NETWORK TRAFFIC ENGINEERING USING LINEAR REGRESSION APPROACH

VATHSALA DEVI D/O KUNALAN

FACULTY OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2006

NETWORK TRAFFIC ENGINEERING USING LINEAR REGRESSION APPROACH

A thesis submitted to
the Faculty of Computer Science & Information Technology,
University of Malaya
in Partial Fulfillment of the Requirements for
the Degree of Master of Computer Science

(12 Credit Hours)

by

VATHSALA DEVI D/O KUNALAN
(WGA020044)

May, 2006

ABSTRACT

Quality of Service (QoS) routing is becoming a very important criteria in Internet for supporting the ever-increasing diverse multimedia applications that demand very high level quality of service from the underlying network. To achieve this, QoS-enabled routers need to maintain accurate view of the network resource availability by exchanging global network state information among themselves at appropriate intervals. Frequent dissemination of global network state information introduces two major problems: increased *computational costs* and higher *protocol overheads*. In this thesis, a new mechanism that disseminates link state updates based on the bandwidth utilization trend of a link, called **TE-LR** (***T***raffic ***E***ngineering using ***L***inear ***R***egression), is proposed. The main idea of the mechanism is to sample the bandwidth utilization ratio of a link at regular intervals and use these sampled data to construct linear regression line equations. The tangent value obtained from the equation is used to decide when to send link state updates. Simulation results showed that the proposed mechanism had been successful in reducing the update message overheads by almost 78% with minimal impact to other parameters such as packet loss ratio, packet commit ratio, average link utilization and average end-to-end delay.

ACKNOWLEDGEMENT

First and foremost, I would like to thank GOD for giving me strong heart and willpower to endeavor into this study. Without HIS grace and blessings, I would not have been able to complete this thesis. I also owe my gratitude to a number of kind souls who had made my thesis research work a great experience.

First in the list is my wonderful supervisor, Dr.Phang Keat Keong. His kindness and understanding had improved my level of self-confidence that helped me to carry on with this work despite many times meeting the dead end. His words of encouragement and motivation had also helped me to boost my self-estime.

A special thanks goes to Dr.Ling Teck Chaw for his valuable ideas and thoughts that helped me to finalize my research scope and objectives. Without his support and constructive criticisms, I would have still been struggling to decide which area of research to undertake. My gratitude also goes to Mr.Por Lip Yee, Mr.Liew Chee Sun and Mr.Ang Tan Fong for their insightful ideas and opinions during discussions.

There is a very special person in my life that I would like to convey my thanks to, my fiancé Mr.Sajilal Divakaran. Without his persuasion and encouragement, I would not be sitting here writing this acknowledgment. His unconditional love and support had really been an energy booster for me to continuously concentrate on my research work without any interference. I would like to dedicate this thesis to him as it is his dream that I fulfilled. This thesis would not have been possible without his love. He is GOD's greatest gift to me.

Next comes my parents, brother and sister-in-law. Their patience and understanding throughout this study had really given me motivation to complete this thesis. My special thanks goes to my mother for bringing me to this world in the first place, educating me and being there for me whenever I needed her. Her love and support had been my source of energy and strength.

Last but not least, I would like to thank the following individuals who worked with me in the Network Research Laboratory of University of Malaya: Soo Wooi King, Ng Eng Seong, Chan Siew Yin, Lim Gek Pei, Ngo Foong Kiew, Ng Wai Keat, Gawri Kumar and Judy Anne Sharmini. These people had made my working environment lively. Heartfelt thanks especially goes to Chan Siew Yin and Lim Gek Pei who had helped me to learn and use UMJaNetSim to implement and simulate my proposed mechanism. They had been supporting and guiding me from the beginning till the end of this thesis.

Finally, I would like to extend my gratitude to Mr. Trevor Ward for helping me proofread this thesis report.

This research is supported by the Postgraduate Scheme.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ALSP	Aggregated Level Simulation Protocol
AS	Autonomous System
ATM	Asynchronous Transfer Mode
ATMLSR	ATM Label Switching Router
BE	Best Effort
BGP	Border Gateway Protocol
BTE	Broadband Terminal Equipment
B-TE	Broadband Terminal Equipment
CA-OSPF	Cost-Adaptive Open Shortest Path First
CBR	Constraint-Based Routing
CBRIPApp	Constant Bit Rate IP Application
CL	Controlled-Load Service
COL	Competitive On-Line Routing Algorithm
DARPA	Defence Advanced Research Projects Agency
DCU	Dynamic Cost-based Update Policy
DCUCT	DCU Update with Cost-Threshold Policy
DCUCTT	DCU Update with Cost-Threshold and Hold-Down Timer Policy
DCUH	DCU Update with Hyteresis Policy
DiffServ	Differentiated Services
DIS	Distributed Interactive Simulation
DS field	Differentiated Services field
EF-PHB	Expedited-Forwarding PHB
EGP	Exterior Gateway Protocol

FIFO	First-In First-Out
FTP	File Transfer Protocol
FU	Full Update Policy
GS	Guaranteed Service
GUI	Graphical User Interface
HFC	Hybrid Fiber Coax
HLA	High Level Architecture
IDRP	Inter-Domain Routing Protocol
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IntServ	Integrated Services
IP	Internet Protocols
IPBTE	IP Broadband Teminal Equipment
LAN	Local Area Network
LBNL	Lawrence Berkeley National Laboratory
LRBU	Logarithm of Residual Bandwidth Update Policy
LSA	Link State Advertisement
LSR	Label Switching Router
MPLS	Multi-Protocol Label Switching
OSPF	Open Shortest Path First
PHB	Per-Hop Behaviour
PNNIU	PNNI Time-based Update Approach
QoS	Quality of Service
QOSPF	QoS Extension to OSPF

QoS-OSPF	QoS Extension to OSPF
RIP	Routing Information Protocol
TBP	Ticket-Based Probing
TCP	Transmission Control Protocol
TE-LR	Traffic Engineering using Linear Regression
TE-QOSPF	Traffic Engineering enhancements to QoS-OSPF
VINT	Virtual InterNetwork Testbed

CHAPTER 1

INTRODUCTION

The sudden burst of Internet traffic in recent years has inspired and forced researchers to design a new framework that incorporates Quality-of-Service (QoS) in Internet Protocol (IP) networks. The increasing growth of diverse Internet applications ranging from real-time multimedia applications to mission-critical business transactions demand certain level of guaranteed service from the underlying network technology. Unfortunately, today's Internet which is connectionless and best-effort has become inadequate to provide QoS-guaranteed services to these applications.

The best-effort Internet treats all packets equally regardless of their importance. They are forwarded on a First-In-First-Out (FIFO) basis without any regards to the resource availability on the chosen path and the traffic's service requirements. This results in significant performance deterioration in terms of longer packet delivery delay and excessive packet drops during heavy congestion periods. This scheme may work well for conventional text-based applications but it does not support the stringent bandwidth requirements and delay constraints of high-speed applications.

To cater for such applications, a real connection-oriented infrastructure has to be established over the underlying connectionless IP infrastructure, i.e., a QoS-based routing mechanism that runs on an IP network without impacting the performance of the existing best-effort services. QoS-based routing or simply QoS routing can be defined as the process of selecting the path to be used by the packets of a flow based on its QoS requirements, e.g., bandwidth and delay [Apostolopoulos98i]. This will not only improve the services provided to user applications but also improves the overall network efficiency in terms of better resource utilization. Over the past several years, the Internet Engineering Task Force (IETF) group has been working to define new Internet QoS models and proposed a framework that contains the following components: Integrated Services (IntServ) [Braden94], Differentiated Services (DiffServ) [Carpenter02], Multi-Protocol Label Switching (MPLS) [Armitage00] and Constraint-Based Routing (CBR) [Younis03]. Till now, there is not a single solution that can provide the exact QoS support required by today's assorted Internet applications.

Most QoS routing algorithms require the most accurate information of the current state of the network on which they operate. Maintaining accurate global network state information in a large network with dynamic nature is impossible due to many factors such as non-negligible propagation delay, infrequent link-state update due to overhead concerns, link-state update policies used and hierarchical state aggregation [Yuan02]. Inaccurate network state information causes non-optimal path selection and higher call blocking rate which could degrade the network's performance significantly. In order to ensure higher percentage of accuracy, nodes in the network must be updated with the link state information at the proper time interval. The rule that governs when to perform an update is called the ***link state update policy*** [Guerin97(i)] which will be discussed in detail in Chapter 2.

1.1 Motivation

QoS routing, though sounds theoretically simple, is difficult to be deployed due to various problems that it manifests such as diverse QoS specifications, dynamically changing network state and the need to co-exist with best effort traffic. These issues will be discussed in detail in Chapter 2 and readers are referred to [Oliveira01, Bruin06] for more information. One of the main problems of deploying QoS routing mechanism is the dynamic nature of the network itself. The network grows in terms of size and the number of user applications dramatically every second and with it the demand for more sophisticated QoS-based services grows too. To provide better QoS-based routing services, the underlying QoS routing algorithm must know exactly the current state of the network such as the network topology and the link state information (i.e., available bandwidth, delay of the links, etc) of every link on the network. However, obtaining accurate network state information is impossible due to the continuously changing network states caused by traffic load fluctuations.

One way to achieve this is by transmitting the network state information to every router on the network frequently so that whenever there is a change in the network state each router is made known immediately. This is a very impractical solution because frequent update messages themselves have the potential of congesting the network and consuming network resources causing higher protocol overheads. Apart from that, routers are also burdened with the extra job of computing routing tables which in turn increases the computational overhead.

On the other hand, routers have the option of transmitting update messages periodically at a fixed interval. This however introduces another major problem. If the interval is very short, the updates will be sent frequently resulting in increased protocol overheads . If the interval is very long, then the network states transmitted will be outdated and no longer reflect the actual state of the network at that point of time. This will cause routing algorithms to make incorrect routing decisions and thus degrade the entire network performance.

Having these problems in mind, it is the motivation of this thesis to devise a simple mechanism which can be incorporated into the existing QoS routing protocol that operates with the existance of inaccurate link state information. Specifically, the objective of the mechanism is to control the frequency of the link state generation and distribution so as to reduce the overheads incurred by the update frequency as discussed above.

1.2 Objectives of the Thesis

After conducting a detailed study of IP QoS models and their related issues, the next important task of the thesis is to study various mechanisms that are capable of reducing both the protocol and computational overheads in a network. Thus, the main objectives of the thesis can be summarized as follows.

- To study and undertand various problems related to QoS routing especially the problem of routing overheads.
- To propose a new link state update mechanism that can overcome the routing overheads problems.
- To develop and evaluate the proposed mechanism on a suitable simulation environment.
- To analyze the effectiveness of the proposed mechanism in reducing routing overheads, specifically the protocol overheads.

1.3 Scope of the Thesis

As outlined in the objectives section above, it is clear that the main emphasis of this thesis is to propose and implement a mechanism that reduces protocol overheads due to inappropriate dissemination of link state update messages. Hence, the scope of the thesis has been narrowed down to focus on this part of the objective and has been specified as follows:

- To study the impact of inaccurate link state information on the existing link state update mechanism called QoS-OSPF.
- To investigate how QoS-OSPF overcome the identified problems.
- To propose a new link-state update mechanism that can improve performance especially in reducing protocol overheads.
- To develop the proposed mechanism and test it using an appropriate QoS routing algorithm to ensure its correctness.
- To develop a suitable simulation environment on UMJaNetSim network simulator.
- To analyze and evaluate the performance of the proposed mechanism.

1.4 Thesis Organization

The remainder of the thesis is organized as follows: Chapter 2 discusses the concept of QoS routing and the main building blocks of a QoS routing mechanism. Further, various mechanisms proposed in literatures to control the amount of information flooded by link state advertisements are described in detail. Finally, the fundamentals of data sampling and regression concepts which form the basis of this research work are also presented. Chapter 3 lays out the design of the proposed link state update mechanism. The details of how the bandwidth utilization ratios of the links are sampled and how the construction of the linear regression line equation is performed is provided. Also, a detailed explanation of how the linear regression line is used to determine when to trigger link state updates is given in this chapter.

Chapter 4 presents the implementation of the simulation environment and the methodology used. Some of the existing network simulators are reviewed in this chapter followed by a brief overview of the simulator used in the implementation and testing of the propsed mechanism A number of test cases are proposed to test the proposed mechanism and the test results are layed out as output traces. Finally, the proposed mechanism is tested using various network topologies and parameter settings to evaluate its correctness.

Chapter 5 provides the actual topology and parameter setting details during the actual simulation run of the proposed mechanism. The results were analyzed and the proposed mechanism's effectiveness was evaluated. Chapter 6 concludes the thesis and the main contribution of the thesis is presented. Finally, the possible future enhancements to the proposed mechanism is highlighted.

CHAPTER 2

LITERATURE REVIEW

This chapter presents an overview of the QoS routing concept and its strategies. A detailed literature review will be conducted to identify the various mechanisms proposed to achieve QoS routing in a network with dynamic nature. The problem of overheads related to link state update policies are discussed thoroughly. Other issues related to QoS routing deployment and the proposed solutions are also presented. Finally, the linear regression model that will be used by the proposed mechanism is discussed.

2.1 Network Traffic Engineering and Quality of Service

Traditional routing protocols were merely aimed at providing faster packet processing and quicker convergence in the case of failures, without any regards to the security and packet delivery guarantees. Traffic flows with different service requirements were treated equally and network resources were not efficiently allocated and utilized. To handle this situation, the concept of network ***traffic engineering*** was brought into the picture whose main objective is *performance optimization* by efficient resource utilization. Two main benefits offered by good traffic engineering are [Lim04]:

- Improving the efficiency of overall network resources usage.
- Providing greater end-to-end quality of service to each traffic flow.

Hence, there has been a general move towards networks that provides services based on the QoS demands of user applications. As stated in the previous chapter, the IETF

QoS models such as IntServ, DiffServ, MPLS and CBR exist with the intention to induce QoS capabilities into the underlying network architecture.

2.1.1 Internet QoS Models

In this sub-section, a detailed description of the above-mentioned IETF QoS models with their pros and cons are presented.

IntServ

The prime goal of IntServ model is to provide QoS support to individual flows by reserving the required resources apriori. This is done using the most commonly adopted reservation setup protocol called Resource Reservation Protocol (RSVP) [Braden97]. RSVP protocols use two types of messages to reserve the required resources along the path. They are the PATH message and RESV message. A PATH message contains the characteristics of the traffic sent by the source node and will be forwarded by the routers along the path to the destination node. Based on the traffic characteristics specified in the received PATH message, the destination node makes the resource reservation via the RESV message. Upon reception of the RESV message, an intermediate router can either reject the request if it cannot provide the requested resource or reserve the requested resource by allocating the link bandwidth and buffer space for the flow.

Two service classes are proposed by IntServ in addition to the Best-Effort Service [Wu99], namely 1) Guaranteed Service (GS), that provides a fixed delay bound for applications; and 2) Controlled-Load Service (CL), that provides reliable and enhanced best-effort service. The main problem of IntServ model is scalability. The amount of state information increases proportionally with the number of flows, placing huge storage and processing overhead on the routers.

DiffServ

The scalability problem imposed by IntServ model has led to the introduction of DiffServ model [Carpenter02, Chen03]. Unlike IntServ model, the DiffServ model places very light load on core routers by assigning the packet classification and conditioning tasks to the edge routers, leaving only the implementation of Behaviour Aggregate (BA) classification tasks to the core routers. DiffServ classifies packets entering the network by means of the Differentiated Services (DS) field [Stallings92]. The DS field is made up of 8 bits of which the leftmost 6 bits form the DS codepoint and the rightmost 2 bits are currently unused. The DS codepoint functions as a DS label used to classify packets for differentiated services. All IP packets having the same DS codepoint will be treated the same way by the routers regardless of which flow they belong to.

Packets entering the DS domain are forwarded by the core routers based on the Per-Hop-Behaviour (PHB) specification. PHBs define the queuing disciplines and packet dropping rules for each packet. Different PHBs can be associated with different differentiated services and at the moment two such PHBs exist: 1) Assured Forwarding PHB (AF-PHB) that guarantees reliable end-to-end packet delivery service during heavy congestion periods; and 2) Expedited Forwarding PHB (EF-PHB) that guarantees low-loss, low-delay, low-jitter and assured bandwidth services for packets. The main problem with the DiffServ model is its method of marking packets as IN and OUT. Packets that conform to its traffic profile are marked as IN and those that does not as OUT. All packets whether marked as IN or OUT are stored in a shared queue together regardless of their priority and will be treated equally. When congestion occurs, the IN packets may be dropped first and this causes performance degradation for TCP traffics.

MPLS

The emergence of MPLS was motivated by the need to standardize a base technology that integrates label swapping paradigm with network level routing [Rosen01]. In other words, MPLS can be seen as a convergence of connection-oriented forwarding techniques and the Internet routing protocol. In an MPLS network, every packet is associated with a fixed length label which makes them more manageable and flexible. Routers in an MPLS networks are called Label Switching Routers (LSRs) and they are classified as edge LSRs and core LSRs. Every incoming packet is assigned a fixed length label by the ingress edge LSRs and then forwarded to the egress edge LSRs via a number of core LSRs. Since packet forwardings are performed just by referring to the labels, scalability can be achieved. MPLS has a number of advantages compared to the other two models described above and these can be found in [Rosen01].

CBR

Most routing algorithms perform route computations that are subject to only one QoS constraint. CBR [Younis03], on the other hand, is a mechanism that computes routes subject to multiple constraints. Though the use of multiple constaints in real networks can lead to NP-Complete problem, simplifying the routing algorithm makes it possible. The goals of CBR is to select routes that meet a flow's QoS requirements and at the same time increase the network utilization. CBR takes into consideration a lot of factors when determining a route. These includes the network topology, the flow's QoS requirements, the link's resource availability and other network policies. Therefore CBR is capable of distributing network traffics more evenly by choosing a longer and lightly loaded path rather than a shorter but heavily loaded path. However, CBRs do have some drawbacks: 1) higher communication and computation overhead; 2) larger routing table size; 3) more resource consumption by longer paths; and 4) potential routing instability.

Though all the above models have their own ways of supporting QoS services, their limitations as discussed above suggests that relying on one single solution is not feasible. Therefore, a solution that is scalable, reliable and provides quantitative guarantees is required.

2.2 Concepts of QoS Routing

In any network-based communication system, the most integral part to its operation is routing. *Routing* is the process of forwarding data packets from one router to another within a single network or across networks, via the most appropriate path. Finding the most appropriate path on the other hand is the most crucial aspect of any routing algorithm and therefore every router in a network has to have a common view about the network topology and its resource availability.

There are two ways in which routing can be performed [Ma97]. The first is called *static routing* in which the network administrator himself or herself manually configures each router with a pre-computed set of routes that he or she knows to be suitable. The second method is called *dynamic routing* in which individual routers in the network perform the route configuration dynamically. Each router collects the link state of each of its interface and disseminates this information to every other router in the network. The collective link state information will then be used by a *routing algorithm* operating at the router to build a routing table consisting of all the feasible paths which can be used to forward the data packets to their destination(s). A comparison of both static routing and dynamic routing in terms of their advantages and disadvantages is given in [Ballew97].These are summarized below:

Static Routing

Advantages:

- The network administrator can easily control the path a packet takes to reach its destination since it is known in advance.
- Since there is no need to exchange information among routers the message overhead is minimal.
- Easy to configure on a small network due to its simplicity.

Disadvantages:

- When the number of nodes in the network grows, it becomes difficult for the network administrator to configure the routes manually thus making the algorithm non-scalable.
- Because of its *static* nature, it cannot adapt to changes in the network such as failure of links or nodes, movements of network segments, addition of new routers and so on.

Dynamic Routing

Advantages:

- Highly scalable due to its dynamic nature. This means that the network can grow in size without impacting the underlying routing mechanism.
- Any changes in the network can also be easily adapted in a dynamically routed network.
- When there is a network failure, dynamic routing can still perform its functions, most likely in a degraded fashion without causing any major interference to other mechanisms.

Disadvantages:

- Dynamic routing has increased message overhead due to the exchange of enormous routing information among the huge number of routers.
- In a network, not all routers can run dynamic routing scheme. When this is the case, then the only option is to use static routing.

With all the disadvantages inherent in both static and dynamic routing, it is difficult for a network administrator to choose the best routing scheme between the two. However, a simple solution is to use both schemes together, that is, to use static routing in one part of the network and dynamic routing in other parts. This type of routing is called the *hybrid routing scheme*. Ballew, S.M., in [Ballew97] had proposed one way of implementing this that is to use static routing at the access networks and dynamic routing at the core and distribution networks. The advantage of doing this is that at access network, the main components are user machines which are connected to a small number of routers. Since the route configuration complexity is very small and manageable by the network administrators, default routes will do in most cases. On the other hand, at core and distribution networks, there will be many routers with many connections requiring larger number of routing tables. Also, the network becomes dynamic in nature, thus requires a more scalable dynamic routing scheme. It is in this dynamic routing scheme one can incorporate QoS features to provide better service quality to applications requiring special kind of treatments known as QoS routing.

2.2.1 Objectives of QoS Routing

As stated in the previous chapter, the main goal of QoS routing is to find the most feasible path among the ones that can satisfy the maximum possible number of flows with QoS requirements and at the same time maximize the overall resource utilization. Therefore the objectives of QoS routing can be summarized as follows [Crawley98]:

- *Dynamic determination of feasible paths:* Any QoS routing algorithm should be able to determine the best path that has the highest chance of satisfying the QoS requirements of a given flow even though there are many choices. Feasible paths may be selected based on constraints such as high bandwidth, low path cost, low delay and etc. For instance, if the constraint is high available bandwidth, then a path is said to be *feasible* if the total amount of the unused bandwidth of all links on the path is greater than a flow's requested bandwidth value.
- *Optimization of resource usage:* A QoS routing algorithm that depends on the network's state information can help optimize network resource utilization by maximizing the overall network throughput.

- *Graceful performance degradation:* During heavy network load conditions, a QoS routing algorithm can provide better throughput and a more graceful performance degradation.

2.2.2 Interior Routing Protocols

QoS routing is composed of two basic entities [Oliveira01]. The first is the collection of global network state information, while the second is the path selection algorithm that uses the collected information to perform path calculation and selection. The process of collecting network state information is the responsibility of *routing protocols* such as OSPF which will be discussed in detail later.

There are two types of routing protocols namely Interior Gateway Protocol (IGP) and Exterior Gateway Protocol (EGP). When IGP is used, routers within a single autonomous system (AS) exchange routing information among themselves. An AS is a collection of networks, or more precisely, the routers joining those networks that are under the same administrative authority and share a common routing strategy [Stallings02]. The most common IGPs are Routing Information Protocol (RIP) [Malkin98] and Open Shortest Path First (OSPF) [Moy98]. On the other hand, when EGP is used, routers from more than one AS exchange routing information among themselves. For this, they have to have a minimal information concerning other ASs outside their own AS. The most common EGPs are Border Gateway Protocol (BGP) [Rekhter95] and Inter-Domain Routing Protocol (IDRP) [Huston01]. Since the main focus of this thesis is IGP, EGP protocols will not be discussed any further and readers are referred to literature for more details.

2.2.2.1 Routing Information Protocol (RIP)

RIP is a very simple protocol, which can be implemented in small networks with very little complexity and overhead. It runs on top of UDP (User Datagram Protocol) and uses a technique known as *distance-vector routing* to distribute routing information and perform route computation [Malkin98]. A typical metric used in the computation of shortest-path is the *number of hops* (i.e., the number of routers a packet has to visit before reaching its destination). Since RIP operates in small networks, the maximum number of hops is limited to 15. To indicate that a particular destination is unreachable, the value of 16 (for infinity) is used.

An RIP-enabled router operates by sending the routing information in the form of update messages to all its neighbors every 30 seconds. Two routers are said to be neighbors if they are both directly connected to the same network. If a router, say A does not receive an update message from a neighboring router say B within 180 seconds, it assumes either B has crashed or the link connecting A to B has failed. In either case, router A assigns 16 in the routing table indicating that there is no valid route to B. If another router say C knows a valid route to B, router A replaces 16 with the new route as stated by C. The reason why 180 seconds is used as the update interval is due to UDP's nature of not providing guarantees for data packet deliveries. If there is a link break-down or congestion, data packets may be lost forever and the destination may not be aware of it at all.

Despite its unreliability, RIP continues to be popular because of its simplicity. One of its main advantage is that it reduces routing loops by the use of *split horizon with* *poisoined reverse* rule. Besides that, RIP also reduces routing overheads caused by excessive distribution of update messages by randomly delaying the update trigger.

Though simple, RIP presents a number of disadvantages that deemed the protocol to be less popular when it comes to larger networks [Stallings02]. The first problem is that it cannot be deployed in a network with more than 15 hops because the maximum admittable hop count is 15. Thus, when used in this type of network, data packets addressed for destinations allocated more than 15 hops away cannot be sent. Although modifications can be done to the algorithm, when it is applied in larger networks, convergence time could be lengthy and costly. The second problem is that due to its simplistic metric even though there is a better path the algorithm may only choose a sub-optimal path leading to slower packet delivery.

Thirdly, any RIP-enabled router may still accept updates from other routers including non-RIP-enabled routers. When a misconfigured non-RIP-enabled router sends an update to a RIP-enabled router, the entire configuration may be disrupted. Finally, RIP requires considerable amount of information transmission that eventually leads to considerable amount of propagation time during significant link state change. Therefore, to compensate these problems, a more scalable protocol called OSPF was introduced by [Moy98].

Distance-Vector Routing Protocol

In *distance-vector routing protocol*, routers exchange routing information among themselves that consists of vectors of known distances to other destinations at a fixed interval. Once this information is made available, routers can find a feasible route through the neighbour that provided the information to reach the destination. Three types of vectors are maintained by each router [Stallings02]:

(i) Link cost vector

$$W_x = \begin{bmatrix} w(x, 1) \\ . \\ . \\ . \\ w(x, N) \end{bmatrix} \tag{2.1}$$

where x is the current router, N is the number of networks to which router x is directly connected to and $w(x, i)$ is the link cost associated with the output side of each router for each attached network i.

(ii) Distance vector

$$L_x = \begin{bmatrix} L(x, 1) \\ . \\ . \\ . \\ L(x, N) \end{bmatrix} \tag{2.2}$$

where $L(x, i)$ is the current estimate of minimum delay from router x to network i and N is the number of networks router x is directly attached to.

(iii)Next-hop vector

$$R_x = \begin{bmatrix} R(x, 1) \\ . \\ . \\ . \\ R(x, N) \end{bmatrix} \tag{2.3}$$

where $R(x, i)$ is the next router in the current minimum delay route from router x to network i.

To determine the best path that can accommodate a QoS request, distance-vector routing protocol uses an algorithm called ***Bellman-Ford algorithm***. The algorithm is stated as follows in [Stallings02]: "*Find the shortest path from a given source vertex subject to the constraint that the path contain at most one link, then find the shortest path with a constraint of path of at most two links, and so on*". A formal description of the algorithm follows:

Definition:

s	=	source vertex
w(i, j)	=	link cost from vertex ***i*** to vertex ***j***; ***$w(i,i) = 0$***; ***$w(i, j) = \infty$*** if the two vertices are not directly connected; ***$w(i, j) \geq 0$*** if the two vertices are directly connected.
h	=	maximum number of links in a path at the current stage of the algorithm.
$L_h(n)$	=	cost of the least-cost path from vertex ***s*** to vertex ***n*** under the constraint of no more than ***h*** links.

The Algorithm:

```
1. [Initialization]
       $L_0(n) = \infty$, for all $n \neq s$
       $L_h(s) = 0$, for all $h$

2. [Update]
   For each successive h ≥ 0:
       For each $n \neq s$, compute

           $L_{h+1}(n) - \min_{j} [L_h(j) + w(j, n)]$

       Connect n with the predecessor vertex j that achieves the minimum, and eliminate
       any connection of n with a different predecessor vertex formed during an earlier
       iteration. The path from s to n terminates with the link from j to n.

3. [Repeat]
   Repeat step 2 until no more changes occur in the iteration.
```

2.2.2.2 Open Shortest Path First (OSPF)

OSPF is an interior gateway protocol used for routing between routers belonging to a single AS [Moy98]. In order to perform routing function, every OSPF-enabled router has to maintain an identical database describing the autonomous system's topology. This is achieved by sending each other information about the direct connections and links, which they have to other routers periodically using the *link-state routing protocol*.

