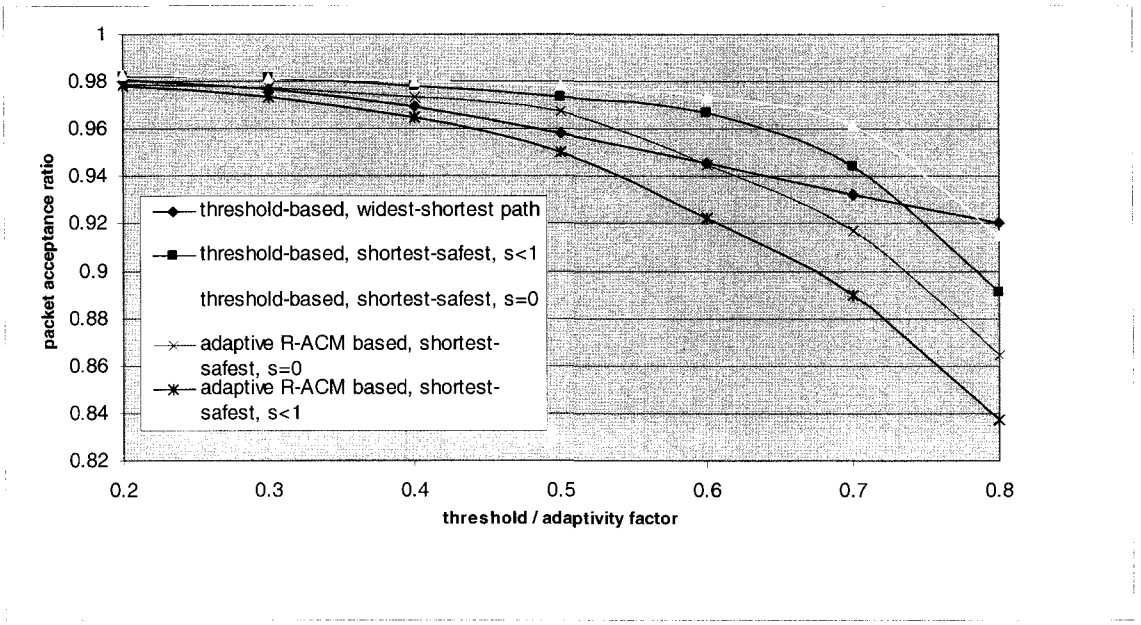


Figure 5-1: The Comparison of Different Methods of Safety-Based Routing

We can see that for smaller threshold values, the threshold-based update policy with safety-based routing performs better than the one with the widest shortest path

algorithm. For larger values of threshold, if applied with the safety-based algorithm, the packet acceptance rate drops. The safety-based method with the adaptive R-ACM based update policy achieves better results if the number of update messages is the objective. The packet acceptance rate for this method is lower than the other methods, although as was mentioned, the number of update messages has been reduced too (Figure 5-2). We believe it might be the estimation error for the running current mean of the available bandwidth that results in the relatively poor performance, which can be a good open research topic for a future work.

