A combined Delay and Throughput proportional scheduling scheme for Differentiated Services

by

Samyukta Sankaran

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Ahmed E. Kamal, Major Professor
Manimaran Govindarasu
Wallapak Tavanapong

Iowa State University

Ames, Iowa

2002

Graduate College
Iowa State University

This is to certify that the master's thesis of

Samyukta Sankaran

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my advisor, Dr. Ahmed E. Kamal, without whom this work would not have been completed. His extraordinary academic guidance, patience and encouragement have been inspiring and invaluable.

I am also grateful to the members of my committee, Dr. G. Manimaran and Dr. W. Tavanapong, for their support and suggestions regarding the technical aspects of my thesis.

On the personal front, I owe thanks to many people who have made my stay in Ames pleasant and productive. To Nandini, my roommate for the majority of my stay in Ames, for all the interesting and challenging times that we have seen each other through. To Murali, for being one of the most sensible and helpful people I know, and for being such a great friend. To Anirban, for always being there. To Jing, for her ability to see things in an often optimistic and always creative light, and for her fun-loving and helpful nature. To Pratima, for her sharp wit and all our analytical conversations together. To Srini, for technical and formatting help with my thesis, and also for his incisive opinions on many subjects, technical and otherwise.

I'd also like to acknowledge the friendship and support of Mamatha Balasubramanian and Sailaja Madineni; and that of Gretchen Jahn, Sharon Ferris, Dr. L. Northup, and Dr. L. Greimann, all from the Civil and Construction Engineering Dept., ISU. Many thanks to Nancy Knight and Pam Myers, for their help with untangling red tape, and for their patience with my endless questions.

I am grateful to all the people at the ECpE Dept., ISU, for having offered me this unique opportunity for personal and professional growth.

ABSTRACT

Traditionally, the Internet has provided the same level of service (best effort service) to all users. Newer applications, however, have wider and diverse Quality of Service (QoS) needs, which makes the current scenario inadequate and restrictive. The IETF has proposed two main approaches for achieving QoS, namely, Integrated Services (IntServ) and Differentiated Services (DiffServ). Although the IntServ model offers absolute end-to-end performance guarantees for every user or flow, the associated overhead contributes to a lack of scalability in the Internet. Differentiated Services proposes a more lightweight approach of grouping flows into few classes, offering more scalability at the expense of absolute end-to-end guarantees. Differentiated Services can take several forms, such as absolute, relative, and proportional differentiation. Proportional Differentiation can be regarded as a general model which can encompass other models as special cases, in which the performance metrics of classes are ratioed according to their weights. This thesis proposes and evaluates a scheduling mechanism for controlling per-hop performance metrics according to the proportional differentiation model. Classes may specify a weight, and the per-hop aggregate throughput and mean delay of the classes may be controllable according to those weights. The scheme uses the well known Little's Law to design such a system. A moving window averaging mechanism and an active queue management scheme are simultaneously and respectively used to achieve control over the relative throughputs as well as the relative delays between classes. This scheme ensures that computationally intensive measurement of actual packet delays is not required, and very little state information must be kept. Mathematical bounds for the feasible conditions under which throughput and delay differentiation is achievable are also presented, and extensive simulations show the effectiveness of the scheme.

CHAPTER 1 INTRODUCTION

1.1 Introduction

The purpose of this chapter is to give an overview of the importance of Quality of Service in the Internet, as well as to introduce recently proposed approaches towards achieving this goal.

1.2 Internet QoS: needs and solutions

The current model of the Internet provides only for best-effort service, which makes no guarantees on packet delivery and provides no bounds on the time of packet arrival. As long as the Internet was a medium of non-real time applications, such as file transfer and e-mail, this model was adequate. However, the proliferation of business and user communities, along with new applications with varying time- and capacity requirements has exposed two weaknesses of this model [1]:

- *lack of performance assurance*: The packet-switched model of the IP network, by itself, provides no assurance that resources will be available at the time communication is desired. As more Internet applications are developing mission-critical needs, this can be a serious limitation.

- *lack of service differentiation*: Currently, all users experience the same level of (best-effort) service. However, applications could range from file transfers, which can tolerate some delay and loss, to streaming video, for which jitter needs to be strictly bounded. Service providers would like to be able to offer their customers different service tiers, depending on the needs of the user.

Research efforts towards QoS, therefore, will need to focus on achieving this performance assurance and service differentiation by resource allocation. The Internet Engineering Task Force

(IETF) has proposed some models and mechanisms towards this, which are briefly discussed below [2] .

1.3 Integrated Services

Integrated Services was IETF's first attempt to improve the Internet's QoS capabilities. The primary motivation at that time was to satisfy the end-to-end requirements of real-time applications, which needed strict jitter and delay bounds. The IntServ architecture [3] aims to provide absolute performance guarantees for each flow, and is based on per-flow resource reservation. Individual users flows use a reservation setup protocol, such as RSVP [4], to reserve the required resources along the path established by the routing protocol. Further, admission control algorithms are used to check for adequate resources at each node. After reservation, packet schedulers in routers on the path forward the packets, using the resources already reserved for that flow. In addition to best-effort service, IntServ supports two service classes:

a. *Guaranteed Service* : This service [5] is designed to offer strict delay and jitter bounds for real time flows. It provides absolute guarantees on the maximal delay experienced by a flow along a particular path. Guaranteed service provides that, the maximum end-to-end queueing delay will remain stable, that is, it will not change as long as the end-to-end path does not change. No assurances are made on minimal or average delay. This approach obviously needs some form of route pinning in order to be effective. Admission control is also assumed. The service represents one extreme end for delay control in networks.

b. *Controlled load Service*: This service [6] is designed to meet the needs of adaptive real time applications, whose performance degrades as the system gets progressively more congested. A Controlled Load Service flow will, even in a overloaded system, approximate the best effort behavior of the flow when the system is under loaded. The scheme specifies that the delay of a high percentage of packets should be close to the minimum delay, and will provide fairly reliable delay bounds for more tolerant applications.

The IntServ approach was the first viable option to achieve QoS, and has the advantages of providing absolute guarantees with a very fine per-flow granularity. However, there are some limitations to this scheme that prevent its widespread deployment [7]. Each flow requires storage of state information on routers, and routers will require processing power to deal

with per-flow book keeping. Further, short-lived flows are wasteful and impractical, with the overhead and connection set-up being greater than the length of the flow. Topology changes will require renegotiation of all current flows, and all participating routers must carry RSVP, per-flow packet classifiers, schedulers, and admission control mechanisms. Per-flow scheduling usually costs O(log n), (n = number of flows) which is restrictive. All these additional sources of overhead increase proportionally with increase in flows. This presents the problem of scalability in the current Internet, which may have thousands of flows passing through a single high-capacity router. These drawbacks led the IETF to propose a more lightweight approach, Differentiated Services.

1.4 Differentiated Services

Differentiated Services, or *DiffServ*, supports a coarser notion of classes than IntServ. Flows with similar requirements may be grouped into a limited number of service classes. The differentiated services architecture [8] is based on a simple model, in which traffic entering a DiffServ-aware network is classified into a limited number of classes based on user requirements. Traffic may also be *policed* or *shaped* at this point. Flows are grouped into classes or *behavior aggregates*, with each class identified by a single DiffServ *code point* (DSCP). Nodes within the DiffServ-aware network select forwarding behavior for packets based on their DSCP, mapping that value to one of the supported *Per-Hop Behaviors* (PHB). Router mechanisms such as packet forwarding and buffering are chosen such that they achieve that Per-Hop Behavior. Nodes at the edge of the DiffServ-aware network (boundary nodes) will need to implement complex classification and conditioning functions; however, nodes within the network (interior nodes) will only need to support only simple, rule-based traffic forwarding. Most of the complexity is pushed to the edges of the network. The next several sections will describe the components of the DiffServ architecture in detail.

1.4.1 Details of the DS field

The DS field is the Type Of Service (ToS) octet in the IPv4 header, or the Traffic Class octet in IPv6, when interpreted from a DiffServ standpoint [9]. Figure 1.1 gives the position of the DS field in the IP header. A packet's DS field is used to designate the packet's class, and consequently determine which forwarding treatment the packet receives.

The former TOS field is replaced by the 8 bit DS field. This byte will carry information

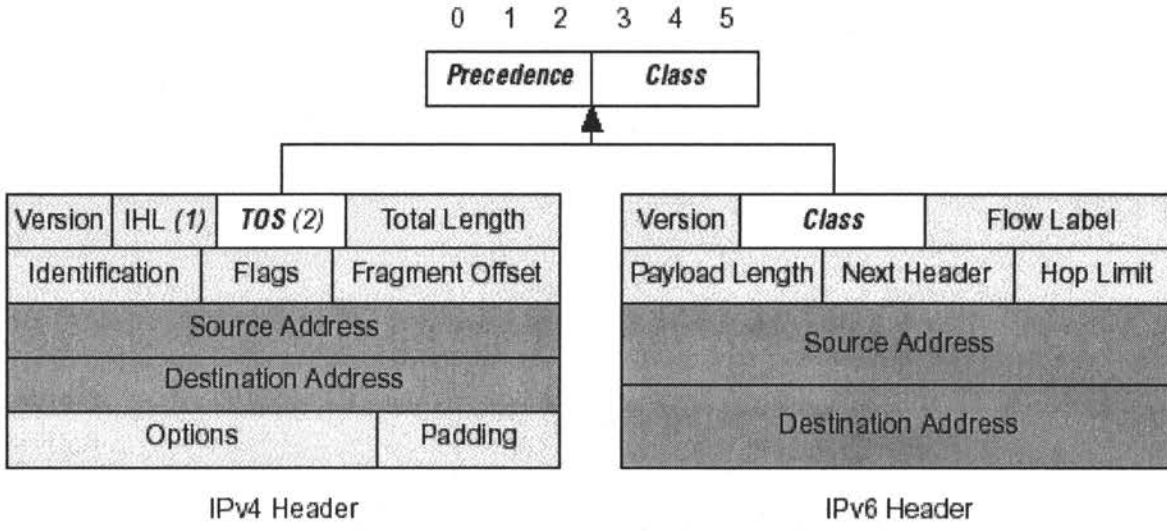


Figure 1.1 The DiffServ field in the IP header

on how the packet will be forwarded. The DS field has exactly the same size and functionality in both IPv4 and IPv6. Figure 1.2 gives the details of the DS-field itself. Six bits of the DS field are used as a codepoint (DSCP) to select the PHB a packet experiences at each node. A two-bit currently unused (CU) field is reserved. To preserve partial backwards compatibility with known current uses of the IP Precedence field, the DS field values with DSCP = 'xxx000' are reserved as a set of Class Selector Codepoints. PHBs which are mapped to by these codepoints must also satisfy the the corresponding precedence requirements. Additionally, the default DSCP ='000000' corresponds to the common, best-effort forwarding behavior already available in existing routers. The mapping of DSCPs to PHBs is not static, and may be locally defined in the context of that DiffServ-aware network.

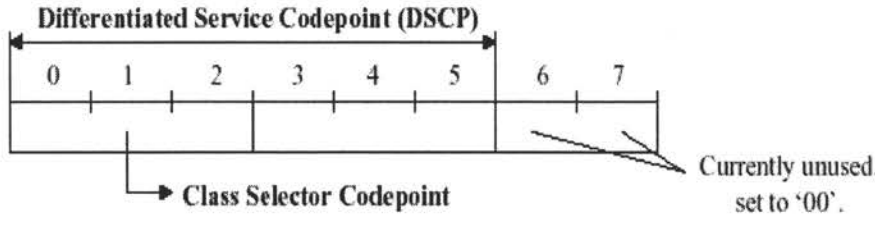


Figure 1.2 Details of the DS field

1.4.2 Boundary Nodes: Traffic Conditioning

The boundary nodes at the edge of the DiffServ-aware network will need to perform traffic conditioning before the traffic enters the network. Traffic conditioning includes performs metering, shaping, policing and/or re- marking, and is necessary to to ensure that the incoming traffic conforms to that users *Service Level Agreement* (SLA). An SLA is a service contract between a customer and a service provider that specifies the profile of the traffic sent by the customer, and also the forwarding service the customer should receive. Figure 1.3 shows the components of a boundary node.

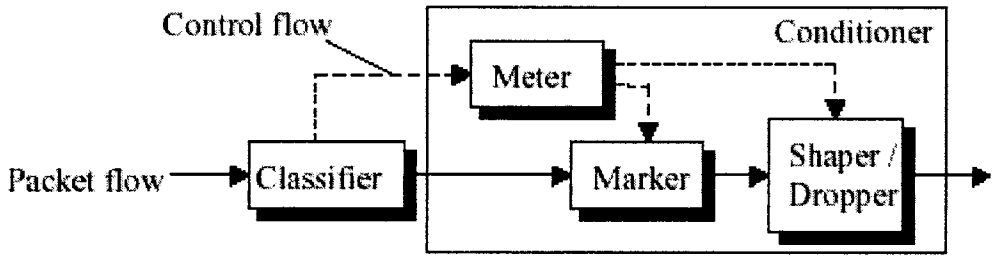


Figure 1.3 The components of a DiffServ boundary node

a. **Classifiers**: Classifiers are used to "steer" packets in a traffic stream matching some specified rule to an element of a traffic conditioner for further processing. Packet classifiers may select packets based on their DSCP's alone, or may use multiple fields such as source and destination addresses, port numbers, and protocol IDs to make this classification.

b. **Meters**: Meters measure the temporal properties of the traffic against the traffic profile specified by the user in the SLA. Different conditioning actions may be applied to the stream, depending on whether it is in-profile or out-of-profile, In-profile packets may be allowed to enter the DS domain without further conditioning; or, in case the DSCP needs to be changed/initialized for that network, the packets may be passed on to the marker. Out-of-profile packets may be queued until they are in-profile (shaped), policed (dropped), marked with a new codepoint (re-marked), or forwarded unchanged while triggering some accounting procedure. The meter may also combine accounting functions, for tracking and billing excess traffic.

c. **Markers**: Markers set the DS field of a packet to a particular DSCP, adding the marked packet to a particular class. The marker may be configured to mark all incoming packets to a single codepoint, or may be configured to mark to one of a set of codepoints according to the state of a meter. When the marker changes the codepoint in a packet it is said to have "re-marked" the packet.

d. **Shapers**: Shapers delay non-conformant packets in a traffic stream, so that the stream may obey its specified traffic profile. Shapers usually have finite-size buffer, and packets may be discarded if there is not sufficient buffer space to hold the delayed packets.

e. **Droppers**: Droppers discard some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile. A dropper is a special case of a shaper with a very small buffer size.

1.4.3 Interior Nodes

Nodes within a DiffServ-aware network have only one requirement: they need to be able to implement the per-hop behaviors associated with the DSCPs of incoming packets. The formal literature defining DiffServ architecture consider per-hop behavior to be only a general concept, or a means by which a node allocates resources to different classes. Given that, defining and implementing effective per-hop behaviors is the key to a useful mechanism to differentiating between classes, and the focus of much current research.

Although the basic architecture assumes that complex conditioning functionality is located primarily in boundary nodes, it is possible to also include these components in interior nodes when needed.

The DiffServ approach has many advantages over the previously described IntServ. Since complexity-intensive functionality may be pushed to the edge of the network, the architecture is inherently scalable. State information is maintained on a per-class basis for a limited number of classes, so DiffServ also scales well with increase in flows.

1.5 Types of Differentiated Services

Research on DiffServ is proceeding along three broad directions [14], *Absolute DiffServ*, *Relative DiffServ*, and *Proportional DiffServ*. The next sections will explain these approaches.