The information in the database is then used by the router to construct shortest-path trees to every destination using the Dijkstra’s algorithm though other algorithms can be equally applied. To achieve scalability, OSPF partitions the AS in which it operates into several parts called *areas*. These areas are connected by a central *backbone area* (i.e Area 0). A router in an area knows the topology of only that area and sends routing information to routers within the same area. This allows significant reduction of routing traffic. When there is a need to send routing information to routers outside the area, *area border routers (ABR)* are used. An ABR summarizes the routing information received from routers belonging to other areas into single aggregated information and forwards it to the requesting router.

For a router to exchange routing information with its neighboring routers, it has to first know who its neighbors are. OSPF uses **Hello Protocol** to learn about other routers on their directly attached networks dynamically. The router does this by sending out a small *hello* packet to each of its interfaces every ten seconds. When a router receives the *hello* packet, it knows about the existence of the originating router and thus they become neighbors. All neighbor routers must have the same topological view of the network. If no *hello* packet is received from a particular neighbor for 40 seconds, that neighbor is considered down.

Link-State Routing Protocol

In contrast with distance-vector routing protocol, the *link-state routing protocol* do not exchange information about the direction to reach the destination. Instead, it provides information about the state of the links to which the router is directly connected. This information is disseminated throughout the network using a simple *flooding* technique. By doing this, routers can construct their own map of the network, compute best paths to every destination and populate their routing tables with these information.

When there is a change in one of the link states, a notification, called *link state advertisement (LSA)* is flooded throughout the network. Upon receiving the LSA, each router re-computes its routing table to reflect the current state of the network. Compared to distance-vector routing protocol, link-state routing protocol is more reliable, easier to debug and less bandwidth-intensive. On the other hand, it is also more complex and requires intensive memory. Despite its complexity the protocol still survives as the most popularly used protocol in OSPF. Due to this, we will assume that a link-state protocol is used in this thesis.

The link-state routing protocol uses ***Dijkstra's algorithm*** to perform route computation and the algorithm is stated as follows in [Stalling02]: *"Find the shortest-path from a given source vertex to all other vertices by developing the paths in order of increasing path length"*. The algorithm proceeds in stages. By the *k*th stage, the shortest-path to *k* vertices closest to (least cost away from) the source vertex have been determined; these vertices are in a set *T*. At stage *(k + 1)*, the vertex not in *T* that has the shortest path from the source vertex is added to *T*. As each vertex is added to *T*, its path from the source is defined.

2.3 QoS Extensions to OSPF

OSPF in its basic form does not provide support for QoS routing. Therefore, Apostolopoulous, et al. had proposed in [Apostolopoulos99iv] some extensions to the basic OSPF protocol that allows the incorporation of QoS routing mechanisms which takes into consideration the QoS demands of flows from various user applications. The OSPF protocol with QoS extension is known as QoS-OSPF or in short QOSPF and it is defined as an intra-domain routing protocol. The difficulty to apply the extensions lie in its need to support demands of QoS flows while at the same time ensuring that the best effort flows are not deprived of their services as well. Thus, the ultimate goal of QOSPF as stated in [Guerin97(i)] is to provide the requested QoS to the flows to improve their performances with minimal impact to the existing OSPF protocol.

To achieve this goal, a number of restrictions have been imposed on the proposed extensions by Apostolopoulos et al in [Apostolopoulos99iv]. The first restriction is that only the link available bandwidth and delay will be advertised as part of the extended LSA packets, and the path selection algorithm should consider only the bandwidth requirements of the flows when performing path computation and selection. These are necessary to ensure that the complexity and the amount of network resources allocated to flows are minimal. The second restriction is that the path selection algorithm should use pre-computation to compute and select path so that the computing overhead can be reduced. Other methods such as hop-by-hop routing and explicit routing can still be used provided that only one single algorithm is used within a single AS for better operation. Finally, whatever mechanism is to be used to advertise link state information must ensure that the link state update overheads due to frequent link state changes is minimal. Two important assumptions are also made [Apostolopoulos99iv]: 1) every QoS-enabled routers are capable of distributing the available resources to both QoS and best-effort flows appropriately by accurately identifying the available resources, and 2) the QoS flows themselves must be capable of defining their QoS requirements in some quantifiable manner.

[Apostolopoulos99iv] had defined three important building blocks of a QoS routing process. The first process involves the specification of QoS requirements in some measurable fashion called *link metrics*. Several useful link metrics are *link available bandwidth*, *link propagation delay* and *hop count*. Once the link metric to be used has been decided, it has to be advertised so that every router in the network can build the network topology database. The accuracy of the link state information is very important and it depends on the frequency of the advertisement. Two options are available and they are *periodic update*, and *trigger-based updates*. The periodic update mechanism has the disadvantage of advertising stale state information if the update interval is very long and introducing additional protocol overheads if the update interval is very short. Conversely, when trigger-based update mechanisms are used, updates are only issued when there is a significant change in the link state metrics.

The final part of the QoS routing process is the path selection. The topology used is the standard OSPF topology given in [Moy98]. The main objective here is to select a path that can accommodate a flow's QoS requirements without compromising the path cost. Hence, QOSPF uses the *widest-shortest path* algorithm to compute the "cheapest" path. If there are more than one such path exist, then the one with the maximum available bandwidth is selected. The aim is to balance the network load as well as to provide guaranteed resources. The QoS path selection algorithm computes paths either *on-demand*, that is, a path is computed whenever there is a new request, or *pre-computed*, that is, a path is computed in advance. Although both techniques are reliable, they are computationally expensive. For more details about the implementation methods and issues of QOSPF, readers are referred to [Apostolopoulos00iv, Apostolopoulos99v].

2.4 QoS Routing Process

As stated in the previous sub-section, the process of routing based on QoS requirements of a flow is made up of three major steps. The first is the metric on which the routing algorithm operates. The second is the mechanism used to propagate updates of this metrics. Finally the path selection algorithm itself.

2.4.1 Link Metric

A link metric is consists of information such as available bandwidth, delay, jitter, path cost, loss probability, reliability, etc. These QoS metrics follow some rules of metric composition as stated in [Wang96].

Let $m(n_1, n_2)$ be a metric for link (n_1, n_2). For any path P = $(n_1, n_2,\ldots,n_i, n_j)$, metric m is:

- ***additive***, if

  $m(P) = m(n_1, n_2) + m(n_2, n_3) + \ldots + m(n_i, n_j)$ (2.4)

  Examples are delay, jitter, path cost and hop-count. For instance, the delay of a path is the sum of the delay of every hop.

- ***multiplicative***, if

  $m(P) = m(n_1, n_2) * m(n_2, n_3) * \ldots * m(n_i, n_j)$ (2.5)

  Examples are reliability and loss probability.

- ***concave***, if

  $m(P) = \min\{m(n_1, n_2), m(n_2, n_3), \ldots, m(n_i, n_j)\}$ (2.6)

  Example is bandwidth, which means that the bandwidth of a path is determined by the link with the minimum available bandwidth.

Though the use of combined metrics is permitable in QoS path computation, using two additive or multiplicative, or one additive and one multiplicative metrics is proven to be NP-Complete [Younis03, Xiao02].

[Apostolopoulos99iv] describes the following metrics on which the path selection process is based:

- *Link available bandwidth*. Each link is assumed to be associated with a maximal bandwidth value. For supporting a particular flow with a certain amount of bandwidth requirement, the link must have minimal bandwidth value which is greater than the requested bandwidth value. Thus, the metric here is the amount of available bandwidth. When there is a change in this metric, it has to be advertised as part of extended LSA in order to provide accurate link state information for the path selection algorithm.

- *Link propagation delay*. The metric is usually used to select a path for a delay-sensitive request, which means pruning the links to identify high latency links. It is not suitable for real-time requests, therefore timely dissemination of this information is not necessary. The change in this metric value is also advertised as part of the extended LSA.
- *Hop count.* This metric represents the cost of the path that a flow takes to reach its destination. The lesser the number of hops on the path the cheaper the cost of the path will be. Since the metric is used implicitly as part of the path selection algorithm, no changes need to be distributed.

2.4.2 Link State Advertisement

The performance of a QoS routing algorithm is greatly influenced by the mechanisms used to trigger link state updates. Link state update policies can be classified as follows [Shaikh98]:

- ***Timer-based link state update policy***

  This is the type of triggering policy that was traditionally used in protocols such as Open Shorted Path First (OSPF). It is also known as *periodical link state update policy*. In this policy, link state updates are periodically generated and disseminated across the network at a fixed time interval. It is very important to define a proper minimal spacing between each consecutive update intervals. If the update interval is too frequent, the link state information will be more accurate but results in more protocol overhead. On the other hand, if the update interval is less frequent, the link state information will become more inaccurate but results in smaller protocol overhead. The spacing between the two consecutive update intervals can be controlled using a timer, i.e., *hold-down timer* or *clamp-down timer*.

- ***Trigger-based update policies***

  In this policy, updates are generated immediately when there is a significant change in the value of the link metric. The change can be measured as either *absolute* or *relative*. When the former is used, the link state metric value is partitioned into several classes of equal size. Update is triggered only when the current link state metric value changes significantly to cross the class boundary. In constrast, when the latter is in use, updates are triggered when the percentage of change of the metric value exceeds a certain pre-defined threshold. In either case, the metric values advertised could either be *actual* or *quantized* [Apostolopoulos99i]. Actual metric values are used when there is a need for frequent updates so that the accuracy can be maintained. In the case of less frequent updates, quantized values are used to increase robustness in the presence of inaccurate link state information.

  There are several ways of implementing the trigger-based update policy as stated in [Apostolopoulos99i]:

  - ***Threshold-based link state update policy***

    A link state update is triggered whenever the relative difference between the current and the previously advertised link state exceeds a pre-defined constant threshold value. For example, if b^o is the last advertised available bandwidth value of a link, b^c is the current available bandwidth value and *th* is the pre-defined threshold, an update is triggered when

$$\frac{|b^o - b^c|}{b^o} > th. \qquad (2.7)$$

The main advantage of this policy is that it allows more accurate link states to be maintained since updates are triggered immediately when there is a significant change rather than waiting for the next update period. To control the update intervals, hold-down timers can be used.

- ***Class-based link state update policy***

  This policy uses a concept similar to absolute threshold where a link's available bandwidth is partitioned into several classes and an update is triggered when the current link state value crosses a class boundary. The class-based link state update policy can be further subdivided into:

  1) ***Equal class-based updates***. The available bandwidth is partitioned into multiple equal sized classes using a constant value ***B***. For instance, (0, B), (B, 2B), (2B, 3B), …, etc. When the available bandwidth of an interface changes significantly until it crosses its own class boundary, an update will be triggered.
  2) ***Exponential class-based updates***. This policy is similar to *equal class based update policy* in that it also partitions the available bandwidth into classes. The difference is that the partitioned classes are of unequal sizes such as (0, B), (B, (f+1)B), ((f+1)B, (f^2 + f + 1)B), …, etc. To achieve this, two constants are used: B and f (f >1).

2.4.3 Path Selection

The final and most important function of a QoS routing process is to select the most feasible path that can satisfy the QoS requests of a flow. Every QoS-enabled router has a routing algorithm that selects best paths based on the link metrics that have been exchanged among the routers. There are different types of routing algorithms in use today, some of which are discussed briefly in [Ma97] as given below:

Shortest Path. A path with the minimum hop count among all paths is selected. The term "shortest" here does not necessarily mean the physical distance. It could also refer to a path with minimum monetary cost.

Widest-Shortest Path. Among all the feasible paths, the one with the least number of hops is selected. If more than one such path exists, then the one with maximum available bandwidth is selected. Again, if more than one widest-shortest path exists, then any one of them will be selected randomly by the algorithm.

Shortest-Widest Path. Among all the feasible paths, the one with the maximum available bandwidth is selected. If more than one such path exists, then the one with the least number of hops is selected. Again, if more than one shortest-widest path exists, then any one of them will be selected randomly by the algorithm.

Shortest Distance Path. Among all the feasible paths, the one with the shortest distance is selected where the distance function is defined by

$$\text{dist}(p) = \sum_{j=1}^{k} \frac{1}{Ri_j} \qquad (2.8)$$

where Ri_j is the bandwidth available on link i_j.

Dynamic Alternative Path. In this algorithm, the number of hops of a minimum hop path is given as *n*. A path is called dynamic alternative if it is a widest-shortest path containing no more than $n+1$ hops.

2.5 Issues of QoS Routing

Although QoS routing is conceptually simple, the distribution of routing information on the network and the path selection algorithms rise several issues in its development and deployment [Oliveira01, Bruin06]. Some of the issues related to QoS routing are discussed in detail in the following sub-sections.

2.5.1 Co-existence with BE traffic

Though significant amount of efforts have been directed towards the development and deployment of efficient QoS routing mechanisms, there is always a need for traffics with QoS requirements to co-exist with Best Effort (BE) traffics [Oliveira01]. This is due to the reason that most Internet traffics do not require specific QoS requirements and simple best effort service is more than adequate. As such, any QoS routing mechanism to be devised has to take into consideration the BE traffics by providing fair and efficient resource sharing so as not to starve them. Starvation of BE traffic could be caused by QoS traffics that consume all the resources leaving very little resource for BE traffics. Limiting the maximum reservable resources, usually bandwidth, for QoS traffics and allocating unused resources to BE traffics, can reduce this impact.

2.5.2 Additional Overheads

The additional overheads of QoS routing can be specified in terms of *computational overheads* [Apostolopoulos99v] and *protocol overheads* [Apostolopoulos99i].

- *Computational Overheads*. To perform QoS routing efficiently, routing information has to be updated frequently so that every router in the network has the same view about the network's global state. The use of more sophisticated link state update mechanisms and the higher frequency of link state updates have contributed towards this added computational overheads. Another reason for this is the complexity of the path selection algorithm, which is usually the direct impact of the increase in the number of constraints that need to be satisfied. This places higher processing demands on the processors. Though the additional computational overheads can be reduced by the use of latest technologies such as faster processors and bigger memories, there can never be a once-and-for-all solution to this problem.

- *Protocol Overheads*. The main reason for the increase in protocol overheads is the amount of protocol messages that penetrates the network every second. These messages come in many forms depending on the type of approach used. For instance, when link-state routing protocol is used, the overhead is caused by the excessive amount of link-state information exchanged among routers via flooding. When on-demand path computation is used, the amount of signaling messages used to request the immediate path computation increases not only the protocol overhead but also the computational overhead. In some other situations, when probe messages are used to collect the network link state information, protocol overheads can be increased due to the large number of probe messages issued, the size of the probe messages and their transmission frequency.

2.5.3 Routing Information Inaccuracy

In large networks, the deployment of QoS routing algorithms depends greatly on the accuracy of the link state information that are exchanged among the routers. Though frequent link state updates could provide more accurate link state information to routers, it can also cause problems in terms of higher resource consumption and increased protocols overheads [Bruin06]. This can be overcome by using periodic updates in which the interval between two consecutive updates is made larger. Unfortunately, this introduces another problem where the link state information distributed becomes stale and could no longer reflect the actual state of the network. This inaccuracy induces major impacts on the QoS routing performance since the path selection algorithms tend to make incorrect routing decisions based on this stale link state information.

Many factors contribute to the link state information inaccuracy which includes non-negligible propagation delay, link state update policies used, resource reservation and hierarchical topology aggregation [Oliveira01, Bruin06]. Hence, any QoS routing algorithm with performance concerns must incorporate mechanisms that can make efficient routing decisions even in the presence of inaccurate link state information.

2.5.4 QoS Routing Instability

QoS routing protocols, which are very sensitive to changes in network states, may produce unnecessary traffic re-routing causing instability of the protocol itself. One major contributor to this problem is the "routing oscillations" [Oliveira01, Ohara03]. Most QoS routing algorithms perform path selection based on the current traffic load on the network. For instance, to achieve greater throughput, a path selection algorithm chooses a path that is lightly loaded to avoid packet losses. However, when the traffic load on the path changes frequently, the algorithm tends to shift from one path to another in a very short period of time causing oscillations. This is especially very obvious during heavy traffic loads or when the traffic is bursty.

2.6 Related Study on Routing Information Inaccuracy

The original OSPF protocol has proposed periodic updates with minimal interval of 30 minutes to disseminate link state information. This large interval is inappropriate for use in QOSPF whose network state may change frequently within this interval due to the network's dynamic nature. Consequently, routing decisions are performed based on outdated link state information and this degrades the performance of the network significantly. To overcome the problem caused by large advertisement intervals, many research works have been conducted mostly concentrating on the use of trigger-based link state update policies.

As mentioned earlier in this chapter, the two main issues of QoS routing are 1) how to perform path selection in the presence of inaccurate link state information, and 2) when the link state information need to be disseminated in order to preserve its validity. A number of proposals have been made in literature which tolerates the inaccuracy of the link state information and still able to make accurate routing decisions. The most notable ones among them which will be discussed later in this section are *Safety-based Routing* [Apostolopoulos99ii], *Multipath Routing* [Chen98i], *Randomized Routing* [Wang00], *ALCFRA* [Kowalik02], *Cost-Adaptive OSPF* [Zhou03], *Distributed Cost-based Update Policies* [Chang02], *Moving Average Filtering* [Lekovic01] and *TE-QOSPF* [Lim04].

2.6.1 Safety-based Routing

Safety-based routing algorithms were first introduced by Apostoloupolous et al in [Apostolopoulos99ii], where each link is ranked based on its safety level in order to select the path that is most likely able to accommodate the requested bandwidth. The safety of a link is measured in terms of the probability that the link can support the requested amount of bandwidth. Hence, the most feasible path among those that have been ranked is the one with the largest safety.

To determine the safety of a link, three things are essential. They are the requested bandwidth value b_{req}, the last advertised bandwidth value b_{adv} , and the type of triggering policy used. The knowledge of the last advertised bandwidth value is necessary to determine the range of values that the actual bandwidth can take before the next update, while the values of the lower bound b_l and upper bound b_u of the bandwidth values can be used to estimate the probability that the link can accommodate b_{req}. b_l and b_u are calculated differently for different types of triggering policies. For example, if the threshold-based triggering policy is used, b_l is given as $b_{adv} * (1 - th)$ and b_u is given as $b_{adv} * (1 + th)$. On the other hand, if the class-based triggering policy is used and the class to which the last advertised bandwidth value belongs to is known, for example C_i, then b_l falls in between [C_{i-1}, C_i], and b_u falls in between [C_i, C_{i+1}].

Assuming that the triggering policy used is class-based and the available bandwidth value is uniformly distributed between b_l and b_u, then the safety of the link for a connection requiring b_{req} units of bandwidth is given as below:

- If $b_l \leq b_{req} \leq b_u$, the safety of the link is given as $(b_u - b_{req}) / (b_u - b_l)$.
- If $b_{req} \leq b_l$, the safety of the link is 1, which means that the link can definitely support the requested bandwidth.
- If $b_{req} \geq b_u$, the safety of the link is 0, which means that the link cannot support the requested bandwidth.

Once the safety of the links has been calculated, one of the following two routing algorithms can be used to perform the selection of the best route [Apostolopoulos99ii]: 1) *safest-shortest routing algorithm*, that selects the shortest path. In the case where more than one such path exists, then the one with the highest safety is selected, and 2) *shortest-safest routing algorithm*, that selects the safest path. In the case where more than one such path exists, then the shortest one is selected.

Using some experimental models, Apostolopoulos et al in [Apostolopoulos99ii] had verified that safety based routing can reduce the update traffic volume and improve the overall performance even when insensitive triggering policies are used. It further demonstrated that the use of uniformly distributed bandwidth values within the range does not cause significant performance loss.

2.6.2 Multipath Routing

The multipath routing scheme introduced by Chen & Nahrstedt [Chen98i] is a distributed routing scheme that works with imprecise link state information. It uses a technique known as ***Ticket-Based Probing (TBP)*** where probing messages consisting of tickets are sent over multiple paths in parallel to maximize the probability of finding a feasible path. Every probe message should consist of at least one ticket and hence the number of probing messages is bounded by the number of tickets in each probe. Since one probe message is capable of searching a single feasible path, the number of feasible paths searched is also bounded by the number of tickets.

Two routing problems, which are ***delay-constrained least-cost routing (DCLCR***) and ***bandwidth-constrained least-cost routing (BCLCR)*** were investigated in [Chen98i] where the former suffers from NP-Completeness problem. The DCLCR algorithm finds a feasible path from source *s* to destination *d* with delay equivalent to or less than the requested delay *D* (i.e., delay ≤ D). When more than one such path exists, then the one with the least-cost among them is selected. The BCLCR algorithm also works in the similar manner except that instead of using delay as the main constraint, it uses bandwidth such that the most feasible path consists of available bandwidth greater than or equals to the requested bandwidth *B* (i.e., bandwidth ≥ B). The main objective of the study conducted by Chen & Nahrstedt is to propose a heuristic algorithm to solve the NP-Complete problem exhibited by DCLCR. The proposed algorithm is capable of finding a delay-constrained path with lowest cost in the presence of imprecise link state information. The same heuristic can also be applied to BCLCR.

The TBP scheme works as follows: Two types of tickets are carried by a probe message and they are differentiated by their colours, for example, *yellow* and *green*. A yellow ticket is used to maximize the probability of finding the most feasible path that satisfies the delay constraint in DCLCR (or bandwidth constraint in BCLCR) and a green ticket is used to maximize the probability of finding the most feasible path that satisfies the cost constraint. The probe messages that are in transit utilize the link state information at the intermediate nodes to guide them through the best possible path. By doing this, the success probability can be maximized while the overhead can be minimized. When a probe message with yellow ticket arrives at a destination node with accumulated delay less than D, that path is identified as a potential feasible path. If another probe message with green ticket arrives at the same node with the lowest path-cost among all the potential feasible paths, then that path can be considered as the most feasible delay-constrained least-cost path.

The algorithm proposed in Chen & Nahrstedt is closely related to the one proposed by Guerin & Orda [Guerin97ii] and Lorenz & Orda [Lorenz98]. In addition to the four state variables defined in the abovementioned papers, which are *connectivity* $R_i(t)$, *delay* $D_i(t)$, *bandwidth* $B_i(t)$ and *cost* $C_i(t)$, Chen & Nahrstedt have defined two more state variables: 1) *delay variation* $\Delta D_i(t)$, that stores the estimated maximum change of $D_i(t)$ before the next update where $D_i(t)$ is the minimum end-to-end delay from i to t, and 2) *bandwidth variation* $\Delta B_i(t)$, that stores the estimated maximum change of $B_i(t)$ before the next update where $B_i(t)$ is the maximum end-to-end bandwidth from i to t.

The algorithm calculates the actual minimum end-to-end delay (or maximum end-to-end bandwidth) for the next update period based on the current historical information accumulated by the probe messages. $\Delta D_i(t)$ (or $\Delta B_i(t)$) and $D_i(t)$ (or $B_i(t)$) are updated periodically and the actual delay (or bandwidth) values are calculated as follows:

$$\Delta D_i^{new}(t) = \alpha \times \Delta D_i^{old}(t) + (1-\alpha) \times \beta \times |\Delta D_i^{new}(t) - \Delta D_i^{old}(t)| \qquad (2.9)$$

where $\Delta D_i^{old}(t)$ and $\Delta D_i^{new}(t)$ are the values of $\Delta D_i(t)$ before and after the update, respectively, and $D_i^{old}(t)$ and $D_i^{new}(t)$ are the values of $D_i(t)$ before and after the update, respectively.

How fast the history information ($\Delta D_i^{old}(t)$) is forgotten and how fast $\Delta D_i^{new}(t)$ converges to $| D_i^{new}(t) - D_i^{old}(t)|$ is determined by the factors α (< 1) and (1 - α) respectively. The actual bandwidth values can also be calculated using the above method. Using the same equation as above (Equation 2.9), the actual bandwidth value can be calculated by substituting $\Delta D_i^{old}(t)$ with $\Delta B_i^{old}(t)$ and $\Delta D_i^{new}(t)$ with $\Delta B_i^{new}(t)$.

The problem with TBR scheme is that tickets of different colors may interfere with each other and redundant searching paths may exist. The most significant problem is that the probe distribution is not optimized. For example, in order to eliminate infinite probing cycles, if a probe message has passed link *(i, j)*, another probe cannot pass the same link again. This may block some useful probes from finding better paths. A solution to this problem has been proposed by Xiao et al in [Xiao02]. The idea is to distribute tickets based on their colours. For example, if a probe message containing one yellow ticket and two green tickets try to pass a link that has already been passed by another probe containing two yellow tickets, instead of blocking the entire probe message, only the yellow ticket will be blocked while the two green tickets are still allowed to pass the link. Through extensive experimentations, Xiao et al proved that this method allows a probe message to be more optimal without increasing the message overhead.

2.6.3 Randomized Routing

A QoS routing algorithm can either be *deterministic* or *randomized*. In deterministic approach, the algorithm always selects the same path as the “best path” before the next update is triggered. This usually happens when there is a frequent network state change before the next update period and the link state is not updated accordingly. Consequently, the path becomes overwhelmed with more traffic than the path can handle leading to heavy congestion. Examples of algorithms that exhibit such behaviour are the safest-shortest path routing and shortest-safest path routing algorithms. On the other hand, the randomized approach is capable of selecting paths randomly by associating each path with some probability value. The approach was adopted by Wang et al in [Wang00] where on-demand path selection and computation method is used to minimize the computational costs and protocol overheads.

Basically, every QoS request consists of a source node u_0, a destination node v_0, the requested bandwidth value b, the “safety rate requirement” S_0 that indicates the probability that b can be satisfied on that path, and the delay requirement N_0 specified in terms of the maximum number of hops allowed in the routing path. A network in randomized routing is represented by a graph $G = (V, E)$ where $V = \{v_1, ..., v_n\}$ is the set of hops in G and $E = \{e_1, ..., e_n\}$ is the set of links in G. A probability function f_p is associated with every link such that the probability that link e_j has available bandwidth of at least b is $f_j(b)$. The calculation of the link's safety rate is similar to the one demonstrated by safety-based routing in section 2.6.1.

A routing path P that satisfies the given QoS request $R = (u_0, v_0, b, N_0, S_0)$ has the following properties [Wang00]:

1) The probability that the bandwidth of the path P is at least b is not smaller than the safety rate requirement S_0.

$$\Pi_{ej \in P} f_i(b) \geq S_0. \tag{2.10}$$

2) The number $|P|$ of hops in the path P is not larger than the delay requirement N_0.

$$|P| \leq N_0 \tag{2.11}$$

Given the network G and QoS request R, the algorithm works as follows: For every node in G, the algorithm records the maximum safety rate and minimum delay from that node to v_0. Then, all nodes that have maximum safety rate of less than S_0 or minimum delay of greater than N_0 are eliminated from the list. The process is repeated until there are no more nodes to be removed. Using the remaining list, the algorithm repeatedly looks for a single routing path that satisfies properties (1) and (2). The identified path will be the most feasible path that tolerates the link state inaccuracy. The algorithm is proven to be effective in achieving good balance of network resource distribution.

2.6.4 ALCFRA

ALCFRA stands for *Adaptive Link Cost Function Routing Algorithm* and was proposed by Kowalik & Collier in [Kowalik02]. The algorithm operates online and to achieve better performance in the presence of inaccurate link state information, ALCFRA uses modified exponential link cost function. The basic idea is to advertise the cost of the link as a function of the bandwidth utilization on that link instead of advertising the bandwidth utilization value itself.

The exponential link cost function used by ALCFRA has the property that when there is a small change in the link load, a lowly utilized link produces only a small change in the link cost while a highly utilized link produces a very huge change. Thus, the most appropriate link cost update policy that can be used to capture these situations is the *increasing density utilization change*, described by Kowalik & Collier. The policy blocks any request that it sees as having the potential of introducing even a slight change in a highly loaded link while assuming that the situation rarely happens for a lightly loaded links. The main objective of ALCFRA is to reduce the effect of route fluctuations/oscillations (i.e., magnet phenomenon) that occurs for both the timer-based update policies with long update intervals and utilization change based policies with long hold-down timers.