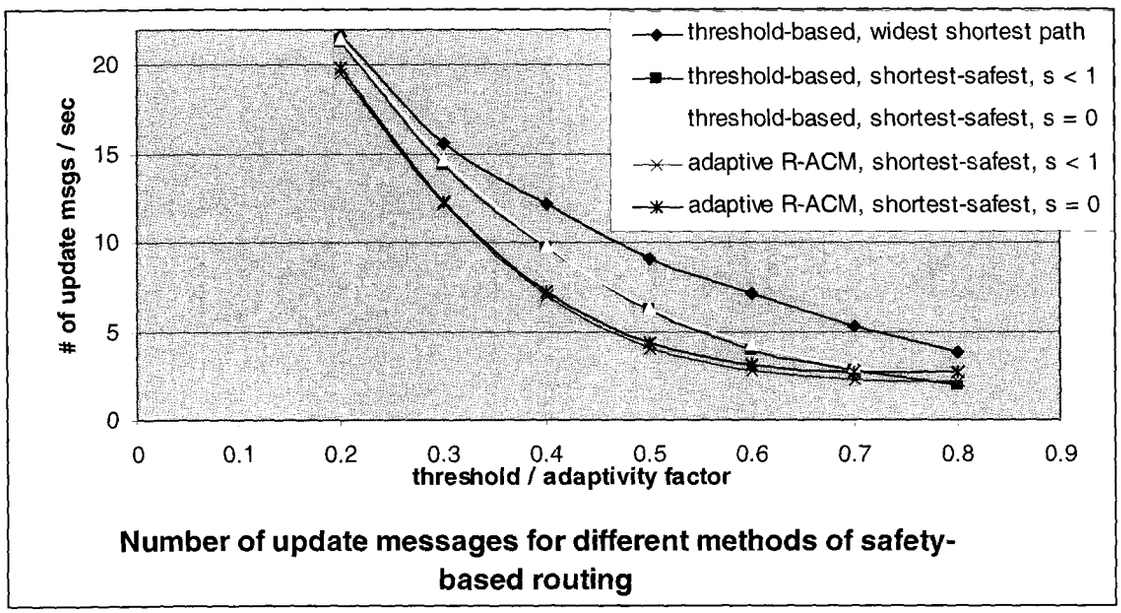


Figure 5-2: Number of Update Messages for Different Methods of Safety-Based Routing

5.2 Conclusion

In this Chapter, we have shown that for smaller threshold values the results of the threshold-based update policy with safety-based routing demonstrate a better performance in terms of its packet acceptance rate, when compared to the results of the threshold-based policy implemented with the widest shortest path. For larger values of the threshold, the packet acceptance rate drops and the performance is rather poor compared to the other algorithm. The adaptive R-ACM update policy implemented with its compatible safety-based routing demonstrates a better performance in terms of the number of update messages when compared with the threshold-based update policy with its own version of safety-based routing. This further reduces the network traffic, while producing a slightly lower performance in terms of the packets acceptance ratio. Therefore, where the main goal is to optimize the number of update messages used to flood the network with the marginal loss of the resultant packet acceptance rate, we propose that the adaptive R-ACM update policy with its compatible safety-based QoS-based routing, be selected as the routing update policy by network designers.

Chapter 6: CONCLUSIONS

6.1 Summary

In this Thesis, we first presented a brief study of routing, routing algorithms, routing protocols and Quality of Service (QoS) based routing. Nowadays the number of real time applications which require guaranteed network resources such as available bandwidth of the links is growing. This makes concepts like QoS-based routing that address the concerns of such applications considerably more important in an integrated services network environment. The importance of the concepts such as QoS-based routing and its associated update policy mechanisms can also be perceived by observing the growing number of research materials and investments in this area of research. We believe that we have succeeded in showing the advantages of applying QoS-based routing in a network. However, we also made it clear that the advantages and benefits of such applications cannot be obtained without jeopardizing the optimal usage of other resources. In the context of the QoS-based routing, trying to keep all routers in the network aware of the latest status of the links of the network has its advantages, yet it is also the effect of adding more computational and protocol overhead costs to the overall costs of the network. The computational cost is the unwanted result of the more and frequent computations in the path determination process of the QoS-based routing, and

the protocol overhead cost is caused by the flood of the update messages carrying the latest information regarding the state of the links of the network. In today's age of super fast and powerful processors and very huge and relatively cheap memories, it is easy to offset and compensate the so-called computational cost. But on the other hand, the job of reducing the cost imposed by the protocol overhead is rather difficult. Therefore, in the context of the QoS-based routing, there are solutions offered by renowned researchers that can address the issue of protocol overhead cost, and will yield better performance. These solutions are called update policies, which have been introduced in the context of the QoS-based routing in order to govern the frequency of the transmission of the update policies. Thus, these try to optimize the traffic level of the update messages, while at the same time trying to maintain the other types of performance such as the packet acceptance and packet loss ratios [6][19][24].

Histograms were introduced in Chapter 3. Histogram-like techniques that deal with optimizing query result size estimation were also introduced to set a baseline for clarifying the objective of this Thesis. The Attribute Cardinality Map (ACM), which is protected under the US Patent No. 6,865,567 issued on 8th of March 2005 [62], is one of these techniques, which was first proposed by Oommen and Thiyagarajah in [59]. The Rectangular ACM (R-ACM) and the Trapezoidal ACM (T-ACM) have been defined as two schemes of ACM. In this Thesis, we started off with the idea of finding a new application for R-ACM in routing. We successfully singled out the trigger-based update policy concept of QoS-based routing as a candidate for our objective.