1.5.1 Absolute DiffServ

Absolute DiffServ is an attempt to meet the same goals as IntServ, of providing absolute performance levels, but without per-flow state maintained in the routers. This can be achieved by providing a strict limit on the resources that the user can expect to receive from the network. Generally, some route pinning is required in order for these services to provide end-to-end QoS. Assured Forwarding and Expedited Forwarding PHB approaches, as specified by the IETF, are examples of Absolute DiffServ.

1.5.2 Relative DiffServ

More flexible than the Absolute model, this approach provides assurances for the relative quality according to some performance metric, rather than absolute service levels for classes. Traffic is grouped into N classes such that class i is better than, or at least no worse than, class *(i-1)*, for $1 < i \leq N$. The comparison is made on the basis of some meaningful performance metric, such as queuing delay, aggregate throughput, or packet loss. In case of a lightly-loaded network, all classes may experience the same quality, so the clause "or no worse than" is therefore meaningful. Users cannot be guaranteed a specific level of performance, but get the assurance that higher classes are allocated more resources than lower classes. It is the user's responsibility to choose a class that meets the current needs of the application.

1.5.3 Proportional DiffServ

A refinement of the Relative DiffServ model, Proportional DiffServ aims at achieving two goals :

a. *Predictability*: Consistent differentiation independent of class loads. Higher classes should always be better, or at least no worse, than lower classes, based on some performance metric.

b. *Controllability*: Ability to externally adjust the quality spacing between classes. Network operators must have "tuning knobs" to adjust the quality spacing between classes, based on pricing and policy.

The Proportional DiffServ model controls some chosen class performance metric, proportional to the differentiation parameters chosen by the network operator. (Differentiation parameters are like weights, and are meant to specify the quality "distance" needed between

classes.) If p_i is the chosen performance metric for class i, and c_i is the corresponding operator-chosen quality differentiation parameter, then the proportional differentiation model attempts to achieve

$$p_i/p_j = c_i/c_j$$

over all classes. So, even though actual service levels vary with class loads, service ratios are always maintained.

This thesis proposes a scheduler based on this model.

1.6 Thesis Contribution and Organization

This work presents the development of a scheme that implements combined delay and throughput differentiation using the Proportional Differentiated Services model. As explained in Section 1.4.3, mechanisms that implement the desired per-hop behavior are key to an effective DiffServ framework. The set of algorithms presented in this work may be implemented in a router at the core of a DiffServ aware network, in order to offer delay and throughput control to the packets passing through that router. Further, when the system is lightly loaded, the scheme reduces to a new implementation of Proportional Delay Differentiation. The scheme involves very little overhead, and no complexity intensive delay measurements are involved.

Chapter 2 discusses several algorithms and scheduling mechanisms for DiffServ which have been proposed in the literature. The discussion also motivates the work of this thesis. Chapter 3 presents the algorithm based on Little's law, strategies used to implement this algorithm, and bounds for feasibility. Experimental results based on simulation are presented and analyzed in Chapter 4. Chapter 5 gives a summary and conclusions.

CHAPTER 2 BACKGROUND

2.1 Introduction

This chapter will overview the scheduling mechanisms which have been developed for DiffServ. These mechanisms will be classified according the the Differentiated Service model they have been developed within.

2.2 Absolute Differentiated Services

2.2.1 Expedited Forwarding

Expedited Forwarding [10], [11] earlier called Premium Service, proposes a low-loss, low latency, low jitter, guaranteed bandwidth end-to-end service. The rationale behind this scheme is that loss, delay and jitter are due to the queues experienced by the traffic as they pass through the network. Therefore, a service which ensures that the traffic experiences very small queues, will be a low delay and loss service. Such a service may be created by enforcing two mechanisms:

a. *Minimum departure rate:* Nodes must be configured so that the EF class has a well-defined minimum departure rate. The minimum departure rate should always be sustained, independent of other traffic at that node. This can be achieved by several simple mechanisms. A example can be a simple priority queue, as long as there is no higher priority queue that can preempt the EF traffic for more than a packet transmission time.

b. *Conditioning incoming traffic:* Incoming traffic from the EF class must be configured so that its arrival rate is always less than each node's configured departure rate. Network boundary conditioners such as those described in Section 1.4.2 can provide this functionality.

If the arrival rate of packets at a node is always less than the departure rate supported by the node, arriving packets will see very little queueing delay. In this way, assurances of low delay and jitter can be met. Expedited forwarding is an example of Absolute DiffServ, since the service provides the absolute contracted bandwidth as long as the users traffic is within its profile.

The IETF has recommended the DSCP for EF traffic to be '101110'.

2.2.2 Assured Forwarding

Another approach towards achieving absolute DiffServ is Assured Forwarding [12]. Similar to Expedited Forwarding, users specify a traffic profile that their traffic is expected to maintain. Incoming packets may be marked as "in-profile" or "out-of-profile" by profile meters placed at the edge of the network, depending on whether the user follows the allocated bandwidth profile or not. When congestion occurs within the network, out-of-profile packets are dropped with a higher probability than in-profile packets, thus assuring bandwidth differentiation during congestion. Based on this scheme, the IETF has proposed an Assured Forwarding PHB [13]. This provides for delivery of packets in 4 independently forwarded classes, with each class having 3 levels of drop precedence. Each class is independently alloted a definite amount of resources, in terms of buffer space and bandwidth, and has a separate queue. In case of congestion, the drop precedence within a class determines the relative importance of that packet within the class. For Assured Forwarding PHB, user traffic need not be hard-limited at the ingress, with the proviso that out-of-profile packets have lower probability of being delivered than in-profile packets. A selective dropping algorithm such as RIO (RED with *in* and *out* packets), [12] may be used to achieve multiple drop precedence levels. Figure 2.1 shows the IETF DSCPs assigned to the AF PHB.

Drop Precedence	*Class 1*	*Class 2*	*Class 3*	*Class 4*
Low	001010	010010	011010	100010
Medium	001100	010100	011100	100100
High	001110	010110	011110	100110

Figure 2.1 IETF recommended DSCPs for the AF PHB

Assured forwarding can be considered a mechanism for Absolute DiffServ from the point of view of bandwidth differentiation, since the aim is to provide the user with absolute values of

bandwidths for in-profile traffic. However, from the viewpoint of loss differentiation, it is closer to Relative DiffServ, since the standard implementations of selective dropping mechanisms offer higher loss rates to lower classes than higher classes.

2.2.3 A two-bit DiffServ Architecture

This implementation [16] of a flexible, multi-level DiffServ architecture was essentially a combination of the Expedited Forwarding and Assured Forwarding mechanisms described in the previous sections. Packets may be marked with a "P" bit for Premium (or Expedited) forwarding, or with an "A" bit for Assured forwarding. Figure 2.2 shows the block diagram of a boundary node implementing this Two Bit scheme.

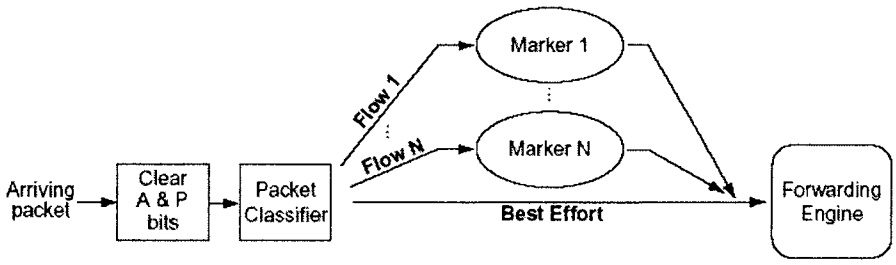


Figure 2.2 Block diagram of a boundary node with the Two Bit scheme

All arriving packets have their A and P bits cleared, and then are separated into different user flows based on the header. Flows then pass through Markers, each of which have been configured with the usage profile for that user/flow. A usage profile can include a service class (P or A), rate, and allowable burst size. Assured flow packets have their A bits set by the marker if they confirm to their profile; otherwise they are not marked. For Premium traffic, the marker will hold/delay packets so that the flow confirms to the stated profile. All premium packets leaving the marker will be marked with the P bit. At the forwarding engine, there are two queues, one each for the A and P bits. The two queues are serviced according to a simple priority scheme, with Premium packets first. However, within the Assured packets queue, RIO [12] must be implemented, so that packets with their A bit set are dropped with a lower probability than those with their A bit cleared.

The work provides a basic flexible framework for DiffServ, and suggests that either or both of these mechanisms can exist along with best effort traffic depending on the needs of the network.

2.2.4 An Adaptive Packet Marking algorithm

The authors in [19] propose a scheme for bandwidth differentiation by using adaptive packet marking at the edge of the network. Since the scheme offers users an absolute bandwidth, this can also be considered within the domain of Absolute bandwidth differentiation. The scheme can be considered an extension of any selective dropping algorithm (ERED, RIO), with more of an emphasis on the algorithm used to tag or mark the packets at the network boundary. Section 2.3.2 gives the details of one such algorithm. There are 2 levels of differentiation, multiple queues are unnecessary, and complexity is mostly restricted to the edges of the network.

The packet marking scheme is designed for use in a network that supports a single bit priority scheme, that specifies best effort traffic and priority traffic. Priority traffic suffers lower loss rates than the best effort traffic. The user or application specifies a minimum throughput rate required for a connection, and this connection parameter is given to a *packet marking engine (PME)* located on or near the host-network interface. By default, all packets are initially marked as best-effort. The role of the PME is to monitor the throughput experienced by the connection, and ensure that the requirements are met. If the observed bandwidth for the connection meets or exceeds its specified minimum, the PME acts as a passive monitor. However, if the actual bandwidth of the connection is less than that required, the PME marks more packets as priority. Once the target bandwidth is reached, the algorithm reduces the number of marked packets as much as possible without falling below the requested rate. The marking probability is scaled by the difference between the actual bandwidth and the target bandwidth. The algorithm is as follows:

```
Define :
tbw - target bandwidth
obw - observed bandwidth
mprob - marking probability
Upon timer expiry, do
   scale = |1 − obw/tbw|
   if (obw < tbw)
      mprob = mprob + scale * increment
   else
      mprob = mprob − scale * increment
```

The marking probability is periodically updated at the edge of the network. Note that the throughput instead of the goodput is used for simplicity, even though packets may be lost/dropped from the connection after the measurement has taken place. Within the core

of the network, any reasonable loss-differentiation scheme can be used to preferentially drop best-effort packets over marked (priority) packets. The authors use *Enhanced Random Early Detection (ERED)* in this work.

This scheme is meant for differentiating bandwidth between regular (best effort) traffic and priority traffic. Hard guarantees are not possible since there is no allocation of resources. It is not clear how this scheme can be reliably extended to multiple classes, as it cannot provide a quantitative bandwidth differentiation even within these 2 classes. Other parameters such as delay and loss are not dealt with.

2.3 Relative Differentiated Services

The Relative Differentiated Services approach, as explained earlier, focuses on maintaining class differentiation by ensuring that, with respect to some important performance parameter, a higher class is always better than (or at least no worse than) a lower class. In Section 2.3.1, some general approaches towards Relative DiffServ are described, along with drawbacks that preclude using these mechanisms for Proportional DiffServ. The next 2 sections describe some specific proposals.

2.3.1 General approaches to Relative DiffServ

Reference [14] describes several general mechanisms which can be used to implement Relative DiffServ. Some of these are briefly explained here:

a. *Price Differentiation*: This mechanism uses no special forwarding behavior, but proposes that a policy of strict differentiation in pricing of classes will naturally lead to differentiation in usage of those classes [15]. The assumption is that higher prices will will lead to lower loads in higher classes, giving users of higher classes better service. The advantage of this approach is that it is very simple to implement. However, since the pricing structure cannot be modified often, this approach is not very flexible. Further, if multiple high-class users become simultaneously active, higher classes may actually receive poorer service than lower classes.

b. *Capacity Provisioning*: This approach involves careful allocation of resources to classes, with higher classes allotted a larger proportion of the resources (bandwidth or buffer space) than lower classes. Schedulers such as Weighted Fair Queuing (WFQ) or Weighted

Round Robin (WRR) can be configured to provide this behavior. However, this approach does not offer predictable differentiation during shorter time-scales, especially given the bursty nature of Internet traffic. This could lead to higher classes actually receiving less service than lower classes.

c. *Strict prioritization*: The simple mechanism of ordering traffic as queues, and serving these queues strictly based on their priority, will lead to sustained predictable differentiation. Higher classes will always be be better than lower classes. However, strict prioritization lacks controllability, that is, the quality spacing between classes cannot be adjusted easily.

2.3.2 Loss Differentiation: the RIO approach

RIO *(RED with in and out packets)* [12] is a scheme for the selective dropping of packets of a lower class, over dropping those from a higher class. This scheme assumes that, somewhere in the edge of the network, packets that are within the profile are tagged as *IN packets*, and those outside the profile are tagged as *OUT packets*. RIO is based on RED [17], by extending it to accommodate selective dropping, so a brief outline of RED is given here. RED *(Random Early Detection)* is a congestion control scheme that is designed to maintain high throughputs and tolerate transient congestion. RED routers drop packets at random, so that the affected TCP connections back off at different times, and avoid global synchronization. Two queue thresholds are used to monitor the dropping. When the queue size is below the first threshold (min_{th}), no packets are dropped. When the queue size is between the two thresholds, packets are randomly dropped with a linearly increasing probability, with the probability being a function of the average queue size and the maximum dropping probability, P_{max}. The packet drops in this congestion avoidance phase are designed to notify the end hosts of congestion in the network, and to force them to reduce their sending rates. When the instantaneous queue size goes above the second threshold (max_{th}), all packets are dropped in order to maintain a short queue size. Figure 2.3 illustrates this behavior.

RIO uses the RED algorithm by using the same mechanism, but is configured with two sets of parameters: $(P_{max}^{IN}, min_{th}^{IN}, max_{th}^{IN})$ for IN packets, and $(P_{max}^{OUT}, min_{th}^{OUT}, max_{th}^{OUT})$ for OUT packets. The average queue sizes are calculated differently for IN and OUT packets, with IN packet calculations being based on the size of the IN packets queue, and OUT packet queue calculations being based on the combined IN and OUT packets queue. By setting the

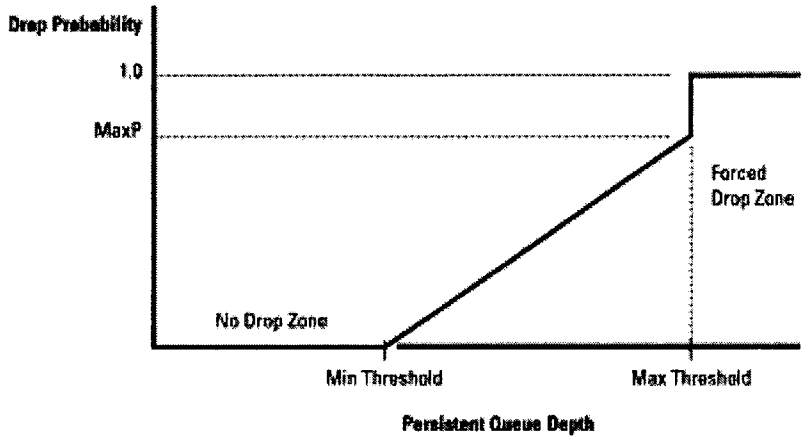


Figure 2.3 RED: Drop Probability vs. Queue Length

parameters appropriately, the OUT packets can be discriminated against.

This approach is designed to push complexity to the edges of the network, where the tagging is done. The routers at the core simply have to run RIO on all incoming packets. Also, with this scheme, multiple queues are not needed to ensure service differentiation between different classes. The routers can treat all IN packets from different flows and users as a single queue. This aspect helps in the ease of deployment.

