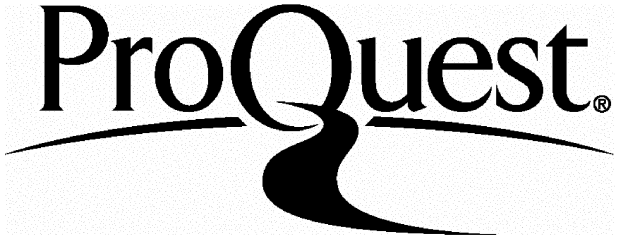
Utility-Based Buffer Management and Scheduling for Networks

Cedric Angelo Mojica Festin

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Department of Computer Science
University College London
2002



Abstract

User satisfaction from a given network service or resource allocation can be viewed as having two aspects, a state and a degree. The state defines whether the user is happy or unhappy. A user is happy when its expectations are met. The degree defines the level of happiness or unhappiness.

In this dissertation, we study the use of perceived knowledge of the state and degree of user satisfaction in managing router resources and functions. We make this information available to the router and examine whether the additional knowledge improves local resource allocation decisions. We recognise that users can have varying requirements, but our main objective is to satisfy all of their expectations fairly. We describe our formulation, Value-Based Utility (VBU), that incorporates both aspects of user satisfaction. From this we derive four further measures, which we then use to evaluate different resource management schemes.

We establish a framework of VBU use for traffic management schemes. In particular, we demonstrate the use of VBU for several buffer management and scheduling mechanisms where we show how they could be adapted to make use of this information. We compare the performance of several existing buffer management schemes against that of our proposed FIFO scheme that uses VBU to determine which packets to drop under a range of conditions. We also present a weighted round-robin server that uses VBU to make scheduling decisions and evaluate its success in making users happy.

The main conclusion we draw from this work is that the VBU framework offers a different perspective of performance definition and analysis. The use of VBU in management allows for the effective distribution of resources especially in times of high demand and low resource availability. Its adoption into existing traffic management schemes is further motivated by the improved performance of our proposed schemes over their non-VBU aware counterparts.

Acknowledgements

I am totally indebted to my supervisor Søren-Aksel Sørensen who with his guidance, patience and personal support encouraged me to persevere and complete this work.

I am grateful to the Computer Science Department at UCL for the environment that allowed me to participate in discussions and seminars that benefited me throughout my stay. I would like to especially acknowledge Stephen Hailes and Jon Crowcroft for their comments on earlier versions of this work.

This work was supported by the Department of Science and Technology of the Philippines and the University of the Philippines. I would like to thank in particular Dr. Estrella Alabastro, Mark Encarnaciõn, Jimmy Caro, Teody Dayoan and June Pereña for all their help.

I would like to thank my friends from UCL, Cambridge and the Pinoy-UK for a life outside studying.

I would like to thank Marissa and Felipe for sharing their home during my studies and for giving their support and understanding. To my first housemates Fe and Zaldy, I thank them for making my move to London less daunting.

I am most especially grateful to Susan who has always been a source of inspiration. I thank her for her motivation, support, patience and love.

Finally, I would like to thank my family for their love and moral support.

Contents

List of Figures

List of Tables

Chapter 1

Motivation

1.1 Context

The last decade has seen the unprecedented growth of the Internet in terms of the number of hosts connected and the number of people using it. This has been brought about by the Internet's ability to scale well while still remaining flexible. Millions of people currently use the Internet for education, business, entertainment, and communication. More users are projected in the next few years. It is not difficult to see that user demands on the network will eventually out-pace the improvements in the capacity and size of transmission facilities. This growth, coupled with the emergence of multimedia applications carrying video and audio streams, has placed a tremendous strain on the existing Internet service model.

In the late 1960's [53, 82], the ARPANET, the precursor of the Internet, was conceived to provide a robust means of communication in the event of war. It was also used as a research and academic network where computing resources were shared through remote sessions. In addition, it allowed the exchange of research data using file transfer programs and communication through electronic mail. These applications required data to be transmitted reliably across sites that used different network technologies. The problem was solved through the introduction of the Internet Protocol (IP) suite which allowed heterogenous networks to be interconnected. However, this

service is on a *best-effort* basis. Packets carrying data are treated equally and transported when network resources are available without guarantees against delay and loss. The two main transport protocols in the Internet are the connection-oriented Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) which provides a connectionless service. TCP was developed to provide reliable, inorder and end-to-end transport of data across the unreliable link level service provided by IP. In contrast, UDP does not have an error recovery mechanism and is commonly used for applications that can tolerate loss or have recovery facilities in the upper layers. This separation of functionality through layering embodies the basic design principle of the Internet – that is the network should just route while the responsibility of control is left to the end systems.