As was mentioned, the rational behind introduction of the update policies was to reduce the protocol overhead cost in the context of the QoS-based routing. Through our

background study, we explained some of the issues we found with the existing update policies. To address those issues, we decided to concentrate on determining how we could transform the R-ACM into a powerful trigger-based update policy so as to carry out the mission of better enhancing the performance of QoS-based routing in terms of bringing down the cost of the protocol overhead without jeopardizing the other criteria of performance. This could only be achieved by triggering less number of update messages in comparison with the existing update policies, while keeping a relatively equal performance in terms of the packet loss and acceptance ratios. We finally were able to transform the R-ACM into two new update policies the first of which uses a constant and the second, which uses an adaptive threshold value in triggering the update messages. Thus they can be perceived to fall into the category of the threshold-based update policies. In this regard, we opted to choose *MaRS* as our simulation tool, and we used a test-bed common in similar studies [6] as our primary network configuration.

In Chapter 4, we explained how the R-ACM could be used to devise a new trigger-based update policy. The algorithm we used was discussed in detail with the introduction of our new trigger-based update policy along with examples of how it affects the routing performance of the network. We found two methods in which R-ACM can be incorporated as a trigger-based update policy, which were called the traditional R-ACM Trigger-Based update policy and the Adaptive R-ACM Trigger-Based update policy. For each of these new methods we have performed several experiments using different sets of parameters and compared their results with the results of the existing update policies on the same test-bed with similar sets of parameters. Figure 6-1 depicts number of update messages sent versus packet acceptance rate achieved from our simulation runs over our

sample network for the existing update policies that we studied, as well as for the two newly-introduced update policies.

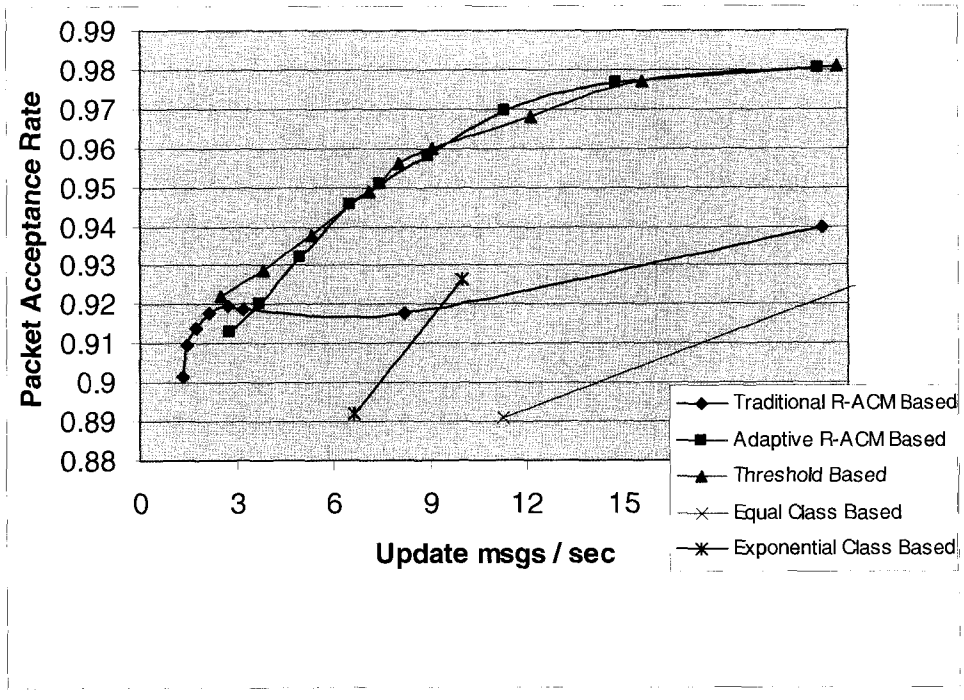


Figure 6-1: The Packet Acceptance Rate vs the Number of Update Messages for Different Update Policies

We know that an update policy outperforms other methods when it produces higher packet acceptance ratio while maintaining a lower number of update messages. In this regard, we can see that in most cases the performance of the adaptive R-ACM trigger-based update policy is as good as the performance of the threshold-based update policy. In some cases depending on the parameter set, if the adaptive R-ACM trigger-based update policy is used we can even perceive a lesser number of update messages sent while achieving the same packet acceptance ratio, hence yielding a better overall performance.