This scheme can be considered a flexible implementation of preferential dropping, and can be modified to offer both relative and absolute differentiation. For example, if min_{th}^{IN} is chosen to be always greater than max_{th}^{OUT}, it is guaranteed that out packets will always be dropped before in packets, thereby providing absolute differentiation. However, a different choice of parameters will provide relative loss differentiation.

2.3.3 Bandwidth Differentiation: the MulTCP approach

The authors of this paper [20] have taken the approach of associating weights with TCP connections at the end user, and applying algorithms at the end user to allocate bandwidth according to the weights. The approach used is weighted proportional fairness, in which the weight of each flow is given by the price paid by the user. Two strategies are used to control the bandwidth allocated to connections using these weights:

a. *Limiting the receiver buffer*: This approach is designed for a scenario in which a set of connections terminate at one host, and share a link that could potentially be a bottleneck.

In this case, limiting the TCP receiver buffers of all connections at the host, using the weights, will amount to weighted proportional fairness. In general, the size of the receiver buffer, B_R, limits the throughput T of a TCP connection as $T \leq B_R/R$, where R is the round trip time. To achieve weighted proportional fairness of the throughputs, receiver buffer sizes of a connection i can be constrained as

$$b_i = B * k_i / \Sigma k_j$$

where k_n is the price that user n is willing to pay, and B is the maximum data that can be in transit over the bottleneck link, given by product of the bottleneck bandwidth and mean RTT of all connections. Doing this sets an absolute upper limit on the throughput of that connection. However, proportional fairness implies that any change in capacity should be divided proportionally and fairly among all existing connections. All buffer sizes will need to be adjusted every time a connection is added, deleted or when a user decides to change the price they are paying.

b. *Changing the TCP control algorithms*: Unlike the previous case, this algorithm is a distributed approach to the weighted proportional fairness problem, and can be used for any network with multiple bottleneck links. It can also be implemented at the end users. The user can associate a number N with their TCP connection, so that the single TCP connection will get the same share of a congested gateway's bandwidth as N standard TCP connections (MulTCP). This is achieved by modifying the TCP control algorithms as follows:

   i. *Slow start phase:* Consider a single TCP connection which has the weight of N virtual standard TCP connections. Applying the regular slow-start phase to a MulTCP flow will result in sending N, 2N, 4N...packets, which may result in bursty patterns of traffic and losses. To counter this, the modified slow start for MulTCP increases the window size by two packets for every acknowledgement received, so that the number of packets sent grows as 1,3,9, 27...for N virtual TCP connections. This continues till the congestion window opens as far as that for N standard TCP connections.

   ii. *Linear Increase:* In the standard TCP control algorithm, after slow start has been reached, the congestion window is increased by 1 packet per RTT. With MulTCP having N virtual connections, the congestion window is increased by N packets per RTT.

iii. *Multiplicative Decrease:* In the standard congestion multiplicative decrease phase of TCP, loss of a packet causes the congestion window to be halved. In MulTCP, when one packet is lost, it is treated as a packet loss from one of the N virtual TCP connections. Hence, only 1/N of the current congestion window is halved, causing the congestion window *cwnd* to be set to ((N-0.5)/N) * *cwnd.*

The advantage of this approach is that the price set by the users themselves is responsible for maximizing the utilization of the network. A self-priced design does away with many sources of overhead usually associated with service differentiation, such as connection admission control, need for multiple queues for different classes, etc. Specifically, this approach makes over-provisioning redundant since the network does not guarantee any level of service at all. This approach controls bandwidth allocated to different flows depending on the weight specified by them. There are no bounds or limits, however, to the bandwidth actually achieved by the individual flows, since this depends on the price that the users perceive as worthwhile, and the number of flows contending for the congested link. Also, it is unclear how such an approach will affect other important performance parameters of the flows, such as delay and loss.

2.4 Proportional Differentiated Services

The Proportional DiffServ model, as explained earlier, is an attempt to introduce *predictability*, or consistent differentiation, and *controllability*, or ability to adjust the quality spacing between classes, within Relative Differentiated Services. Some developments in this direction are briefly described below.

2.4.1 Delay Differentiation: the BPR and WTP schedulers

Reference [21] introduced the concept of Proportional DiffServ, and proposed two schedulers to achieve proportional delay differentiation. Specifically, to achieve delay differentiation: If d_i is the average queueing delay of a class i, then

$$\frac{d_i}{d_j} = \frac{\delta_i}{\delta_j}$$

over all N classes, where δ_i is the *Delay Differentiation Parameter* for class i, ordered as $\delta_1 > \delta_2 > ...\delta_N > 0$. The proportional differentiation model is independent of class loads, since relative ratios are used. Two schedulers have been proposed in [23]:

a. *Backlog Proportional Rate (BPR) Scheduler* : Here, the class service rates are dynamically adjusted so that they are proportional to the backlog suffered by that class. The rationale behind this scheme is that, if a class has received a smaller amount of service than it 'deserves' (based on its delay differentiation parameter and current load) in a recent time interval, then its backlog (or queue) will be proportionally larger. Serving this class will reduce its backlog, and therefore will help in reducing the unfairness in delay for that class.

Formally, the BPR proportionality constraint for N classes is:

$$\frac{r_i(t)}{r_j(t)} = \frac{s_i}{s_j} * \frac{q_i(t)}{q_j(t)}$$

where $r_i(t)$ is the service rate for class i at time t, $q_i(t)$ is the length of the queue i at time t, and s_i is the *Scheduler Differentiation Parameter* for class i. This equation should be valid for all $i, j \leq N$. The actual service rates of the classes are subject to the following constraint:

$$\sum_{i=1}^{N} r_i(t) = R$$

where R is the link capacity. A drawback of this scheme is that, since a small relative backlog produces small service rates, queues that are just emptying or just filling up can experience large(r) delays.

b. *Waiting Time Priority (WTP) Scheduler* : This is a priority scheduler for which the amount of time that a packet has been waiting in a queue, proportional to its differentiation parameter, determines its priority for service. Formally,

$$p_i(t) = w_i(t) * s_i$$

where $w_i(t)$ is the waiting time of a packet at time t. The class priorities have to be recalculated with every packet departure for all backlogged classes. The Scheduler Differentiation Parameters s_i control the rate of increase of priority with queuing delay. This approach has the limitation that, in case of large bursts arriving in a higher class, short term starvation can occur for lower classes.

2.4.2 Loss Differentiation: Proportional Loss Rate Droppers

In this work [22], the authors extend the proportional differentiation model developed for delay (above) to loss differentiation. Specifically, loss differentiation in the proportional

model can be achieved if:

$$\frac{l_i}{l_j} = \frac{L_i}{L_j} \tag{2.1}$$

over all classes, where l_i is the packet loss rate for class i, and L_i is the Loss Differentiation Parameter for class i. A packet dropping mechanism must be developed to implement this. Currently proposed packet dropping schemes are inadequate for the proportional differentiated model, as explained below. Lowest Priority First schemes, that drop packets from the least backlogged class first, assure consistent differentiation but do not provide for quality spacing between classes. Buffer partitioning schemes in which packets of a class are accepted only if the aggregate backlog of a class is less than a certain proportional threshold have also been proposed. The drawback of this approach, however, is that the fineness or granularity of the differentiation is strongly dependent on the statically set thresholds. Other approaches like multi-class RED and RIO (explained earlier) also share this limitation. The authors, then, propose a new dropping mechanism that is designed for proportional loss differentiation.

The Proportional Loss Rate Dropper (PLR) is realized by interpreting equation 2.1 as the normalized loss rate (l_i/L_i) being equal for all classes. The class from which to drop a packet is selected as the class which has the minimum normalized loss rate of l_i/L_i . This will cause equality of the loss ratio over all classes. The authors suggest two versions of PLR, depending on the length of time over which the loss rate l_i is measured. The *PLR(∞)* dropper maintains l_i as a long term fraction of dropped packets. The *PLR(M)* dropper computes l_i as the fraction of dropped packets in class i over the last M arrivals.

2.4.3 Combined Loss and Delay Differentiation

Other researchers [21], [29], [7] have proposed schemes to achieve combined delay and loss differentiation. These schemes are either combinations of the delay and loss approaches mentioned in this chapter, or propose heuristics to achieve optimal delay and loss values in a single step [29], [7]. As will be shown in Proposition 1, combined loss and delay differentiation is not possible without taking into account the actual loss ratios. These schemes therefore consider the actual loss ratios to achieve combined differentiation.

2.5 Thesis Motivation

The schemes of Relative Differentiated model deal exclusively with achieving throughput differentiation. Further, all have the severe limitation of lacking consistent differentiation

and inability to control *spacing* between classes. Proportional Differentiated Service schemes alleviate this, and offer solutions for delay and loss differentiation, both separately and jointly. Current schemes for joint delay and loss differentiation implement heuristics to achieve optimal delay and loss values in a single step. Measuring the actual loss ratios will call for measurement windows, and thereby more state information. Further, throughput differentiation has not been combined with any other metric. There is a need for a proportional differentiation scheme that integrates control of throughput with other significant performance metrics. This work proposes to achieve throughput and delay differentiation simultaneously.

CHAPTER 3 STRATEGIES FOR COMBINED PROPORTIONAL DIFFERENTIATION: LIMITATIONS AND APPROACHES

3.1 Introduction

This section introduces strategies to achieve combined proportional differentiation of delay and throughput. Theoretical foundations for the scheme are introduced, followed by a justification for choice of delay and throughput metrics. The limits on feasible throughput and delay ratios are derived. Proof is provided for the fact that satisfaction of throughput ratios is possible only under certain conditions, and bounds are presented. It must be noted that, in case throughput differentiation cannot be achieved, the scheme reduces to a new protocol for achieving proportional delay differentiation (which is always feasible).

3.2 Little's Result and Proportional Differentiation

This work has its basis in the fundamental result, first proven by J. D. C. Little, and known as Little's Result, e.g., see [26]. Consider any general queueing system in which customers arrive at some arrival rate, receive a certain amount of service from the system, and then depart from the system. If s_i is the throughput of customers in class i, $\overline{w}_i$ is the average delay experienced by such customers, and $\overline{q}_i$ is the average number of class i customers waiting (or average queue size), then Little's result states that

$$\overline{q}_i = s_i \cdot \overline{w}_i$$

This result makes no assumptions about the nature of arrival or departure processes in the queuing system, and is therefore very general. Little's result holds for any work-conserving discipline, where no work is created or destroyed within the system, and also for non work-conserving disciplines. A work-conserving mechanism is also one in which the server is never idle unless there are no customers in the system. This attribute makes the work-conserving approach especially relevant to the design for an Internet router, since high throughput is very

desirable in this context.

Taking multiple classes into account, the following is a straightforward application of the above

$$\frac{\overline{q}_i}{\overline{q}_j} = \frac{s_i}{s_j} \cdot \frac{\overline{w}_i}{\overline{w}_j} \tag{3.1}$$

Predefined weights can be associated with the performance metrics of each class, and the problem of maintaining proportional differentiation between classes for these metrics then reduces to a problem of enforcing equation (3.1).

However, combined proportional differentiation has its limitations with regard to the metrics that can be combined. We propose the following, which is proven in *Appendix A*:

Proposition 1: *it is not possible to achieve combined proportional differentiation in the delay and loss metrics, independent of actual values of packet loss ratios.*

Given the restriction that combined delay and loss differentiation is possible only if the actual loss ratios are taken into account, we have chosen to implement combined delay and throughput differentiation.

3.3 Choice of Proportionality Metrics

The previous section has justified the decision to use, for this work, the form of Little's equation that incorporates throughput rather than loss. That is, for every pair of classes, i and j, equation (3.1) must hold. Control of any two of the three ratios in this equation will result in a proportional control of the third. Controlling the mean delay of a class involves greater complexity, since router bookkeeping and delay measurement has to be done for each packet passing through the router. Mean delays will need to be updated every time a packet is served. Also, accuracy of measurement can vary widely between systems, as it depends on clock granularities. Clearly, measuring and controlling mean delays of classes is a computationally expensive and repetitive task. Overruling active delay control leads to the choice of controlling $\overline{q_i}$ and s_i as the optimal approach.

3.4 Strategies for Combined Throughput and Delay Differentiation

This section discusses our strategies for controlling the queue and throughput proportions, hence achieving combined proportional differentiation. It also establishes bounds on the achiev-

Symbol	Definition
$i,\ j$	indices corresponding to the class number, where $0 \leq i, j < N$
$\overline{b}_i$	mean service time for a packet from class i
$\overline{b^2}_i$	second moment of the service time for a packet from class i
λ_i	arrival rate from class i in packets per second
ρ_i	offered load (in Erlang) from class $i = \lambda_i \overline{b}_i$
s_i	throughput of class i in packets per second
σ_i	carried load from class i in Erlang $= s_i \overline{b}_i$
$\overline{w}_i$	mean packet delay for class i packets
$\overline{q}_i$	mean number of packets from class i in the buffer
S_i	throughput weight for class i
W_i	mean delay weight for class i
Q_i	mean queue size weight for class i
B	buffer size

Table 3.1 List of symbols used in the thesis

able proportions. In Section 3.4.1 we present the Packet Scheduler for departing packets, with separate modes for controlling the throughput and queue ratios. Section 3.4.2 shows how to integrate the two modes. Section 3.4.3 is the Queue Manager, which describes how to handle packet arrivals to a full buffer. The symbols defined in Table 3.1 will be used in the discussion.

3.4.1 The Packet Scheduler: Mechanisms for Packet Departure

Serving a packet from a class increases the throughput of the class, and hence affects the throughput ratios involving that class. Further, service from a class decreases the queue length of that class, affecting the associated queue ratios (and hence delay ratios). Therefore, the packet scheduler must be designed to control both ratios properly. This section explains separate scheduling mechanisms to control each of the two ratios, and the next section describes how to combine them.

a. **Controlling s_i/s_j: The throughput control mechanism**

To control the throughput of all classes proportionally, a moving window averaging mechanism is used. Throughput data for departing packets is collected from each class over a moving window. A moving window of size M will hold throughput information about the M most recently served packets. We have chosen the moving window size to be of the same order as the buffer size. During the mechanism development, both the mov-

ing window and a jumping (discrete) window approaches were considered. It was found that the discrete window mechanism offered abrupt, sharp changes of throughput periodic measurements, which led to inadequate correction of the desired throughput ratios. When the throughput mechanism chooses a class to serve a packet from, it will base its choice of the class on minimizing the difference between the throughput ratios of all flows in the system and the ideal ratio, using min-max optimality. That is, packet departures are scheduled from that class for which the maximum deviation of any of its throughput ratios from the ideal, after service, is minimal. The implementation of this strategy will be defined formally in Section 3.5, and using equation (3.3). This strategy, when used exclusively, will fulfill the requested throughput weights of all flows, provided the input traffic satisfies certain conditions.