The philosophy of decentralised control has made the Internet flexible allowing networks and hosts to interconnect easily. This scaling property, the deployment of LANs [53] and the development of the World Wide Web (WWW) in 1990's [7, 6, 82] provided the impetus for the dramatic growth of the Internet. The WWW gave its user an easy-to-use interface to access the information on the Internet. With the commercialisation of the Internet, information content flourished, new and better applications were developed and the Internet continued to expand. In spite of these innovations, the current Internet is essentially a network for communication and information exchange. The main difference from the Internet of the 1970's and even up to the late 1980's, is that it is now increasingly used to transport data with different traffic characteristics.

Today the Internet is challenged to support a wide spectrum of applications ranging from traditional data applications like electronic mail to real-time applications such as audio and video conferencing. Real-time applications are described by their *Quality of Service* (QoS) requirements often expressed in terms of high bandwidth and low delay. The Internet, because of its decentralised control and best-effort service, is unable to guarantee such QoS requirements. Although live audio and video conferences have been transmitted over the Internet since 1992 [13, 24], such services are far from

common. This is due to the difficulty of sustaining the desired level of service especially in periods of high demand [18].

1.1.1 Evolving Nature of Applications and Their Quality of Service Requirements

Improved transmission facilities and the development of more sophisticated protocols have greatly reduced the effects of the traditional problems associated with early data networks. Data applications now have reliable facilities to manage data loss, ordering, duplication and other data anomalies [74]. As a result of these technical innovations, new applications with multimedia streams and real-time requirements are fast emerging.

These 'new' applications can be categorised in many ways. One early classification was given in [41, 80] where applications were categorised as providing either interactive or distribution services. The interactive services were further classified into three categories namely: conversational, messaging and retrieval. Conversational services are characterised by the interaction of two or more users in real-time. Current applications falling in this service category include audio and video conferencing [24], IP telephony [64], interactive network games [22] and other collaborative [65] and computer-supported co-operative work [36] applications. The messaging and retrieval services are complementary services where the interaction is not time-constrained unlike conversational services. Messaging services update information in a data store while the retrieval services extract information from the data store. The reason these two services were differentiated in [41, 80] is that messaging only involves movement of information in one direction (from source to data store) while retrieval services also involve an initial query before actual transfer from data store to the receiver commences (to and from data store). Applications like sending and receiving multimedia e-mail and the update and retrieval of media databases are representative of these services. Distribution services as defined in [41, 80] involve the transmission of information in one direction. The transmission of multimedia streams can be characterised

as broadcast-like where the receiver may or may not have presentation controls like pause, fast forward, rewind or even slow motion features. A video-on-demand service [62, 50] is a typical example of a distribution service.

Applications can also be categorised according to how they are used [63]. The grouping can be based on: 1) their use as a communication tool; 2) the need to get information and entertainment; or 3) the need to access computing resources and facilities. Still another distinction can be based on whether applications are point-to-point or multi-point. Many of these classification schemes are overlapping. For example, video-conferencing is both conversational and point-to-point. If there are more than two users, video-conferencing is multi-point and depending on its use, could be either be for communication, information or even both.

Regardless of how these wide ranging applications are categorised, it is clear that their development and deployment impose different Quality of Service (QoS) constraints on the underlying service infrastructure, devices, protocols and resources. Although in this thesis we shall concentrate on network QoS, it is by no means the only issue. The provisioning for QoS is an end-to-end issue [40, 57, 58, 77, 5] that involves among others the user, application, operating system, end-system hardware, as well as the network and transport protocols. Architectural frameworks relating different elements of the end-to-end communication such as QoS definitions, translations and specifications are explored in some detail in [40, 57, 58, 77, 46, 59, 5]. The main point we derive from these works on QoS architectural frameworks is that the establishment of network performance targets is coordinated with other entities in the end-to-end path. This allows for some level of functional abstraction and the establishment of specific network performance objectives. Whatever scheme is used to achieve the targets will be in conjunction with the overall framework. This means that the success of QoS provisioning can be viewed at two levels, end-to-end and the network level. We will mainly deal with the latter.

Keshav [48] defines QoS as "network quality to satisfy user needs, however the needs may be expressed". The ITU-T X.902 [42] defines QoS as "a set of quality

requirements on the collective behaviour of one or more objects." These definitions are just a sample of how QoS is defined but whatever definition is used, two aspects or views are common to all. The first one is the network view while the other relates to user perception.

Generally, the network views QoS as measurable performance targets [26, 27, 75, 52] that characterise real-time and multimedia applications. These performance targets are often expressed as bandwidth, delay, loss and jitter requirements. Bandwidth requirements of multimedia applications are determined by several factors, e.g., the application type, the subject of the application, the encoding and compression scheme used, and the frame rate and size. This results in a wide range of bandwidth demands. For example, typical ranges for low bit audio communication are from as low as 2.4 kbps and up to 64 kbps [19]. The requirement for compressed digital video is around 5-10 Mbps while the data rate transmitting digital video uncompressed is higher at 170 Mbps [47].