Chapter 5 introduces some QoS-based routing methods, among which we have focused our study on the concept of safety-based routing. Safety-based routing infers the range of the actual link state value and based on that range finds the safest path, which is a path with the highest probability to satisfy the bandwidth requirements of a request [5]. In this Chapter, we introduced a new safety-based routing technique compatible with the notion of the R-ACM. We then incorporated our Adaptive R-ACM update policy with this new safety-based routing method to present a trigger-based update policy with reasonable sensitivity to network bandwidth changes without a loss in its performance. To achieve this, we had to come up with a tailor-made estimation method to infer an estimation value for the current running mean of a given link based on the last advertised statistics of the link state and, of course, before the arrival of a new advertisement, to compute the estimation of the current running mean of a link bandwidth. We also proposed a simple formulation.

In Chapter 5, we have shown the results of our experiments performed using different sets of parameters for the Adaptive R-ACM update policy incorporated with our new safety-based routing algorithm. We also presented the results of the simulation runs for the existing methods such as threshold-based update policy incorporated with its own version of the safety-based routing method. The results of the threshold-based update policy with safety-based routing (implementing the shortest-safest algorithm) demonstrate a better performance (higher packet acceptance ratio) for smaller threshold values when compared to the results of the threshold-based policy implemented with widest shortest path. For larger values of the threshold, the packet acceptance rate drops and the performance is rather poor compared to the threshold-based policy with widest

shortest path algorithm. Our proposed Adaptive R-ACM update policy with safety-based routing (implemented with safest-shortest algorithm) shows better performance in terms of the number of update messages when compared with the threshold-based update policy with safety-based routing (implemented with the safest-shortest algorithm), and yet it produces slightly lower performance in terms of packets acceptance ratio.

Therefore in a given network if the objective is to achieve higher routing performance with the main focus on reducing the number of update messages flooding the network, we can certainly claim that we have introduced a powerful trigger-based update policy together with its compatible safety-based routing algorithm. By incorporating them together we will have an update policy sensitive enough to deal with changes in the state of the links of that network and hence achieving our utmost goal.

6.2 Future Work

In the Thesis, we have mentioned that for the purpose of comparing different update policies, we have used two key separate independent criteria by presenting their raw results, without using any weighted method to combine them as a single performance indicator. One of these criteria is "the number of update messages flooding the network", which defines the protocol overhead cost, and the other is "the packet acceptance rate", which specifies the ratio of the number of the packets that safely reach the destinations over the total number of packets produced after the transient phase. We believe that the problem of combining these two criteria and determining a single key performance indicator that is constructed as a weighted product of these two indicators, is yet to be solved and is still an open case study for future research.

As was mentioned in the context of the Thesis, in safety-based routing, we have considered a uniform distribution for the actual available bandwidth values for a given link. A better approach would be a measurement-based approach, which tries to find the real distribution of actual available bandwidth values. Other researchers have reported that this method produces a big overhead for the network. We believe that this is one area in which a flavor of ACM can be used.

In the Thesis, we have also mentioned that we are using an estimation method to compute the value of the current running mean of the available bandwidth of a link. More research can be done here to find a better estimation technique so as to revise our proposed estimation method to enhance the performance of the safety-based routing algorithm. It is obvious that a better estimation for the value of the current running mean of the available bandwidth of a link results in predicting a more accurate range for the link state value, and will in turn, result in further bringing down the number of update messages and help in a better packet acceptance ratio, hence achieving a great overall routing performance.

In this Thesis, we focused on applications of R-ACM in networking. We only proposed two applications for this powerful histogram-like algorithm in routing. We believe that the ACM methods can be incorporated with many other applications in networking as well as other areas. Wireless networking is a "hot" research area nowadays, and is the focus of a number of studies such as the wireless mesh. We believe that ACM methods have great potential in being incorporated in one or more aspect(s) of these studies, and we encourage researchers to consider using ACM-based methods to enhance the performance of such networks.

BIBLIOGRAPHY AND REFERENCES

[1] C. Alaettinoglu, K. Dussa-Zieger, I. Matta, O. Gudmundsson, A. U. Shankar, "*MaRS (Maryland Routing Simulator) – Version 1.0 – Programmer's Manual*". July 1991.

[2] C. Alaettinoglu, K. Dussa-Zieger, I. Matta, A. U. Shankar, "*MaRS (Maryland Routing Simulator) – Version 1.0 – User's Manual*". June 1991.