Considering two classes, 1 and 2, and given that ρ_i is the offered load from class i, and S_i is the weight associated with the throughput for class i, we propose the following, which is proven in Appendix B:

Proposition 2: *The necessary and sufficient conditions for the throughput ratios to be satisfied depend on the offered load, and are as follows:*

Case 1: When $\rho_1 + \rho_2 \leq 1$, *then* λ_1/λ_2 *must equal* S_1/S_2

Case 2: When $\rho_1 + \rho_2 \geq 1$*:*

2a: if $\rho_1/\rho_2 > S_1/S_2$, *then* $\lambda_2 \geq S_2/(S_1\bar{b}_1 + S_2\bar{b}_2)$

2b: if $\rho_1/\rho_2 < S_1/S_2$, *then* $\lambda_1 \geq S_1/(S_1\bar{b}_1 + S_2\bar{b}_2)$

When the system is unsaturated, that is, when the offered traffic is less than the system capacity, it may not be possible to satisfy the throughput ratios. All incoming packets are served, hence throughput differentiation is not possible unless the offered traffic satisfies the desired throughput ratios as in *Case 1* (we assume a work conserving system). However, when the input traffic exceeds the server capacity, the system is considered overloaded. Since buffers are limited in capacity, packets must be dropped. Without route pinning in DiffServ domains, this case may arise. In such a case, it might be possible to achieve throughput differentiation according to the above condition.

b. **Controlling $\overline{q_i}/\overline{q_j}$: The queue control mechanism**

Serving a packet affects the delay ratios, so it is important to incorporate delay control as well in the scheduling mechanism. Ratios of mean queue sizes of different classes need to be also controlled in order to control the mean class delays proportionally. The scheduler can control the queue lengths by serving a packet from that class for which the maximum deviation of the queue lengths, after service, is minimal. This strategy, when exclusively used, will control the queue lengths (and hence delays) proportionally, with the following caveat: when one or more of the queues are small, the above min-max approach presents anomalous behavior. This is due to the fact that when one or more of the queue sizes is very small (0 or 1), considering q_1/q_2 may give a different service decision than by considering q_2/q_1. The frequency of this anomalous behavior increases as the input traffic decreases until, under very sparse incoming traffic, the achieved delay ratios are actually the reciprocal of the target values. For example, consider a system with only two classes, Class 1 and Class 2. If q_2 is frequently very small or zero, considering q_1/q_2 in the algorithm will lead to erroneous results (often yielding ∞), while considering q_2/q_1 will be more accurate (often yielding zero.) In such a case, considering q_1/q_2 will lead to the exactly wrong decision, consistently choosing the wrong queue to serve. Hence, in the worst case, achieved ratios may the reciprocal of the target value. This issue has been dealt with by applying a simple, deterministic heuristic in such cases, i.e., serving the class with the smallest delay weight W_i. Note that delay differentiation using this mechanism is possible under all scenarios, unlike throughput differentiation.

Additionally, there are certain constraints to the delay control using min-max optimality. Proposition 3 states the bounds on the achievable delay ratios when system throughput is less than 1:

Proposition 3: *When $\sigma_1 + \sigma_2 < 1$, and under Poisson arrivals and general service times, the achievable delay ratio is such that:*

$$\frac{(1-\sigma_1-\sigma_2)\cdot[\bar{b}_0+\bar{b}_1(1-\sigma_1)]}{\bar{b}_0+\bar{b}_2(1-\sigma_1)(1-\sigma_1-\sigma_2)} \leq \frac{\overline{w}_1}{\overline{w}_2} \leq \frac{\bar{b}_0+\bar{b}_2(1-\sigma_1)(1-\sigma_1-\sigma_2)}{(1-\sigma_1-\sigma_2)\cdot[\bar{b}_0+\bar{b}_1(1-\sigma_1)]}$$

where $\bar{b}_0$ is the residual service time as seen by an arrival, and is given by

$$\bar{b}_0 = \sum_{i=1}^{2} \sigma_i \frac{\overline{b_i^2}}{2\bar{b}_i} \tag{3.2}$$

Proof is provided in Appendix C. It must be noted that these two bounds are also consistent with equation (4) in [28], with this result being the limit on the mean delay ratio when the Waiting Time Priority Scheduler is used, and when the dynamic priority control parameters for class 1 is much higher (respectively lower) than that for class 2.

Proposition 4 states the achievable delay bounds when the system is saturated, i.e., when $(\sigma_1 + \sigma_2 = 1)$:

Proposition 4: *Assuming Poisson arrivals and general service times, when a system is saturated,*

$$\frac{(\bar{b}_0 - \sigma_1\bar{b}_1 + \bar{b}_1)\sigma_2\bar{b}_1}{[(B-1)(1-\sigma_1)\bar{b}_1 - (\bar{b}_0 - \sigma_1\bar{b}_1)\sigma_1]\bar{b}_2} \leq \frac{\overline{w_1}}{\overline{w_2}} \leq \frac{[(B-1)(1-\sigma_1)\bar{b}_1 - (\bar{b}_0 - \sigma_1\bar{b}_1)\sigma_1]\bar{b}_2}{(\bar{b}_0 - \sigma_1\bar{b}_1 + \bar{b}_1)\sigma_2\bar{b}_1}$$

where $\bar{b}_0$ is the residual service time given by equation (3.2).

Proof of this proposition is in Appendix D. Taken together, Propositions 2,3 and 4 present the achievable bounds for delay and throughput differentiation.

3.4.2 Packet Scheduler: Modes of Operation

The throughput control mechanism, if used exclusively (i.e., if packet scheduling is controlled by using this mechanism only), will exactly satisfy throughput ratios, subject to the input constraints described earlier (Proposition 2). Likewise, the queue control mechanism, when exclusively used, will provide the exact delay ratios required in all cases. However, our work aims at combined delay and throughput differentiation, when feasible. Accordingly, the system should employ both mechanisms in a complementary manner in order to satisfy both requirements. We define two modes of operation for the packet scheduler, namely, the *light* and *heavy* load modes[1].

a. *Lightly loaded state*: When the total instantaneous input traffic is less than or equal to the server capacity, there is no need for packets to be dropped. In this case, each class obviously receives all the throughput it has requested. Since the system is work-conserving, throughput control is neither needed nor possible under this condition. However, the delays of the classes may be controlled in this stage by controlling the queue sizes. Therefore, when the system is lightly loaded, the queue control mechanism is called upon to choose

[1]We use the terms light and heavy loads here in an informal context in order to define two different modes of operation.

the class to be served, therefore satisfying the queue ratios.

b. *Heavily loaded state*: When the input traffic to the server is greater than the server capacity, some packets will need to be discarded. In this case, the throughputs of all classes are controlled by calling the throughput control mechanism which works to maintain the necessary throughput ratios.

When the number of arrivals to the system within a predefined, discrete time frame, is less than a predefined packet threshold, the system is considered lightly loaded; otherwise it is heavily loaded.

3.4.3 The Queue Manager: Mechanisms for packet arrival

To complete the mechanism, we must decide on how to handle packet arrivals. Notice that the arrival of a packet from a class changes the associated queue ratios (and hence delay ratios).

a. *Instantaneous light load:* When the system is lightly loaded, it corresponds to the buffer state being non-full. Since the system is work-conserving, queue control (by dropping packets to adjust queue ratios) is not a feasible option in this state. Recall that in this state, the packet scheduler works to control queue ratios proportionally. So in the lightly loaded state, delay control is achieved by the packet scheduler. Further, it is impossible to control packet arrivals.

b. *Instantaneous heavy load:* When the buffer is full, the system is considered to be in a state of instantaneous heavy load. In this case, the packet scheduler does not control delay, but works to control throughput. In this case packet arrivals can be controlled, as explained below. When an arriving packet faces a full buffer, multiple queuing decisions are possible: *an arriving packet may be dropped, or it may be accepted by discarding an already-queued packet from one of the other classes.* This decision must be made with a view to satisfying the delay ratios. The queue manager performs this function.

In this case, we use an active queue management scheme to achieve combined proportional differentiation. Deviations of the queue ratios from the ideal are computed for each of the above-mentioned decisions. That decision is chosen for which deviation of the queue ratios from the ideal is optimal, using the min-max optimality criterion alluded to earlier.

3.5 Formal Description of the Strategy

This section presents the pseudocode for the modules of our combined proportional differentiation strategy. In this pseudocode, a set of pairwise parameter ratio offsets for N classes is defined as:

$$\Delta(x)_i = \frac{x_i}{x_{(i+1) \bmod N}} - \frac{X_i}{X_{(i+1) \bmod N}} \tag{3.3}$$

for $0 \leq i < N$, where the argument x can take the value q, s and w for the queue length, the throughput, or the mean delay, respectively, while X corresponds to x's target weight. Note that X can be a fixed weight, such as in delay and throughput, or a computed weight, such as in the case of queue length.

3.5.1 Selection of operational mode of Packet Scheduler

Every time a packet is to be served, the system state must be determined as follows:

```
if number of packet arrivals in window ≤ packet-threshold,
    SYSTEM-STATUS = LIGHT
else
    SYSTEM-STATUS = HEAVY
```

3.5.2 Packet Scheduler for Departing Packets

If a packet is to be served, it is chosen from the class that will minimize the largest difference between the actual, and the target queue ratios under light load. However, under heavy load, it is chosen from the class that will minimize the largest difference between the actual, and the target throughput ratios.

```
if SYSTEM-STATUS = LIGHT
  If (exactly two non-empty queues i and j )
    If ( (q_i = 1) OR (q_j = 1) )
      If ( W_i = W_j )
        k = random(i, j)
      else
        Compute W_k = min(W_i, W_j)
    Serve a packet from class k
  else
  For each class i
    compute Δ(q)_i|_{q_j=q_j-1}
  find class j for which
    max_{0≤i<N}{Δ(q)_i|_{q_j=q_j-1}} ≤ max_{0≤i<N}{Δ(q)_i|_{q_l=q_l-1}}, for j ≠ l
  Serve a packet from class j
```

```
else if SYSTEM-STATUS = HEAVY
   For each class i
      compute Δ(s)_i|_{q_j=q_j-1}
   find class j for which
      max_{0≤i<N}{Δ(s)_i|_{q_j=q_j-1}} ≤ max_{0≤i<N}{Δ(s)_i|_{q_l=q_l-1}}, for j ≠ l
    Serve a packet from class j
```

3.5.3 Queue Manager for Arriving Packets

When a packet arrives and the buffer is full, it is decided whether to discard that packet, or queue it and discard a packet from another class depending on the action that will minimize the largest difference between the actual queue ratios and the target ones.

```
      /* j = class of incoming packet */
if buffer = NOT-FULL
   accept incoming packet
else
   for class i
         compute Δ(q)_i|_{q_j=q_j}
   for class k
         compute Δ(q)_i|_{q_j=q_j+1,q_k=q_k-1}
   find the queuing decision and class j for that decision in which:
         max_{0≤i<N}{Δ(q)_i|_{q_j=q_j}, Δ(q)_i|_{q_j=q_j+1,q_k=q_k-1}, ∀k ≠ j} ≤
         max_{0≤i<N}{Δ(q)_i|_{q_l=q_l}, Δ(q)_i|_{q_l=q_l+1,q_k=q_k-1}, ∀k ≠ l}
   Apply the chosen queueing decision.
```

3.6 Summary

In this chapter, two separate and independent schemes were introduced, one each for delay and throughput differentiation. A simple mechanism to combine them completes the strategy. Delay differentiation is observed to be possible under all loads; however, there are limitations to the achievable delay ratios. Throughput differentiation is feasible only under certain conditions. The chapter also presents mathematical bounds that justify these achievable ratios and feasibility requirements.

CHAPTER 4 PERFORMANCE EVALUATION AND SIMULATION RESULTS

4.1 Introduction

This chapter presents several scenarios and results to demonstrate the effectiveness of the scheme. All results are based on a simulator developed by the author, and written in C. Section 4.2 shows the system performance under Markov Modulated Poisson Process (MMPP) input traffic. Results verifying the bounds presented in Chapter 3 are included. Section 4.3 shows similar results with the input traffic being governed by a heavy tailed Pareto distribution, which is used to describe standard Internet traffic. Finally, Section 4.4 demonstrates the applicability of the scheme in an end-to-end environment.

4.2 Input traffic with MMPP distribution

4.2.1 The MMPP model

In this section, several numerical examples are provided to show the effectiveness of the scheme with *MMPP (Markov Modulated Poisson Process)* input traffic. The MMPP model, explained in [33], is a well known distribution commonly used to model voice and telephony traffic. In this work, MMPP is used as a general model that can encompass other traffic models. An MMPP consists of a number of states, where the duration of each state is exponentially distributed. Transitions between the states are memoryless and independent. During state k, packets are generated according to a Poisson process with rate λ_k. This process can be used to model a number of other processes, for example, an aggregation of N independent and identically distributed ON-OFF modulated Poisson sources. In this section, we assume three such ON-OFF modulated Poisson processes, which generate traffic from three different classes, respectively.

Consider a single server which is fed by three such packet streams. In simulations, all packets from all streams are assumed to require the same transmission time, which is exponentially distributed with a unity mean. The buffer at the server can accommodate a maximum of

100 packets, and the sliding window size (M), which is used for throughput measurement, is 100 time units. The mean ON and OFF periods are exponentially distributed, and can be used to control the burstiness of the incoming traffic. For maximum effectiveness, two packet thresholds are used, one each for ON and OFF states. In this simulation setup, 1000 packets in 60 time units is the threshold used when the system in ON; 60 packets in 60 time units is the threshold in the OFF state. If the number of packets arriving within a window exceeds this threshold, then the system is considered to be in the heavy load state; otherwise it is in the light load state.

Delay control is achieved in all scenarios, regardless of input load. Further, during heavy load periods in any scenario, throughputs are limited by their weights; during light load periods, throughput is equal to the offered load.

4.2.2 Results for Light, Heavy and Mixed loads

The first scenario considered is the heavy traffic case, where the system is almost always in the heavily loaded state. This is the case in which the scheme should be able to achieve full control over the throughput and mean delay ratios. The traffic from the three streams is adjusted such that their offered loads are 1, 1 and 2 Erlang, respectively. The throughput weights are 2, 2 and 1, while the mean delay weights are 2, 1 and 1, respectively. Mean ON time is 100 time units, and mean OFF time is 1 time unit. Table 4.1 shows both the target and achieved ratios for throughput and delay.

Table 4.1 MMPP input traffic: Throughput and Delay ratios for a Heavily loaded system

Throughput Ratio	Target	Measured	Delay Ratio	Target	Measured
s_1/s_2	1.0	1.003	$\overline{w_1}/\overline{w_2}$	2.0	2.048
s_2/s_3	2.0	1.96	$\overline{w_2}/\overline{w_3}$	1.0	0.968
s_3/s_1	0.5	0.507	$\overline{w_3}/\overline{w_1}$	0.5	0.504

The achieved values are very close to the target values, with maximum errors of 2% in the throughput ratios, and 3% in the mean delay ratios. Figure 4.1(a) plots the instantaneous and cumulative throughput ratios over a total duration of 1000 packet transmission times. The ratio measures are from throughput and delay values collected over the sliding window. The instantaneous and cumulative mean delay ratios, measured over the same interval, and using

the same sliding window, are also shown in Figure 4.1(b). In the figures, the ideal values are represented in braces in the legend.