A request in ALCFRA is of the form $\beta_i = (s_i, d_i, r_i)$ where s_i is the source node, d_i is the destination node and r_i is the bandwidth requirement. If a least-cost path P that can satisfy r_i exists, the path is marked as $P_{\mathrm{i}}^{\mathrm{A}}$ for the duration of the i^{th} connection or else it is rejected. How A is calculated is not given in this thesis but can be found in [Gawlick95]. To tolerate the link state inaccuracy, the link cost function has been modified so that a small change in a highly loaded link produces huge link cost. To achieve this, every link is associated with a parameter a_e which can take a maximum value of A and can increase or decrease by some constant value Δ. A utilization threshold u^{tr} is pre-defined so that when the long term utilization is under u^{tr}, a_e takes a positive value and when it is greater than u^{tr}, a_e takes a negative value. Whether a_e takes positive value or negative value is determined by the following rule:

$$a_e = \begin{cases} a_e + \Delta, & \text{when: } a_e \leq A\,[\,u^{tr} - u_c(e)]\,/\,u^{tr} \\ a_e - \Delta, & \text{otherwise} \end{cases} \tag{2.12}$$

where $u_c(e)$ is the utilization of link e defined by

$$u_c(e) = \sum_{i,e \in P_i^A} \frac{r_i}{c(e)} \tag{2.13}$$

The value of a_e changes according to the above rule after every unit of time, say 1 second. Using the value of a_e, ALCFRA defines its link cost function as:

$$cost\,(e) = \begin{cases} \dfrac{a_e^{\,u_c(e)} - 1}{a_e - 1} & \text{if } a_e > 1 \\ u_c(e) & \text{if } -1 \leq a_e \leq 1 \\ \log_{|a_e|}((|a_e| - 1)u_c(e) + 1) & \text{if } a_e < -1 \end{cases} \tag{2.14}$$

Thus, a_e makes the link cost function more convex under low link load and concave under heavy link load. Once the cost of a link has been computed, ALCFRA generates and advertises the new cost value using the *equal density cost change mechanism*. The experiments conducted by Kowalik & Collier found that even when the link state update is not frequent, ALCFRA works well and it allows QoS routing to be introduced in Internet at a reasonable cost.

2.6.5 Cost-Adaptive OSPF (CA-OSPF)

In the basic OSPF, the cost of the interfaces belonging to a router is fixed during the router installation phase itself and do not change thereafter. The routing algorithm in OSPF always selects the shortest path (i.e., the least-cost path) without regards to the bandwidth availability of that path. Also, the long periodic update interval (which is 30 minutes), gives incorrect information of the global network state. This causes the path to be congested with more and more traffic that eventually leads to dramatic performance degradation over time. One solution to this problem is to select other better paths with low utilization ratio even though the interface costs could be higher.

An improvement to the basic OSPF has been suggested by Zhou et al in [Zhou03] known as Cost-Adaptive OSPF (CA-OSPF), where the costs of the interfaces can be changed dynamically according to the link's bandwidth utilization ratio. It is claimed that, by doing this the resource utilization ratio can be improved despite the presence of inaccurate link state information in a dynamic OSPF environment. According to CA-OSPF, the bandwidth utilization ratio U $(0 \leq U \leq 100\%)$ of an interface is bounded by an upper bound threshold U_a and a lower bound U_b. The state of an interface is classified as either ***over-used state*** $(U_a \leq U \leq 100\%)$, ***middle state*** $(U_b \leq U \leq U_a)$ and ***under-used state*** $(0 \leq U \leq U_b)$. CA-OSPF allows both non-CA-OSPF routers that fixes the interfaces costs at the initialization phase and CA-OSPF routers that is capable of adjusting the interface costs based on changes in bandwidth utilization ratio to co-exist.

The parameters required by CA-OSPF routers to modify the link costs are the initial cost of the current interface C_0, the current cost C of the interface, minimum allowable increment or decrement of the cost Δ, and the maximum allowable total interface cost increment Δ_{max}. Given all these parameters and link state classifications, the interface's cost is adjusted at regular intervals as follows:

1) When the interface's state falls into the *over-used state*, the cost of the interface is changed only when the bandwidth utilization ratio of that interface exceeds U_a thrice successively and the total increment do not exceed Δ_{max}. Once the cost has been changed, the CA-OSPF router generates and distributes the link state information to other routers.
2) When the interface's state falls into the *middle state*, no changes are done to the interface's cost.
3) When the interface's state falls into the *under-used state*, nothing is done to change the cost of the interface if its current cost C is equal to its initial cost C_0. If C is greater than C_0, then the cost will be decreased to not less than C_0.

The Δ_{max} parameter is used to reduce the link state update storm resulting from unbounded interface cost increment during heavy network congestion. To minimize response delay, U_a is usually set to 95% of the network utilization instead of 100%. Using CA-OSPF, the overall performance of the network can be improved only when the network load is light. Under heavy load, CA-OSPF failed to show any improvement. Though CA-OSPF is adaptive to local change and is better than OSPF, it can improve performance only to certain a extent. Furthermore, the increment and decrement of interface costs at a regular interval introduces higher protocol overheads.

2.6.6 Dynamic Cost-based Update Policies

Most mechanisms that were proposed in literature to overcome the effects of stale link state information and excessive updates concentrate on improvements to the instantaneous link state information advertisement. Also, most of them consider only the use of hold-down timers to achieve the above improvements. In [Chang02], Chang & Hwang have proposed a new link state update policy called *dynamic cost-based update policy (DCU)* which aims at reducing the inaccuracy of aggregated link state information while at the same time reducing the frequency of aggregation and information distribution. DCU is mainly intended for use in hierarchical networks to reduce the effect of inaccurate aggregated link state information.

In DCU the link state information are aggregated as in hierarchical networks and distributed based on the COL link cost function defined in [Gawlick95]. The COL link cost function sets the cost of each link in the network as an exponential function of the residual bandwidth of the link. That is, when the bandwidth utilization of link ℓ is i, then the cost of the link can be expressed as;

$$W_l(i)=\mu^{i/C_\ell}, \tag{2.15}$$

where μ denotes the chosen constant parameter, and C_ℓ represents the capacity of link. ℓ. The cost of the path is then computed by summing up the cost of all the links in the path.

The DCU policy basically divides the link utilization into 3 regions based on how fast the link cost function changes: *slow*, *moderate* and *high*. This is shown in Figure 2.1.

The boundaries of the regions are determined based on the equations given in [Chang02]. Once the boundaries have been set, the individual regions are further sub-divided into several link states. The accuracy of the aggregated link state information becomes higher if there are more link states but this has to be traded-off with the increased update overheads. The number of link state updates generated depends on in which region the link state resides. The link state update becomes more frequent when the link state is in the fast region because of the higher cost variation. Conversely, when the link state is in the slow region, the cost variation will be smaller and hence reduces the frequency of updates.

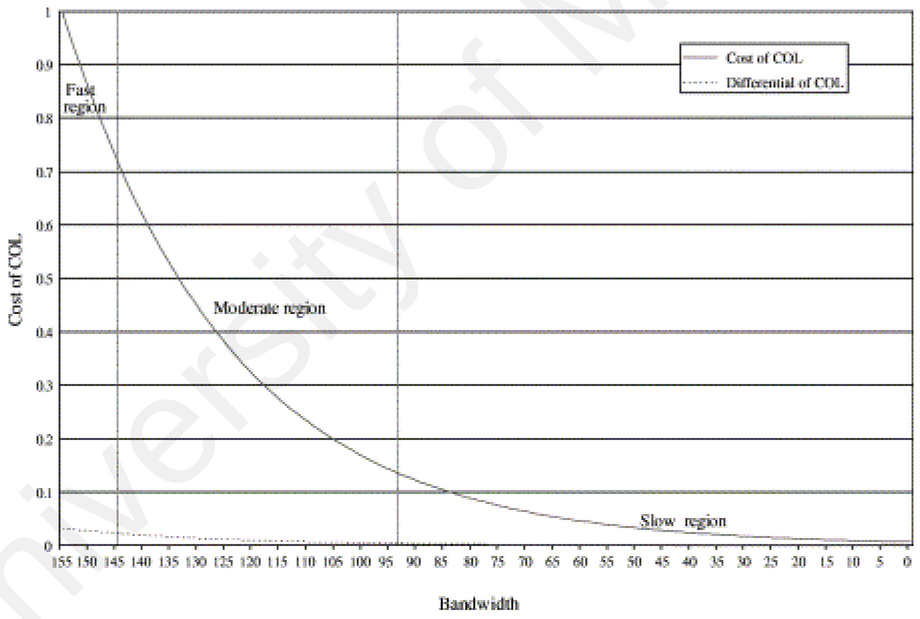


Figure 2.1 The slow, moderate and fast regions of COL function

DCU in its basic form suffers from the problem of routing oscillations due to smaller class boundaries and dynamic flow arrivals. To confront this problem, Chang & Hwang have extended the DCU policy by proposing three more update policies referred to as the *DCU update with hysteresis policy (DCUH)*, the *DCU update with* *cost threshold policy (DCUCT)*, and the *DCU update with cost threshold and hold-down timer policy (DCUCTT)*. A brief discussion of how these policies operate comes next and readers are referred to [Chang02] for more information.

DCU update with hysteresis policy (DCUH). This policy applies guard bands between the boundaries of two link states so that a sudden spike in the cost change will not trigger an unnecessary update which is the main cause of routing oscillations. The idea is to allow the link state to change from state *k-1* to *k* only when the cost of the link exceeds C^H_k. Again, the link state is allowed to change from state *k* back to state *k-1* only when the cost of the link becomes less than C^L_k. This is illustrated in Figure 2.2.

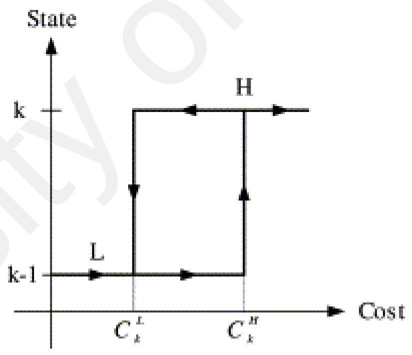


Figure 2.2 The characteristics of DCUH

The technique is applied only in moderate region because the cost intervals of low and fast region are very small.

DCU update with cost threshold policy (DCUCT). The main objective of DCUCT is to further reduce the update frequency by deciding whether or not the QoS parameters should be aggregated and distributed [Chang02]. The policy is based on the dynamic cost update policy with aggregated cost threshold where the aggregated cost threshold is defined as:

$$\frac{\Psi_{new} - \Psi_{previous}}{\Psi_{previous}} \leq \beta, \ \beta \geq 0 \qquad (2.16)$$

where Ψ_{new} is the new aggregated cost, $\Psi_{previous}$ is the last advertised aggregated cost and β is the cost threshold parameter. If the cost ratio exceeds β, then the QoS parameters will be aggregated and advertised; otherwise nothing is done. The choice of β value is an important factor that determines the success of this technique. A larger β value could result in larger fractional reward loss due to inaccurate link state information while a lower value causes more update overheads without any improvement to fractional reward loss. Through the use of various simulation topologies and traffic loads, Chang & Hwang have proposed a value that ranges from 0.15 to 0.45.

DCU update with cost threshold and hold down timer policy (DCUCTT). The primary goal of DCUCTT is to minimize update triggering actions in DCUH and DCUCT with the use of hold down timers. Basically, the procedure works as follows [Chang02]: First the cost threshold β and the hold down timer are initialized with some initial value. Then the hold down timer is started. When the hold down timer expires, the process of deciding whether or not to aggregate and advertise the QoS parameters as given in DCUCT is performed. If there is a need to aggregate and advertise the QoS parameters, then the parameters are advertised and the hold-down timer will be restarted; otherwise nothing is done. On the other hand, if the hold down timer has not expired, then none of the above is performed.

Chang & Hwang compared the performance of all four of the proposed policies with the PNNI time-based update policy (PNNIU), the full update policy (FU), and the logarithm of residual bandwidth update policy (LRBU). It is proven that all four proposed policies perform better in terms of lesser revenue loss and aggregation overhead. The DCUCTT has been identified as the most successful policy as it is able to reduce the aggregation and advertisement overheads significantly.

2.6.7 Moving Average Filtering

In [Lekovic01], Lekovic & Miegham has considered the use of threshold-based and exponential class-based triggering policies instead of using the conventional timer-based triggering policies. The objective is to provide consistent link state information to the underlying QOS routing algorithms. They have proposed a new technique called "*moving average filtering*" that proved to be effective in limiting the number of link state updates generated. Instead of triggering updates for every instant of link state change, the technique first monitors the trend of the link state and based on this information decides when to trigger the update.

The main motivation of this technique is to overcome the problems caused by inversatile hold down timers. It is very difficult to set an appropriate hold down interval when the flow arrival rate is unpredictable. This condition is further complicated by the sudden utilization spikes after the hold down time causing unnecessary link state updates. Thus, the *moving average filtering* technique suggests the monitoring of bandwidth utilization trends before deciding when to trigger the updates. The idea behind this is that the smoothed bandwidth utilization curve carries only valuable information on the utilization trend rather than carrying unnecessary information which could cause unnecessary updates.

The technique requires the routing algorithm to obtain a set of successive bandwidth utilization values of the link over some period of time. The size of the averaged set is called the sample size *N*. After N number of samples has been taken, the mean of the sampled data is computed. This mean value is compared to the mean value of the previous set of samples. The difference is then compared to either the absolute or relative thresholds to decide when to trigger the link state updates. Since the technique is adaptive in nature, a high flow arrival rate causes the next moving average point to occur earlier thus generating frequent updates. On the other hand, if the flow arrival rate is low, the moving average point occurs much later causing less frequent updates. Hence the hold down timer is no longer required. Lekovic & Miegham have proven that the moving average filtering technique outperforms the hold down timer concept in terms of lesser call blocking rates and, reduced link state update errors and overheads.

2.6.8 TE-QOSPF

Most routing algorithms in use today selects shortest paths (minimum hop paths) as best paths when performing routings. One main disadvantage of such algorithms is the inefficient utilization of network resources and poor load balancing. For example, the *widest-shortest path* routing algorithm always selects the shortest paths among all the available paths without giving any consideration to the path's resource sufficiency. Selecting the same shortest path over and over again eventually leads to congestions resulting in greater propagation and queuing delays, and higher packet drop probabilities. These effects are undersirable since the Internet traffic is growing quickly demanding more and more improvements in the services provided. Therefore, Lim, S.H., in [Lim04] had proposed an algorithm called TE-QOSPF (***T***raffic ***E***ngineering enhancements to ***Q***oS-***O***SPF) that selects non-shortest paths instead of the usual shortest paths.

The algorithm works as follows: If there are two candidate paths, say P1 and P2, and P2 is one hop longer than P1, then P1 will be selected if its available bandwidth value is two times greater than that of P1. Similarly, if P2 is two hops longer than P1, it will only be selected if the available bandwidth value is three times greater than that of P1. To ensure that very long paths are not selected, a hop-count difference threshold, ***c***, is used. Again, if P1 and P2 are candidate paths, the shorter one will always be selected if their hop-count difference is greater than ***c***. The algorithm is summarized below:

Given souce ***S***, desination ***D***, two candidate paths ***P1*** and ***P2*** with a distance of ***d1*** and ***d2*** respectively to reach *D* from *S*, and the available bandwidth on path *P1* and *P2* are ***bw1*** and ***bw2*** respectively, the selection of non-shortest path abides the following rules [Lim04]:

1) If $|d1-d2| \leq c$, then

$$r = bw1 / bw2 \tag{2.17}$$

   (a) If $d1 > d2$ and $r > \boldsymbol{k}$, then P1 will be selected.

   (b) If $d2 > d1$ and $1/r > \boldsymbol{k}$, then P2 will be selected.

2) If $d1=d2$, then

   (a) If $r \geq 1$, P1 will be selected.

   (b) If $r < 1$, P2 will be selected.

3) If $|d1-d2| > c$, then the normal shortest path rule is applied.

The choice of ***k*** value is very important as it effects the choice of a better longer path. If it is too small, then a slightly longer path with a very little available bandwith may be chosen. On the contrary, if it is too large, then a longer path with larger available bandwidth may have very little chance to be chosen. Hence, Lim, S.H., had proposed the following value for k in [Lim04]: $k = |d1-d2| + 1$. However, the value greatly depends on many factors such as network load pattern, network topology and duration of the existing LSPs.

In terms of link state updates, TE-QOSPF uses a mechanism similar to QoS-OSPF that sends an update message for every instantaneous change of the available bandwidth. In other words, the percentage of available bandwith on a link is sampled at every pre-defined interval and this value is compared to a pre-defined threshold value. If it exceeds the threshold, then an update message is immediately sent out. Otherwise no updates will be triggered. The problem with this mechanism is that it fails to capture false positives which eventually lead to increased update message overheads.

Through extensive simulations, Lim, S.H., had verified that TE-QOSPF had been successful in reducing the packet loss ratio and packet delay, while at the same time improves the link utilization. As per the QoS-OSPF mechanism, even though the update message overhead increases as the network load increases, the mechanism manages to achieve almost 100% load balancing and higher throughput.

2.7 Concept of Linear Regression

The concept of linear regression was first introduced by Sir Francis Galton who conducted a study to compare the heights of sons to the height of their fathers . Today, linear regression is used to show the relationship between two variables in terms of mathematical equations [Motulsky03]. Observed data samples are plotted on a *scatter diagram* and a straight line that best fits the sampled data are drawn to represent the relationship between the two variables. The main objective of establishing a linear relationship between two variables is to show the correlation between the variables and to predict the value of one variable based on the value of another. The value of a variable can only be predicted provided that the value of another is known in advance. The variable whose value is known in advance is called ***independent variable*** and is denoted by **X**, while the variable whose value has to be predicted is called ***dependent variable*** and is denoted by **Y**. The stronger the correlation between X and Y, the more accurate the predicted value of Y will be.

For instance, given a set of data points, (x_1, y_1), (x_2, y_2),…,(x_i, y_i), a straight line that "best fits" these data points is drawn on the scatter diagram (see Figure 2.3). A *best fit* line has all the data points scattered very close around it, hence giving greater correlation among the data points. Looking at the scatter diagram in Figure 2.3 at one glance gives the impression that many straight lines can be drawn for the data points, but through careful analysis, one that best fits the data points has to be selected. One way to do so is by using ***least-square methods*** as described in the next section.

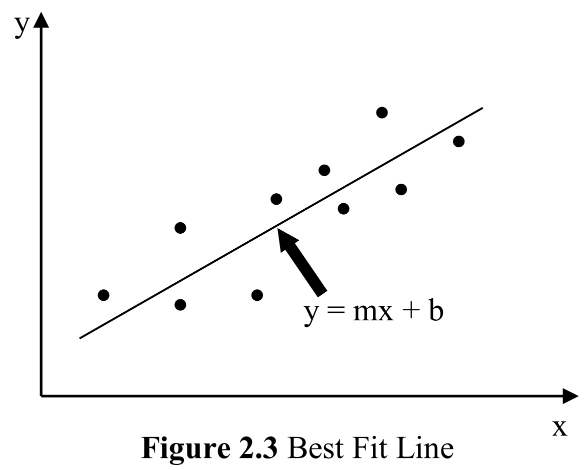


Figure 2.3 Best Fit Line

2.7.1 Least Square Methods

The least-square method attempts to minimize the sum of the squares of deviations of the actual data points from the calculated data points (see Figure 2.4) [Waner02, Motulsky03]. The smaller the sum, the more accurate the predicted value of Y will be. The least square line that best fits a set of data point, say (x_1, y_1), (x_2, y_2),…,(x_i, y_i), has the form:

$$y = mx + b; \tag{2.18}$$

where

y = the y-axis variable

x = the x-axis variable

b = the intercept (i.e., the value of y when x = 0)

m = the slope of the line (also known as the ***tangent*** of the line)

The formulated equation is called a ***linear regression equation*** and the line is called the ***linear regression line*** of *y* on *x*. The ***regression coefficients***, ***b*** and ***m*** are constants whose values can be derived from the following two equations:

$$\text{Intercept} = b = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2} \qquad (2.19)$$

$$\text{Slope (tangent)} = m = \frac{(\Sigma y) - m(\Sigma x)}{n} \qquad (2.20)$$

If the estimated value of *m* is known, then it is possible to estimate the ***change in y per unit change in x***. The following graph shows the linear regression line with a deviation.

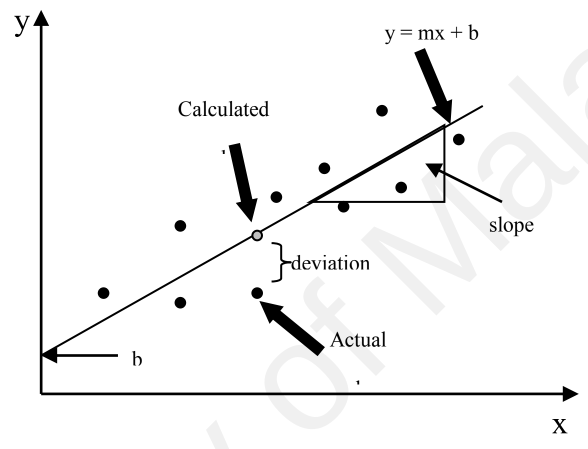


Figure 2.4 Linear Regression Line with Deviation

2.7.1.1 Why square the sum of the errors?

One may ask why not simply use the sum of the actual deviations instead of summing the squares of the deviations. This can be explained using two sets of data, where each data set has two (x, y) points [Motulsky03]. In the first set each of the points deviates 3 units away from the straight line while in the second data set, one point deviates 1 unit away and the other 5 units away from the straight line. If the absolute sum of the deviations are taken, then the absolute sum of the deviations for the first data set will be 6 (i.e., 3+3) and for the second data set will also be 6 (i.e., 1 + 5). If the random scatter follows Gaussian distribution, then it is far more likely that the line given by the second data set will be chosen. This is not preferable because the data of the second set is more scattered then the first data set and the line produced by this data set will not accurately represent the relationship between X and Y.

To solve this problem, the least-square method sums the squares of the deviations so that the sum-of-squares of the first data set will be 18 (i.e., $3^2 + 3^2$) and the sum-of-squares of the second data set will be 26 (i.e., $1^2 + 5^2$). Now it is evident that the first data set is much more accurate then the second and thus the first data set can produce a more accurate best fit line for the given data.

2.8 Chapter Summary

In this chapter, the concept of QoS routing and mechanisms to provide QoS-based services to Internet users was studied and reviewed in detail. Various solutions proposed in literatures to overcome the problem of excessive link state update message overheads was studied thoroughly as this is essential in the development of the proposed mechanism. This is followed by an in-depth discussion of the main mechanism applied in the proposed mechanism called linear regression. The discussion presented in this chapter had given useful ideas for the design and construction of the proposed mechanism which is discussed in the next chapter.

CHAPTER 3

PROPOSED LINK STATE UPDATE MECHANISM

Effective QoS routing algorithms rely heavily on the accuracy of the link state information. Accurate link state information can be obtained only if the link state update is performed frequently enough to capture every instant of the link state change. However, this imposes significant burden on the network resources. On the other hand, allowing longer update intervals based on hold-down timers do not allow for the link state upate policy to be adaptive with respect to the changes in the link state. In this thesis, a new link state update mechanism has been proposed which aims at providing consistent link state information to the QoS routing algorithm, at the same time limiting the number of excessive link state update messages. The mechanism is called **TE-LR** (***T***raffic ***E***ngineering using ***L***inear ***R***egression).

3.1 Assumptions

The following are the assumptions made for the proposed link state update mechanism:

1. Bandwidth is assumed to be the primary QoS metric.
2. The routing protocol used is link state which gives a complete view of the overall network topology.
3. The bandwidth utilization ratio of a link is sampled randomly.

3.2 Link State Update Mechanism using Linear Regression

The basic idea of the proposed link state update mechanism is to observe the link bandwidth utilization trend over some period of time. Based on the observed trend, a decision on whether or not a link state update message should be sent out will be made. The mechanism allows an update message to be sent out only when the observed trend shows a significant utilization change rather than sending an update for every instant of change as in other existing trigger-based update mechanisms, such as QoS-OSPF. In QoS-OSPF, link state updates are sent whenever the bandwidth utilization ratios exceeds a pre-defined threshold value, ***thr***. This produces excessive overhead messages as updates may be sent due to unnecessary bandwidth utilization fluctuations. Hence, the proposed mechanism's main goal is to suppress the unwanted update message overheads while at the same time maintaining the link state information accuracy. The proposed mechanism is further illustrated using a flowchart in Figure 3.1.

In general, the mecahnism is made up of three main functions: 1) the sampling of bandwidth utilization ratios of every link, 2) the construction of linear regression line equations, and 3) the advertisement of link state updates based on the tangent difference. A detailed description of the mechanism follows in the next sub-sections.

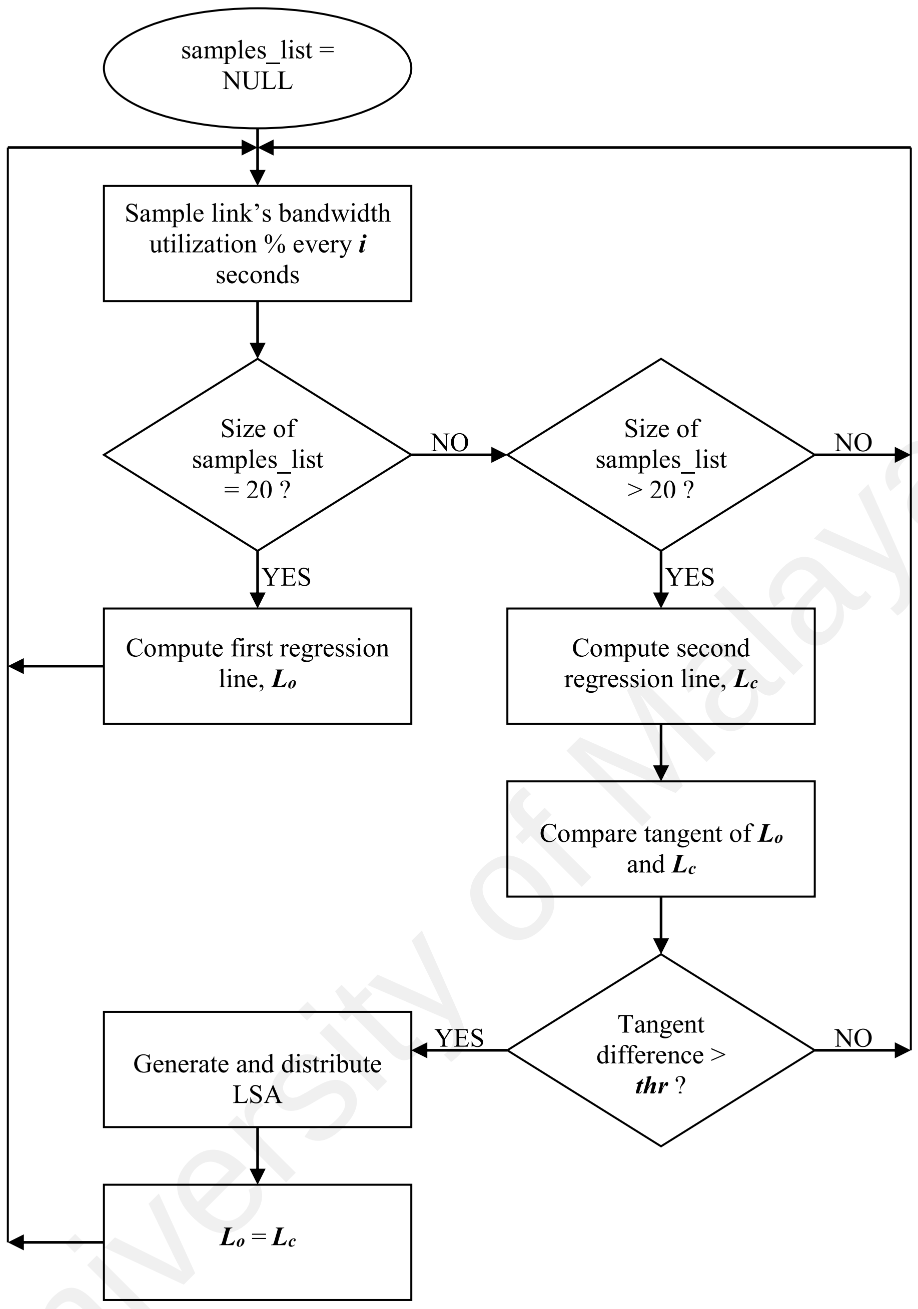

samples_list = NULL
Sample link's bandwidth utilization % every *i* seconds
Size of samples_list = 20 ?
NO
Size of samples_list > 20 ?
NO
YES
YES
Compute first regression line, L_o
Compute second regression line, L_c
Compare tangent of L_o and L_c
Tangent difference > *thr* ?
YES
NO
Generate and distribute LSA
$L_o = L_c$

Figure 3.1 A flowchart that illustrates the proposed TE-LR mechanism

3.2.1 Sampling of bandwidth utilization ratios

A network is modeled as a set of ***V*** nodes (i.e., routers) that are interconnected by a set ***E*** of full-duplex, directed communication links. Each node keeps an up-to-date local state about all the outgoing links. The state information of link ***(i, j)*** is consist of the bandwidth utilization ratio of the link, i.e., how many percentage of the link's original capacity is in use. At every sampling interval, ***i*** seconds, the bandwidth utilization ratio of link ***(i, j)*** is continuously sampled. The mechanism maintains two lists each with 20 elements. One list is used to store the sampled bandwidth utilization ratios, while the other is to store the time at which each sample was taken as this is required during the linear regression line construction process. Table 3.1 shows an example of the sampled bandwidth utilization ratio and time values.