[3] C. Alaettinoglu, A.U. Shankar, K. Dussa-Zieger, I. Matta, "*Design and Implementation of MaRS: A Routing TestBed*", Institute for Advanced Computer Studies and Department of Computer Science, University of Maryland, Maryland.

[4] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg and M. Terpstra, "*Routing Policy Specification Language (RPSL)*", RFC 2622, June 1999.

[5] G. Apostolopoulos, R. Guerin and S. Kamat, "*Improving QoS Routing Performance under Inaccurate Link State Information*", In Proceedings of ITC'16, June 1999, pp. 1351-1362.

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

[7] G. Apostolopoulos, D. Williams, S. Kamat, R. Guerin, O. Orda and T. Przygienda, "*QoS Routing Mechanisms and OSPF Extensions*", RFC 2676, August 1999.

[8] A. Ballardie, "*Core Based Trees (CBT version 2) Multicast Routing*", RFC 2189, September 1997.

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

[10] A. Berson, S. Smith, and K. Thearling, "*An Overview of Data Mining Techniques*", Excerpted from the book "Building Data Mining Applications for CRM".

[11] S. Bhattacharyya, J. T. Buck, ..., "*The Almagst, Vol. 1 – Ptolemy 0.7 User's Manual*", College of Engineering, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California.

[12] S. Bhattacharyya, J. T. Buck,..., "*The Almagst, Vol. 2 – Ptolemy 0.7 Programmer's Manual*", College of Engineering, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California.

[13] R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC 1633, June 1994.

[14] J. Case, M. Fedor, M. Schoffstall and J. Davin, "*Simple Network Management Protocol (SNMP)*", RFC 1157, May 1990.

[15] M. C. Chan, Y. Lin and X. Wang, "*A Scalable Monitoring Approach for Service Level Agreements Validation*", In Proceedings of International Conference on Network Protocols (ICNP), pages 37-48, Osaka, Japan, November 2000.

[16] S. Chen and K. Nahrstedt, "*Distributed QoS Routing with Imprecise State Information*" *ICCCN'98*, October 1998.

[17] J. Chung and M. Claypool, "*NS By Example*", WPI Computer Science.

[18] Cisco, "*Open Shortest Path First*", Internetworking Technology Handbook. 20 Aug 2005, http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ospf.htm

[19] Cisco, *"Routing Basics"*, Internetworking Technology Handbook. 20 Aug 2005, http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/routing.htm.

[20] Cisco, "*Routing Information Protocol*", Internetworking Technology Handbook. 20 Aug 2005, http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/rip.htm.

[21] L. Coene., "*Multirouting*", Internet Draft, Siemens, February 2002.

[22] L. Coene, "*Multihoming issues in the Stream Control Transmission Protocol.*", Internet Draft, Siemens, February 2002.

[23] D. Comer, "*Internetworking with TCP/IP, Principles, Protocols, and Architecture*", Prentice-Hall, Englewood Cliffs, NJ, 1988.

[24] E. Crawley, R. Nair, B. Rajagopalan, H. Sandick, "A Framework for QoS-based Routing in the Internet", RFC 2386, August 1998.

[25] R. Draves and B. Akbari., "*Default Router Preferences and More Specific Routes.*", Internet Draft, Microsoft, November 2000.

[26] D. Farinacci, T. Li, T. Hanks, D. Meyer and P. Traina, "*Generic Routing Encapsulation (GRE)*", RFC 2784, March 2000.

[27] B. Fortz, J. Rexford and M. Throup, "*Traffic Engineering with Traditional IP Protocols*". IEEE Communication Magazine, Volume 40, Issue 10, pages 118-124, October 2002.

[28] V. Fuller, T. Li, J. Yu and K. Varadhan, "*Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy*", RFC 1519, September 1993.

[29] R. Guerin and A.Orda, "*QoS Based Routing in Networks with Inaccurate Information: Theory and Algorithms*", in proceedings of INFOCOM, 1997.

[30] T. Hain, "*An IPv6 Provider-Independent Global Unicast Address Format*", Internet Draft, Cisco, March 2002.

[31] B. Harangsri, "*Query Result Size Estimation Techniques in Database Systems*", University of New South Wales, Dissertation, 1998.

[32] D. Haskin, "*A BGP/IDRP Route Server alternative to a full mesh routing*", RFC 1863, October 1995.