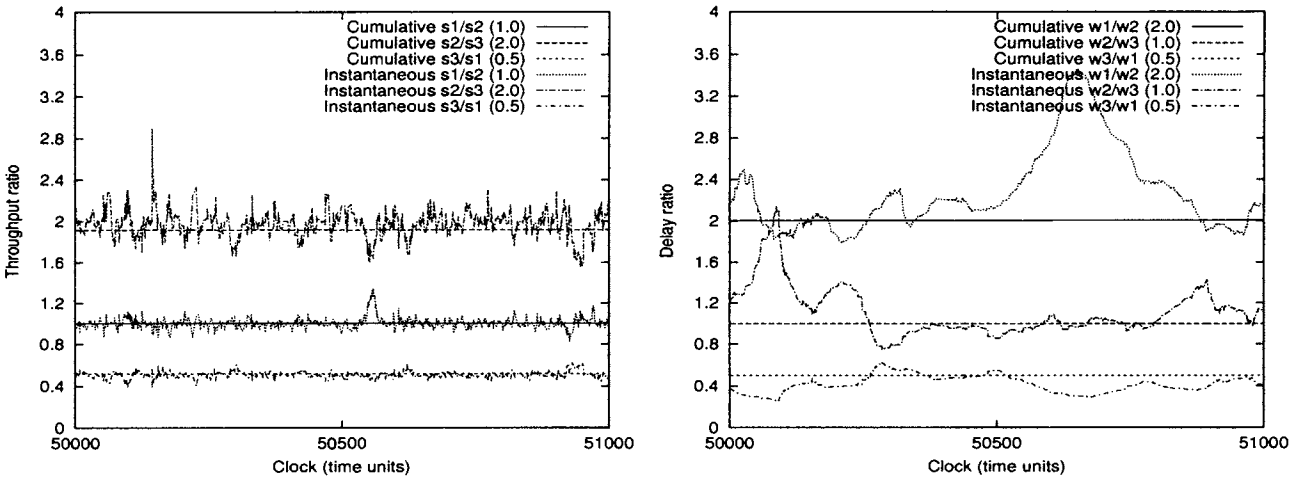


Figure 4.1 MMPP input traffic: (a) Throughput and (b) Delay ratios in a Heavily loaded system

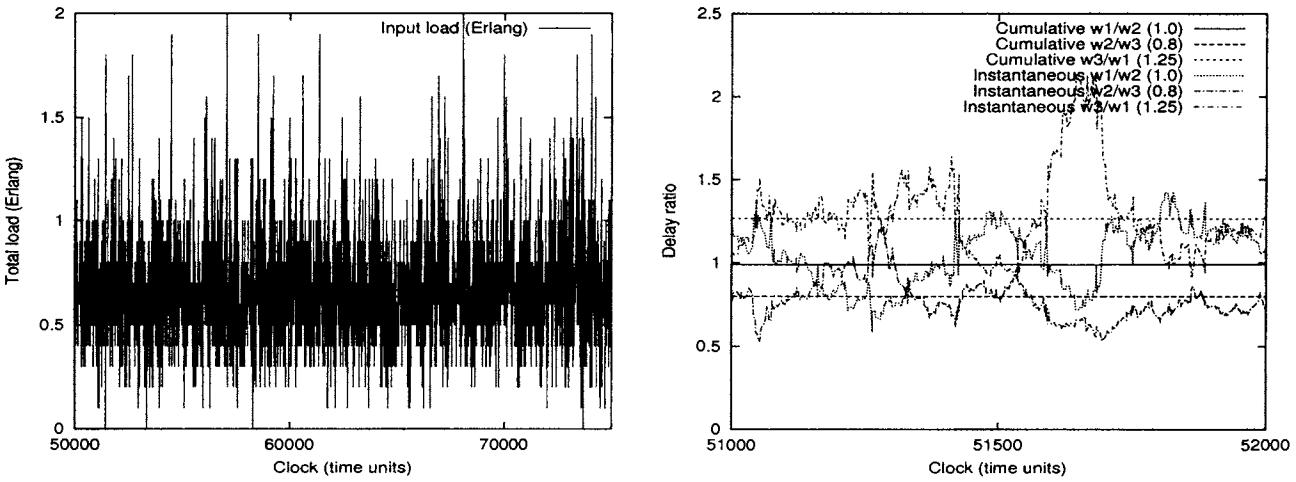
Note that, for both delay and throughput, the cumulative value graph is constant and very close to the ideal, and the maximum deviations of instantaneous ratios from the ideal are reasonable [1]. Such fluctuations occur due to the lack of packets from one or more flows at a particular instant. However, these fluctuations are corrected as soon as packets from the under-served flow arrive.

The second scenario deals with light traffic, where the system is almost always lightly loaded. The three streams generate Poisson traffic with aggregate loads of of 0.1, 0.3 and 0.2 Erlang, for a total of 0.6 Erlang. Incoming traffic is not very bursty, with mean ON times of 10 time units and mean OFF times of 120 time units. Figure 4.2(a) shows a snapshot of input load, between 50,000 and 75,000 time units.

In this case, the simulation results show that the queue size never exceeds 20 packets, therefore no packet losses are ever encountered. The individual stream throughputs are equal to the offered load, and therefore cannot be controlled. However, setting the target mean delay weights to 1, 1 and 1.25, our scheme was able to control the mean delay ratios, as shown in Table 4.2. The error in the ratios is less than 3%. Note that the traffic is uneven, with different input loads, and a higher weight is assigned to a class with lower input traffic.

Also, the graphs of the cumulative mean delay ratios were almost flat and close to the ideal,

[1] Comparing these fluctuations to those produced by the WTP and BPR algorithms in [23]



(a) (b)

Figure 4.2 MMPP input traffic : (a) Total input traffic and (b) Delay ratios in a Lightly loaded system

Table 4.2 MMPP input traffic : Delay ratios in a Lightly loaded system

Delay Ratio	Target	Measured
$\overline{w_1}/\overline{w_2}$	1.0	0.976
$\overline{w_2}/\overline{w_3}$	0.8	0.812
$\overline{w_3}/\overline{w_1}$	1.25	1.261

as shown in Figure 4.2(b). Note that instantaneous delay ratio graph for the light traffic case exhibit more frequent sharp transitions than in the heavy traffic case (Figure 4.1(b)) and also larger variations from the cumulative value. This is due to the fact that the idle-system case, when queues are momentarily empty, occurs much more frequently when the system is lightly loaded.

The third scenario shows the effectiveness of the scheme in mixed loads, when the system alternates between periods of heavy load and underloaded states, which are generated using the MMPP process. The three classes offer average loads of 0.4, 0.4 and 0.85 Erlang during the heavy load periods, and average loads of 0.2, 0.2 and 0.425 Erlang during the light load, or underloaded periods. The durations of the heavy and light load periods were exponentially distributed with a mean of 2,000 time units. The length of the simulation run was 300,000 time units. Figure 4.3 shows a snapshot of the input load in the period of 50,000 to 75,000 time units.

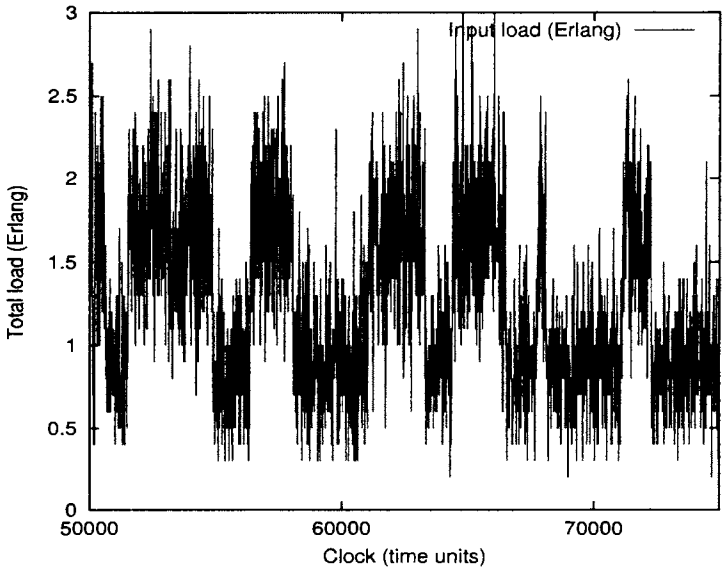


Figure 4.3 MMPP input traffic : a snapshot of total input traffic in a Mixed load system

The target delay weights are set as 1.0, 1.0 and 1.25, which are to be enforced over the entire simulation run. The target throughput weights are set to 1.2,1.0 and 1.4, and are enforced under periods of heavy load. Under the underloaded system condition, throughput is equal to the offered load. Table 4.3 shows the average delay ratios for the heavy and light load periods, as well as the overall delay ratios. The overall averages are reasonably close to the target ratios, and so are the light load period ratios. The average ratios for the heavy period show somewhat larger deviations from the target.

Table 4.3 MMPP input traffic : Delay ratios in a Mixed load system

Delay Ratio	Target	Heavy period Avg.	Light period Avg.	Overall Avg.
$\overline{w_1}/\overline{w_2}$	1.0	0.78	0.93	0.84
$\overline{w_2}/\overline{w_3}$	0.8	0.73	0.77	0.83
$\overline{w_3}/\overline{w_1}$	1.25	1.74	1.38	1.42

Table 4.4 shows similar results for throughput. Note that very precise control is achieved over the ratios in the heavy load phase. During the light load duration, no throughput control is exercised, and all incoming traffic is served. Throughput ratios in this phase are the same as that for the incoming traffic.

Table 4.4 MMPP input traffic : Throughput ratios in a Mixed load system

Throughput Ratio	Target (for heavy load)	Heavy period Avg.	Light period Avg.	Overall Avg.
s_1/s_2	1.2	1.19	0.99	1.11
s_2/s_3	0.71	0.71	0.47	0.59
s_3/s_1	1.16	1.17	2.1	1.50

It is important to point out that, for the lightly loaded system, where no control over the throughput is possible, this scheme reduces to a new implementation of *Proportional Delay Differentiation.*

4.2.3 Effect of Distribution of the Offered Load

This section demonstrates the simulator's capacity to handle variations in distribution of the load. Unlike the limitations of some simulators[2], this scheme works well under varying load conditions. To demonstrate this, the following scenarios are presented:

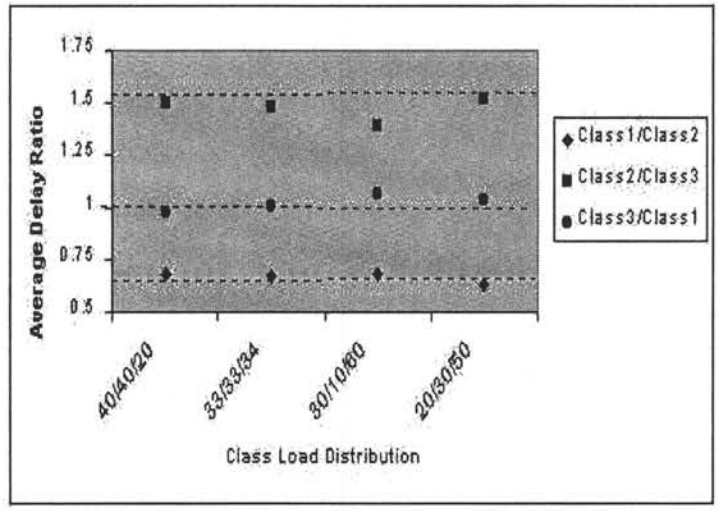


Figure 4.4 MMPP, Unsaturated system: Delay ratios as functions of load distribution

[2]Reference [23] presents results for the BPR proportional delay scheduler, in which uneven loads are shown to strongly and adversely affect the achieved ratios.

a. *Unsaturated system: Effect of class load distribution on delay*

Figure 4.4 shows the achieved delay ratios when the system is unsaturated. All data is for a total load of 96% (unsaturated) and with delay weights of 1.0, 1.6, and 1.0. Throughput weights are irrelevant, since throughputs cannot be controlled in an unsaturated system. Different distributions of the total load among the different classes were considered, namely, 40/40/20, 33/33/34, 30/10/60, and 20/30/50. It is seen that delay correction is uniformly accurate, regardless of the distribution of the input traffic. The maximum deviation from the ideal is in the case when the input loads of the classes are distributed as 30/10/60, i.e., when the class loads show a large difference. Such a deviation from the ideal is, however, still limited.

b. *Saturated system: Effect of class load distribution on delay and throughput*

For this scenario, it is not very useful to take the same approach as in the unsaturated case, as explained below. Consider the following scenario: Throughput weights of 1.0, 1.0 and 1.2 for the three classes. These numbers are chosen to generate fairly small ratios. Proposition 2 places restrictions on the achievable throughput ratios depending on the input load, so an arbitrary combination of input load cannot be expected to satisfy the throughput weights. Given this, it is not useful to take the approach taken for the unsaturated case, of choosing class input loads as a percentage of a total load. Instead, different class loads are chosen so as to keep the total input load as 2 Erlang, while maintaining each class load as at least 0.5 Erlang. Further, delay weights are taken as 1.0, 1.0, and 1.8.

The following 4 scenarios have been chosen to demonstrate combined delay and throughput differentiation:

- Class 1: 0.75 Erlang; Class 2: 0.5 Erlang; Class 3: 0.75 Erlang
- Class 1: 0.9 Erlang; Class 2: 0.5 Erlang; Class 3: 0.6 Erlang
- Class 1: 0.5 Erlang; Class 2: 0.5 Erlang; Class 3: 1.0 Erlang
- Class 1: 0.667 Erlang; Class 2: 0.667 Erlang; Class 3: 0.667 Erlang

Delay and throughput differentiation results are presented in Figure 4.5 and Figure 4.6 respectively. The graphs show that the combined delay and throughput differentiation is uniformly achievable in all these cases. As before, the maximum deviation from the ideal is in the case where the class loads are 0.5, 0.5, and 1.0 Erlang respectively, which is

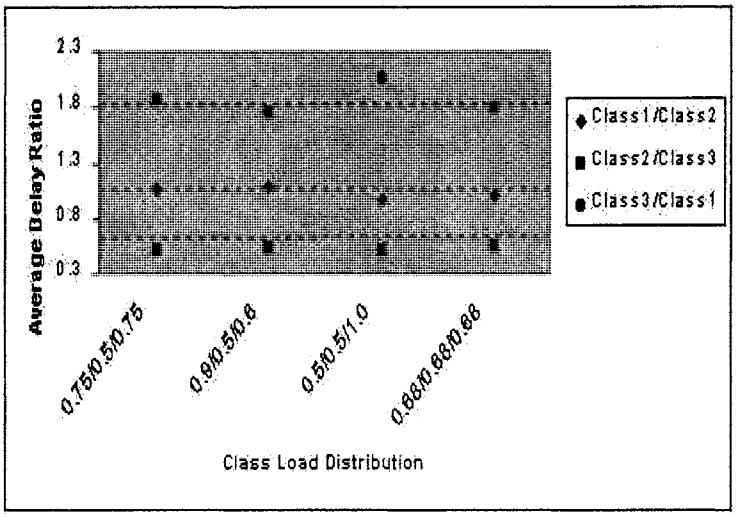


Figure 4.5 MMPP, Saturated system: Delay ratios as functions of load distribution

the case with the most divergent loads. Note that if both weight sets are taken together, Class 3 requires both high delay and high throughput. This, however, is satisfied by our scheme.

4.2.4 Effect of variations in the Offered Load

This section presents results for delay and/or throughput differentiation with varying loads. Typically [3], delay ratios get closer to ideal as the load increases and the system saturates, with moderate loads (75-90 %) unable to offer close correction.

Similar to the previous section, two cases, saturated and unsaturated, are considered.

a. *Unsaturated system: Effect of class load on delay*

For this scenario, total system loads of below 100 % were considered, making the system unsaturated. Throughput control is not achievable in this state. Hence, delay differentiation is targeted, with delay weights chosen as 1.4, 1.0, and 1.0. In all cases, the individual class loads are distributed as 1:1:1.5 of the total system load. The nature of the load is fairly uniform, with ON periods as 10 time units, and OFF periods of 0.1 time units. Figure 4.7 shows the results of the delay differentiation over several system loads

[3]Proportional delay differentiation schemes presented in [23] show up to 30 % variation from ideal, under moderate loads of 75 % utilization.

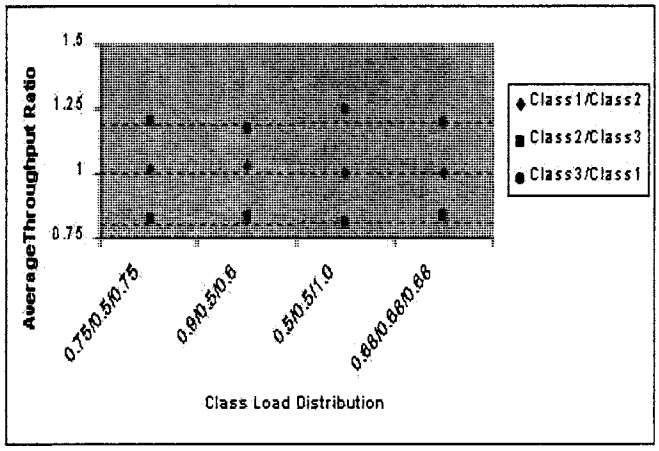


Figure 4.6 MMPP, Saturated system: Throughput ratios as functions of load distribution

of 75%, 80%, 85%, 90%, 95%, and 99.9%. Since the system is unsaturated, the average system utilization in each of these cases is the same as the average system load.