Table 3.1 Example of sampled data values

Sample Time (secs)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Bw Utilization (%)	52.79	54.27	43.25	50.03	50.67	49.18	49.61	52.36	48.76	55.54

3.2.2 Construction of linear regression line equations

For every *N* values sampled (i.e., both bandwidth utilization ratio and time), the intercept and tangent associated with the ***n* = *N*** samples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ are computed using equations (2.19) and (2.20). The tangent, ***m*** and intercept, ***b*** values are then applied into equation (2.18) to get the linear regression line equation. Lets assume that the first-most linear regression line computed for the first set of *N* samples is called **original line**, $\boldsymbol{L_o}$ and each subsequent line is called **current line**, $\boldsymbol{L_c}$. The tangent of line $\boldsymbol{L_o}$ and $\boldsymbol{L_c}$, which are $\boldsymbol{m_o}$ and $\boldsymbol{m_c}$, are obtained from the respective line equations. Figure 3.2 shows an example of the linear regression lines which have been computed and plotted on the scatter diagram. The linear regression line equations and the tangent values for each of the lines are also shown on the diagram.

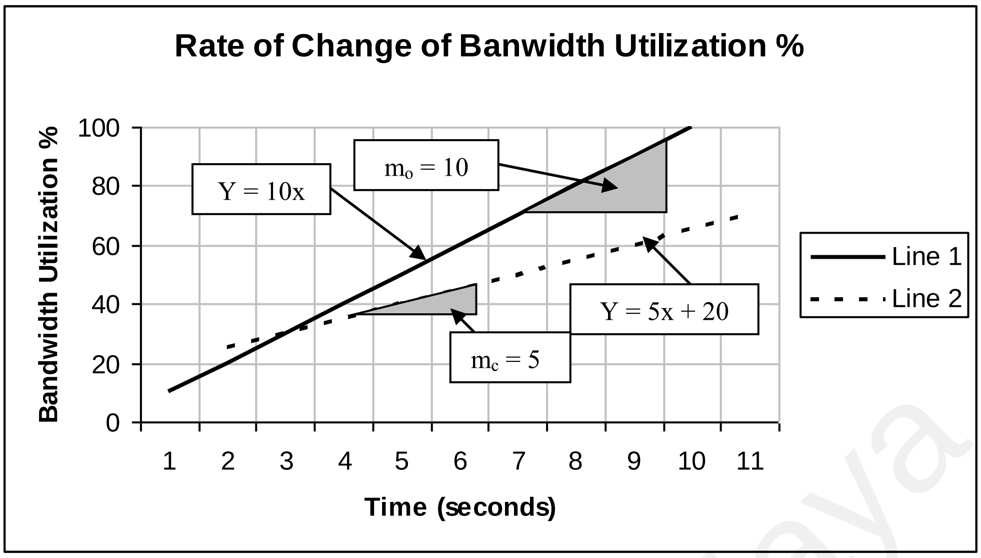


Figure 3.2 Linear Regression Line plotted on a Scatter Diagram

Let *N = 10.* As can be seen on the diagram above, the linear regression line equation for line 1, L_o is Y = 10x where m_o is 10, while the linear regression line equation for line 2, L_c is Y = 5x + 20 where m_c is 5. The process of linear regression line equation computation will be repeated for every successive sets of N samples obtained throughout the entire simulation.

3.2.3 Advertisement of link state updates

The computed tangent of L_o, which is $\boldsymbol{m_o}$ is compared with the tangent of the L_c, which is $\boldsymbol{m_c}$ using the following equation.

$$\textbf{\textit{tangent difference}} = \textbf{\textit{absolute}}[\, m_c - m_o\,] \qquad (3.1)$$

If the computed absolute tangent difference exceeds the pre-defined threshold ***thr***, that is $\boldsymbol{m_c - m_o > thr}$, the router will immediately send a link state update message to all its neighboring routers. Upon doing this, the current line is set to act as the original line against which the next linear regression lines, i.e., the next current lines, will be compared to. On the other hand, if the computed absolute tangent difference do not exceed ***thr***, that is $\boldsymbol{m_c - m_o < thr}$, then no update will be sent out. In this case, the original line remains as the original line against which the next linear regression lines will be compared to until the next update trigger is issued.

For example, in Figure 3.1, $\boldsymbol{m_o}$ and $\boldsymbol{m_c}$ are given as 10 and 5 respectively. Assume the value of ***thr*** is 4, then ***(10 - 5) > 4***, hence an update will be triggered. L_c will become the new L_o. If ***thr*** is 6, then ***(10 - 5) < 6***, so no update will be triggered and the current L_o remains as L_o for subsequent comparisons.

3.4 Chapter Summary

The idea behind the proposed mechanism TE-LR was discussed in detail in this chapter. Important elements of the mechanism such as the process of bandwidth utilization ratio sampling, the costruction of linear regression line equations and the process of deciding when to send link state updates were covered in the discussion. In the next chapter, the changes done to UMJaNetSim in order to implement TE-LR and the testings performed to evaluate TE-LR are presented.

CHAPTER 4

IMPLEMENTATION AND TESTING

This chapter begins with a discussion about the methodology used in implementing the proposed TE-LR mechanism. Since simulation had been selected as the tool to evaluate the performance of TE-LR, then a review of exisiting network simulators is presented, followed by an overview of the selected network simulator, UMJaNetSim. Enhancements in UMJaNetSim to incorporate new functionalities of TE-LR is later described in detail. Finally, several test cases are produced to evaluate the correctness of TE-LR and the results of the test are presented.

4.1 Methodology

The performance of a computer system is usually evaluated by constructing an appropriate model of the system and executing the model to predict the system's behaviour and characteristics. Models allow the observed relationships to be summarized in a compact way and it assists in making predictions and inferences. Choosing the most appropriate model that suits the specific purpose of the system testing is a very complex task. [Stephenson00] had given the following stages that can be followed to identify a model that best suits the current problem:

1. ***Identification*** - A number of potentially suitable models are identified by critically analyzing the data using descriptive techniques.
2. ***Estimation*** - Using least-square methods or maximum likelihood methods, the model is fitted to the sample data to estimate the models' parameters and their confidence intervals.

3. ***Validation*** - By carefully analyzing the residuals (errors) of the fit and other diagnostics, the model's fit to the data are critically assessed.
4. ***Prediction*** - To select the best model that best suits the current situation, the model is used to make predictions. However, it is not necessary that a model that provides a good fit will also produce good predictions.

In order to get the best model, the above stages have to be repeated. Once the most appropriate model has been identified, it can be used to model a system and evaluate its performance. Various techniques are available to test the performance of a system either by using a model or without using a model. Three common techniques are [Bank00, Law00]:

- Empirical experimentation

  Empirical experimentation is also known as "real world" testing where it involves the construction of the actual system and evaluating it in its own environment. Though the technique allows the most intricate details of the system performance to be captured, it involves very expensive and complicated laboratory tests and "test beds". It also requires the acquisition of equipment, network and other facilities which are very costly and infeasible rendering the technique to be impractical.
- Analytical model

  In analytical model approach, the system is modeled as a set of mathematical equations to predict the result and performance of the system. The approach is very simple and allows complete control over the created model but it may result in some of the key features of the system to be overlooked due to over simplification.

- Simulation

  Simulation is the most effective technique to analyze and evaluate the behaviour of a system without building the actual system. It allows the testing of various types of topologies under various traffic conditions and characteristics. Simulation is most useful to explore the unknown and unproven. Due to its simplicity, feasibility in terms of cost and time saving, and flexibility, simulation has been employed in this thesis to test the proposed TE-LR mechanism.

4.2 What is Network Simulation?

Network simulation is the process of modeling real world computer network systems on a computer in order to determine how the real system performs and predict the effect of changes to the system as time progresses. The technology provides the analytical power required to evaluate the dynamic aspects of current processes as well as to investigate the potential for innovative new processes.

Simulation in general can be classified according to several criteria [Nance93]:

1. ***Stochastic Process*** versus ***Deterministic Computation***

   A *stochastic process* (also known as *Monte Carlo* simulation) is a process of change governed by probabilities at each step. The process uses *random number generators* to model the chance or random events. The stochastic process uses one or more time-dependent random variables to determine the behaviour of a system.

   On the other hand, when *deterministic computation* is used, the system will always produce the same final state given the same initial state and the same input. The most widely used deterministic computation is the ***deterministic finite state machine*** that has only one transition from the current state for each pair of state and input symbol.

2. ***Discrete Event Simulation*** versus ***Continuous Event Simulation***

   Discrete event simulation views the process under simulation as a series of state changes as the process progresses through a series of events. Associated with each process is its own state variables and each event has a starting and ending time. In contrast, a ***continuous event simulation*** represents those processes whose state and output changes continuously throughout the simulation as mathematical expressions. Once the choice of the mathematical expression has been made, it is easy to use *continous event simulation*.

3. ***Local Simulation*** versus ***Distributed Simulation***

   Local simulations involves the execution of the simulation model on a stand-alone system while in the *distributed simulations* environments, simulations are run on a network of interconnected computers. The most common examples of distributed simulations are the Aggregated Level Simulation Protocol (ALSP), the Distributed Interactive Simulation (DIS) and the High Level Architecture (HLA).

4.3 Existing Network Simulation Software

Network simulation software (also known as *network simulators*), enables us to evaluate network protocols under varying network conditions and over a long period of time at low cost. Some systems may have to be operated in harsh environments that may be too dangerous for human beings to be physically present. Therefore, network simulators provide secured atmosphere to test the system before using it in real life environments. Many network simulators, such as NS2, OPNET, MIT'S NETSIM, etc are widely available for studying and understanding the behaviour and characteristics of network protocols. In this research, a network simulator called UMJaNetSim developed at the Networks Research Laboratory of University of Malaya, Malaysia has been used. Detailed description of UMJaNetSim will be presented in Section 4.4, while a brief description of other commercially available network simulators follows next.

4.3.1 MIT's NETSIM

NETSIM is an event driven simulator designed by the MIT LCS Advanced Network Architecture group intended for modeling packet-switched networks. Though it was originally developed for personal use, it was later made available online at [Barnett94] for those who are interested in learning and using it. NETSIM's simulation engine is a single process witten in C.

Any kind of network that involves packet transmission can easily be modeled using NETSIM. The simple X window graphic user interface (GUI) representation allows network topology and its data to be displayed in graphical mode. The parameters of a component can either be an input given by the user or outputs produced by the simulator. They are stored as a singly-linked list and may be modified during run time, thus allowing effects of the changes to be analysed immediately.

The NETSIM package contains an event manager, I/O routines, various structural tools such as queues and lists, and a library of C functions. The components provided by NETSIM are links, hosts, switches and network connections. Whenever an event is sent to any of these components, the action routine is called. Three types of events are handled by a component: 1) *private*, these are events sent by a component to itself, 2) *regular*, these are events that are concerned with the actual running of the simulation, and 3) *command*, these are events that are concerned with housekeeping actions.

The simulator provides very limited functionality, that is, it only provides the means to schedule events and to communicate with users [Ernst97]. Other than that, adding and removing events are also very slow. However, its simplicity and its ability to allow code modifications to suit the needs of different users renders the simulation to be one of the best options available. For example, even though it supports only *static routing*, users are allowed to modify the code to implement and test any kind of routing algorithm. Users can also easily create their own components by just making slight modifications to the code since there is no specific data structure for the components.

4.3.2 NIST ATM/HFC Simulator

A simulator for studying and evaluating the performance of ATM and HFC (*Hybrid Fiber Coax*) networks which is based on NETSIM has been developed by the **N**ational **I**nstitute of **S**tandards and **T**echnology (***NIST***) [Golmie98i]. The simulator is called ***ATM/HFC*** *simulator* where HFC network is a network that uses coax and fiber technology to provide high speed digital services to cable TV subscriber premises.

The ATM/HFC simulator is written using C and uses X window graphic user interface (GUI) similar to NETSIM to allow interactive modeling.

The simulation engine of the simulator consists of components, event managers, I/O routines and other tools that help the creation of the simulation topology and setting up of the simulation environment. Two types of windows are available to display the components and simulation information: the *information windows* are used to display the component parameters and the *meter windows* are used to record the network activities.

The ATM/HFC simulator plays two important roles: one as ***ATM network planning tool*** and the other as ***ATM protocol analysis tool***. As a *network planning tool*, it runs simulations with various network configurations and traffic loads, and it produces statistics for analysis purpose. On the other hand, as a *protocol analysis tool*, it is used to study the total system effect of a particular protocol. Several classes of components are available within the simulator such as physical links, ATM switches, broadband terminal equipment (B-TE) and various types of ATM applications. It is claimed that the modules simulating the components can be changed or added. The source code and user manual are made available freely on NIST's official website for those who are interested in using it for teaching or research purposes.

4.3.3 OPNET

OPtimized ***N***etwork ***E***ngineering ***T***ools (OPNET) is a generic commercial network simulator developed by Third Millenium Technologies (MIL3) Inc., to provide sophisticated network modeling and simulation environment [Duan06]. It is a discrete-event simulator that provides very flexible and scalable approach towards the design and study of communication networks, devices, protocols and applications.

A number of features are added into OPNET's software package, such as an event-driven scheduled simulation kernel, integrated analysis tools for interpreting and synthesizing output data, graphical specification of models and a hierarchical object-based modeling tool. The package provides a range of useful tools to allow users specify models in great detail. The tools can be classified into four main categories, namely *OPNET Modeler*, *OPNET Planner*, *Analysis Tools* and *Modeling Libraries*.

The OPNET Modeler employs hierarchical modeling structure where each level describes different aspects of the simulated model. This way the models can be reused during other simulations. It comes with four types of editors that aid in the development of the simulated models:

- ***Network Editor***. It provides graphical user interface to allow users to design their topologies with convenience and ease. The communication entities in OPNET are nodes and links. OPNET also allows wireless networks to be modeled using the radio links. The characteristics of both nodes and links can be customized via the user-friendly GUI-based property windows.
- ***Node Editor***. It is used to specify the communication entities created by the Network Editor in terms of interconnected modules.
- ***Process Editor***. The process editor is used to create process models that describes the processes and queues as a finite state machine where the states and transitions are graphically specified using state-transition diagrams written using C-like programming language called *Proto-C*.

The OPNET Planner allows users to analyse the network performance and behaviour using discrete-event simulations. Two types of tools are available to define and run simulations, and to debug the simulation at run-time. They are the *Simulation Tool* and *Debugging Tool*. Data and output collected during the simulation can be evaluated and manipulated via the use of various tools such as the Probe Editor, Analysis Tool, Filter Tool and Animation Viewer. Both OPNET Modeler and OPNET Planner are equipped with modeling libraries that provide protocols and analysis environments such as ATM, TCP, IP, and so on.

One significant advantage of OPNET is its support for large communication systems ranging from a single LAN to global satellite networks. Due to this, the complexity of the network model may be very huge causing scalability problems. OPNET handles this by applying an abstraction method called *subnetworking* [Ernst97]. Despite the advantage, OPNET also has some disadvantages. Firstly, OPNET do not allow users to define new modules. Users have to choose a model from a pre-defined set of models that comes with the package. Secondly, to use OPNET for either teaching or research purpose, one has to acquire the license which is very costly. Finally, users are not allowed to modify the OPNET software except for researchers who have obtained special licenses permitting them to do so.

4.3.4 REAL

REAL (***R***ealistic ***A***nd ***L***arge) is a network simulator written by S.Keshav [Keshav88i, Keshav88ii] at Cornell University for studying the dynamic behaviour of flows and congestion control schemes in packet-switched data networks. REAL was built based on the modified version of NEST 2.5 which was developed at Columbia University.

REAL adopts the client-server arhictecture where the server located at Cornell University is connected to the client via Berkeley UNIX socket. Simulations are carried out on the server machine, while users set up their simulation environment and control their simulations via the display client.

The initial version of REAL allows networks to be modeled only as graphs where the descriptions of the network topology, protocols work-load and control parameters are transmitted to the server using a single ASCII representation of the network called *NetLanguage*. However, newer releases come with GUI written in Java to allow users to create topologies graphically without having to download and build the simulator. Once the server machine completes running a simulation, it returns the status information and results to the client via the same socket.

REAL's network layer is datagram-oriented while the transpost layer is modeled based on Transmission Control Protocol (TCP) that provides reliable, sequenced packet transmissions using standard techniques of flow control, timeout estimation and packet retransmission. It provides 30 modules to emulate flow control protocols such as TCP and 5 scheduling disciplines such as FIFO (First In First Out), Fair Queuing, DEC congestion avoidance and Hierarchical Round Robin. The workload presented to the simulator is of three types [Ernst97]: 1) *FTP (File Transfer Protocol)*, that allows the transfer of large files, 2) *TELNET*, that uses exponential inter-packet spacing to transfer minimal sized packets, and 3) *ill-behaved sources*, that continuously sends maximally sized packets which is not more than the network's capacity.

The main advantage of using REAL is it client-server architecture itself. Users can run large simulations parallely on several servers with just a single point of control. It also provides library of routines that collect simulator statistics to generate report automatically. In contrast to OPNET, REAL is available for free and users can modify the source code to suit their specific requirements. The main disadvantage of REAL worth-noting is its platform dependency where it runs only on UNIX platform.

4.3.5 Network Simulator 2 (NS2)

NS is an object-oriented discrete-event simulator intended to study packet switched networks. It is mainly used for small scale simulations of queuing algorithms, transport protocol congestion control and some multicast related work. The initial version of NS, *NS version 1.0* was developed by the Network Research Group at the Lawrence Berkeley National Laboratory (LBNL) but its later developments and evolutions was made part of the VINT project funded by the DARPA in collaboration of XEROX PARC and LBNL [Fall03]. The main aim of the VINT project is to unite the efforts of people working on network simulations. NS is a very powerful simulator in that it is written using two different programming languages, C++ and OTcl. The simulator supports a class hierarchy in C++ (also known as *compiled hierarchy*) and a class hierarchy in OTcl interpreter (also known as *interpreted hierarchy*). Both class hierarchies are related to each other by a one-to-one correspondence.

New simulator objects are created through the interpreter and they are closely mirrored by a corresponding object in the compiled hierarchy. The idea behind this is that, all tasks that require greater run-time speed but lower turn-around time (e.g., run simulation, find and fix bugs, re-compile, re-run) can be implemented using C++, while tasks requiring greater interation time (e.g., changing parameters or configurations, quickly exploring various scenarios,etc) but lower run-time are implemented using OTcl [Ernst97]. Hence, detailed protocol implementation can be done using C++ that is faster to run but slower to change and manipulating simulation configurations can be done using OTcl that is slower to run but quicker to change.

The core class in NS2 is the *class Simulator*, written in C++ to provide procedures to create and manage the topology, to initialize the packet format and to choose the scheduler. The components supported are nodes, links, agents, FTP and Telnet traffics. NS2 simulation engine is single-threaded, meaning that it allows only the execution of one complete event at one time. Four types of event schedulers are available in NS2: a simple linked-list (default), heap, calendar queue and real-time. Unlike NETSIM that supports only static routing, NS2 supports both static and dynamic routing strategies. User can choose their choice by specifying it in *class Simulator*.

Whenever a new simulator object is created, three main operations are performed by the initialization procedure in *class Simulator*.

- initialize packet format
- create a scheduler
- create a "null agent" (i.e., discard sinks)

class Simulator also provides a number of other methods to set up the simulation which are categorized as:

- methods to create and manage topology

- methods to perform tracing
- helper functions to deal with the scheduler

One problem with NS2 is scaling. When the number of nodes increases, storage requirements also increase due to the need to store large routing tables. The VINT project has proposed the use of more efficient hierarchical routing table structure instead of using the existing flat addressing structure. It also proposes to further improve the performance by replacing the linear search insertion algorithm with heap or calendar queue. Soon, a distributed version of the simulator will be available via the VINT project.

4.4 Overview of the Simulation Environment

In this section, a general overview of the network simulator used to evaluate the correctness and to demonstrate the performance of the proposed mechanism is presented. The simulator, known as UMJaNetSim is based on NIST ATM/HFC network simulator and was developed by Lim, S.H., et al. [Lim00] for research purposes.

- **Usage**

  UMJaNetSim can be used to model and evaluate any kind of network whose components interact with each other by exchanging messages.

- **The Architeture**

  The use of Java programming language in its development has brought significant simplicity and flexibility in creating simulation models and controlling the overall simulation process. Architecturally, UMJaNetSim is composed of a number of classes that are grouped into two main parts, namely the *simulation engine* and the *simulation topology* (see Figure 4.1).

Classes that reside inside the dotted rectangle area form the main simulation engine while claases that reside outside the dotted rectangle area form the simulation topology [Lim00]. *class Javasim* in the simulation engine acts as the main controller of the simulator and performs two major management tasks: ***event management*** and ***GUI management***. As an event manager, *JavaSim* manages an event queue, an event scheduler and a simulator clock. As a GUI manager, *JavaSim* with the support of an helper object called *SimPanel* handles the creation of simulation components, manages the dialog boxes associated to the component parameters and, handles user inputs and manipulation of component properties. Other classes in the simulation engine provides helper functions to assist in the proper functioning of the simulator. For instance, the *SimEvent object* stores descriptions about an event such as the event ID, the source and destination (target) components of the event, the time of the event invocation and some other parameters. The *SimClock object* provides functions to convert the real time to "ticks" and vice versa.

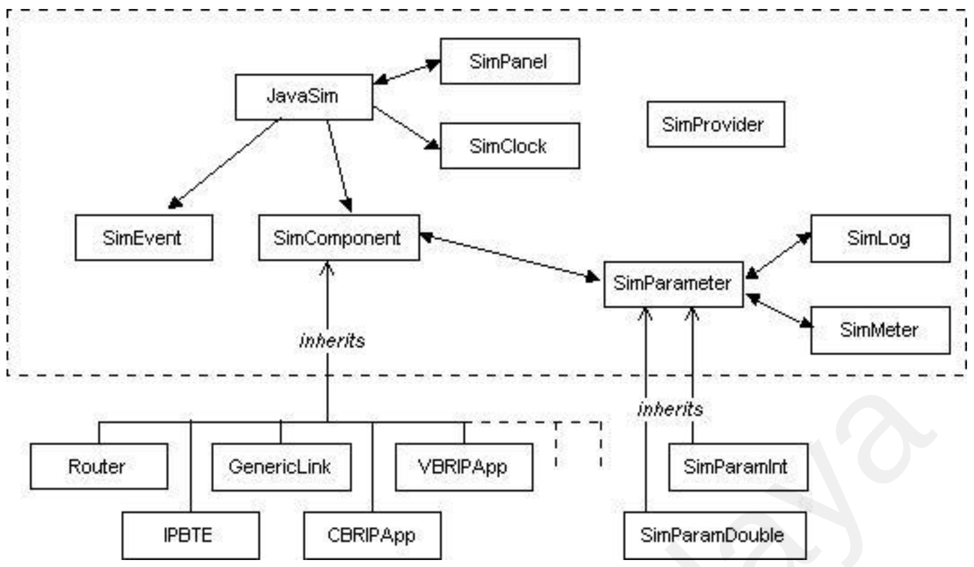


Figure 4.1 The Architecture of UMJaNetSim

On the other hand, the simulation topology is consist of classes that support the creation of simulation components and settings of the component parameters. To create a new component, for instance a router, *class SimComponent* can be inherited. The components of a network and their properties can easily be viewed via the GUI representation capability of UMJaNetSim. During the execution of a simulation, the simulation engine interacts with the components within the simulation topology. The interaction goes through two important steps [Lim00]:

1. Any simulation component can schedule an event targeted for any other simulation component (target component) or for itself by enqueuing the event using the *enqueue* function. Every event has a specified time (implemented using a *timer*) upon reaching which the event must be fired.
2. When an event's timer expires, the simulation engine invokes the event handler of the target component in order for the component to react accordingly.

- **Implementation of the Simulator**

  Components in UMJaNetSim are such as ATM switches/routers, Broadband Terminal Equipments (BTEs), traffic sources, generic links and applications. They are inherited from the *SimComponent* class. Every component has associated with it a set of parameters that describe its behaviour. The parameters are displayed on simulation window using GUI-based dialog boxes and they can be classified as either external or internal parameters.

  External parameters are those that can be edited by the user at run-time and are displayed in the parameter dialog box. For example, the packet transmission rate and simulation start time. In contrast, internal parameters are those that are not editable by the user at run-time and are displayed in the meter dialog box. Packet loss ratio and bandwidth utilization ratio are examples of such parameters. UMJaNetSim implements the event queue as lists using the *java.util.List*. Events are added to the list and are fired in the order of their arrival. The time is represented as a "tick" where the duration of a tick is configurable by the user. The default duration of a tick is 10 nanoseconds.

4.5 Implementation of TE-LR

In this thesis, UMJaNetSim has been used to implement and evaluate the proposed TE-LR mechanism due to its flexibility (i.e., components can be extended and added quickly) and portability (i.e., can run virtually on any platform).

The simulation environment used to evaluate the proposed algorithm is composed of the following existing components of UMJaNetSim:

- ATMLSR

  Generally, a router allows nodes in a network to interact with each other by passing messages and data packets to each other. To model the simulation environment, ATMLSRs (Asynchronous Transfer Mode Label Switching Routers) were used which are capable of routing MPLS's native L3 packets. ATMLSR inherits *SimComponent*.

- IPBTE

  An IPBTE (IP Broadband Terminal Equipment) is responsible of aggregating traffics from customer sites. Customer sites are represented by traffics sources attached to an IPBTE such as CBRIP and VBRIP. There can be more than one traffic source attached to one IPBTE and more than one IPBTE can be attached to one ATMLSR. IPBTE inherits *SimComponent*.

- CBRIPApp

  CBRIPApp (Constant Bit Rate IP Application) is an application that generates constant bit rate traffics that require a fixed amount of bandwidth available continuously as long as the connection is active. Traffics generated by CBRIP applications will be aggregated by an IPBTE. CBRIPApp inherits *SimComponent*.

- VBRIPApp

  VBRIPApp (Variable Bit Rate IP Application) is an application that generates traffics at various rates and requires varying amount of bandwidth (must be within the total capacity of the link). It is used to emulate web traffics. Traffics generated by VBRRIP applications will be aggregated by an IPBTE. An IPBTE can have both CBRIP and VBRIP sources attached to it at the same time. VBRIPApp inherits *SimComponent*.

- GenericLinks

  To connect two ATMLSRs or an ATMLSR to an IPBTE, generic links are used. The capacity of the links can be set by the user. The default capacity is 155Mbps. GenericLink also inherits *SimComponent*.

In addition to the components described in the previous section, the implementation of **TE-LR** requires some modification to UMJaNetSim. The discussion of the modification is broken into several categories:

- Collection of bandwidth utilization ratio samples.
- Construction of linear regression line equation.
- Advertisement of link state updates.