[33] C. Hedrick, "*Routing Information Protocol.*", RFC 1058, June 1998.

[34] R. Hinden, "*Applicability Statement for the Implementation of Classless Inter-Domain Routing (CIDR)*", RFC 1517, September 1993.

[35] C. Huitema, "*An Experiment in DNS Based IP Routing*", RFC 1383, December 1992.

[36] G. Huston, "*Management Guidelines & Operational Requirements for the Address and Routing Parameter Area Domain ("arpa").*", RFC 3172, September 2001.

[37] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, H. Zheng, "*Overview of the Ptolemy Project*", Department of Electrical Engineering and computer Science, University of California, Berkeley, California.

[38] T. Imielinski and J. Navas, "*GPS-Based Addressing and Routing*", RFC 2009, November 1996.

[39] Y. Ioanidis and S. Christodoulakis, "*On the propagation of errors in the size of join resulrs*", In Proceedings of the ACM-SIGMOD Conference, pages 268-277, 1991.

[40] Y. E. Ioannidis and S. Christodoulakis, "*Optimal Histograms for Limiting Worst-case Error Propagation in the Size of the Join Results*", ACM Transactions on Database systems, 1993.

[41] V. Jacobson, "*Compressing TCP/IP Headers for Low-Speed Serial Links*", RFC 1144, February 1990.

[42] G. Kessler and S. Shepard, "*A Primer on Internet and TCP/IP Tools and Utilities*", RFC 2151, June 1997.

[43] S. Kille, "*X.400-MHS use of the X.500 Directory to support X.400-MHS Routing*", RFC 1801, June 1995.

[44] G. F. Lucio, M. P. farrera, E. Jammeh, M. Fleury, M. J. Reed, "*OPNET modeler and NS-2: Comparing the Accuracy of Network Simulators for Packet-Level Analysis using a Network testbed*", University of Essex, Colchester, United Kingdom.

[45] Q. Ma and P. Steenkiste, "*On Path Selection for Traffic with Bandwidth Guarantees*", Proceedings of the 1997 International Conference on Network Protocols (ICNP '97), 1997.

[46] Q. Ma and P. Steenkiste, and H. Zhang, "*Routing High-Bandwidth Traffic in Max-Min Fair Share Networks.*", In ACM SIGCOMM96, pages 206-217, Stanford, CA, August 1996.

[47] G. Malkin, "*RIP Version 2*", RFC 2453, November 1998.

[48] G. Malkin, "*RIP Version 2 Carrying Additional Information*", RFC 1723, November 1994.

[49] G. Malkin, "*RIP Version 2 Protocol Applicability Statement*", RFC 1722, November 1994.

[50] P. Marques and F. Dupont, "*Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing*", RFC 2545, March 1999.

[51] A. Marshall and I. Olkin, "*Inequalities: Theory of Majorization and Its applications*", academic Press, New York, 1979.

[52] K. McCloghrie, "*Management Information Base for Network Management of TCP/IP-based internets.*", RFC 1156, May 1990.

[53] K. McCloghrie, F. Baker and E. Decker, "*IEEE 802.5 Station Source Routing MIB using SMIv2*", RFC 1749, December 1994.

[54] K. McCloghrie, D. Farinacci and D. Thaler, "*IPv4 Multicast Routing MIB.*", RFC 2932, October 2000.

[55] K. McCloghrie and M. Rose, "*Management Information Base for Network Management of TCP/IP-based internets: MIB-II*", RFC 1213, March 1991.

[56] D. L. Mills, "*Exterior Gateway Protocol Formal Specifications*", RFC 904, April 1984.

[57] J. Moy, "*OSPF version 2.*", RFC 2328, April 1998.

[58] S. Nelakuditi, Z.L. Zhang, and Rose P. Tsang, "*Adaptive Proportional Routing: A Localized QoS Routing Approach*", In IEEE Infocom, April 2000.

[59] B. J. Oommen and M. Thiyagarajah, "*Benchmarking Attribute Cardinality Maps for Database Systems Using the TCP-D Specifications.*", IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-33(B), December 2003, pp. 913-924.

[60] B. J. Oommen and M. Thiyagarajah, "*Query result size estimation using a novel histogram-like technique: The rectangular attribute cardinality map*", Proceedings of the 1999 International Symposium on Database Engineering & Applications, 1999.