It is seen that the delay ratios are reasonably accurate over all class ranges. It is seen that the ratio of delay of Class 2 to Class 3 shows slight deviations from the normal, and these deviations decrease as the total load increases. These deviations range from 7% at 75% load, to 3.7% at 99.9% load.

Other ratios are well corrected under all loads. Note that ratios of the delay weights of Class 2 to Class 3 (equal to 1.4) is the highest numerical value among all the delays; the reciprocal of this ratio, equal to 0.71, is well corrected. This behavior is consistent with the observation in [23] that, as the differentiation spacing between classes increases, even while maintaining the same relative ratios to each other, the deviations of the results from the ideal also increase. That is, a set of delay weights (1.0, 2.0, 4.0), will achieve better correction than the delay weights of (1.0, 4.0, 16.0).

b. *Saturated system: Effect of class load on delay and throughput*

When the system is saturated, that is, when the system utilization is 100%, combined delay and throughput differentiation is possible. To demonstrate this, several loads exceeding system capacity were considered. Input traffic of 105%, 110%, 120%, 130%, 150%, 175%, 185% and 200% were considered, and the resulting delay and throughput ratios are presented in Figure 4.8 and Figure 4.9 respectively. In all cases, the delay

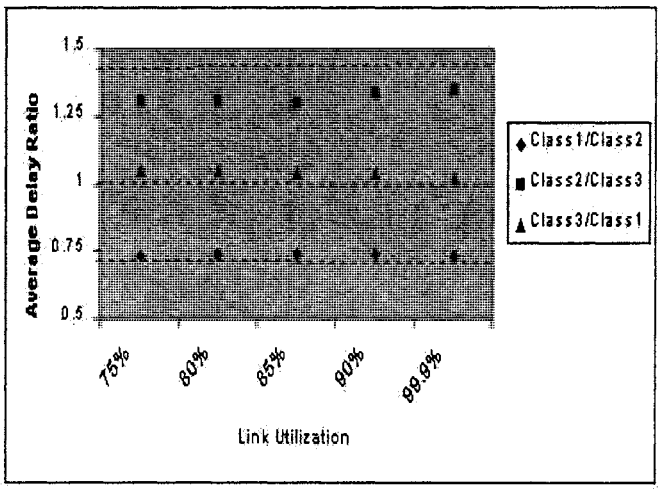


Figure 4.7 MMPP, Unsaturated system: Delay ratios as functions of utilization

weights were set as 1.0, 1.4, and 1.0; the throughput weights were set as 1.0, 1.0 and 1.2. Class loads were distributed as 1:1:1.5 for all loads, and the ON and OFF periods are the same as in the previous case.

The graphs show the correction in delay and throughput improves as the input loads increase. It is to be noted that the delay remains well corrected over all loads. Throughput ratios are observed to be slightly better corrected as the system load increases. Also, even better throughput correction can be achieved by changing the threshold (or by increasing the ON time), at the expense of delay correction. This will be explained in more detail in Section 4.2.6.

4.2.5 Validating the Delay and Throughput Bounds

This section demonstrates the validity of the delay and throughput bounds stated in Propositions 2, 3 and 4 in Chapter 3. Each of the following sections considers a scenario, derives related bounds, and shows results that prove the applicability of these bounds. There are 3 bounds:

a. *Proposition 2: Bounds for throughput ratios*

Proposition 2 states that, in case the system is unsaturated, the throughputs are uncontrollable. For a saturated system, however, the achievable throughput ratios are as

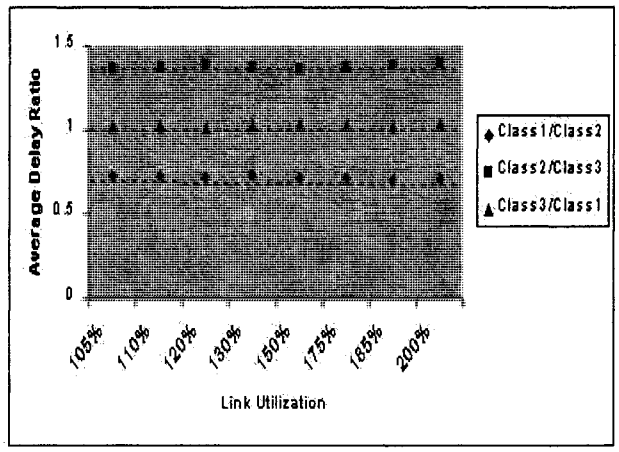


Figure 4.8 MMPP, Saturated system: Delay ratios as functions of utilization

stated. Consider a saturated system with 2 classes, having input loads of 0.7 Erlang and 0.6 Erlang respectively (Scenario 1). Let the throughput weights be 5 and 1. Then, Proposition 2 states that, in order for these throughput ratios to be achieved, Class 1 must have a minimum arrival rate of 0.833 packets per time unit. Table 4.5 shows that, as expected, Scenario 1, which has an input rate of 0.7 packets/time unit for Class 1, is unable to satisfy the throughput ratios. However, if the input load of Class 1 is increased to 0.9 Erlang (Scenario 2), the desired throughput ratio is obtained.

Table 4.5 Validating bounds: Proposition 2

	Scenario 1 Input load	Scenario 2 Input load
Class 1	0.7	0.9
Class 2	0.6	0.6
Throughput Ratio	2.29	5.39

b. *Proposition 3: Bounds for delay ratios in a unsaturated system*

If the system is unsaturated, i.e., if $\sigma_1 + \sigma_2 < 1$, then the achievable delay ratios are as stated in Proposition 3. Consider two classes with input loads of 0.2 and 0.3 Erlang respectively. Since the throughputs are uncontrollable in an unsaturated system, the throughputs of the two classes are also 0.2 and 0.3 Erlang. In this case, according to

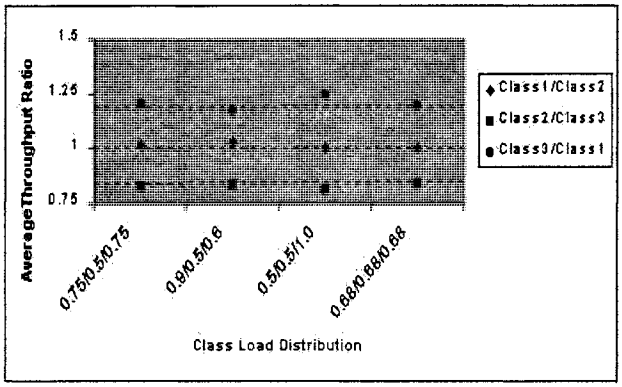


Figure 4.9 MMPP, Saturated system: Throughput ratios as functions of utilization

Proposition 3, the achievable delay is as $0.722 \leq (\overline{w_1}/\overline{w_2}) \leq 1.39$. If the delay weights of the two classes are 1.2 and 1.0 respectively (Scenario 1), the delays should satisfy these weights, such that (Class 1 delay)/(Class 2 delay) = 1.2. However, if the delay weights are increased to 2.0 and 1.0 (Scenario 2), then, according to the bounds, the delays can no longer satisfy these weights, that is, (Class 1 delay)/(Class 2 delay) cannot be 2.0 Table 4.6 verifies these conclusions.

Table 4.6 Validating bounds: Proposition 3

	Scenario 1 Delay	Scenario 2 Delay
Class 1	2.658	2.86
Class 2	2.226	2.13
Delay Ratio	1.194	1.342

c. *Proposition 4: Bounds for delay ratios in a saturated system*

Consider two classes with throughputs of 0.6 and 0.4 Erlang respectively. Then, since the system is saturated, $(\sigma_1 + \sigma_2 = 1)$ then the bounds on the achievable delay ratios are $0.014 \leq (\overline{w_1}/\overline{w_2}) \leq 70.3$. To show the applicability of these bounds, two cases are considered. If the delay weights are taken as 50.0 and 1.0 (Scenario 1), they are within

the bounds and must be satisfied. If the weights are 90, 1 (Scenario 2), they will no longer be satisfied. Table 4.7 shows the results obtained.

Table 4.7 Validating bounds: Proposition 4

	Scenario 1 Delay	Scenario 2 Delay
Class 1	146.311	140.89
Class 2	3.325	2.056
Delay Ratio	44.0	68.498

It must be noted that this condition of precise saturation ($\sigma_1 + \sigma_2 = 1$) is difficult to achieve, due the bursty nature of arriving traffic and dropped packets.

4.2.6 Importance of the Threshold Parameter

The threshold parameters used in the scheme play a very important role in the effectiveness of the scheme. Two distinct strategies are used to combine the two metrics of delay and throughput; the system must switch between these two strategies in a way that optimizes both delay and throughput correction. The threshold parameters are the means to do this. In all the simulation scenarios presented so far, the same thresholds (1000 and 60 packets in 60 time units, for OFF and ON periods, respectively) have been used. These values were arrived at by trial and error, and are justified by the following rationale:

a. During the OFF period, the system will experience a lower level of input traffic, and should mostly be in the lightly loaded state, as defined in Section 3.4.2. (Recall that, when the system is (instantaneously) lightly loaded, packets are served with a view to satisfying their delay ratios.) In order to drive the system to the instantaneous light state, the threshold must be high.

b. During the ON period, the arriving traffic will have shorter interarrival times, and should mostly be in the heavily loaded state. During this phase, packets are served such that the throughput ratios move towards the target ratio. The threshold is set lower for the ON state, in order to ensure that the threshold is crossed and the heavily loaded state is achieved.

It is important to note that the choice of thresholds significantly affects the precision of delay and throughput ratio correction. To demonstrate this, consider results extracted from

Table 4.8 Effect of threshold parameters: Throughput and Delay ratios with ON, OFF periods of 10, 0.1

Throughput Ratio	Target	Measured	Delay Ratio	Target	Measured
s_1/s_2	1.0	0.960	$\overline{w_1}/\overline{w_2}$	0.714	0.719
s_2/s_3	0.833	0.701	$\overline{w_2}/\overline{w_3}$	1.4	1.364
s_3/s_1	1.2	1.485	$\overline{w_3}/\overline{w_1}$	1.0	1.018

the discussion in Section 4.2.4. Table 4.8 shows the delay and throughput ratios under a total input of of 130 %. The delay weights are set as 1.0, 1.4, and 1.0, and the throughput weights are set as 1.0, 1.0 and 1.2. Class loads are distributed as 1:1:1.5, and ON and OFF periods are 10 and 0.1 time units respectively (Scenario 1). Threshold parameters of 1000 and 60 are used. Table 4.9 shows the results for the same scenario, with the same aggregate system load, but with ON and OFF periods of 100 and 1 time units (Scenario 2). It is observed that Scenario 1 offers a maximum error in the delay ratios of 2.5 %, while maximum error in the throughput ratios is 23.75 %. In Scenario 2, the maximum deviation of the delay ratios from ideal is 20 %, while the corresponding figure for the throughput ratios is 5.5 %. This is because in Scenario 1, the ON period of 10 time units may not be sufficiently long for the heavy state to occur. (The system state is considered heavy when 60 or more packets arrive in 60 time units.) This results in the under-correction of the throughput ratios in Scenario 1. On the other hand, the system state is set as heavy load more often in Scenario 2, where the ON period (with heavy arrivals) is 100 time units. This results in a better correction of the throughput ratios. A similar argument can be made for the delay ratios.

It is obvious that there is a trade off between achieving accuracy in the delay and throughput metrics. The nature of the input traffic, especially the lengths and sizes of bursts, strongly influence the accuracy of the average delay and throughput ratios. A set of thresholds that produce optimal results for a certain traffic pattern need not necessarily do the same for other traffic profiles. Further, the user or network operator may have different application dependent goals, preferring more precise control over delay rather than throughput, or *vice versa*. For all these reasons, threshold parameters are a significant design parameter. An algorithm to dynamically adjust thresholds, based on the nature and burstiness of the incoming traffic, will be needed to produce optimal results over all traffic patterns.

Table 4.9 Effect of threshold parameters: Throughput and Delay ratios with ON, OFF periods of 100, 1

Throughput Ratio	Target	Measured	Delay Ratio	Target	Measured
s_1/s_2	1.0	0.986	$\overline{w_1}/\overline{w_2}$	0.71	0.663
s_2/s_3	0.833	0.801	$\overline{w_2}/\overline{w_3}$	1.4	1.25
s_3/s_1	1.2	1.266	$\overline{w_3}/\overline{w_1}$	1.0	1.2

4.3 Pareto Modulated Input Traffic

4.3.1 The Pareto Distribution

It is a well known fact [34], [35] that the bursty nature of Internet traffic may be modeled by a heavy tailed distribution. A heavy tailed distribution can be simply understood as one in which most bursts are small, but there are a few bursts that are very large in size. Intuitively, one scenario which may produce this behavior is a user visiting many web sites, occasionally downloading an image, and even more rarely, downloading some software. One model proposed for the distribution of burst sizes in web transfers is the Pareto distribution, where that the probability that a given burst size is larger than x bytes is $(1/x)^b$, where b is the shape parameter of the Pareto distribution, for $x \geq a$.

This section verifies the feasibility of this scheme for a standard Pareto distribution. The interarrival times of the input traffic during the ON and OFF periods are still modeled as Poisson, as in the previous section. However, the ON and OFF periods (which are analogous to burst sizes) are modeled as a Pareto distribution, and the results are presented in the following sections.

4.3.2 Results for Light, Heavy and Mixed loads

This section presents results for heavy, light, and mixed loads, when the ON and OFF periods are modeled as a Pareto distribution. The simulation set-up is substantially the same as in Section 4.2.2. Packet lengths are exponentially distributed with a mean of 1 time unit, the buffer size is taken as 100 packets, and the size of the sliding window for throughput measurements is 100. Packet thresholds are 1000 packets in 60 time units when the system in the ON state, and 60 packets in 60 time units when the system is in the OFF state. If the number of packets arriving within a window exceeds this threshold, then the system is

Table 4.10 Pareto Modulated input traffic: Throughput and Delay ratios for a Heavily loaded system

Throughput Ratio	Target	Measured	Delay Ratio	Target	Measured
s_1/s_2	1.0	1.004	$\overline{w_1}/\overline{w_2}$	2.0	2.033
s_2/s_3	2.0	1.94	$\overline{w_2}/\overline{w_3}$	1.0	0.992
s_3/s_1	0.5	0.513	$\overline{w_3}/\overline{w_1}$	0.5	0.495

considered to be in the heavy load state; otherwise it is in the light load state. A Pareto shape parameter of 5.0 is used.

The first case taken here is the heavy traffic case, when the system is almost always in the heavy state. This is the case in which the scheme should be able to achieve full control over the throughput and mean delay ratios. The traffic from the three streams is set to be 1, 1 and 2 Erlang, respectively. The throughput weights are 2, 2 and 1, while the mean delay weights are 2, 1 and 1, respectively. Mean ON time is 100 time units, and mean OFF time is 1 time unit. Table 4.10 shows both the target and achieved ratios for throughput and delay.

The achieved values are very close to the target values, and similar in accuracy to the results obtained for MMPP traffic. Figure 4.10 plots the instantaneous and cumulative throughput ratios over a total duration of 1000 packet transmission times. In the figure, the ideal values are represented in braces in the legend. Note that the deviation of the instantaneous values from the ideal (or cumulative) values are very reasonable.