4.5.1 Collection of bandwidth utilization ratio samples

The construction of linear regression line equation requires the sampling of the bandwidth utilization ratio of a link at a fixed sampling interval, *i*. The implementation of the sampling process is done in *class ATMLSR*'s private method called ***private void sw_averaging_interval().*** The code is shown below with some parts omitted to shorten the length:

```
private void sw_averaging_interval() {
    long t1,t2;
    java.util.Iterator i=voports.values().iterator();
    while(i.hasNext()) {
      Port voport=(Port)i.next();
      ////// Calculate link utilization
        Double linkspeed = (Double)(voport.to_link.compInfo
                                        (GenericLink.GET_CAPACITY,this,null))[0];
        if(sw_ef_no_traffic.getValue()) {
        //fake the cellsInWindow
```

-- Continued on the next page --

-- Continued from previous page --

```
      voport.cellsInWindow += (int)(voport.EFcurrent / 1000.0 *
                                                          sw_ai.getValue() / 424.0);
    }
    voport.utilization.setValue(voport.cellsInWindow * 424.0 /sw_ai.getValue() /
                                                    linkspeed.doubleValue() * 100.0);

                  -------- Code omitted to reduce length --------

     //The following are the codes to collect samples
     Double utilization = new Double (voport.utilization.getValue());
     double myTime = SimClock.Tick2Sec(theSim.now());
    Double sampleTime = new Double(myTime);//to store the time in a list

     int listSize = 20;//number of samples to collect
     boolean second = false; //make sure that two lines have been computed
     boolean trigger = false; //if trigger == true, send LSA

     if(myTime > 5.0) { //give some time for convergence
       //Keep on adding samples to the list until there are 20 samples
       if(voport.bwUtilPercent_list.size() < listSize) {
             voport.bwUtilPercent_list.add(utilization);
             voport.time_list.add(sampleTime);
       }
       else {
             if(voport.bwUtilPercent_list.size() == listSize){
               Object element = voport.bwUtilPercent_list.getLast();
               //20 samples collected, compute the regression line
               voport.firstLine_M = calculateM(voport, listSize);
               voport.bwUtilPercent_list.add(utilization);
               voport.time_list.add(sampleTime);
               //Start counting number of link state updates advertised
               startLSACount = true;
             }
             else{
                second = true;
                voport.bwUtilPercent_list.removeFirst();
                voport.time_list.removeFirst();
                Object element = voport.bwUtilPercent_list.getLast();
                voport.nextLine_M = calculateM(voport, listSize);
                voport.bwUtilPercent_list.add(utilization);
                voport.time_list.add(sampleTime);
             }
       }

       //Already have two 'm' values, so compare the lines now
       if(second == true){
             trigger = compareM(voport.firstLine_M, voport.nextLine_M);
             if(trigger == true) {
              voport.firstLine_M = voport.nextLine_M;
              }
       }

     }

    if(delayedLSA) { //if there is a pending advertisement
      //good, we use the latest value :)
      delayedLSA=false;
      voport.lastAdvertisedBandwidth = curpercent *
                                            linkspeed.doubleValue()/100.0;
      generate_RouterLSA();
    }
    else if (myThreshold.getValue() == 0) {

     if (trigger == true){
             voport.lastAdvertisedBandwidth = curpercent *
                                                          linkspeed.doubleValue()/100.0;
             if(sw_dynamic_ospf.getValue()==true) {
                     voport.lsaSent = true;
                     generate_RouterLSA();
             }

     }
     else{
```

-- Continued on the next page --

-- Continued from previous page --

```
              voport.lastAdvertisedBandwidth = curpercent *
                                               linkspeed.doubleValue()/100.0;
         }

       }
       else if(myThreshold.getValue() == 1) {
          if((Math.abs(lastpercent-curpercent)/lastpercent*100) >
                                                   sw_dospf_threshold.getValue()) {
          //ok, changes exceed the threshold, update the available bandwidth
          voport.lastAdvertisedBandwidth = curpercent *
                                                      linkspeed.doubleValue()/100.0;
               if(sw_dynamic_ospf.getValue()==true) {
                      generate_RouterLSA();
               }
          }
       }

                  -----------Code omitted to reduce length-----------
}
```

-- THE END --

The following attributes have been added to method ***sw_averaging_interval()*** in order to perform the abovementioned operation of the proposed mechanism:

1. ***Double utilization***

   utilization is an object of type *Double* that is declared and used in method ***sw_averaging_interval()*** to store the bandwidth utilization ratio each time it is sampled. When the value is sampled, it is in the primitive data type *double*. In order to add the value to Java linked-list it has to be converted to an object of type Double.

2. ***Double sampleTime***

   Similar to utilization, *sampleTime* is also declared to be an object of type *Double* to store the time values at which each sampling is done. Each time the time value is read, it will be added to the Java linked-list.

3. ***int listSize***

   This is a variable of *integer* type used to specify the number of samples to be collected in order to construct the linear regression line equation. Whenever the number of sampled bandwidth utilization ratio reaches *listSize*, the method to construct the linear regression line equation will be invoked. The value of *listSize* can be changed to any preferable value.

4. ***boolean second***

   The variable *second* can take either the value of ***true*** or ***false***. When it is set to *true*, it is an indication that two or more linear regression lines have already been computed and the process of comparing the lines can be performed. Its value will be *false* only during the initial stage where only one line has been computed. For subsequent lines, it will remain as *true*.

5. ***boolean trigger***

   An LSA can only be sent if the computed tangent difference between two linear regression lines exceeds a pre-defined threshold value. Whenever this happens, the *boolean* variable *trigger* will take the value of ***true*** indicating that an LSA has to be sent out. Otherwise, it remains as ***false***.

6. ***java.util.LinkedList bwUtilPercent_list***

   bwUtilPercent_list is a linked-list that stores all the sampled bandwidth utilization ratio as objects. The maximum size of the list is limited by *listSize*. It is declared in *class Port*, so that for each interface of a router, there will be a separate list.

7. ***java.util.LinkedList time_list***

   time_list is also a linked-list declared in *class Port* to store the time at which each sample is taken. Similar to bwUtilPercent_list, the maximum size of time_list is also limited by *listSize*.

8. ***double firstLine_M***

   This is a variable of type *double* added to *class Port* to stores the computed tangent value of the first-most linear regression line, L_o. The value will then be compared with the tangent value of the subsequent linear regression lines. In the event that an LSA is triggered, *firstLine_M* will be substituted with the tangent value of L_c.

9. ***double nextLine_M***

   This is a variable of type *double* that stores the computed tangent value of subsequent linear regression lines, L_c. Each time a value is stored in *nextLine_M*, both *firstLine_M* and *nextLine_M* will compared using Equation (3.1). If the absolute difference exceeds the threshold, *firstLine_M* will be substituted by the value in *nextLine_M*.

10. ***boolean startLSACount***

    This is a *boolean* variable declared as a private variable in *class ATMLSR* that indicates when the counting of LSA should be started. The counting will be started the moment the first comparison takes place. Each time the router floods the LSA into the network, the number of LSAs sent out will be traced. The final value will be used in the calculation of update message overheads to evaluate the proposed mechanism's performance.

11. ***SimParamIntTag myThreshold***

    myThreshold is a component of type *SimParamIntTag* that specifies which mechanism should be used in the simulation. Two options are available: QoS-OSPF and TE-LR. It appears on the property window of each router as ***V Threshold_Type*** so that users can make their selection.

4.5.2 Construction of linear regression line equations

The construction of the linear regression line equation itself is performed by a new method added to class ATMLSR called ***calculateM( )*** which takes the port id and the size of the sample list as inputs. Equations (2.19) and (2.20) are applied in the computation of tangent and intercept values of a line. The implementation of the method follows:

```
private double calculateM(Port voport, int n){
  double sumX=0.0, sumY=0.0, sumXY=0.0, sumX2=0.0, sumXX=0.0;
  double [] arrayX = new double [n];//holds sample time
  double [] arrayY = new double [n];//holds bw utilization in percentage
  double M = 0.0;
  double B = 0.0;

  java.util.Iterator itr1 = voport.time_list.iterator();
  java.util.Iterator itr2 = voport.bwUtilPercent_list.iterator();

  int count = 0;

  //Transfer sample time into arrayX
  while(itr1.hasNext()) {
     Object element = itr1.next();
     double sampleTime = ((Double)element).doubleValue();
     arrayX[count++] = sampleTime;
  }

  count = 0;

  //Transfer bw utilization into arrayY
  while(itr2.hasNext()) {
     Object element = itr2.next();
     double bwUtil = ((Double)element).doubleValue();
     arrayY[count++] = bwUtil;
  }

  //Calculate sum of x, sum of y, sum of xy, sum of x squared,
  //squared sum of x
  for(int i=0; i<n; i++)
  {
       sumX = sumX + arrayX[i];
       sumY = sumY + arrayY[i];
       sumXY = sumXY + (arrayX[i] * arrayY[i]);
       sumX2 = sumX2 + (arrayX[i] * arrayX[i]);
  }

  sumXX = sumX * sumX;

  //Calculate M (tangent of the line)
  M = ((n * sumXY) - (sumX * sumY)) / ((n * sumX2) - (sumXX));

  //Calculate B (intercept of the line)
  B = (sumY - M * (sumX)) / n;

  return M;
}
```

4.5.3 Advertisement of link state updates

Finally, the most important part, that is, the process of deciding when an LSA should be distributed was implemented by adding another new method into *class ATMLSR* called ***compareM( )***. The main function of this method is to compare the tangent values of two successive linear regression lines using Equation (3.1) and then return the value of ***true*** if there is a need to send an LSA or ***false*** otherwise. One important attribute that has been defined to allow users to set the preferred threshold value is ***Threshold_M***. It is a component that inherits *SimParamDouble* displayed on every router's property window requiring user input. The implementation of the method follows:

```
private boolean compareM(double m1, double m2){

  double difference = Math.abs(m2 - m1);

  //If difference is greater than threshold, return true, otherwise false
  if(difference > Threshold_M.getValue()){
       return true;
  }
  else {
          return false;
  }
}
```

4.6 The Testing Process

Testing is the process of validating and verifying a system to ensure that its design and implementation meets the actual requirement specifications and that it is acceptable by the user. Among all the stages in a software development process, testing plays a very important role due to several factors:

1. Identifying errors and problems in early stages of the software development cycle helps reduce the cost of the software development significantly.
2. When a system is tested, the actual functioning of the system can be verified and validated against its expected functions.
3. Systems that are well tested and documented are easy to learn and use, thus reduces the cost of staff re-training.

4. Customers who are satisfied with the system tend to be loyal to the system developer and will bring in more business opportunities.

A system, whether a software system or a hardware system, is consists of a number of sub-systems, each of which may in turn be composed of many modules. A module, on the other hand is made up of many procedures and functions. Therefore system testing should not only concentrate on testing the system as a single unit but must also test the workings of each individual component and the interactions among them. Five main testing stages have been proposed in [Sommerville98], and these are illustrated in Figure 4.2.

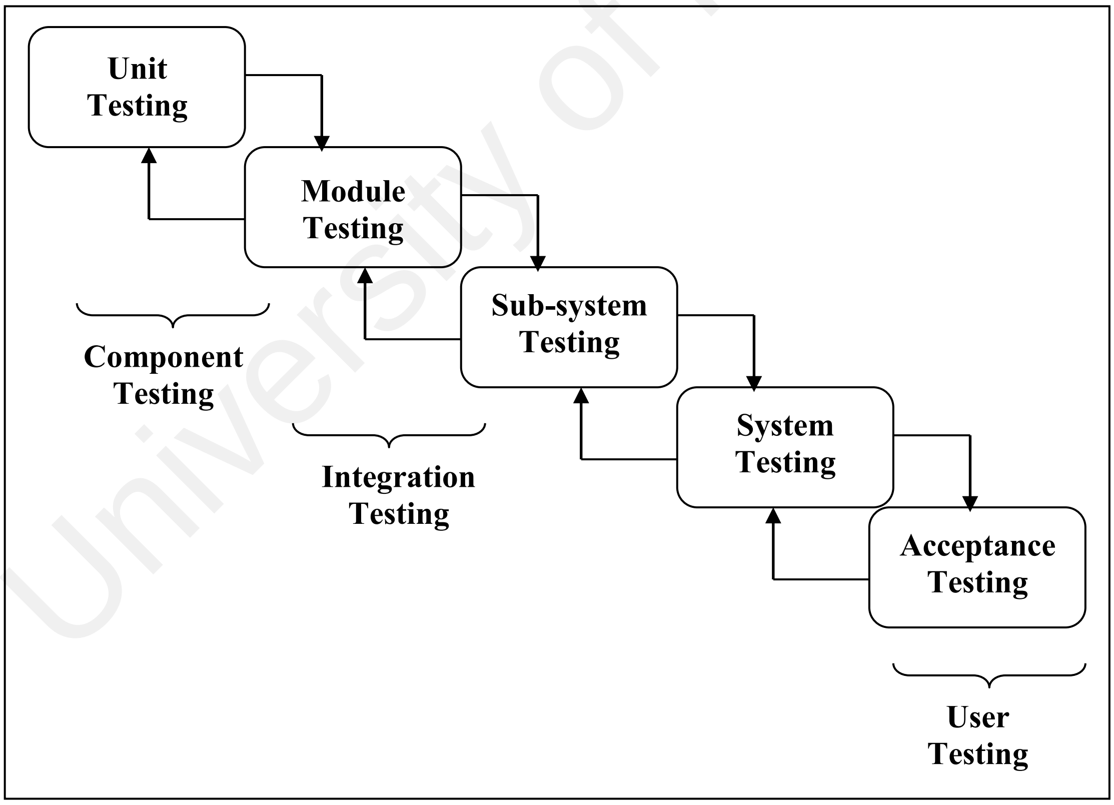


Figure 4.2 The testing process

4.6.1 Testing TE-LR

A number test cases have been developed to test TE-LR to ensure that all the components and the interactions among them are working as expected. The testing process started with component testing, then followed by an integration testing and finally a full system testing. A detailed description of the tests carried out are presented in the following section.

4.6.1.1 Testing the bandwidth utilization ratio sampling process

A test was carried out to determine whether the code is able to sample the bandwidth utilization ratios at the expected sampling interval. The sampling interval here is actually determined by the interval between each successive entry into the ***sw_averaging_interval()*** method. This test was also intended to ensure that exactly ***N*** number of samples were collected before the linear regression line equation can be constructed. The topology used for this test is as shown in Figure 4.3. Two ATM switches denoted by R1 and R2 are connected by a point-to-point generic link, L1. Each router has one BTE that aggregates traffics from the customer sites. BTE1 is attached to two types of applications, CBR and VBR. CBR is responsible of generating constant traffic while VBR is responsible for emulating web traffic. To allow protocol comvergence, the simulation was left to run for 5.0 seconds before the data sampling was performed.

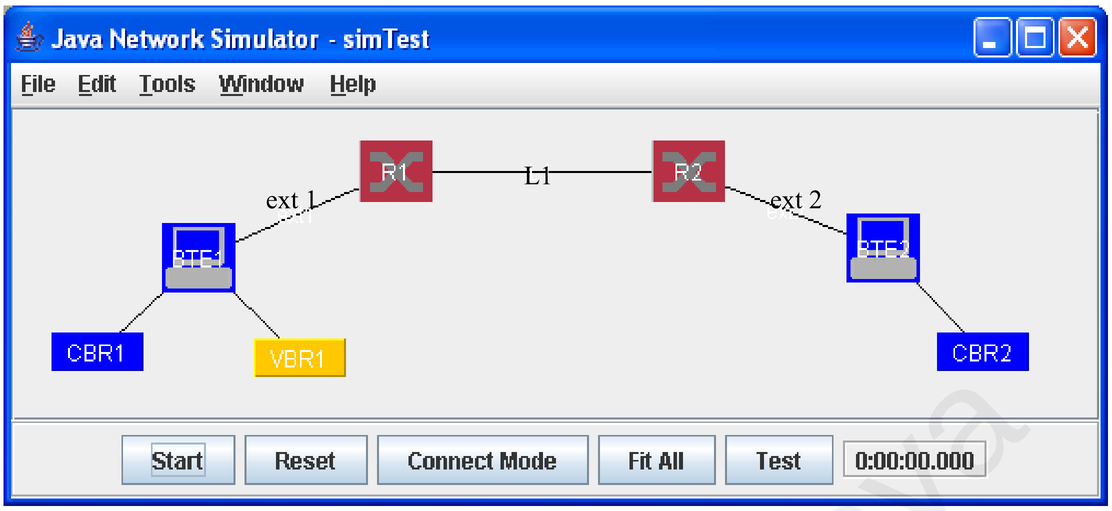


Figure 4.3 Test Topology for TE-LR

Each router is required to sample the bandwidth utilization ratio at every sampling interval, ***S*** = 0.1 seconds. This means that the method ***sw_averaging interval()*** will be entered every 0.1 second. Once there are enough samples, that is, N = 20 samples, the module should construct the equation of the linear regression line to obtain the tangent value for that line. The parameter settings and the trace produced by the simulation test run follows:

Table 4.1 IP Settings for R1 and R2

Router Name	Interface to Link	Interface IP Address
R1	L1 ext1	1.0.0.1 101.0.0.1
R2	L1 ext2	1.0.0.2 102.0.0.1

Table 4.2 Property Settings for R1 and R2

Router Properties	Settings
V Threshold Type	TE-LR
Threshold_M	5.0
Sampling Interval, *i* (secs)	0.1
Sample Size, *N*	20
Sampling Start Time (secs)	5.0

Table 4.3 Characteristics of VBR1

VBR Source Properties	Settings
Bit Rate (Mbits/s)	0.8
Mean Burst Length (usecs)	50000
Mean Interval Bet. Bursts (usecs)	15000
Start Time (secs)	1.0
Number of Mbits to be sent	3
Repeat count (-1 = inf)	-1
Delay between calls (usecs)	1.0
DiffServ Class	Best Effort
Poisson Model	TRUE
Random Data Size	TRUE
Random Delay Bet. Calls	TRUE
Enable Starting Delay	TRUE
Random Destination	FALSE
Avoid Local Destination	TRUE
Destination IP	102.0.0.0

Table 4.4 Characteristics of CBR1

CBR Source Properties	Settings
Bit Rate (Mbits/s)	0.8
Bit Rate Lower Bound (Mbits/s)	0.1
Bit Rate Upper Bound (Mbits/s)	1.0
Start Time (secs)	1.0
Number of Mbits to be sent	3.0
Repeat count (-1 = inf)	-1
Delay between calls (usecs)	1.0
DiffServ Class	Best Effort
Random Data Size	TRUE
Random Delay between Calls	TRUE
Enable Starting Delay	TRUE
Random Destination	FALSE
Avoid Local Destination	TRUE
Destination IP	102.0.0.0

Table 4.5 Source Address Settings for BTE1 and BTE2

BTE Name	Source Address
BTE1	101.0.0.0/24
BTE2	102.0.0.0/24

Table 4.6 Link Speed Settings for L1, ext1 and ext2

Link Name	Link Speed (MBits/sec)
L1	2.0
ext1, ext2	155.0

Expected Output of the Test Run:

- The process should be able to sample every link at the specified sampling interval.
- Two lists must be maintained, one for storing the sampled bandwidth utilization ratios and the other for storing the time at which each sampling was done.
- Both lists should be able to store a maximum of 20 samples each.

(***Note***: *to shortent the length of the traces produced, some of the traced codes have been omitted from the sample traces given in this and subsequent sub- sections.*)

Actual Output Trace:

```
5029.34443ms: R2-> Interface 1.0.0.2 : 1 sample(s) added to list
Elements in bw list: [40.068]
Elements in time list: [5.02934443]            Two lists

5029.34443ms: R2-> Interface 102.0.0.1 : 1 sample(s) added to list
Elements in bw list: [0.6401032258064516]
Elements in time list: [5.02934443]

5039.618920000001ms: R1-> Interface 1.0.0.1 : 1 sample(s) added to list
Elements in bw list: [52.788000000000004]
Elements in time list: [5.03961892]

5039.618920000001ms: R1-> Interface 101.0.0.1 : 1 sample(s) added to list
Elements in bw list: [0.5170064516129033]
Elements in time list: [5.03961892]

5129.34443ms: R2-> Interface 1.0.0.2 : 2 sample(s) added to list
Elements in bw list: [40.068, 40.068]
Elements in time list: [5.02934443, 5.12934443]      Proper sampling interval

5129.34443ms: R2-> Interface 102.0.0.1 : 2 sample(s) added to list
Elements in bw list: [0.6401032258064516, 0.7221677419354838]
Elements in time list: [5.02934443, 5.12934443]

5139.618920000001ms: R1-> Interface 1.0.0.1 : 2 sample(s) added to list
Elements in bw list: [52.788000000000004, 54.272]
Elements in time list: [5.03961892, 5.13961892]

5139.618920000001ms: R1-> Interface 101.0.0.1 : 2 sample(s) added to list
Elements in bw list: [0.5170064516129033, 0.5170064516129033]
Elements in time list: [5.03961892, 5.13961892]
```

-- Continued on the next page --

-- Continued from previous page --

```
                    ------output omitted------

6829.344430000001ms: R2-> Interface 1.0.0.2 : 19 sample(s) added to list
Elements in bw list: [40.068, 40.068, 39.856, 40.068, 40.068, 39.856, 40.068, 40.068,
39.856, 40.068, 40.068, 39.856, 40.068, 40.068, 39.856, 40.068,
40.068, 39.856, 40.068]
Elements in time list: [5.02934443, 5.12934443, 5.22934443, 5.32934443, 5.42934443,
5.52934443, 5.62934443, 5.72934443, 5.82934443, 5.92934443, 6.02934443, 6.12934443,
6.22934443, 6.32934443, 6.42934443, 6.52934443, 6.62934443, 6.72934443, 6.82934443]

6829.344430000001ms: R2-> Interface 102.0.0.1 : 19 sample(s) added to list
Elements  in  bw  list:  [0.6401032258064516,  0.7221677419354838,  0.5771870967741936,
0.6401032258064516,    0.6428387096774193,    0.6510451612903226,    0.6291612903225806,
0.6838709677419355,    0.6318967741935484,    0.689341935483871,    0.752258064516129,
0.752258064516129,     0.6018064516129031,     0.32825806451612904,0.19421935483870967,
0.17233548387096775, 0.13950967741935483, 0.12583225806451612, 0.10941935483870968]
Elements  in  time  list:  [5.02934443,  5.12934443,  5.22934443,  5.32934443,  5.42934443,
5.52934443,  5.62934443,  5.72934443,  5.82934443,  5.92934443,  6.02934443,  6.12934443,
6.22934443, 6.32934443, 6.42934443, 6.52934443, 6.62934443, 6.72934443, 6.82934443]

6839.618920000001ms: R1-> Interface 1.0.0.1 : 19 sample(s) added to list
Elements  in  bw  list:  [52.788000000000004,  54.272,  43.248,  50.032,  50.668,  49.184,
49.608000000000004,  52.364,  48.76,  55.544000000000004,  59.36,  55.120000000000005,
50.456, 21.412, 11.236, 16.112000000000002, 9.964, 8.692, 7.632]
Elements  in  time  list:  [5.03961892,  5.13961892,  5.23961892,  5.33961892,  5.43961892,
5.539618920000001,  5.63961892,  5.73961892,  5.83961892,  5.93961892,  6.039618920000001,
6.13961892,    6.23961892,    6.33961892,    6.43961892,    6.539618920000001,    6.63961892,
6.73961892, 6.83961892]

6839.618920000001ms: R1-> Interface 101.0.0.1 : 19 sample(s) added to list
Elements   in   bw   list:   [0.5170064516129033,  0.5170064516129033,  0.5142709677419356,
0.5170064516129033,    0.5170064516129033,    0.5142709677419356,    0.5170064516129033,
0.5170064516129033,    0.5142709677419356,    0.5170064516129033,    0.5170064516129033,
0.5142709677419356,    0.5170064516129033,    0.5170064516129033,    0.5142709677419356,
0.5170064516129033, 0.5170064516129033, 0.5142709677419356, 0.5170064516129033]
Elements  in  time  list:  [5.03961892,  5.13961892,  5.23961892,  5.33961892,  5.43961892,
5.539618920000001, 5.63961892, 5.73961892, 5.83961892, 5.93961892, 6.039618920000001,
6.13961892,    6.23961892,    6.33961892,    6.43961892,    6.539618920000001,    6.63961892,
6.73961892, 6.83961892]

6929.344430000001ms: R2-> Interface 1.0.0.2 : 20 sample(s) added to list
Elements in bw list: [40.068, 40.068, 39.856, 40.068, 40.068, 39.856, 40.068, 40.068,
39.856, 40.068, 40.068, 39.856, 40.068, 40.068, 39.856, 40.068,40.068, 39.856, 40.068,
40.068]
Elements  in  time  list:  [5.02934443,  5.12934443,  5.22934443,  5.32934443,  5.42934443,
5.52934443,  5.62934443,  5.72934443,  5.82934443,  5.92934443,  6.02934443,  6.12934443,
6.22934443,  6.32934443,  6.42934443,  6.52934443,  6.62934443,  6.72934443,  6.82934443,
6.92934443]
```

20 samples added to list

```
6929.344430000001ms: R2-> Interface 102.0.0.1 : 20 sample(s) added to list
Elements  in  bw  list:  [0.6401032258064516,  0.7221677419354838,  0.5771870967741936,
0.6401032258064516,    0.6428387096774193,    0.6510451612903226,    0.6291612903225806,
0.6838709677419355,    0.6318967741935484,    0.689341935483871,    0.752258064516129,
0.752258064516129,     0.6018064516129031,     0.32825806451612904,0.19421935483870967,
0.17233548387096775,   0.13950967741935483,   0.12583225806451612,   0.10941935483870968,
0.22978064516129032]
Elements  in  time  list:  [5.02934443,  5.12934443,  5.22934443,  5.32934443,  5.42934443,
5.52934443,  5.62934443,  5.72934443,  5.82934443,  5.92934443,  6.02934443,  6.12934443,
6.22934443,  6.32934443,  6.42934443,  6.52934443,  6.62934443,  6.72934443,  6.82934443,
6.92934443]
```

-- Continued on the next page --

-- Continued from previous page --

```
6939.618920000001ms: R1-> Interface 1.0.0.1 : 20 sample(s) added to list
Elements in bw list: [52.788000000000004, 54.272, 43.248, 50.032, 50.668, 49.184,
49.608000000000004, 52.364, 48.76, 55.544000000000004, 59.36, 55.120000000000005,
50.456, 21.412, 11.236, 16.112000000000002, 9.964, 8.692, 7.632, 20.988]
Elements in time list: [5.03961892, 5.13961892, 5.23961892, 5.33961892, 5.43961892,
5.539618920000001, 5.63961892, 5.73961892, 5.83961892, 5.93961892, 6.039618920000001,
6.13961892, 6.23961892, 6.33961892, 6.43961892, 6.539618920000001, 6.63961892,
6.73961892, 6.83961892, 6.93961892]

6939.618920000001ms: R1-> Interface 101.0.0.1 : 20 sample(s)added to list
Elements in bw list: [0.5170064516129033, 0.5170064516129033, 0.5142709677419356,
0.5170064516129033, 0.5170064516129033, 0.5142709677419356, 0.5170064516129033,
0.5170064516129033, 0.5142709677419356, 0.5170064516129033, 0.5170064516129033,
0.5142709677419356, 0.5170064516129033, 0.5170064516129033, 0.5142709677419356,
0.5170064516129033, 0.5170064516129033, 0.5142709677419356, 0.5170064516129033,
0.5170064516129033]
Elements in time list: [5.03961892, 5.13961892, 5.23961892, 5.33961892, 5.43961892,
5.539618920000001, 5.63961892, 5.73961892, 5.83961892, 5.93961892, 6.039618920000001,
6.13961892, 6.23961892, 6.33961892, 6.43961892, 6.539618920000001, 6.63961892,
6.73961892, 6.83961892, 6.93961892]
```

-- THE END --

Test Run Result:

As was expected, the code is able to perform sampling at the specified sampling interval and maintain 2 lists, each containing 20 elements to store the bandwidth utilization ratio and time values that were sampled.

4.6.1.2 Testing the linear regression line equation construction process

For this testing, the same topology as shown in Figure 4.3 was used with the same parameter settings. The purpose of this test is to ensure that the construction of linear regression line equations was performed correctly using Equations (2.18), (2.19) and (2.20).

The Expected Output of the Test Run

- After the first 20 samples have been obtained, the equation of the first linear regression line is constructed, i.e, L_o.
- Once the subsequent 20 samples have been obtained, the equation of the next linear regression line is constructed, i.e, L_c.