[61] B. J. Oommen and M. Thiyagarajah, "*Rectangular Attribute Cardinality Map: A New Histogram-Like Technique for Quality Optimization*". Technical Report, TR-99-01, School of Computer Science, Carleton University, Ottawa, Canada, January 1999.

[62] B. J. Oommen and M. Thiyagarajah, "*Method of generating attribute cardinality maps*", United States Patent No. 6,865,567 issued on March 8, 2005.

[63] J. Parker, "*Management Information Base for IS-IS*", Internet Draft, Axiowave Networks, January 2004.

[64] G. Piatetsky-Shapiro and C. Connell, "*Accurate Estimation of the number of Tuples Satisfying a Condition*", in Proceedings of the ACM SIGMOD Conference, 1984.

[65] "*Ptolemy II*", http://ptolemy.eecs.berkeley.edu/ptolemyII.

[66] "*Ptolemy Seamlessly Supports Heterogeneous Design*", RASSP Enterprise Newsletter, Vol. 1, No. 3, August 1994.

[67] R. Razavi, "*How Routing Algorithms Work*", http://computer.howstuffworks.com/routing-algorithm2.htm.

[68] Y. Rekhter and T. Li, "*Implications of Various Address Allocation Policies for Internet Routing*", RFC 2008, October 1996.

[69] W. Richard Stevens, "*TCP/IP Illustrated, Volume 1, The Protocols*". Addison-Wesley, MA, 1994.

[70] M. Rose, "*ISO Presentation Services on top of TCP/IP-based internets*", RFC 1085, December 1988.

[71] M. Rose and K. McCloghrie, "*Structure and Identification of Management Information for TCP/IP-based Internets*", RFC 1155, May 1990.

[72] P. Selinger, D.C. M.M. Astrahan, R. Lorie and T, Price, "*Access path selection in a Relational Database Management System*", In Proceedings of ACM-SIGMOD Conference, 1979.

[73] A. Sheikh, J. Rexford and K. G. Shin, "*Evaluating the Overheads of Source-Directed Quality-of-Service Routing*". Proc. IEEE International Conference on Network Protocols (ICNP '98), October 1998, Austin, TX, pp. 42-51.

[74] A. Singh, W. Dally, A. Gupta and B. Towles, "*GOAL: A Load Balanced Adaptive Routing Algorithm for Torus Networks*", In the Proceedings of 30th Annual International Symposium on Computer Architecture, pp. 194- 205, .June 2003.

[75] T. Socolofsky and C. Kale, "*A TCP/IP Tutorial.*", RFC 1180, January 1991.

[76] C. Spurgeon, "*List of University of Texas Network System (UTnet) Guides and Documents*", The University of Texas at Austin, Austin, TX, May 1990.

[77] M. Steenstrup, "*An Architecture for Inter-Domain Policy Routing*", RFC 1478, June 1993.

[78] M. Steenstrup, "*Inter-Domain Policy Routing Protocol Specification: Version 1*", RFC 1479, July 1993.

[79] R. Stine, "*FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices*", RFC 1147, April 1990.

[80] M. Thiyagarajah, "*Attribute Cardinality Maps: New Query Result Size Estimation Techniques for Database Systems*", Ph.D. Thesis, School of Computer Science, Carleton Univ., Ottawa, Canada, May 1999.

[81] N. Toumba and I. Miloucheva, "*Pattern based spatio-temporal Quality of Service analysis for capacity planning*".

[82] C. Villamizar, C. Alaettinoglu, R. Govindan and D. Meyer, "*Routing Policy System Replication*", RFC 2769, February 2000.

[83] C. Villamizar, C. Alaettinoglu, D. Meyer and S. Murphy, "*Routing Policy System Security*", RFC 2725, December 1999.

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

[85] Z. Wang and J. Crowcroft, "*Quality-of-Service Routing for Supporting Multimedia Applications*", IEEE JSAC, 14(7): 1288-1234, September 1996.

[86] U. Warrier and L. Besaw, "*The Common Management Information Services and Protocol over TCP/IP (CMOT)*", RFC 1095, April 1989.

[87] X. Yuan and A. Saifee, "*Path Selection Methods for Localized Quality of Service Routing*", IC3N'01, Oct. 2001.

[88] X. Yuan, W. Zheng, "*A Comparative Study of Quality of Service Routing Schemes That Tolerate Imprecise State Information*", Florida State University, Department of Computer Science, Technical Report TR--010704, 2001.