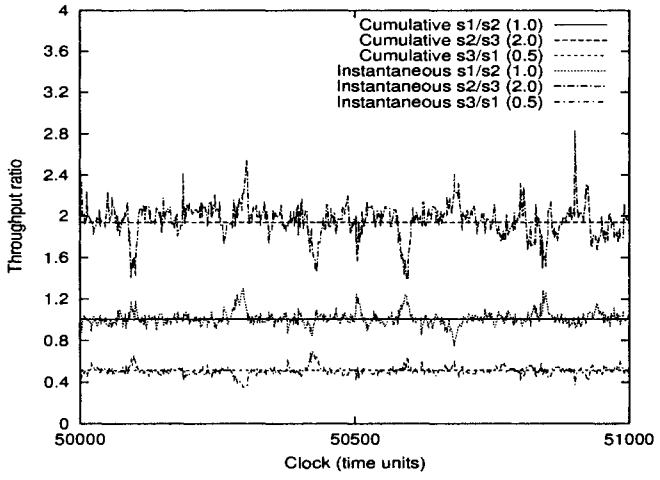


Figure 4.10 Pareto Modulated input traffic : Throughput ratios in a Heavily loaded system

The second scenario is under light traffic, where the system is almost always lightly loaded. The three streams generate Poisson traffic at rates of 0.1, 0.3 and 0.2 Erlang, for a total of 0.6 Erlang. Mean ON times are 10 time units and mean OFF times are 120 time units. The queue size does not exceed a maximum of 29 packets, and no packets are dropped. The individual stream throughputs are equal to the offered load, and therefore cannot be controlled. However, setting the target mean delay weights to 1, 1 and 1.25, the scheme was able to control the mean delay ratios precisely, as shown in Table 4.11.

Table 4.11 Pareto Modulated input traffic : Delay ratios in a Lightly loaded system

Delay Ratio	Target	Measured
$\overline{w_1}/\overline{w_2}$	1.0	1.008
$\overline{w_2}/\overline{w_3}$	0.8	0.784
$\overline{w_3}/\overline{w_1}$	1.25	1.264

The third scenario shows the usefulness of the scheme in mixed loads, when the system alternates between periods of heavy load and underloaded states. The three classes offer average loads of 0.4,0.4 and 0.85 Erlang during the heavy load periods, and average loads of 0.2,0.2 and 0.425 Erlang during the light load, or underloaded periods. The ON and OFF periods were generated with a Pareto distribution, and have a mean of 2000 time units each. Figure 4.11 shows a snapshot of the input load in the period of 50,000 to 75,000 time units.

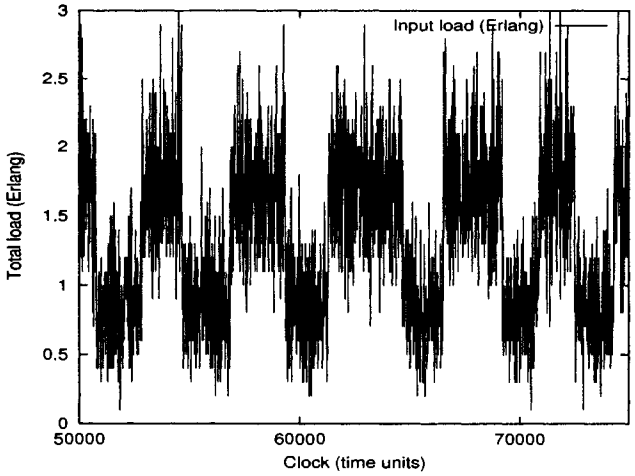


Figure 4.11 Pareto Modulated input traffic : a snapshot of total input traffic in a Mixed load system

The target delay weights are set as 1.0, 1.0 and 1.25, which will be achieved over the entire simulation run. The target throughput weights are set to 1.2, 1.0 and 1.4, and will be enforced under periods of heavy load. In the underloaded system condition, throughputs are uncontrollable and equal to the offered load. Table 4.12 shows the average delay ratios for the heavy and light load periods, as well as the overall delay ratios.

The overall averages are reasonably close to the target ratios, as do the ratios during light load periods. The average ratios for the heavy period show somewhat larger deviations from the target.

Table 4.12 Pareto Modulated input traffic : Delay ratios in a Mixed load system

Delay Ratio	Target	Heavy period Avg.	Light period Avg.	Overall Avg.
$\overline{w_1}/\overline{w_2}$	1.0	0.76	0.93	0.82
$\overline{w_2}/\overline{w_3}$	0.8	0.74	0.76	0.83
$\overline{w_3}/\overline{w_1}$	1.25	1.78	1.38	1.38

Table 4.13 shows the results for throughput. In the heavy load phase, throughput ratios are satisfied very accurately. During the light load duration, no throughput control is exercised, and all incoming traffic is served. Throughput ratios in this phase are the same as that for the incoming traffic.

Table 4.13 Pareto Modulated input traffic : Throughput ratios in a Mixed load system

Throughput Ratio	Target (for heavy load)	Heavy period Avg.	Light period Avg.	Overall Avg.
s_1/s_2	1.2	1.19	0.99	1.11
s_2/s_3	0.71	0.71	0.47	0.59
s_3/s_1	1.16	1.17	2.1	1.52

The results presented here clearly show that the scheme is equally effective for the well known Pareto distribution.

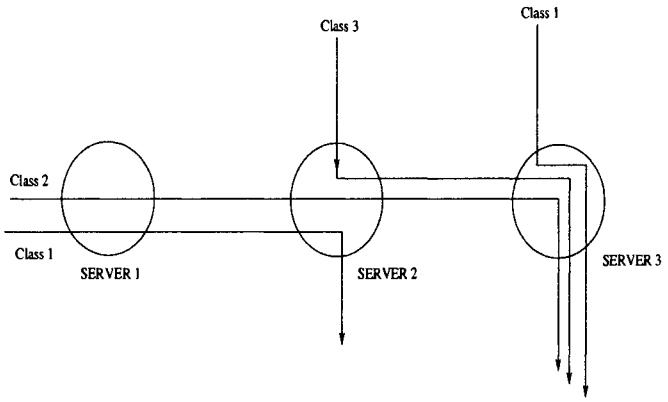


Figure 4.12 An example Multi hop topology

Table 4.14 Throughput and Delay weights for the 3 server Multi hop topology

		Throughput Weight	Delay Weight
Server 1	Class 1	1.25	1.0
	Class 2	1.0	1.6
	Class 3	1.0	1.0
Server 2	Class 1	1.0	1.0
	Class 2	1.0	1.4
	Class 3	2.0	1.0
Server 3	Class 1	2.0	1.0
	Class 2	1.0	1.0
	Class 3	2.0	1.0

4.4 A Network Scenario: Proportional Differentiation over Multiple hops

This chapter has so far dealt with our proposed scheme in a single server environment with 3 classes of traffic. The set of strategies proposed are meant to implement specific per hop behaviors (combined delay and throughput control) at a router. However, for a end-to-end connection to perceive the benefits of this scheme, this behavior will need to translate to effective performance on a end-to-end basis. This section validates the use of this scheme on an end-to-end network.

Figure 4.12 describes a simple end-to-end network with 3 servers, each of which has the same capability of the single server described in the previous sections. The class throughput

Table 4.15 Throughput and Average Delay values for the 3 server Multi hop topology

		Throughput	Avg. Delay
Server 1	Class 1	$s_1 = 0.558$	$\overline{w_1} = 77.24$
	Class 2	$s_2 = 0.439$	$\overline{w_2} = 122.62$
	Class 3	s_3 = -	$\overline{w_3}$ = -
Server 2	Class 1	$s_1 = 0.268$	$\overline{w_1} = 111.0$
	Class 2	$s_2 = 0.255$	$\overline{w_2} = 142.38$
	Class 3	$s_3 = 0.474$	$\overline{w_3} = 85.82$
Server 3	Class 1	$s_1 = 0.469$	$\overline{w_1} = 98.74$
	Class 2	$s_2 = 0.186$	$\overline{w_2} = 114.38$
	Class 3	$s_3 = 0.342$	$\overline{w_3} = 112.52$

Table 4.16 Throughput and Delay ratios for a 3 server Multi hop topology

	Throughput Ratio	Target	Measured	Delay Ratio	Target	Measured
Server 1	s_1/s_2	1.25	1.27	$\overline{w_1}/\overline{w_2}$	0.625	0.629
	s_2/s_3	1.0	-	$\overline{w_2}/\overline{w_3}$	1.6	-
	s_3/s_1	0.8	-	$\overline{w_3}/\overline{w_1}$	1.0	-
Server 2	s_1/s_2	1.0	1.04	$\overline{w_1}/\overline{w_2}$	0.714	0.779
	s_2/s_3	0.5	0.54	$\overline{w_2}/\overline{w_3}$	1.4	1.65
	s_3/s_1	2.0	1.77	$\overline{w_3}/\overline{w_1}$	1.0	0.77
Server 3	s_1/s_2	2.0	2.51	$\overline{w_1}/\overline{w_2}$	1.0	0.863
	s_2/s_3	0.5	0.55	$\overline{w_2}/\overline{w_3}$	1.0	1.016
	s_3/s_1	1.0	0.73	$\overline{w_3}/\overline{w_1}$	1.0	1.139

weights and delay weights chosen by each server are shown in Table 4.14. The classes offer heterogeneous loads, with Class 1 offering an aggregate load of 0.7 Erlang, Class 2 offering 0.8 Erlang, and Class 3 offering 0.9 Erlang. The classes have the same characteristic input traffic across all servers. Table 4.15 shows the actual delay and throughput values after the traffic exits each of the servers. Note that, out of the 0.439 Erlang of Class 2 traffic that leaves Server 1 (and enters Server 2), only 0.255 Erlang leaves Server 2. This is because of the queue manager at Server 2, which drops arriving packets when faced with a full buffer. When the system is saturated, packets are dropped in order to maintain the desired queue ratios. Similar observations can be made regarding Class 2 and Class 3 traffic leaving Server 2 and entering Server 3. Table 4.16 shows the delay and throughput ratios achieved, for the entire network. The achieved delay and throughput ratios are very close to the target ones.

To see the efficacy of the scheme for an end-to-end topology, consider that a Source *Src1* (at Server 1) is sending a stream of packets to a destination *Dst 1* (at Server 3). The source may request a delay weight of 1.2, and be sending Class 1 traffic. By setting the weight of Class 1 to be 1.2 at every server, it obvious that this can be achieved. In the case of throughput, it is sufficient to set the desired throughput weight at Server 3. Regardless of the path taken by the Src1's traffic, Dst1 will receive its weighted share of inputs into Server 2, provided the input traffic into Server 3 is adequate to satisfy the throughput weights.

Based on the above discussion, it can be observed that the scheme can implement different PHB's at different servers. Thus end-to-end delay differentiation is achievable, while the throughput differentiation depends on the traffic and the throughput weights at the different servers.

4.5 Summary

This chapter presented the results of extensive simulations of the proposed scheme. The effectiveness of the scheme under different types of input traffic including the Markov Modulated Poisson process, and the Pareto modulated Poisson process was shown. The performance of the scheme for various load distributions and varying class loads was analyzed, and the importance of the threshold parameter was explained. Finally, the feasibility of the scheme in an end-to-end network topology was presented.

CHAPTER 5 CONCLUSION

5.1 Thesis Contribution

The proliferation of resource-intensive applications in the Internet, coupled with the limited availability of network resources, has fueled interest in Quality of Service issues. Two QoS architectures have been recently proposed, Integrated Services (IntServ) and Differentiated Services (DiffServ). Both architectures address different QoS needs, with IntServ offering strict per-flow guarantees at the expense of lack of scalability and ease of deployment. On the other hand, DiffServ offers a flexible and scalable architecture by aggregating flows into a limited number of classes. Within DiffServ, several approaches have been proposed by the research community. Absolute DiffServ offers hard guarantees similar to IntServ, and Relative DiffServ offers service that is "better, or at least no worse", for higher classes over lower classes. Proportional DiffServ is a refinement of this model, and offers *predictability* and *controllability* of differentiation between classes. This work proposes a packet scheduler based on this model.

The key element in a DiffServ aware network are those router mechanisms that implement the specified per-hop behavior on streams passing through that router. This work introduces a set of schemes to achieve combined delay and throughput proportional differentiation. Previous schemes for implementing Proportional DiffServ have included mechanisms for delay differentiation, loss differentiation, throughput differentiation, and combined loss and delay differentiation. In this work, we introduce combined delay and throughput differentiation, which, to the best of our knowledge, has not been achieved before. Light-weight schemes have been proposed for both delay and throughput differentiation, based on the use of Little's Law. A simple strategy is used to switch between these schemes, to obtain combined delay and throughput differentiation.

This work also presents bounds on the feasibility limits for delay and throughput differentiation. We show that delay differentiation is feasible under all input traffic loads, and propose limits to the achievable delay ratios. It is to be noted that our scheme for delay differ-

entiation is a new implementation of Proportional Delay Differentiation. Further, we observe that throughput differentiation is not possible under all input loads, and derive conditions for feasible throughput differentiation.

Extensive simulation results show the effectiveness of our scheme in per-hop and end-to-end scenarios, as well as for different input traffic distributions. Further, the scheme avoids complexity-intensive measurement of time (for delay), and involves minimal state information to be stored. It can be concluded that the proposed scheme is an effective and light weight option for achieving combined delay and throughput proportional differentiation in the Internet.

5.2 Future Work

This thesis proposes two separate schemes for delay and throughput differentiation, and a simple threshold-based strategy to combine them. In the results section, we noted that the simple threshold may not be optimal for all traffic burstiness profiles. Interesting future avenues for enhancing our scheme will include a dynamic algorithm for choosing the best possible threshold, based on the burstiness of the incoming traffic. This will result in our scheme providing optimal delay and throughput differentiation for a wide range of traffic profiles.

Another significant area for future development is variance control for class delay. Controlling the *delay variance* and minimizing it is an additional aspect that will be particularly relevant to multimedia and real time applications, which are sensitive to delay jitter.

The construction of a mathematical performance model for this system is another future direction, which can give more insight into operation of the protocol, and allow fine tuning of its parameters.

APPENDIX A

Proposition 1: *It is not possible to achieve combined proportional differentiation in the delay and loss metrics, independent of the actual values of packet loss ratios.*

Proof: Let l_i be the fraction of lost packets for class i, and λ_i be the mean arrival rate for class i. Then the throughput for class i, s_i can be expressed as $\lambda_i(1 - l_i)$. Little's Result states that $\overline{q_i} = s_i\overline{w_i}$, which yields

$$\frac{\overline{q_i}}{\overline{q_j}} = \frac{s_i}{s_j} \cdot \frac{\overline{w_i}}{\overline{w_j}} = \frac{\lambda_i}{\lambda_j} \cdot \frac{(1 - l_i)}{(1 - l_j)} \cdot \frac{\overline{w_i}}{\overline{w_j}}$$

Let the target proportions be $\overline{w_i}/\overline{w_j} = W_i/W_j$ and $l_i/l_j = L_i/L_j$. Then the ratio $\overline{q_i}/\overline{q_j}$ must be expressible in terms of these proportions only. We prove the infeasibility of this using contradiction.