Actual Output Trace:

```
                            -----output omitted-----

Computing line 1 for R2-> Interface 1.0.0.2
Equation of Line (Y = mX + b): Y = 2.1882579290778118E-13x + 40.00439999999868

7029.344430000001ms: R2-> Interface 1.0.0.2 21 sample(s) added to list

Elements in bw list: [40.068, 40.068, 39.856, 40.068, 40.068, 39.856, 40.068, 40.068,
39.856, 40.068, 40.068, 39.856, 40.068, 40.068, 39.856, 40.068,40.068, 39.856, 40.068,
40.068]
Elements in time list: [5.02934443, 5.12934443, 5.22934443, 5.32934443, 5.42934443,
5.52934443, 5.62934443, 5.72934443, 5.82934443, 5.92934443, 6.02934443, 6.12934443,
6.22934443, 6.32934443, 6.42934443, 6.52934443, 6.62934443, 6.72934443, 6.82934443,
6.92934443]

Computing line 1 for R2-> Interface 102.0.0.1
Equation of Line (Y = mX + b): Y = -0.32233461072036274x + 2.4230193366263744

7029.344430000001ms: R2-> Interface 102.0.0.1 21 sample(s) added to list

Elements  in  bw  list:  [0.6401032258064516,  0.7221677419354838,  0.5771870967741936,
0.6401032258064516,   0.6428387096774193,   0.6510451612903226,   0.6291612903225806,
0.6838709677419355,    0.6318967741935484,    0.689341935483871,    0.752258064516129,
0.752258064516129,     0.6018064516129031,     0.32825806451612904,0.19421935483870967,
0.17233548387096775,  0.13950967741935483,  0.1258322580645161 2,  0.10941935483870968,
0.22978064516129032]
Elements  in  time  list:  [5.02934443,  5.12934443,  5.22934443,  5.32934443,  5.42934443,
5.52934443,  5.62934443,  5.72934443,  5.82934443,  5.92934443,  6.02934443,  6.12934443,
6.22934443,  6.32934443,  6.42934443,  6.52934443,  6.62934443,  6.72934443,  6.82934443,
6.92934443]

Computing line 1 for R1-> Interface 1.0.0.1
Equation of Line (Y = mX + b): Y = -25.12757894736917x + 188.87662227695608

7039.618920000001ms: R1-> Interface 1.0.0.1 21 sample(s) added to list

Elements  in  bw  list:  [52.788000000000004,  54.272,  43.248,  50.032,  50.668,  49.184,
49.608000000000004,  52.364,  48.76,  55.544000000000004,  59.36,  55.120000000000005,
50.456, 21.412, 11.236, 16.112000000000002, 9.964, 8.692, 7.632, 20.988]
Elements  in  time  list:  [5.03961892,  5.13961892,  5.23961892,  5.33961892,  5.43961892,
5.539618920000001,  5.63961892,  5.73961892,  5.83961892,  5.93961892,  6.039618920000001,
6.13961892,    6.23961892,    6.33961892,    6.43961892,    6.539618920000001,    6.63961892,
6.73961892, 6.83961892, 6.93961892]
```

Equation of line *Lo* correctly constructed

```
Computing line 1 for R1-> Interface 101.0.0.1
Equation of Line (Y = mX + b): Y = -3.4191530141840344E-15x + 0.5161858064516335

7039.618920000001ms: R1-> Interface 101.0.0.1 21 sample(s) added to list

Elements  in  bw  list:  [0.5170064516129033,  0.5170064516129033,  0.5142709677419356,
0.5170064516129033,    0.5170064516129033,    0.5142709677419356,    0.5170064516129033,
0.5170064516129033,    0.5142709677419356,    0.5170064516129033,    0.5170064516129033,
0.5142709677419356,    0.5170064516129033,    0.5170064516129033,    0.5142709677419356,
0.5170064516129033, 0.5170064516129033, 0.5142709677419356,   0.5170064516129033,
0.5170064516129033]
Elements  in  time  list:  [5.03961892,  5.13961892,  5.23961892,  5.33961892,  5.43961892,
5.539618920000001,  5.63961892,  5.73961892,  5.83961892,  5.93961892,  6.039618920000001,
6.13961892,    6.23961892,    6.33961892,    6.43961892,    6.539618920000001,    6.63961892,
6.73961892, 6.83961892, 6.93961892]

Computing line 2 for R2-> Interface 1.0.0.2
Equation of Line (Y = mX + b): Y = 0.019127819548940102x + 39.8881153967671

7129.344430000001ms: R2-> Interface 1.0.0.2 22 sample(s) added to list
```

-- Continued on the next page --

-- Continued from previous page --

```
Elements in bw list: [40.068, 39.856, 40.068, 40.068, 39.856, 40.068, 40.068, 39.856,
40.068, 40.068, 39.856, 40.068, 40.068, 39.856, 40.068, 40.068, 39.856, 40.068,
40.068, 40.068]
Elements in time list: [5.12934443, 5.22934443, 5.32934443, 5.42934443, 5.52934443,
5.62934443, 5.72934443, 5.82934443, 5.92934443, 6.02934443, 6.12934443, 6.22934443,
6.32934443, 6.42934443, 6.52934443, 6.62934443, 6.72934443, 6.82934443, 6.92934443,
7.02934443]

Computing line 2 for R2-> Interface 102.0.0.1
Equation of Line (Y = mX + b): Y = -0.33673189425176847x + 2.522536262497032

7129.344430000001ms: R2-> Interface 102.0.0.1 22 sample(s) added to list

Elements in bw list: [0.7221677419354838, 0.5771870967741936, 0.6401032258064516,
0.6428387096774193, 0.6510451612903226, 0.6291612903225806, 0.6838709677419355,
0.6318967741935484, 0.689341935483871, 0.752258064516129, 0.752258064516129,
0.6018064516129031, 0.32825806451612904, 0.19421935483870967, 0.17233548387096775,
0.13950967741935483, 0.12583225806451612, 0.10941935483870968, 0.22978064516129032,
0.23525161290322583]
Elements in time list: [5.12934443, 5.22934443, 5.32934443, 5.42934443, 5.52934443,
5.62934443, 5.72934443, 5.82934443, 5.92934443, 6.02934443, 6.12934443, 6.22934443,
6.32934443, 6.42934443, 6.52934443, 6.62934443, 6.72934443, 6.82934443, 6.92934443,
7.02934443]

Computing line 2 for R1-> Interface 1.0.0.1
Equation of Line (Y = mX + b): Y = -26.182796992482427x + 195.92845594394012

7139.618920000001ms: R1-> Interface 1.0.0.1 22 sample(s) added to list

Elements in bw list: [54.272, 43.248, 50.032, 50.668, 49.184, 9.608000000000004,
52.364, 48.76, 55.544000000000004, 59.36, 5.120000000000005, 50.456, 21.412, 11.236,
16.112000000000002, 9.964, 8.692, 7.632, 20.988, 15.052]
Elements in time list: [5.13961892, 5.23961892, 5.33961892, 5.43961892,
5.539618920000001, 5.63961892, 5.73961892, 5.83961892, 5.93961892, 6.039618920000001,
6.13961892, 6.23961892, 6.33961892, 6.43961892, 6.539618920000001, 6.63961892,
6.73961892, 6.83961892, 6.93961892, 7.039618920000001]
```

Equation of line *Lc* correctly constructed

```
Computing line 2 for R1-> Interface 101.0.0.1
Equation of Line (Y = mX + b): Y = -1.4397283531711227E-4x + 0.516925771959977

7139.618920000001ms: R1-> Interface 101.0.0.1 22 sample(s) added to list

Elements in bw list: [0.5170064516129033, 0.5142709677419356, 0.5170064516129033,
0.5170064516129033, 0.5142709677419356, 0.5170064516129033, 0.5170064516129033,
0.5142709677419356, 0.5170064516129033, 0.5170064516129033, 0.5142709677419356,
0.5170064516129033, 0.5170064516129033, 0.5142709677419356, 0.5170064516129033,
0.5170064516129033, 0.5142709677419356, 0.5170064516129033, 0.5170064516129033,
0.5142709677419356]
Elements in time list: [5.13961892, 5.23961892, 5.33961892, 5.43961892,
5.539618920000001, 5.63961892, 5.73961892, 5.83961892, 5.93961892, 6.039618920000001,
6.13961892, 6.23961892, 6.33961892, 6.43961892, 6.539618920000001, 6.63961892,
6.73961892, 6.83961892, 6.93961892, 7.039618920000001]

Computing line 3 for R2-> Interface 1.0.0.2
Equation of Line (Y = mX + b): Y = 0.007969924812341026x + 39.944551089503335

                    ----output omitted---
```

-- THE END --

Test Run Result:

The process of constructing the linear regression line equations were correctly invoked at the appropriate time, that is, for every 20 samples obtained. The construction of two linear regression lines and their respective line equations can be seen on the sample trace.

4.6.1.3 Testing the link state advertisement process

Using the same topology shown in Figure 4.3 and the respective parameter settings, the following test was conducted to ensure that an LSA is triggered each time the absolute tangent difference between two linear regression lines exceed the pre-defined threshold. For the testing purpose, the threshold value is set to 5.0.

The Expected Output of the Test Run

Whenever two linear regression line equations have been computed, the absolute difference of their tangent values should be computed and compared with the threshold value to decide whether or not an LSA should be distributed.

Actual Output Trace:

```
7129.344430000001ms: R2-> Interface 1.0.0.2 22 sample(s) added to list
Elements in bw list: [40.068, 39.856, 40.068, 40.068, 39.856, 40.068, 40.068, 39.856,
40.068, 40.068, 39.856, 40.068, 40.068, 39.856, 40.068, 40.068,39.856, 40.068, 40.068,
40.068]
Elements in time list: [5.12934443, 5.22934443, 5.32934443, 5.42934443, 5.52934443,
5.62934443, 5.72934443, 5.82934443, 5.92934443, 6.02934443, 6.12934443, 6.22934443,
6.32934443, 6.42934443, 6.52934443, 6.62934443, 6.72934443, 6.82934443, 6.92934443,
7.02934443]
```

Tangent of two lines available for comparison

```
Comparing tangent of two lines

Tangent of line 1 = 2.1882579290778118E-13 Tangent of line 2 = 0.019127819548940102
Tangent Difference = 0.019127819548721278

Difference < Threshold => DON'T send LSA
```

No significant difference, so no LSA sent

```
Computing line 2 for R1-> Interface 1.0.0.1

7139.618920000001ms: R1-> Interface 1.0.0.1 22 sample(s) added to list
Elements in bw list: [54.272, 43.248, 50.032, 50.668, 49.184, 49.608000000000004,
52.364, 48.76, 55.544000000000004, 59.36, 55.120000000000005, 50.456, 21.412, 11.236,
16.112000000000002, 9.964, 8.692, 7.632, 20.988, 15.052]
Elements in time list: [5.13961892, 5.23961892, 5.33961892, 5.43961892,
5.539618920000001, 5.63961892, 5.73961892, 5.83961892, 5.93961892, 6.039618920000001,
6.13961892, 6.23961892, 6.33961892, 6.43961892, 6.539618920000001, 6.63961892,
6.73961892, 6.83961892, 6.93961892, 7.039618920000001]
```

-- Continued on the next page --

-- *Continued from previous page* --

```
Comparing tangent of two lines

Tangent of line 1 = -25.12757894736917 Tangent of line 2 = -26.182796992482427
Tangent Difference = 1.0552180451132571

Difference < Threshold => DON'T send LSA

                                        ---- Output omitted ---


Computing line 10 for R1-> Interface 1.0.0.1

7939.618920000001ms: R1-> Interface 1.0.0.1 30 sample(s) added to list
Elements in bw list: [55.544000000000004, 59.36, 55.120000000000005, 50.456, 21.412,
11.236, 16.112000000000002, 9.964, 8.692, 7.632, 20.988, 15.052,9.751999999999999,
14.628, 17.384, 16.112000000000002, 3.604, 10.388, 12.931999999999999, 40.068]
```

-- *Continued from previuos page* --

```
                           : [5.93961892, 6.039618920000001, 6.13961892, 6.23961892,
                             6.539618920000001, 6.63961892, 6.73961892, 6.83961892,
6.93961892, 7.039618920000001, 7.13961892, 7.23961892, 7.33961892, 7.43961892,
7.539618920000001, 7.63961892, 7.73961892, 7.83961892]

Comparing tangent of two lines

Tangent of line 1 = -24.06279699248262 Tangent of line 2 = -17.63425563909774
Tangent Difference = 6.42854135338488

Difference > Threshold => send LSA
```

There is a significant difference, so send LSA

-- THE END --

Test Run Result:

The code managed to obtain the absolute difference of tangents and correctly compare the tangent values to make correct decisions on whether an LSA should or should not be sent.

4.6.1.4 Testing the proper working of the routing algorithm

The purpose of conducting this test is to ensure that whenever a link is congested, TE-LR is able to make decision to distribute LSA and appropriate action is taken . For this, the underlying routing algorithm will be tested to see if it can react accordingly and re-route packets according to the link state update invoked by TE-LR. The routing algorithm used for this testing is the *widest-shortest path routing* described in sub-section 2.4.3.

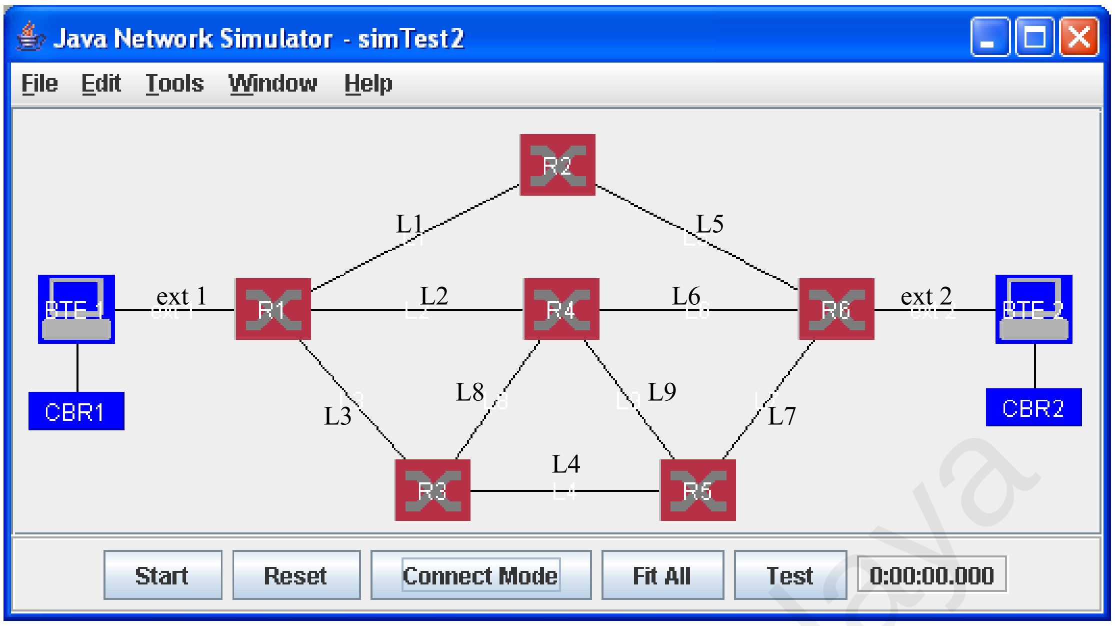


Figure 4.4 Test Topology for Widest-Shortest Path Routing Algorithm

For testing the proper working of TE-LR in terms of invoking the underlying routing algorithm appropriately, the test topology shown in Figure 4.4 was used. There are 6 routers in the topology with R1 and R6 acting as the source and destination respectively. R1 and R6 are connected to one IPBTE each, that is, BTE1 and BTE2 respectively. Each BTE generates one CBR type application traffic.

Looking at the topology given in Figure 4.4, it is evident that there are two possible shortest paths available from R1 to R6, which are R1->R2->R6 through links L1->L5 and R1->R4->R6 through links L2->L6. Initially, the *widest-shortest path* routing algorithm should select path R1->R2->R6 since it has the highest available link bandwidth of 2 Mbps. In the event that the path gets congested, then the algorithm should shift to path R1->R4->R6 even though it has the lowest available link bandwidth (1 Mbps) compared to other paths. The idea behind this algorithm is that whenever there is a path with the shortest length (i.e., specified in terms of either path cost or the actual physical distance) available, it must be selected first without any regards to its available link bandwidth. The parameter settings for all the components used in the topology are given in the following tables:

Table 4.7 IP Settings for all Router Interfaces

Router Name	Interface to Link	Interface IP Address
R1	L1 L2 L3 ext1	1.0.0.1 2.0.0.1 3.0.0.1 101.0.0.1
R2	L1 L5	1.0.0.2 5.0.0.2
R3	L3 L4 L8	3.0.0.3 4.0.0.3 8.0.0.3
R4	L2 L6 L8 L9	2.0.0.4 6.0.0.4 8.0.0.4 9.0.0.4
R5	L4 L7 L9	4.0.0.5 7.0.0.5 9.0.0.5
R6	L5 L6 L7 ext2	5.0.0.6 6.0.0.6 7.0.0.6 102.0.0.1

Table 4.8 Property Settings for all Routers

Router Properties	Settings
V Threshold Type	TE-LR
Threshold_M	5.0
Sampling Interval, *i* (secs)	0.1
Sample Size, *N*	20
Sampling Start Time (secs)	5.0

Table 4.9 Source Address Settings for BTE1 and BTE2

BTE Name	Source Address
BTE1	101.0.0.0/24
BTE2	102.0.0.0/24

Table 4.10 Characteristics of CBR1

CBR Source Properties	Settings
Bit Rate (Mbits/s)	2.0
Bit Rate Lower Bound (Mbits/s)	0.1
Bit Rate Upper Bound (Mbits/s)	3.0
Start Time (secs)	1.0
Number of Mbits to be sent	5.0
Repeat count (-1 = inf)	-1
Delay between calls (usecs)	1.0
DiffServ Class	Best Effort
Random Data Size	TRUE
Random Delay between Calls	TRUE
Enable Starting Delay	TRUE
Random Destination	FALSE
Avoid Local Destination	TRUE
Destination IP	102.0.0.0

Table 4.11 Link Speed Settings for all Links

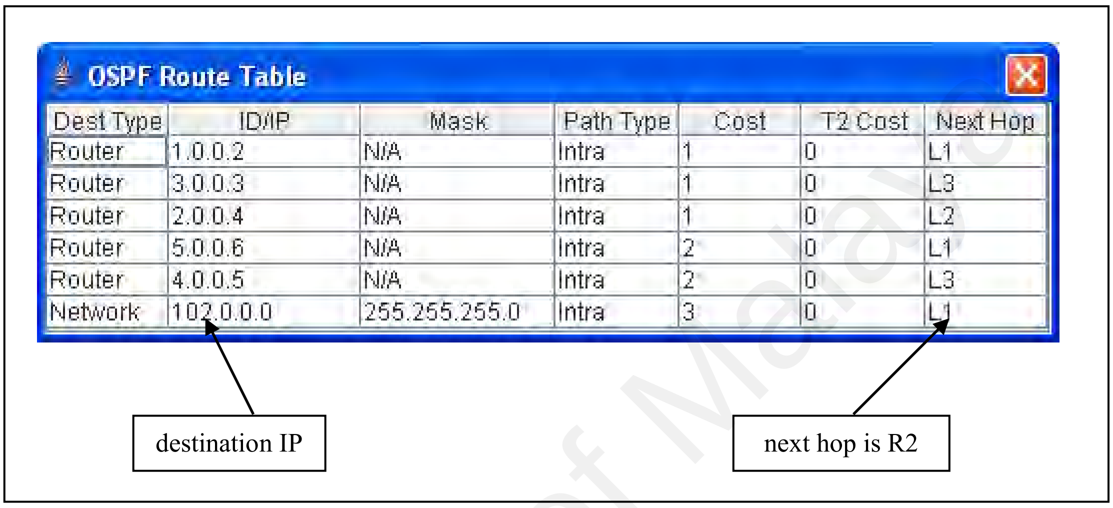
Link Name	Link Speed (MBits/sec)
L1, L3, L4, L5, L7, L8, L9	2.0
L2, L6	1.0
Ext1, ext2	155.0

The Expected Output of the Test Run

- Initially path R1->R2->R6, that is shortest and with maximum available bandwidth of 2 Mbps, should be selected by the *widest-shortest path* algorithm.
- When the path becomes congested, the next shortest path R1->R4->R6 should be selected even though it has lower available bandwidth compare to the other longer paths.

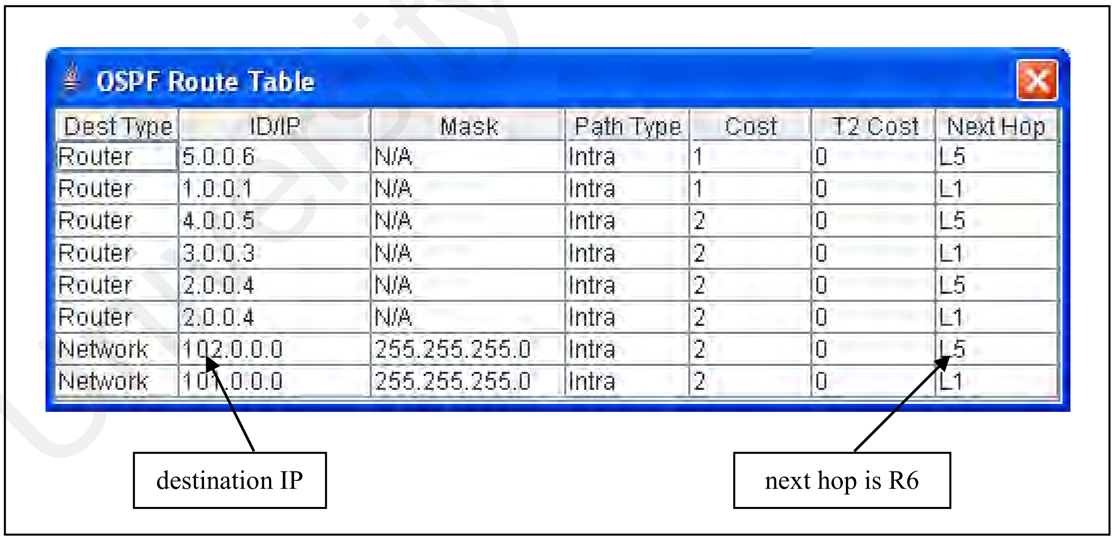
Actual OSPF Routing Table:

The result of the test simulation run is presented as the OSPF routing table produced by R1, R2 and R4 before and after the congestion. This can be seen in Figures 4.5, 4.6, 4.7 and 4.8.



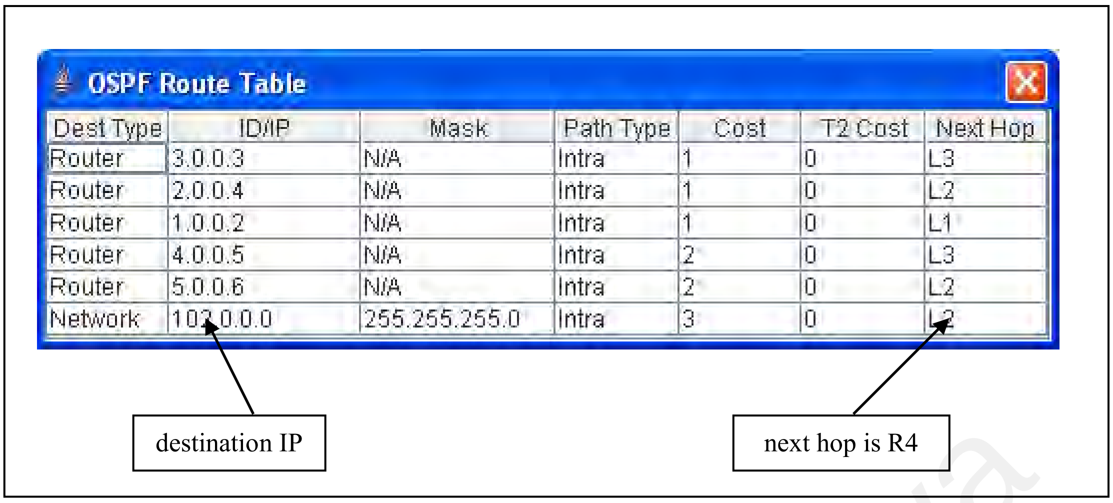
Dest Type	ID/IP	Mask	Path Type	Cost	T2 Cost	Next Hop
Router	1.0.0.2	N/A	Intra	1	0	L1
Router	3.0.0.3	N/A	Intra	1	0	L3
Router	2.0.0.4	N/A	Intra	1	0	L2
Router	5.0.0.6	N/A	Intra	2	0	L1
Router	4.0.0.5	N/A	Intra	2	0	L3
Network	102.0.0.0	255.255.255.0	Intra	3	0	L1

Figure 4.5 OSPF Routing Table for R1 *Before* Congestion



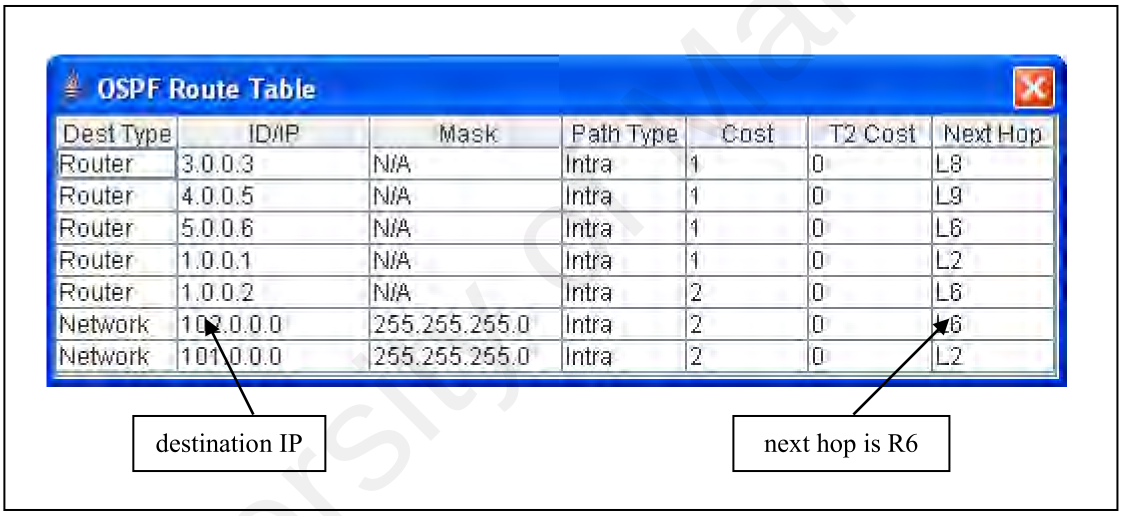
Dest Type	ID/IP	Mask	Path Type	Cost	T2 Cost	Next Hop
Router	5.0.0.6	N/A	Intra	1	0	L5
Router	1.0.0.1	N/A	Intra	1	0	L1
Router	4.0.0.5	N/A	Intra	2	0	L5
Router	3.0.0.3	N/A	Intra	2	0	L1
Router	2.0.0.4	N/A	Intra	2	0	L5
Router	2.0.0.4	N/A	Intra	2	0	L1
Network	102.0.0.0	255.255.255.0	Intra	2	0	L5
Network	101.0.0.0	255.255.255.0	Intra	2	0	L1

Figure 4.6 OSPF Routing Table for R2 *Before* Congestion


OSPF Route Table

Dest Type	ID/IP	Mask	Path Type	Cost	T2 Cost	Next Hop
Router	3.0.0.3	N/A	Intra	1	0	L3
Router	2.0.0.4	N/A	Intra	1	0	L2
Router	1.0.0.2	N/A	Intra	1	0	L1
Router	4.0.0.5	N/A	Intra	2	0	L3
Router	5.0.0.6	N/A	Intra	2	0	L2
Network	102.0.0.0	255.255.255.0	Intra	3	0	L2

Figure 4.7 OSPF Routing Table for R1 *After* Congestion


OSPF Route Table

Dest Type	ID/IP	Mask	Path Type	Cost	T2 Cost	Next Hop
Router	3.0.0.3	N/A	Intra	1	0	L8
Router	4.0.0.5	N/A	Intra	1	0	L9
Router	5.0.0.6	N/A	Intra	1	0	L6
Router	1.0.0.1	N/A	Intra	1	0	L2
Router	1.0.0.2	N/A	Intra	2	0	L6
Network	102.0.0.0	255.255.255.0	Intra	2	0	L6
Network	101.0.0.0	255.255.255.0	Intra	2	0	L2

Figure 4.8 OSPF Routing Table for R4 *After* Congestion

Test Run Result:

Figure 4.5 is the OSPF routing table for R1 populated by the routing algorithm at the beginning of the simulation, that is at time 1.0 second (after allowing the network to converge). The next hop from R1 to reach R6 is given as L1 which indicates the link connecting R1 to R2. In Figure 4.6, the OSPF routing table for R2 is given that shows the next hop to reach R6 from R2 is L5 that connects R2 and R6. After running the simulation for 8 minutes, path R1->R2->R6 became congested, and thus the routing algorithm had to choose an alternative path. Figure 4.7 and 4.8 shows the new path taken by R1. In Figure 4.7, the next hop to reach R6 is given as L2 that connects R1 and R4. From here, R4 has two options to choose, that is either L6 or L9. However, in Figure 4.8, it is obvious that the next hop chosen by R4 is L6 that connects R4 and R6 directly. This is because, taking L6 requires only one hop to reach R6. On the other hand, taking L9 requires two hops. Therefore, it is proven that the algorithm is working correctly and managed to select the most appropriate widest-shortest path. This in turn proves the correct working of TE-LR's link state advertisement operation.