Assume these proportions can be simultaneously satisfied, then

$$\frac{\overline{q_i}}{\overline{q_j}} = \frac{\lambda_i}{\lambda_j} \cdot \frac{(1 - l_j \cdot L_i/L_j)}{1 - l_j} \cdot \frac{W_i}{W_j} \tag{5.1}$$

However, equation (5.1) depends on l_j in addition to L_i and L_j. This contradicts the assumption, proving that combined proportional delay and loss differentiation is possible, but only if the actual loss ratios are taken into account. □

APPENDIX B

Proposition 2: *The necessary and sufficient conditions for the throughput ratios to be satisfied depend on the offered load, and are as follows:*
Case 1: When $\rho_1 + \rho_2 \leq 1$, *then* λ_1/λ_2 *must equal* S_1/S_2
Case 2: When $\rho_1 + \rho_2 \geq 1$*:*

2a: if $\rho_1/\rho_2 > S_1/S_2$, *then* $\lambda_2 \geq S_2/(S_1\bar{b}_1 + S_2\bar{b}_2)$

2b: if $\rho_1/\rho_2 < S_1/S_2$, *then* $\lambda_1 \geq S_1/(S_1\bar{b}_1 + S_2\bar{b}_2)$

Proof: We first prove sufficiency by direct proof.
Case 1: Since the system is work-conserving, $\rho_1 + \rho_2 \leq 1$, implies $s_1 = \lambda_1$, $s_2 = \lambda_2$. Then, it must be that $s_1/s_2 = \lambda_1/\lambda_2 = S_1/S_2$.
Case 2: $\rho_1 + \rho_2 \geq 1$ implies that some traffic will be discarded. Further,
Case 2a: $\lambda_1/\lambda_2 > S_1/S_2$ means that $s_1 < \lambda_1$, and that at least some traffic from class 1 must be dropped. In the worst case, all Class 2 traffic will be carried, i.e., $s_2 \leq \lambda_2$, and

$$s_2 = s_1 \cdot \frac{S_2}{S_1} \leq \lambda_2 \tag{5.2}$$

Since the system is saturated, then

$$s_1\bar{b}_1 + s_2\bar{b}_2 = 1 \tag{5.3}$$

Combining equations (5.2) and (5.3), we have

$$\lambda_2 \geq \frac{S_2}{S_1\bar{b}_1 + S_2\bar{b}_2}$$

which is the required sufficient condition.

Case 2b: The proof is similar to *Case 2a*.

To prove the converse, i.e., that this is the the necessary condition, we use contradiction.

Case 1 is obvious.

Case 2: Let $\rho_1 + \rho_2 \geq 1$. Further,

Case 2a: It is given that $\lambda_1/\lambda_2 > S_1/S_2$. Assume that the necessary condition is not satisfied, i.e., $\lambda_2 < S_2/(S_1\bar{b}_1 + S_2\bar{b}_2)$. Then,

$\lambda_1/\lambda_2 > S_1/S_2$ means that a fraction, X_1, of the Class 1 offered load must be discarded until $(\lambda_1 - X_1)/\lambda_2 = S_1/S_2$, where $0 < X_1 < \lambda_1$. Then, both $(\lambda_1 - X_1)$ and λ_2 can be reduced proportionally with the S_1/S_2 ratio satisfied. This means that

$$\lambda_1 - X_1 = \lambda_2 \cdot \frac{S_1}{S_2}. \tag{5.4}$$

By the assumption of $\lambda_2 < S_2/(S_1\bar{b}_1 + S_2\bar{b}_2)$, equation (5.4) yields

$$\lambda_1 - X_1 < \frac{S_1}{S_1\bar{b}_1 + S_2\bar{b}_2}$$

The reduction in λ_1 by X_1 should affect the throughputs as well, to give $s_1 \leq \lambda_1 - X_1$, and $s_2 \leq \lambda_2$. This yields,

$$\bar{b}_1 s_1 + \bar{b}_2 s_2 \leq (\lambda_1 - X_1)\bar{b}_1 + \lambda_2\bar{b}_2 < \frac{S_1\bar{b}_1}{\bar{b}_1 S_1 + \bar{b}_2 S_2} + \frac{S_2\bar{b}_2}{\bar{b}_1 S_1 + \bar{b}_2 S_2} = 1$$

Since the system is work conserving, it must be $s_1 + s_2 = 1$. Therefore, this last equation is a contradiction.

Case 2b: Proof is similar to Case 2a. □

APPENDIX C

Proposition 3: *When $\sigma_1 + \sigma_2 < 1$, and under Poisson arrivals and general service times, the achievable delay ratio is such that:*

$$(1-\sigma_1-\sigma_2)\cdot\frac{\bar{b}_0+\bar{b}_1(1-\sigma_1)}{\bar{b}_0+\bar{b}_2(1-\sigma_1)(1-\sigma_1-\sigma_2)} \leq \frac{\overline{w}_1}{\overline{w}_2} \leq \frac{1}{1-\sigma_1-\sigma_2}\cdot\frac{\bar{b}_0+\bar{b}_2(1-\sigma_1)(1-\sigma_1-\sigma_2)}{\bar{b}_0+\bar{b}_1(1-\sigma_1)}$$

where $\bar{b}_0$ is the residual service time as seen by an arrival, and is given by

$$\bar{b}_0 = \sum_{i=1}^{2}\sigma_i\frac{\overline{b^2}_i}{2\bar{b}_i} \tag{5.5}$$

Proof: We assume that the buffer will never overflow. The minimum value of $\overline{w}_1/\overline{w}_2$ will be achieved when Class 1 has a non-preemptive, higher priority than Class 2. Stating the result given in [27] regarding the mean waiting time in the queue (excluding service) in classes 1 and 2, t_1 and t_2, we have

$$\begin{aligned}\bar{t}_1 &= \frac{\bar{b}_0}{(1-\sigma_1)} \qquad \text{and}\\ \bar{t}_2 &= \frac{\bar{b}_0}{(1-\sigma_1-\sigma_2)(1-\sigma_1)}.\end{aligned}$$

Taking into account that $\overline{w}_i = \bar{t}_i + \bar{b}_i$, then we have

$$\frac{\overline{w}_1}{\overline{w}_2} = (1-\sigma_1-\sigma_2)\cdot\frac{\bar{b}_0+\bar{b}_1(1-\sigma_1)}{\bar{b}_0+\bar{b}_2(1-\sigma_1)(1-\sigma_1-\sigma_2)}.$$

Conversely, $\overline{w}_1/\overline{w}_2$ will be maximum when Class 2 has a non-preemptive, higher priority than Class 1. In this case, and similar to the above, the lower bound is

$$\frac{\overline{w}_1}{\overline{w}_2} = \frac{1}{1-\sigma_1-\sigma_2}\cdot\frac{\bar{b}_0+\bar{b}_2(1-\sigma_1)(1-\sigma_1-\sigma_2)}{\bar{b}_0+\bar{b}_1(1-\sigma_1)}$$

□

APPENDIX D

Proposition 4: *Assuming Poisson arrivals and general service times, when a system is saturated,*

$$\frac{(\bar{b}_0 - \sigma_1\bar{b}_1 + \bar{b}_1)\sigma_2\bar{b}_1}{[(B-1)(1-\sigma_1)\bar{b}_1 - (\bar{b}_0 - \sigma_1\bar{b}_1 + \bar{b}_1)]\bar{b}_2} \leq \frac{\overline{w}_1}{\overline{w}_2} \leq \frac{[(B-1)(1-\sigma_2)\bar{b}_2 - (\bar{b}_0 - \sigma_2\bar{b}_2 + \bar{b}_2)]\bar{b}_1}{(\bar{b}_0 - \sigma_2\bar{b}_2 + \bar{b}_2)\sigma_1\bar{b}_2}$$

where $\bar{b}_0$ is the residual service time given by equation (3.2).

Proof: We prove this proposition by considering a packet that arrives when the buffer has only one empty location, and will therefore be accepted, and eventually served. Or, the packet arrives to find the queue full, and will remove a packet from the other class. This packet will also be served eventually. We also assume that the numbers of packets in the buffer are kept in accordance with the target ratio.

To obtain the lower bound, let class 1 have a higher non-preemptive priority over class 2. As such, the effect of class 2 packets is only through the residual service time of class 2 packets found in service upon arrival.

The mean waiting time of an arriving packet from class 1 which will be accepted in the buffer is given by

$$\begin{aligned} \overline{w_1} &= \bar{b}_0 + (\overline{q}_1 - \sigma_1)\bar{b}_1 + \bar{b}_1 \\ &= \frac{\bar{b}_0 - \sigma_1\bar{b}_1 + \bar{b}_1}{1-\sigma_1} \end{aligned} \tag{5.6}$$

Since the queue size changes by ± 1, then the distribution of the number of packets in the system seen by a departing packet is the same distribution seen by an arriving packet. This property was used in the above equation. Also, Little's result was used in order to relate the mean queue size to the mean waiting time. Notice that the mean number of class 1 packets which must be served ahead of the target packet is reduced by 1 if a class 1 packet is already in service when the target packet arrives; hence, the subtraction of the σ_1 term in the first equation.

Class 2 packets, which are assumed to have a lower priority, will be also affected by class 1 packets. Since the system is heavily loaded, then

$$\overline{q}_1 + \overline{q}_2 = B - 1$$

Therefore,

$$\overline{q}_2 = B - 1 - \overline{q}_1 = s_2 \overline{w}_2$$

The last term is due to the application of Little's result. Hence, after simple algebraic manipulations while using the expression given in equation (5.6) and substituting $\overline{q}_1 = s_1 \overline{w}_1$, we have

$$\begin{aligned} \overline{w}_2 &= \frac{B - 1 - \overline{q}_1}{s_2} \\ &= \frac{[(B-1)(1-\sigma_1)\bar{b}_1 - (\bar{b}_0 - \sigma_1 \bar{b}_1 + \bar{b}_1)\sigma_1]\bar{b}_2}{\sigma_2(1-\sigma_1)\bar{b}_1} \end{aligned} \tag{5.7}$$

Dividing equation (5.6) by equation (5.7), we obtain the lower bound. The upper bound can similarly be obtained by assuming that Class 2 has a higher non-preemptive priority over Class 1. □

BIBLIOGRAPHY

[1] Z. Wang, "Internet QoS - Architecture and Mechanisms for Quality of Service," *Morgan-Kaufman Publishers,* 2001.

[2] X. Xiao and L. Ni, "Internet QoS: A Big Picture," *IEEE Network,* vol. 13, no. 2, pp. 8-18, Apr. 1999.

[3] R. Braden, D. Clark, and S. Shenkar, "Integrated Services in the Internet Architecture: an Overview," *RFC 1633,* June 1994.

[4] R. Braden, L. Zhang, F. Baker, and D.L. Black, "Resource ReSerVation Protocol (RSVP) Version 1, Functional Specification," *RFC 2205,* Sept. 1997.

[5] S. Shenkar, C. Partridge, and R. Guerin, "Specification of Guaranteed Quality of Service," *RFC 2212,* Sept. 1997.

[6] J. Wroclawski, "Specification of the Controlled-Load Network Element Service," *RFC 2211,* Sept. 1997.

[7] A. Striegel and G. Manimaran, "Packet Scheduling with Delay and Loss Differentiation," *Computer Communications,* vol. 25, no. 1, pp. 21-31, Jan. 2002.

[8] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," *RFC 2475,* Dec. 1998.

[9] K. Nichols, S. Blake, F. Baker, and D.L. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 headers", *RFC 2474,* Dec. 1998.

[10] V. Jacobson, "Differentiated Services Architecture", *talk in Int-Serv WG, IETF Munich* Aug. 1997.

[11] V. Jacobson, K. Nichols, and K. Poduri, "An Expedited Forwarding PHB", *RFC 2598,* July 1999.

[12] D. Clark and W. Fang, "Explicit Allocation of Best Effort Packet Delivery Service", *IEEE/ACM Trans. on Networking*, vol. 6, pp. 362-373, Aug. 1998.

[13] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured Forwarding PHB group", *RFC 2597,* June 1999.

[14] C. Dovrolis and P. Ramanathan, "A Case for Relative Differentiated Services and the Proportional Differentiation Model", *IEEE Network,* vol. 13, no. 5, pp. 26-35, Sept. 1999.

[15] A.M. Odlyzko, "Paris Metro Pricing: The Minimalist Differentiated Services Solution" in *Proc. IEEE/IFIP Intl. Workshop on Quality of Service, London, U.K.*, pp. 159-161, June 1999.

[16] K. Nichols, V. Jacobson, and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", *RFC 2638,* July 1999.

[17] S. Floyd and V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", *IEEE/ACM Trans. on Networking*, vol. 1, no. 4, pp. 397-413, Aug. 1993.

[18] C. Dovrolis and D. Stiliadis,"Relative Differentiated Services in the Internet: Issues and Mechanisms", in *Proc. ACM SIGMETRICS*, pp. 204-205, May 1999.

[19] W. Feng, D. Kandlur, D. Saha and K. Shin, "Adaptive Packet Marking for Maintaining end-to-end throughput in a Differentiated-services Internet", *IEEE/ACM Transactions on Networking,* vol. 7, no. 5, pp. 685-697, Oct. 1999.

[20] J. Crowcroft and P. Oechslin, "Differentiated end-to-end Internet Services using a Weighted Proportional Fair sharing TCP", in *ACM Computer Communication Review,* vol. 28, no. 3, pp. 53-67, July 1998.

[21] C. Dovrolis, D. Stiliadis and P. Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling", in *IEEE/ACM Transactions on Networking*, vol. 10, no. 1, pp. 12-26, Feb. 2002.

[22] C. Dovrolis and P. Ramanathan, "Proportional Differentiated Services, Part II: Loss Rate Differentiation and Packet Dropping", in *Proc. IEEE/IFIP Intl. Workshop on Quality of Service, Pittsburgh, PA*, pp. 52-61, June 2000.

[23] C. Dovrolis, D. Stiliadis and P. Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," in *Proc. ACM SIGCOMM,* pp. 109-120, 1999

[24] T. Nandagopal, N. Venkitaraman, R. Sivakumar, and R. Bharghavan, "Delay Differentiation and Adaptation in Core Stateless Networks", in *Proc. IEEE INFOCOM,* pp. 421-430, Mar. 2000.

[25] R. Sivakumar, T. Kim, N. Venkitaraman, J. Li, and V. Bhargavan, "Achieving per-flow Weighted Rate Fairness in a Core Stateless network", in *Proc. Intl. Conf. on Dist. Comp. Systems*, pp. 188-196, 2000

[26] L. Kleinrock, "Queueing Systems, Vol. I: Theory", *John Wiley, New York,* 1975.

[27] L. Kleinrock, "Queueing Systems, Vol. II: Computer Applications", *John Wiley, New York,* 1976.

[28] M. Leung, J. Lui, and D. Yau, "Adaptive Proportional Delay Differentiated Services: Characterization and Performance Evaluation", in *IEEE/ACM Transactions on Networking*, vol. 9, no. 6, pp. 801-817, Dec. 2001.

[29] J. Liebeherr and N. Christin, "JoBS: Joint Buffer Management and Scheduling for Differentiated Services", in *Proc. IEEE/IFIP Intl. Workshop on Quality of Service, Karlsruhe, Germany,* pp 404-418, June 2001.

[30] J. Keilson and L.D. Servi, "A Distributional Form of Little's Law", *Operations Research Letters*, vol. 7, pp. 223-227, Oct. 1988.

[31] H. Zhang, "Service Disciplines for Guaranteed Service in Packet Switching Networks", *Proc. of IEEE*, vol. 83, no. 10, pp. 1374-1396, Oct. 1995.

[32] W. Weiss, "QoS with Differentiated Services", *Bell Labs Technical Journal*, pp. 44-62, Oct. 1998.

[33] A. Adas, "Traffic models in Broadband Networks", *IEEE Communications Magazine*, pp. 82-89, July 1997.

[34] S. Floyd and V. Paxson, "Difficulties in Simulating the Internet", in *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 392-403, Aug. 2001.

[35] D. Clark and W. Lehr, "Provisioning for Bursty Internet Traffic: Implications for Industry and Internet Structure", presented at *MIT ITC Workshop on Internet Quality of Service*, Nov. 1999