4.7 Chapter Summary

In this chapter, the methodology used to develop the proposed mechanism was discussed. An overview of network simulation concept and a brief review of existing network simulators were presented. This was followed by the details of major components in the chosen network simulator, UMJaNetSim and the changes done to it in order to implement the proposed mechanism, TE-LR. Finally, a number of tests were conducted to evaluate the correctness of TE-LR and the result of the tests were discussed in detail. In the next chapter, a detailed discussion on how the proposed mechanism will be evaluated using the simulation environment created in this chapter will be presented.

CHAPTER 5

SIMULATION RESULTS AND PERFORMANCE ANALYSIS

In this chapter, extensive simulations were done to evaluate the performance of the proposed mechanism, TE-LR. Firstly, a detailed discussion on the simulation environment such as the simulation topology, the parameter settings and the simulation sessions conducted are presented. This is followed by the performance analysis of TE-LR in comparison with QoS-OSPF based on the simulation results obtained. The performance of TE-LR is evaluated using a number of selected performance metrices and these comes later in this chapter.

5.1 Simulation Environment

To evaluate and compare TE-LR mechanism with QoS-OSPF, the topology shown in Figure 5.1 was created and used. The topology depicts the typical ISP topology known as MCI backbone topology which is used in [Apostolopoulos99i] and [Lim04]. The topology is consist of 18 routers, 32 links and 216 traffic sources which are a combination of both CBR and VBR applications. The capacity of the links in the topology are classified into 3 types: *low capacity* (4 Mbps), *medium capacity* (5 Mbps) and *high capacity* (7 Mbps). They have been scaled down from actual capacity values to reduce the simulation time and volume.

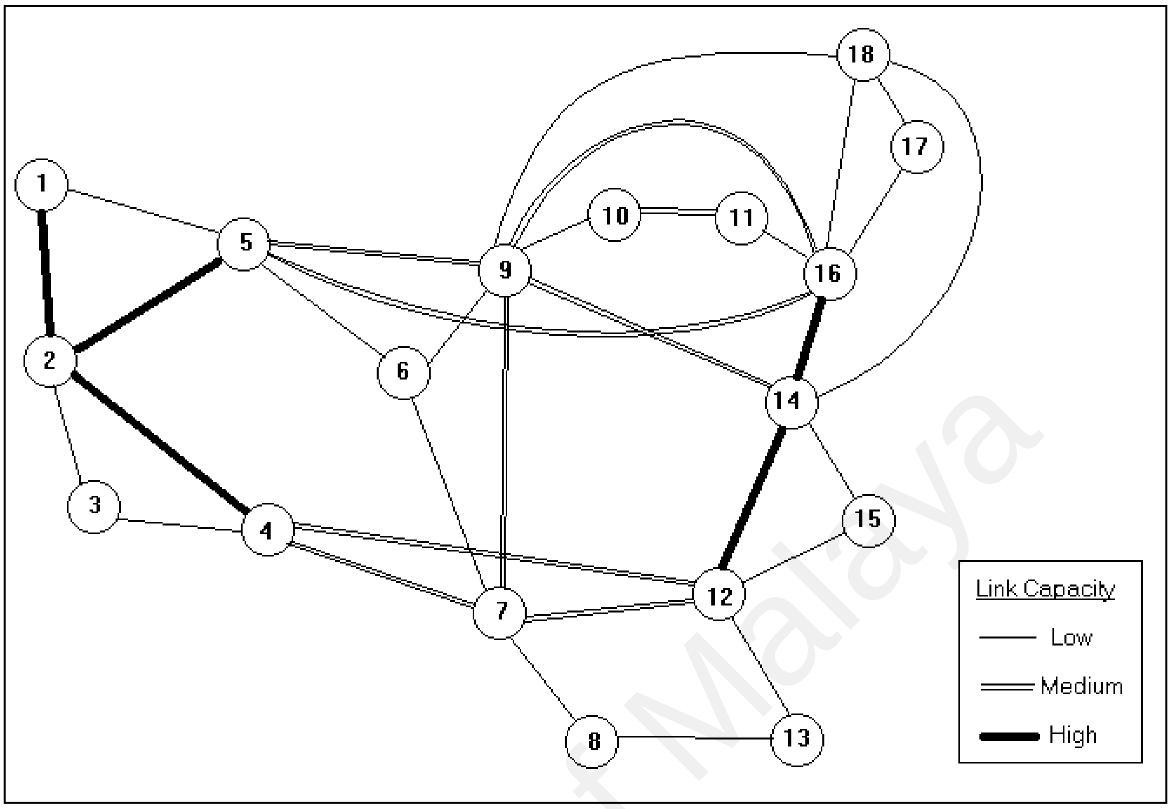


Figure 5.1 Simulation Toplogy

All the assumptions made in Section 3.1 were considered during the preparation of the simulation environment. Each ATMLSR router is connected to a single customer site represented by IPBTE, and each IPBTE is connected to 12 applications that generates both CBR and VBR traffics. The traffic arrival rates of each VBR traffic follow the Poisson model. A total of 10 simulation sessions were executed, each with increasing traffic load, starting with a normalized load of 1. For each subsequent session, the traffic rate and size are increased by 10%. Each simulation session is run for 180 seconds (i.e., 3 minutes or 1.8 x 10^{10} ticks). To allow for convergence of the network traffic, data transmission starts only after 1 second. Table 5.1, 5.2 and 5.3 shows the characteristics of the routers and traffic sources used in this simulation.

Table 5.1 Property Settings for all Routers

Router Properties	Settings
V Threshold Type	TE-LR
Threshold_M (%)	10.0
Sampling Interval, *i* (secs)	0.1
Sample Size, *N*	20
Sampling Start Time (secs)	5.0

Table 5.2 Characteristics of VBR Traffic Sources

VBR Source Properties	Settings
Bit Rate (Mbits/s)	0.8
Mean Burst Length (usecs)	50000
Mean Interval Bet. Bursts (usecs)	15000
Start Time (secs)	1.0
Number of Mbits to be sent	2.0
Repeat count (-1 = inf)	-1
Delay between calls (usecs)	3.0
DiffServ Class	Best Effort
Poisson Model	TRUE
Random Data Size	TRUE
Random Delay Bet. Calls	TRUE
Enable Starting Delay	TRUE
Random Destination	TRUE
Avoid Local Destination	TRUE

Table 5.3 Characteristics of CBR Traffic Sources

CBR Source Properties	Settings
Bit Rate (Mbits/s)	0.8
Bit Rate Lower Bound (Mbits/s)	0.1
Bit Rate Upper Bound (Mbits/s)	1.0
Start Time (secs)	1.0
Number of Mbits to be sent	2.0
Repeat count (-1 = inf)	-1
Delay between calls (usecs)	3.0
DiffServ Class	Best Effort
Random Data Size	TRUE
Random Delay between Calls	TRUE
Enable Starting Delay	TRUE
Random Destination	TRUE
Avoid Local Destination	TRUE

5.2 Performance Analysis

The performance of TE-LR was investigated using the performance metrices described in the next sub-sections.

5.2.1 Packet Loss Ratio

Figure 5.2 shows the packet loss ratio of TE-LR compared to QoS-OSPF. Packet loss ratio is a parameter used to get an idea of the level of congestion in the network. Indirectly, it provides some useful information about the throughput of the evaluated routing mechanisms. For instance, higher packet loss ratio indicates lower throughput and vice versa. In this thesis, the packet loss ratio is calculated as a weighted average packet loss ratio amongst all the routers in the network. The weight used here is the total number of packets received by each router. Hence, packet loss ratio can be defined as:

$$packet\ \ loss\ \ ratio\ \ \frac{total\ number\ of\ packets\ dropped}{total\ number\ of\ packets\ received}\ \text{x } 100\% \qquad (5.1)$$

From Figure 5.2, it can be observed that the packet loss ratio of both mechanisms, TE-LR and QoS-OSPF are comparable. Both mechanisms exhibit moderate increment in packet loss as the network load increases. As the network load grows larger, TE-LR experiences a slightly higher packet loss compared to QoS-OSPF. As shown in Table 5.4 and Figure 5.2, it can be seen that TE-LR experiences only 3% to 8% more packet loss compared to QoS-OSPF. This outcome is anticipated due to the bursty nature of the VBR traffics. VBR traffics are intended to emulate web traffics whose transmission bit rate is variable. Therefore, when the flow arrival rate is high (i.e.,the interval between successive flow arrivals are short), the link utilization becomes very high causing frequent link state advertisements (i.e., the construction of linear regression line equation and comparison of tangent difference appear fast). Conversely, when the flow arrival rate is very low, the link utilization will also be very low causing delays between successive link state advertisements (i.e., the construction of linear regression line equation and comparison of tangent difference appear late). This delay leads to a delay in conveying the network condition message to all the routers and hence packet loss may be experienced before the routers decide to select a new path.

Table 5.4 Packet Loss Ratio (%)

	Normalized Load									
	1	2	3	4	5	6	7	8	9	10
QoS-OSPF	0.064	0.224	0.395	0.652	0.940	1.480	1.904	2.487	3.129	3.559
TE-LR	0.066	0.254	0.414	0.699	0.983	1.701	2.175	2.904	3.024	3.866

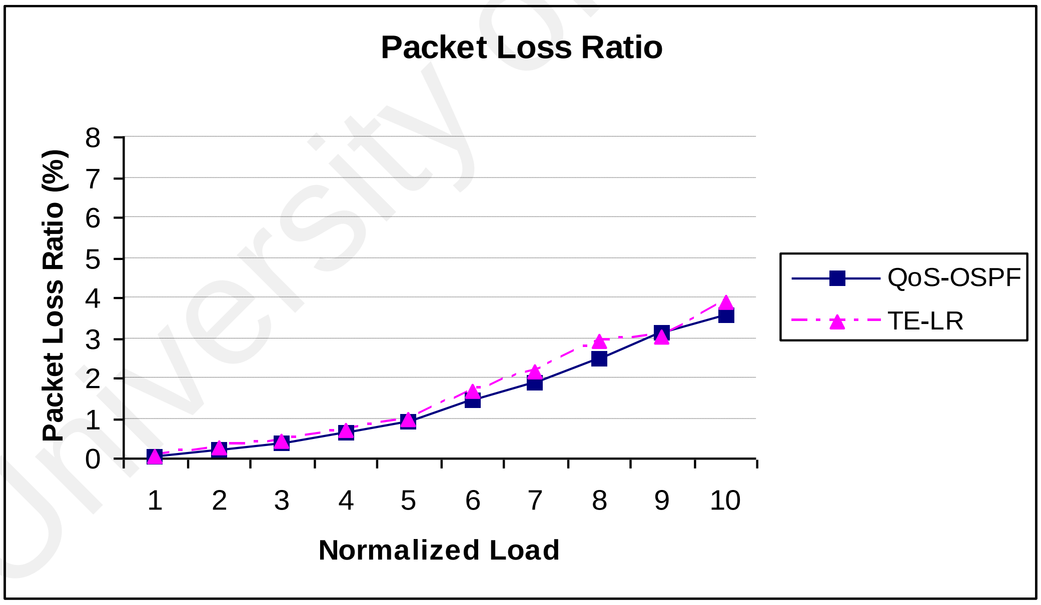


Figure 5.2 Packet Loss Ratio

In contrast, QoS-OSPF slightly outperforms TE-LR due to the fact that for every instantaneous change in a link's banwidth utilization, link state advertisements are triggered giving a better picture of the networks congestion condition. However, QoS-

OSPF may suffer from unnecessary link state advertisements caused by sudden spikes in link utilizations creating more overheads. This is because QoS-OSPF is quite sensitive to changes in the link state and according to [Apostolopoulos99ii], this sensitivity leads to frequent triggering of link state advertisements. TE-LR, on the other hand handles these sudden spikes by observing the link state trend before deciding to send link state advertisements.

5.2.2 Link Utilization

Measuring the utilization of a link gives some idea about the network resources consumption which is crucial for successful traffic engineering. Link utilization can be measured as either average link utilization or peak link utilization. In this thesis, both of these have been used as each of them provides different information about the network resource utilization. Average link utilization provide information about the overall network throughput. For instance, higher throughput is generally an implication of higher average link utilization. The average link utilization is calculated by measuring the longterm link utilization at each outgoing interface of a router (assume every link is bi-directional) separately for every router in the network. It is defined as:

$$\textit{average link utilization} = \frac{\sum_{i=1}^{n} \textit{longterm link utilization of interface } i \times \textit{capacity of link } i}{\sum_{i=1}^{n} \textit{capacity of link } i} \qquad (5.2)$$

Assume that there are *n* router interfaces in the network, then the longterm link utilization of each interface is calculated as:

$$\textit{longterm link utilization} = \frac{\textit{total cells scheduled}}{\textit{total simulation time} \times \textit{link capacity}} \times 100\% \qquad (5.3)$$

In contrast, the peak link utilization gives an indication of how evenly had the network load been distributed among all the network links. The lower the peak link utilization the better the load distribution is. The peak link utilization is actually the highest longterm link utilization measured among all the interfaces using Equation (5.3).

Table 5.5 Average Link Utilization

	Normalized Load									
	1	2	3	4	5	6	7	8	9	10
QoS-OSPF	33.607	36.572	39.449	41.700	44.901	47.281	49.313	51.426	53.287	55.567
TE-LR	33.301	36.262	39.132	41.393	44.507	46.654	48.592	50.582	53.035	54.787

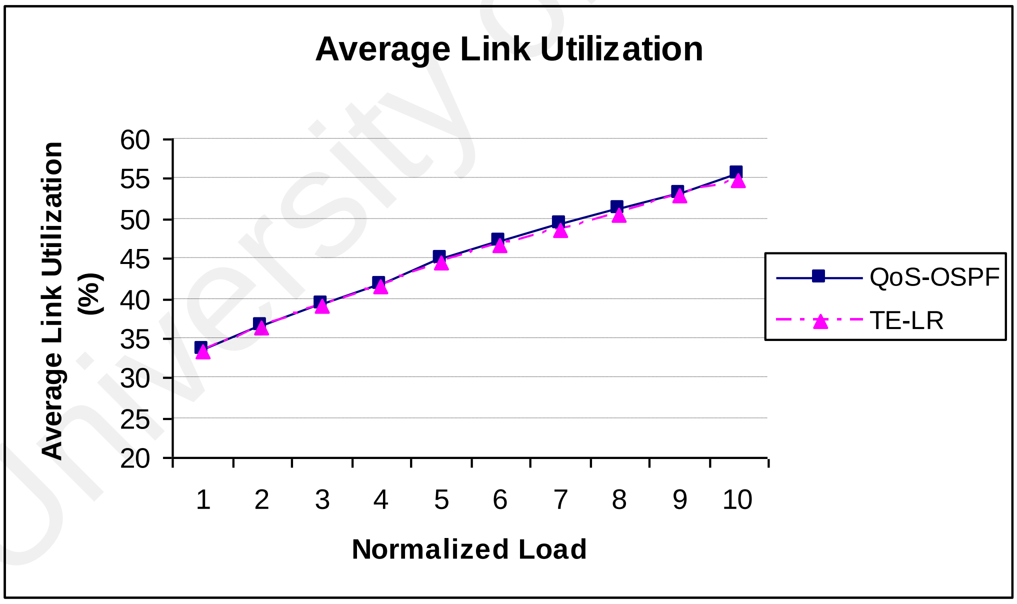


Figure 5.3 Average Link Utilization

Figure 5.3 and Figure 5.4 shows the average link utilization and peak link utilization measured for both TE-LR and QoS-OSPF. For both measurement, these two mechanisms exhibit the same behaviour and they are comparable. In terms of average link utilization, it can be observed that QoS-OSPF slightly outperforms TE-LR. As mentioned earlier in Section 5.2.1, TE-LR experiences slight delay in advertising link state advertisements due to the variable nature of VBR traffics. This causes the link utilization to be slightly lower than QoS-OSPF. However, it can be seen that the difference observed is very tiny and insignificant (i.e., the largest difference observed is when the load is 8, which is around 1.6%), therefore this can be ignored. In terms of peak link utilization, again there is no significant difference between TE-LR and QoS-OSPF. As the network load increases, both mechanisms show a comparable performance in achieving almost 100% load balancing, thus making the network more stable. Overall, TE-LR has exhibited a slightly better load balancing compared to QoS-OSPF for all loads except when the load is 3. This has been achieved without incurring additional cost.

Table 5.6 Peak Link Utilization

	Normalized Load									
	1	2	3	4	5	6	7	8	9	10
QoS-OSPF	83.981	88.557	91.616	93.130	94.627	96.714	96.453	96.624	97.889	98.884
TE-LR	81.781	87.605	92.052	92.971	94.162	95.913	96.256	96.210	97.399	98.145

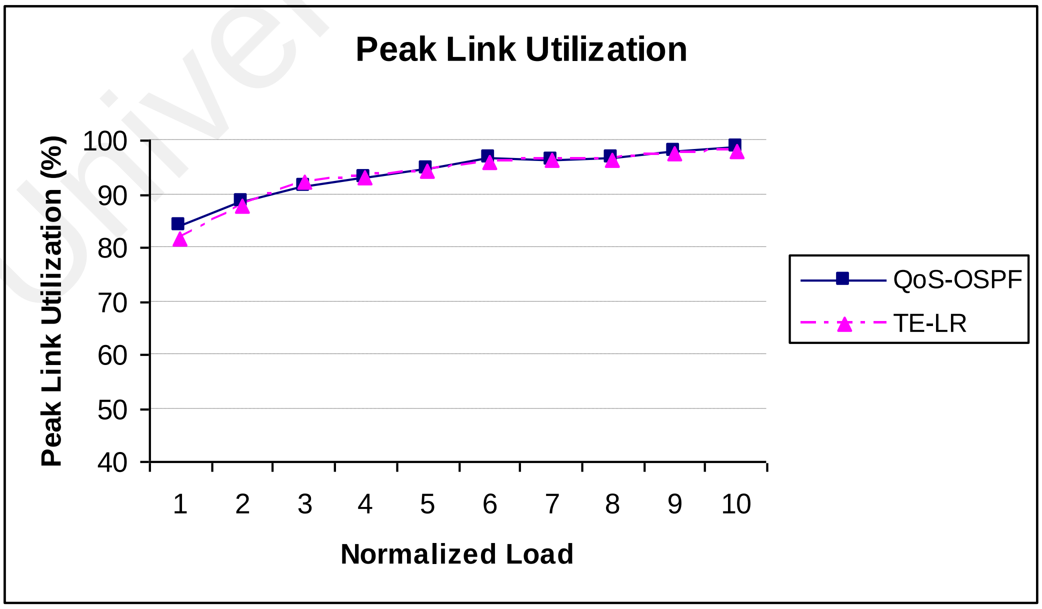


Figure 5.4 Peak Link Utilization

5.2.3 Packet Commit Ratio

The packet commit ratio parameter can be used to analyse the percentage of packets that have been successfully received by the destination compared to the number of packets that have been transmitted by the source. It provides only an approximate indication about the per source packet loss ratio rather than an accurate measurement since some packets may still be in transmission at the time of the measurement. Packet commit ratio is defined as:

$$\textit{packet commit ratio} = \frac{\textit{total delivered cells}}{\textit{total sent cells}} \text{ x } 100\% \qquad (5.4)$$

In most cases, performing multiple simulation sessions under the same network load requires exactly the same traffic source characteristics (i.e., the same transmission rate) to be used for each session. This allows the network throughput to be analysed given the data about the packet commit ratio since network throughput and packet commit ratio are approximately proportional to each other. This relationship is shown below:

$$\begin{aligned}\textit{packet commit ratio} &= \frac{\textit{total delivered cells}}{\textit{total sent cells}} \text{ x } 100\% \\ &= \frac{\textit{total delivered cells}}{\textit{transmission rate} \text{ x } \textit{total time}} \text{ x } 100\% \\ &= \frac{\textit{total delivered cells}}{\textit{total time}} \text{ x } \frac{100\%}{\textit{transmission rate}} \\ &= \textit{throughput} \text{ x } \textit{constant} \qquad (5.5)\end{aligned}$$

$\therefore$ **p**acket commit ratio $\propto$ throughput

In this section, the packet commit ratio of both TE-LR and QoS-OSPF is evaluated and the results of the simulations is presented in Figure 5.5. Since packet commit ratio indicates the total number of cells successfully received by the receiver, it can be seen as an inverse to packet loss ratio. Higher packet loss ratio denotes lower packet commit ratio. This result is anticipated as the packet loss experienced by TE-LR is unavoidable. However, both TE-LR and QoS-OSPF still demonstrate a comparable performance and a steady degradation of performance can be observed as the network load increases.

Table 5.7 Packet Commit Ratio (%)

	Normalized Load									
	1	2	3	4	5	6	7	8	9	10
QoS-OSPF	99.805	99.303	98.750	98.009	97.158	95.438	94.262	92.462	90.738	89.443
TE-LR	99.796	99.209	98.704	97.864	96.998	94.784	93.391	91.216	90.997	88.560

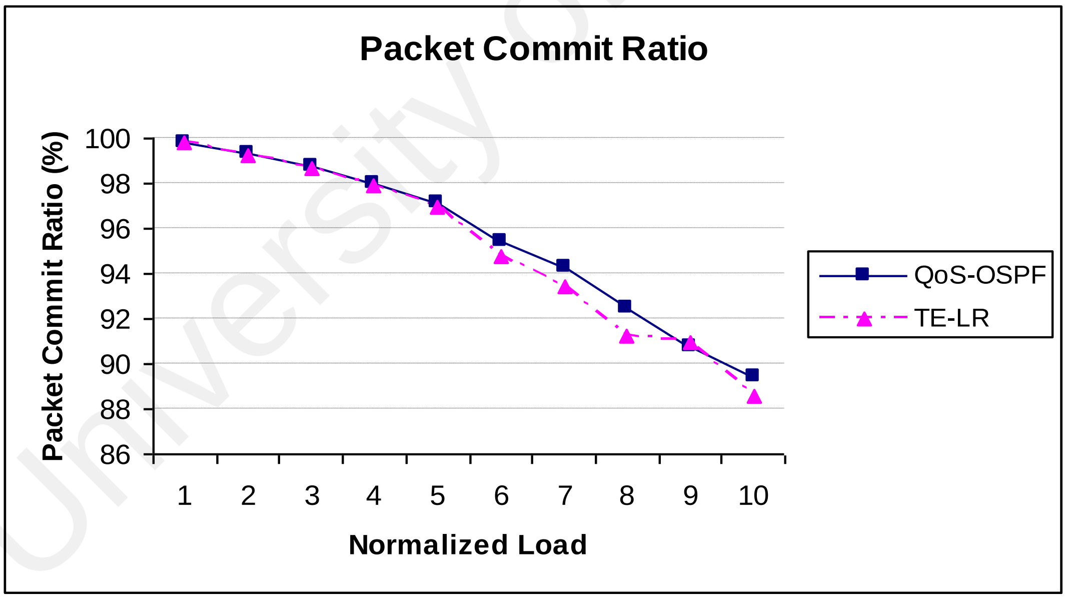


Figure 5.5 Packet Commit Ratio

5.2.4 Update Message Overhead

Packets being transmitted in a network do not necessarily be data packets. They may be comprised of other packets such as Hello packets and acknowledgement packets carrying control messages vital for the correct operations of the network. The additional packets may contribute to increasing load in the network placing heavier burden on the network administration and control. One of the most important control packets necessary to ensure the correct workings of the underlying routing algorithm is the link state update messages. Inappropiate distribution of these messages may cause additional overheads to the network leading to higher processing costs. Hence, the objective of measuring the update message overhead in this section is to identify which mechanism produces excessive message overheads and incur higher processing costs. The update message overhead is calculated as:

$$\textit{update message overhead} = \frac{\textit{total LSA generated}}{\textit{total time (seconds)}} \tag{5.6}$$

The main objective of the proposed link state update mechanism, TE-LR is to reduce both communication overhead and processing cost by suppressing the total amount of update messages produced. Looking at Figure 5.6, it is apparent that TE-LR had successfully achieved its objective. Compared to QoS-OSPF, TE-LR had drastically reduced the update message overhead up to 78% (load 1). The difference here is very significant although both mechanisms reveal a more constant overhead as the network load increases.

In the case of QoS-OSPF, the link state advertisement initiated for every instant of bandwidth utilization change caused the update message overheads to shoot up so high. The mechanism has also failed to capture the fluctuations in bandwidth utilization provoked by the inconsistent VBR traffics. It is obvious that for all levels of network loads, the update message overhead of QoS-OSPF remains high due to the variations in the traffic source leading to unnecessary advertisements of link state update. In contrast, TE-LR is more insensivitve to fluctuations since it monitors the utilization trend before making any decision to advertise link state updates. Any false positives will be captured and turned into data that provides better information about the overall link utilization. Therefore, it can be concluded that TE-LR has been successful in reducing the overall update message overheads and processing costs with minimal impact on other parameters such as packet loss ratio, packet commit ratio, average link utilization and average end-to-end delay.

Table 5.8 Update Message Overhead

	Normalized Load									
	1	2	3	4	5	6	7	8	9	10
QoS-OSPF	1656.583	1649.297	1632.646	1645.886	1635.680	1631.583	1646.451	1612.343	1629.954	1624.051
TE-LR	361.766	363.914	366.869	369.554	372.509	367.943	368.211	363.646	365.257	368.749

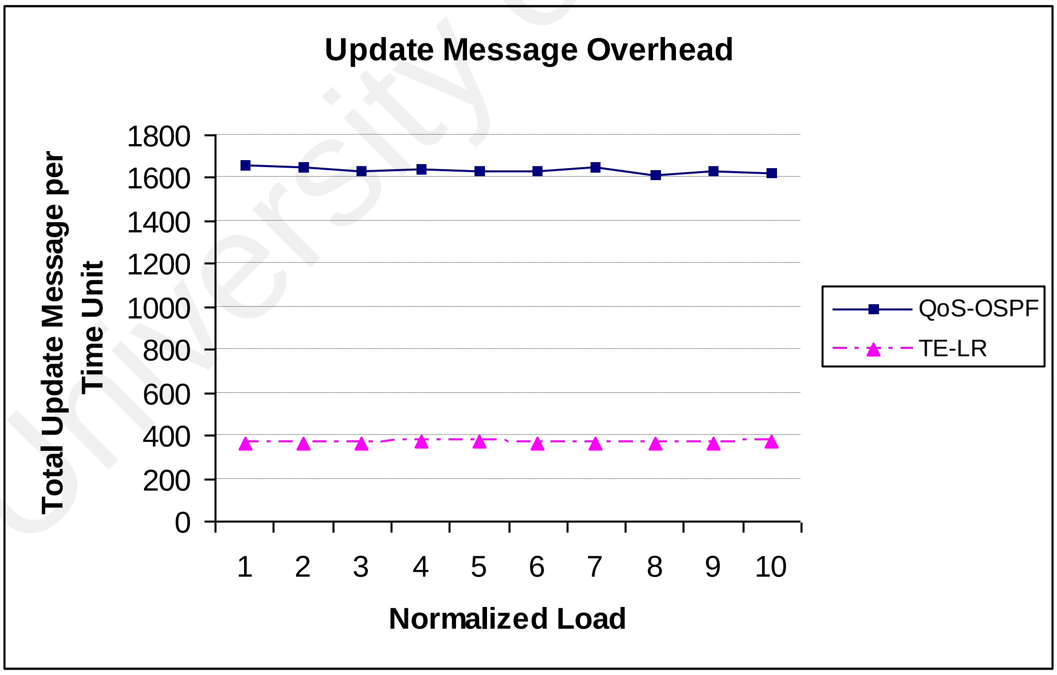


Figure 5.6 Update Message Overhead

5.2.5 End-to-End Delay

The end-to-end delay parameter measures the total time taken by a packet from the time it is inserted into the network for transmission until it is received by the destination. The delay is comprised of queuing delay and propogation delay. Both are usually caused by network congestions. Peak end-to-end delay is taken as the highest end-to-end delay experienced by a packet. Together, both average end-to-end delay and peak end-to-end delay provides an insight into the network congestion condition.

Table 5.9 Average End-to-End Delay (in ms)

	Normalized Load									
	1	2	3	4	5	6	7	8	9	10
QoS-OSPF	2.949	7.370	11.388	14.498	19.246	27.095	31.959	38.886	43.883	47.699
TE-LR	2.901	7.419	10.822	14.912	20.830	29.148	33.925	40.151	42.756	49.555

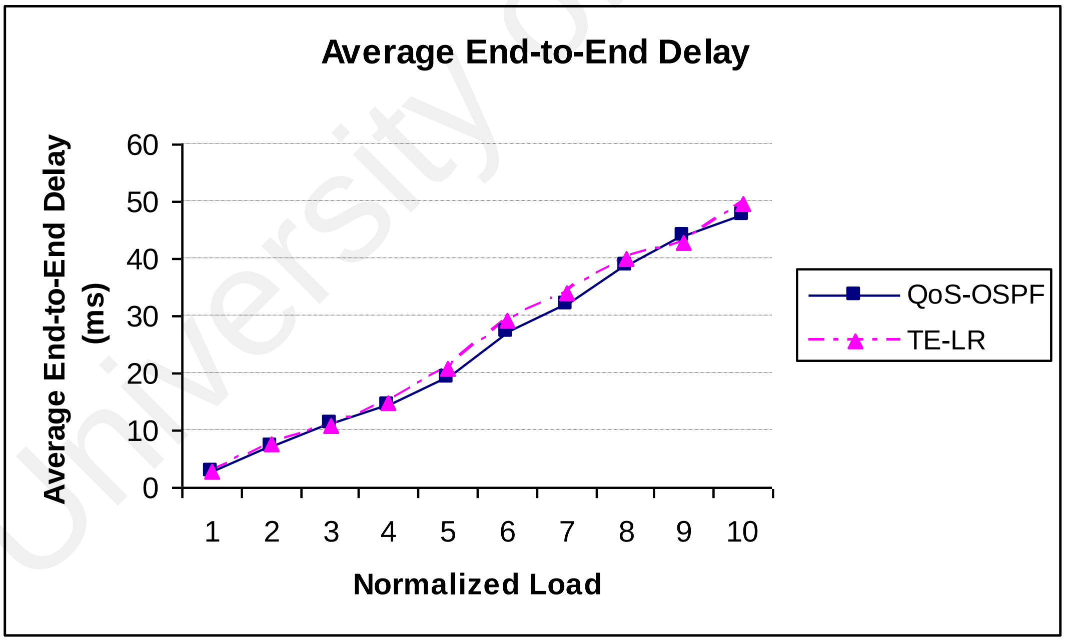


Figure 5.7 Average End-to-End Delay

Looking at Figure 5.7, it can be observed that the average end-to-end delay of both mechanisms are comparable. As discussed earlier, end-to-end delays are comprised of both queuing and propagation delays. Queuing delays in TE-LR is the result of delays in disseminating link state updates immediately when there is a change. Thus, packets had to wait a little longer in the queue before being forwarded. Queuing delays in turn aggravates propagation delays in a network with varying traffic conditions.

To better evaluate the average end-to-end delay and peak end-to-end delay of TE-LR, the packet loss ratio and packet commit percent parameters should be taken into consideration. When the congestion in the network increases, TE-LR had shown a slight increase in packet loss ratio and a slight decrease in packet commit percent. These condition are obviously caused by the end-to-end delays in the network. Hence, it be can concluded that due to these delays TE-LR had resulted in slightly higher packet loss ratio and lower packet commit ratio compared to QoS-OSPF, as demonstrated in Section 5.2.1 and 5.2.3. Similarly, the peak end-to-end delay of both mechanisms experience consistent increment as the network load increases. This is shown in Figure 5.8. However, the increment in delay experienced by TE-LR is very little and does not effect other parts of the routing process, thus this can be ignored.

Table 5.10 Peak End-to-End Delay (in ms)

	Normalized Load									
	1	2	3	4	5	6	7	8	9	10
QOS-OSPF	93.473	124.416	137.862	150.599	162.797	191.190	201.256	218.928	232.493	230.038
TE-LR	89.575	128.102	140.734	152.908	161.970	189.772	207.620	222.684	228.891	237.478

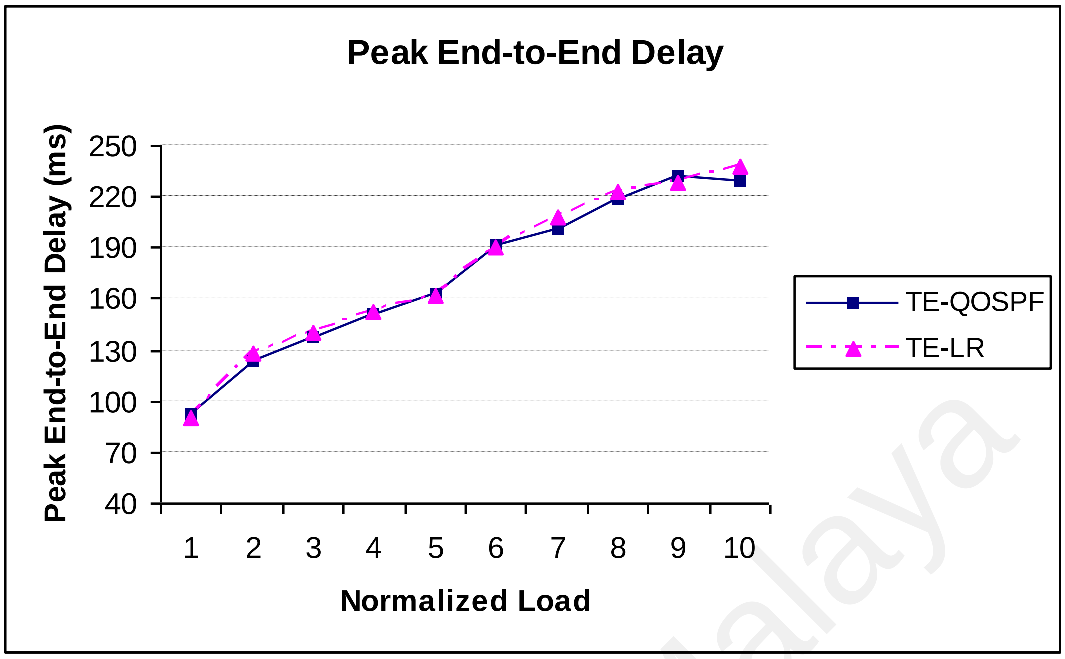


Figure 5.8 Peak End-to-End Delay

5.3 Simulation Conclusion and Discussion

As discussed earlier, the main objective of the proposed mechanism, TE-LR is to reduce the total number of update messages produced by monitoring the bandwidth utilization trend of every link in the network over some period of time. This is in contrast with QoS-OSPF which advetises link state updatse for every instant of bandwidth utilization change causing excessive update overheads. By looking at the simulation results produced by both TE-LR and QoS-OSPF in the previous sub-sections, it is proven that TE-LR had managed to successfully reduced the number of update message overheads thus reducing the communication overheads and processing costs compared to QoS-OSPF. Though the performance of QoS-OSPF in other aspects such as packet loss ratio, packet commit ratio, average link utilization and average end-to-end delay is slightly better than TE-LR, these are extremely insignificant compared to the higher costs saving in terms of processing cost offered by TE-LR. It had also been proven in this section that TE-LR is able to provide a performance comparable to QoS-OSPF under any traffic condition and with the uncertainty brought in by the bursty VBR traffics. Therefore, it can be concluded that the TE-LR is potentially useful in containing the update message overheads in web traffics with uncertainty.

5.4 Chapter Summary

The most important part in running any simulation is to plan and design the simulation environment to be used. This had been covered in the beginning of this chapter. The performance metrices applied to test the proposed mechanism was discussed in detail and a number of simulation sessions was run to obtain the performance criteria of both TE-LR and QoS-OSPF. Finally, the performance of TE-LR was compared with the performance of QoS-OSPF by analyzing the simulation results generated by the planned simulation sessions.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this last chapter, a summary of the thesis contributions is presented followed by some proposals for future enhancements to the proposed mechanism, TE-LR.

6.1 Thesis Contribution

The thesis provides a detailed report of the research work conducted focusing on IP QoS. The report is comprised of a thorough review of related literatures, design of the proposed mechanism, examination of existing network simulators, selection of appropriate network simulator to develop the simulation environment, the implementation and testing of the proposed mechanism, evaluation of both proposed and exisiting link state update mechanisms, and finally a detailed analysis of the simulation results.

In most of the traditional networks such as the Internet, distributing the link state information places heavy burden on the network resources. For the routing algorithm to work correctly, it has to have some degree of precision in the link state information that it receives from neighboring routers. With the diversity of Internet traffics plus constantly varying traffics loads and QoS requirements, the distribution of accurate link state information is almost impossible. Therefore, in Chapter 2, the thesis begun with a thorough investigation of existing solutions proposed in literatures to identify areas of improvements. A number of proposed solutions were carefully studied. In Chapter 3, a mechanism had been proposed that is capable of distributing link state information more efficiently and as precisely as possible, while at the same time reducing communication overheads and processing costs. The mechanism is called **TE-LR** (***T***raffic ***E***ngineering using ***L***inear ***R***egression) whose main plus point is the use of Linear Regression model to observe each link's bandwidth utilization behaviour before initiating link state update distributions. The mechanism monitors the bandwidth utilization ratios of each interface and samples them at regular intervals to construct linear regression line equations for the sampled observations. The constructed lines are then used to decide whether or not link state updates should be advertised. Thus, the major contribution of the proposed mechanism is its ability to suppress the number of update messages sent into the network compared to the current mechanism, QoS-OSPF.

The next most vital contribution of this thesis is the extensions made to the functionalities of UMJaNetSim in Chapter 4. Implementation of TE-LR required some of the functionalities in UMJaNetSim to be modified and extended in order to support the correct working of the mechanism. The existing mechanism, QoS-OSPF performs link state update advertisements entirely based on the instantaneous change of bandwidth utilization and assumes that the link state information exchanged among the routers are accurate. Although it can be argued that the mechanism provides a better picture of the underlying network's congestion condition, it is inevitably true that it increases the protocol overheads and processing costs considerably. Hence, the enhancements done to the simulator allowed a more realistic TE-LR mechanism to co-exist with the existing one to provide alternative solution to the users.

In addition to the above contributions, in Chapter 4, a platform for evaluating the performance of TE-LR was designed and created. The implementation and testing of

TE-LR on the created simulation environment was also featured here. Testing the mechanism on a well-design environment allowed the mechanism's behaviour to be examined and issues related to the actual implementation to be highlighted.

Finally, in Chapter 5, the performance of TE-LR was compared against the performance of QoS-OSPF using the simulation environment developed earlier. The simulation results produced by running a number of simulation sessions for each mechanism were demonstrated using a line graph and they were critically analysed. From the analysis report, it is verified that TE-LR is capable of controlling the unnecessary disseminations of link state update messages without incurring additional computation and processing costs compared to QoS-OSPF.

6.2 Suggestions for Future Research

In this research, the concept of QoS routing and issues related to its deployment had been studied and understood. Problems and solutions identified in literatures were discussed and a mechiasm had been proposed to overcome one of the main problems of QoS provisioning in Internet, that is reducing protocol overheads. There are a number of areas identified for potential future enhancements to the proposed mechanism:

1) One of the most valuable improvements that can be done to TE-LR is the classification of the sampled bandwidth utilization ratios into 3 different classes: *low*, *medium* and *high*. Two thresholds will be defined, upper T_u and lower T_l. When the bandwidth utilization ratio falls into the *high* class, link state updates will be immediately generated and distributed. This is to avoid the link from

getting over congested. When it falls into *medium* class, updates will only be generated and distributed when the bandwidth utilization ratio exceeds T_u thrice successfully. This is to reduce the effect of false positive caused by oscillations. When the bandwidth utilization ratio falls into the *low* class, no updates will be generated until the value falls below T_l. This is to avoid the link from being under-utilized.

2) TE-LR was initially proposed to work in intra-domain environment (single AS) where only routers belonging to the same AS will apply this mechanism. In future, TE-LR will be extended to work across multiple ASs.

REFERENCES

[Ansari04] Ansari,N., Cheng, G., & Krishnan, R.N. (2004). *Efficient and Reliable Link State Information Dissemination*, IEEE Communications Letters, vol.8, no.5, pp.317-319, May 2004.

[Alaettinoglu92] Alaettinoglu, C., Zieger, K.D., Matta, I., Shankar, A.U., & Gudmundsson, O., (1992). *Introducing MaRS, a Routing Testbed*, ACM SIGCOMM Computer Communication Review, vol.22, issue 1, pp.95-96.

[Alghannam01] Alghannam, A.N., Woodward, M.E., & Mellor, J.E. (2001). *Security As A QoS Routing Issues*,in PGNET 2001 Symposium. ***URL:*** *http://www.cms.livjm.ac.uk/pgnet2001/papers/aalghannam.doc*.

[Apostolopoulos99i] Apostolopoulos, G., Guérin, R., Kamat, S., & Tripathi, S.K. (1998). *Quality of Service Based Routing: A Performance Perspective*, in Proceedings of ACM SIGCOMM'98, Vancouver, BC, Canada, August 31- September 4.

[Apostolopoulos99ii] Apostolopoulos, G., Guérin, R., Kamat, S., & Tripathi, S.K. (1999). *Improving QoS Routing Performance Under Inaccurate Link State Information*, in Proceedings of the 16th International Teletraffic Congress (ITC-16), Edinburgh, UK, June 1999.

[Apostolopoulos99iii] Apostolopoulos, G., Guerin, R., Kamat, S., Orda, A., & Tripathi, S.K. (1999). *Intradomain QoS Routing in IP Networks: A Feasibility and Cost Benefit Analysis*, IEEE Network, 13(5):42–54, September/October 1999.

[Apostolopoulos99iv] Apostolopoulos, G., Kamat, S., Williams, D., Guerin, R., Orda, A., & Przygienda, T. (1999). *QoS Routing Mechanism and OSPF Extensions*, RFC 2676, August 1999.

[Apostolopoulos99v] Apostolopoulos, G., Guerin, R., & Kamat, S. (1999). *Implementation and Performance Measurements of QoS Routing Extensions to OSPF*, IEEE INFOCOM '99, New York, vol. 2, pp.680-688, March 21 - 25.

[Armitage00] Armitage, G., (2000). *MPLS: The Magic Behind the* Myths, IEEE Communications Magazine, January 2000.

[Ballew97] Ballew, S.M., (1997). *Managing IP Networks with CISCO Routers*, Sample Chapter 5: Routing Protocol Selection, First Edition, 1-56592-320-0, O'Reilley Online Catalog, October 1997.

URL: *http://www.oreilly.com/catalog/cisco/chapter/ch05.html*.

[Barnett94] Barnett, L., (1994). *NetSim User's Manual*, Technical Report TR-92-01, Department of Mathematics and Computer Science, University of Richmond, June 3.

URL:

http://www.mathcs.richmond.edu/~barnett/techrpt/tr_1992_01.pdf.

[Braden94] Braden, R., Clark, D., & Shenker, S. (1994). *Integrated Services in the Internet Architecture: An Overview*, RFC 1633, June 1994.

[Braden97] Braden, R., Zhang, L., Berson, S., & Herzog, S. (1997). *Resource ReSerVation Protocol (RSVP)*, RFC 2205, September 1997.

[Bruin06] Bruin, X.M., Yannuuzzi, M., Pascal, J.D., et al. (2006). *Research Challenges in QoS Routing*, Computer Communications, vol.29, pp.563-581, 2006.

[Camarillo00] Camarillo, G. (2000). *Influence of Link State Update Algorithms on the Cost of QoS Path Computations*, Advance Signaling Research Laboratory, Finland, 2000.
URL:
http://www.netlab.tkk.fi/tutkimus/ipana/paperit/QoSR/Gonzalo_simulation.pdf.

[Carpenter02] Carpenter, B.E., & Nichols, K. (2002). *Differentiated Services in the Internet*, in Proceedings of the IEEE, vol.90, no.9, September 2002.

[Chang99] Chang, X. (1999). *Network Simulation with OPNET*, Proceedings of the 1999 Winter Simulation Conference, pp.307-314.

[Chang02] Chang, B.J., & Hwang, R.H. (2002). *Distributed Cost-based Update Policies for QoS Routing on Hierarchical Networks,* Technical Report CYUT-CSIE-TR-T-2002-006, Chaoyang University of Technology, 2002.

[Chen03] Chen, S., Ling, Y., & Ting, Y. (2003). *A New Routing Architecture for DiffServ Domains*, in Proceedings of 2nd IASTED International Conference on Communications, Internet and Information Technology (CIIT 2003), Scottsdale, AZ, November 17-19.

[Chen98(i)] Chen, S., & Nahrstedt, K. (1998). *Distributed QoS Routing with Imprecise State Information*, in Proceedings of 7th IEEE International Conference on Computer, Communications and Networks (ICCCN'98), Lafayette, LA, pp. 614-621, October 1998.

[Chen98(ii)] Chen, S., & Nahrstedt, K. (1998). *An Overview of Quality of Service Routing for the Next Generation High-Speed Networks:Problems and Solutions*, IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video, 12(6), pp.64-79.

[Cheng04] Cheng, G., & Ansari, N. (2004). *An Information Theory Based Framework for Optimal Link State Update*, IEEE Communication Letter, vol.8, issue 11, pp.692-694, November 2004.

[Crawley98] Crawley, E., Nair, R., Rajagopalan, B., & Sandick, H. (1998). *A Framework for Qos-based Routing in the Internet*, RFC 2386 August 1998.

[Duan06] Duan, G., & Ma, J. (2006). QoSIP Models User Guide, OPNET Technologies,Inc. ***URL:*** *http://www.opnet.com*.

[Ernst97} Ernst, T. (1997). *Notes About Network Simulators*, INRIA Sophia-Antipolis MISTRAL Team, France, October 30. ***URL:*** *http://www.inrialpes.fr/planete/people/ernst/Documents/simulator.html*.

[Fall03] Fall, K., & Varadhan, K. (2003). *The ns Manual*, by The VINT Project, December 2003.

[Gawlick95] Gawlick, R., Kamath, A., Plotkin, S., & Ramakrishnan,K. (1995). *Routing and Admission Control in General Topology Networks*, Standford Technical Report, STAN-CS-TR-95-1548, Stanford University.

[Ghaoui 97] Ghaoui, L.E., & Lebret, H. (1997). *Robust Solutions to Leastsquares Problems with Uncertain Data*, SIAM Journal on Matrix Analysis and Applications, vol.18, no.4, pp. 1035 1064, October 1997.

[Gojmerac01] Gojmerac, I., Ziegler, T., Ricciato, F., & Reichi, P. (2001). *Adaptive Multipath Routing for Dynamic Traffic Engineering*, in Proceedings of QoSIS 2001-2nd International Workshop on Quality of Future Internet Services, San Francisco, USA.

[Golmie98(i)] Golmie, N., Mouveaux, F., et.al. (1998). *ATM/HFC Network Simulator, Operation and Programming Guide*, version 4.0, U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Gaithersburg, MD 20899, December 1998.

[Golmie98(ii)] Golmie, N., Mouveaux, F., Hester, L., Saintillan, Y., Koenig, A., & Su, D. (1998). *The NIST ATM/HFC Network Simulator: Operation and Programming Guide*, version 4.0, NISTIR5703R2, U.S. Department of Commerce, December 1998.

[Grout04] Grout, V., Davies, J., Hughes, M. & Houlden, N. (2004). *A New Distributed Link-State Routing Protocol with Enhanced Traffic Load Distribution*, in Proceedings of BCS/IEE International Network Conference - INC 2004, University of Plymouth, pp165-172, July 6 - 9.
URL: *http://www.newi.ac.uk/groutv/Papers/NDLSRA.pdf*

[Guerin97(i)] Guerin, R., Orda, A., & Williams, D. (1997). *QoS Routing Mechanism and OSPF Extensions*, Internet Draft, in Proceedings of GLOBECOM 1997, March 1997.

[Guerin97(ii)] Guerin, R., & Orda, A. (1997). *QoS-based Routing in Networks with Inaccurate Information: Theory and Algorithms*, IEEE INFOCOMM '97, April 1997.

[Hou99] Hou, T., Wu, D., Yao, J., Hamada, T., & Taniguchi, T. (1999). *A Differentiated Services Architecture for Multimedia Traffic in IP Networks,* in Proceedings of International Conference on Computer Communication (ICCC'99), vol.1, pp.114-123, September 14-16.

[Huston01] Huston, G. (2001). *Commentary on Inter-Domain Routing in the Internet,* RFC 3221, December 2001.

[Kamei01] Kamei, K., & Kimura, T. (2001). *Evaluation of Routing Algorithms and Network Topologies for MPLS Traffic Engineering*, IEEE Global Telecommunications Conference 2001 (GLOBECOM 01), San Antonio, TX. USA, vol.21, pp. 25-29.

[Keshav88(i)] Keshav*,* S. (1988). *REAL: A Network Simulato*r, Technical Report 88/472, University of California,Berkeley.

[Keshav88(ii)] Keshav, S. (1998). *REAL 5.0 Overview*, Correll University, ***URL:*** *http://www.cs.cornell.edu/skeshav/real*.

[Kowalik02] Kowalik, K., & Collier, M. (2002). *ALCFRA – A Robust Routing Algorithm Which Can Tolerate Imprecise Network State Information*, in Proceedings of 15th ITC Specialists Seminar, Wurzburg, Germany, July 22-24.

[Kowalik03] Kowalik, K., & Collier, M. (2003). *Should QoS Routing Algorithms Prefer Shortest Paths?*, IEEE International Conference on Communications, IEEE-ICC, Anchorage, Alaska, May 2003.

[Kuipers05] Kuipers, F.A., & Mieghem*,* P.V. (2005). *Conditions that Impact the Complexity of QoS Routing*, IEEE/ACM Transactions on Networking, vol.13, no.4, pp.717-730.

[Lekovic01] Lekovic, B., & Mieghem, P.V. (2001). *Link State Update Policies for Quality of Service Routing*. ***URL:*** *www.nas.its.tudelft.nl/publications/2001/Bojan_LinkStateUpdatePolicies.pdf*.

[Lim00] Lim, S.H., Phang, K.K., Yaacob, M.H., & Ling, T.C. (2000). *UMJaNetSim User Manual*, Networking Research Laboratory, Faculty of Computer Science and Information Technology, University of Malaya.

[Lim04] Lim, S.H., Yaacob, M.H, Phang, K.K & Ling, T.C. (2004). *Traffic Engineering Enhancement to QoS-OSPF in DiffServ* *and MPLS Networks*, IEEE Proceedings – Communications, 151(1), pp.101-106.

[Lorenz98]	Lorenz, D.H., & Orda, A. (1998). *QoS Routing in Networks with Uncertain Parameters*, IEEE INFOCOMM '98, March 1998.
[Ma96]	Ma, Q., & Steenkiste, P. (1996). *Routing High-Bandwidth Traffic in Max-Min Fair Share Networks*, Proceedings of ACM SIGCOMM96, Stanford, CA, August 1996.
[Ma97]	Ma, Q., & Steenkiste, P. (1997). *On Path Selection for Traffic with Bandwidth Guarantees*, in Proceedings of IEEE International Conference on Network Protocols, Atlanta, Georgia, October 1997.
[Malkin98]	Malkin, G. (1998). *RIP Version 2*, RFC 2453, November 1998
[Mieghem04]	Mieghem, P.V., & Kuipers, F.A. (2004). *Concepts of Exact QoS Routing Algorithms*, IEEE/ACM Transactions on Networking, vol.12, no.5, pp.851-864.
[Motulsky03]	Motulsky, H., & Christopoulos, A. (2003). *Fitting Models to Biological Data using Linear and Nonlinear Regression: A Practical Guide to Curve Fitting,* GraphPRISM, version 4.0, pp.47-57, April 2003.
[Moy98]	Moy, J. (1998). *OSPF Version 2*, RFC 2328, April 1998
[Nance93]	Nance, R.E. (1993). *A History of Discrete Event Simulation Programming Languages*, in Proceedings of the 2nd ACM SIGPLAN History of Programming Language Conference.

	Reprinted in *ACM SIGPLAN Notices*, 28(3), pp. 149-175, April 1993.
[Nealan04]	Nealan, N. (2004). *An As-Short-As-Possible Introduction to the Least Squares, Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation*, Technical Report, May 2004. ***URL:*** *http://www.nealen.com/projects/.*
[Ohara03]	Ohara, Y., Bhatia, M., Osamu, N., Murai, J. (2003). *Route Flapping Effects on OSPF*, Symposium on Applications and the Internet Workshops (SAINT '03 Workshops), pp. 232, January 2003.
[Oliveira01]	Oliveira, M., & Monteiro, E. (2001). *An Overview of Quality of Service Routing Issues*, in Proceedings of the 5th World Multiconference on Systemics, Cybernetics, and Informatics (SCI 2001) Orlando, USA, June 2001.
[Orda00]	Orda, A., & Sprintson, A. (2000). *QoS Routing: The Precomputation Perspective*, in Proceedings of the Conference on Computer Communications (IEEE INFOCOM '00), March 2000.
[Rekhter95]	Rekhter, Y., & Li, T. (1995). *A Border Gateway Protocol 4 (BGP-4)*, RFC 1771, March 1995.
[Rosen01]	Rosen, E., Viswanathan, A., & Callon, R. (2001). *Multiprotocol Label Switching Architcture*, RFC 3031, January 2001.

[Shaikh98]	Shaikh, A., Rexford, J., & Shin, K. (1998). *Evaluating the Overheads of Source-Directed Quality-of-Service Routing*, International Conference on Network Protocols (ICNP).
[Shen04]	Shen, S., Xiao, G., & Cheng, T.H. (2004). *Evaluating Link State Update Triggers in Wavelength-Routed Networks*, in Proceedings of Asia-Pacific Optical Communications Conference and Exhibition (APOC) 2004, November 7-11.
[Sobrinho02]	Sobrinho, J.L. (2002). *Algebra and Algorithms for QoS Path Computation and Hop-by-Hop Routing in the Internet*, IEEE Transactions on Networking, 10(4):541-550, August 2002.
[Sommerville98]	Sommerville, I. (1998). *Software Engineering*, Fifth Edition, Addison-Wesley, pp.445-459.
[Stallings02]	Stallings, W. (2002). *High Speed Networks and Internets: Performance and Quality of Service*, Second Edition, Prentice Hall.
[Stephenson00]	Stephenson, D.B. (2000). *A Few Words About Modeling Strategy*, September 2. ***URL:*** *http://web.gfi.uib.no/~ngbnk/kurs/notes/node66.html*.
[Waner02]	Waner, S., & Costenoble, S.R. (2002). *On-Line Tutorials for Finite Mathematics, Applied Calculus and, Finite Mathematics and Applied Calculus: Linear Regression*, Department of Mathematics, Hofstra Univeristy, January 2002. ***URL:*** *http://people.hofstra.edu/faculty/Stefan_Waner/RealWorld/tutindex.html*.

[Wang00] Wang, J., Wang, W., Chen, J., & Chen, S. (2000). *A Randomized QoS Routing Algorithm On Networks with Inaccurate Link State Information*, in Proceedings of International Conference on Communication Technology, pp.1617-1622, August 21-25.

[Wang96] Wang, Z., & Crawcroft, J. (1996). *Quality of Service Routing for Supporting Multimedia Applications*, IEEE Journal Selected Area in Communications, 14(7):1228-1234, September 1996.

[Wu03] Wu, W., Misra, A., Das, S.K., & Das, S. (2003). *Scalable QoS Provisioning for Intra-Domain Mobility*, IEEE Global Telecommunications Conference (GLOBECOM '03), vol.7, pp.3615-3619, December 1-5.

[Wu99] Wu, D., Hou, Y.T., Zhang, Z.L., and Chao, H.J. (1999). *On Implementation Architecture for Achieving QoS Provisioning in Integrated Services Networks,* in Proceedings of the IEEE International Conference on Communications (ICC'99), Vancouver, British Columbia, Canada, pp. 461-468, June 1999.

[Xiao02] Xiao, L., Wang, J., & Nahrstedt, K. (2002). *The Enhanced Ticket-Based Routing Algorithms*, in Proceedings of IEEE ICC 2002, New York, NY USA, April 28 – May 2.

[Xiao99] Xiao, X., & Ni, L.M. (1999). *Internet QoS: A Big Picture*, IEEE Network, vol.13, issue 2, pp.8-18, March-April 1999.

[Younis03] Younis, O., & Fahmy, S. (2003). *Constraint-Based Routing in the Internet: Basic Principles and Recent Research*, IEEE

Communications Surveys and Tutorials, vol.5, issue 1, 3rd Quarter 2003.

[Yuan02] Yuan, X., Zheng, W., & Ding, S. (2002). *A Comparative Study of Quality of Service Routing Schemes That Tolerate Imprecise State Information*, the 11th IEEE International Conference on Computer Communications and Networks (IC3N'02), Miami, FL, October 14-16.

[Zhou03] Zhou, H., Pan, J., & Shen, P. (2003). *Cost Adaptive OSPF*, Fifth International Conference on Computational Intelligence and Multimedia Applications (ICCIMA ’03), p.55.