The delay requirement is important, especially when applications are interactive and in real-time. Packets of real-time fiows arriving late are useless and simply waste bandwidth. In addition to having bounds on delay, it is equally important that the jitter of packets is minimised to enhance the understanding of communication. Excessive packet loss on the other hand seriously degrades interaction especially if multimedia traffic is compressed or layered. Whatever values are set for these requirements, be it 150 to 400 ms for one-way delay [47, 50] or 10% packet loss [49], these are often influenced by user perception of quality.

The user view of QoS is very subjective because it is dependent on how well the user can understand the information content from using the application [33]. In addition, perception differs from one user to another [73] which makes it more difficult to define quality. There are several techniques for assessing quality [16, 9]. Current approaches normally involve asking a group of users to rate and evaluate multimedia presentations based on some aspect of perceptual quality [2, 33, 16] while some media distortion is introduced. For example, the work in [2] has studied how a video server can support

additional users by degrading the quality of the streams received by the current users. In [33], the effects of varying the frame rate of video streams on perceptual quality were investigated, while others [16, 73] have looked at inducing loss and jitter in audio and video streams. Although the mapping of distortion to user perception is still evolving, we can use the fact that in certain conditions, users can tolerate some level of service deterioration. This knowledge of user tolerance can be used in the policies and schemes implemented inside the network.

1.1.2 Traffic Management and Utility

The provisioning for network QoS to satisfy user requirements is not just one scheme or policy. Rather it is an aggregation of mechanisms and policies. Collectively, the set of mechanisms and policies employed by the network to attain a level of service given resource constraints is called *Traffic Management*. Some of the key mechanisms employed by networks are: admission control, traffic shaping, packet classification, packet marking, scheduling, queue management and congestion control. Different levels of service differentiation can be achieved depending on the policies used by these mechanisms. The policies determine how packets will be treated by the mechanisms, e.g., which packet to drop or which packet is to be served first.

In this thesis we adopt a policy based on utility. However, the way we define and use utility is slightly different from its normal usage. Utility is mainly used to express user preferences for choices. These preferences have often been linked to pricing [17, 20, 55, 67, 66]. This means they have assumed that if you prefer A over B, you are willing to pay more for A than for B. We do not assume this link between preference and pricing. We use utility to express a preference for A but this preference is not necessarily linked to the willingness and capability to pay for A. The preference we deal with is solely based on need.

The policy we employ inside the network is therefore determined by the way we define and use utility. In general, we use a policy where resources are shared between users with a range of needs. The idea here is that when one or more users are satisfied

with the network QoS they receive, it maybe possible to degrade the service they are getting to help others.

1.2 Thesis Statement

User satisfaction from a given network service or resource allocation can be viewed as having two aspects, a state and a degree. The state describes whether the user is happy or unhappy. A user is happy when its expectation or minimum requirements are met, otherwise it is unhappy. The second aspect, the degree, describes how happy or unhappy the user is.

This dissertation aims to study the use of perceived knowledge of the state and degree of satisfaction of users in the management of router resources and functions. The basic idea is that if a router has this information locally, then this router can possibly make better resource management and control decisions to meet its users' expectations. There is little work investigating the use of the state and degree of user satisfaction in this manner. Most of the work done so far has been limited to using one aspect of user satisfaction or the other.

Numerous control algorithms and policies have been designed offering a range of service and resource allocation capabilities. Most of these schemes operate with the goal of achieving the performance requirements of network users. Under this objective, success is only measured in terms of whether users are happy or not. How well these schemes accomplish their goal is secondary or even unimportant. There is a danger that in some situations such as high demand, services and resources are not effectively distributed.

Previous work that uses the degree of satisfaction in management primarily revolves around economic indices such as utility. These works often use utility as part of an optimisation objective such as the maximisation of overall utility and/or the minimisation of network cost. In this context, it is often assumed that with better service or increased allocation, utility increases. The problem with this view is that it fails to consider whether the users are happy or not with the service they receive.

We believe that the marriage of the state and degree of satisfaction could potentially provide useful information for traffic management. In particular, we aim to find out if it is possible for a router using a local view of this information to effectively distribute router resources and services so that user expectations can be achieved.

1.3 Research Contribution

The main contributions of our work are as follows:

- We offer a different perspective in specifying user expectation that redefines the definition and analysis of acceptable performance. The use of our notion of user satisfaction in management allows for the effective distribution of resources especially in times of high demand and low resource availability.
- We model user satisfaction as a utility function which provides a measurable index of performance.
- We apply our formulation in traffic management and examine schemes such as admission control, packet classification and marking, traffic shaping, scheduling and packet discard in general terms and establish a framework where our formulation can be used.
- In particular, we show how our formulation can naturally offer service differentiation and provide improved performance in terms of user happiness. We provide specific examples of our formulation's use for managing loss and delay and show the improved performance achieved by the proposed schemes.

1.4 Organisation of the Thesis

Our work is composed of four main areas. The areas and their relationship with each other are shown in Figure 1.1.

The first area deals with our definition of satisfaction. Here we motivate the need to incorporate both the state and degree of satisfaction and establish key terms and

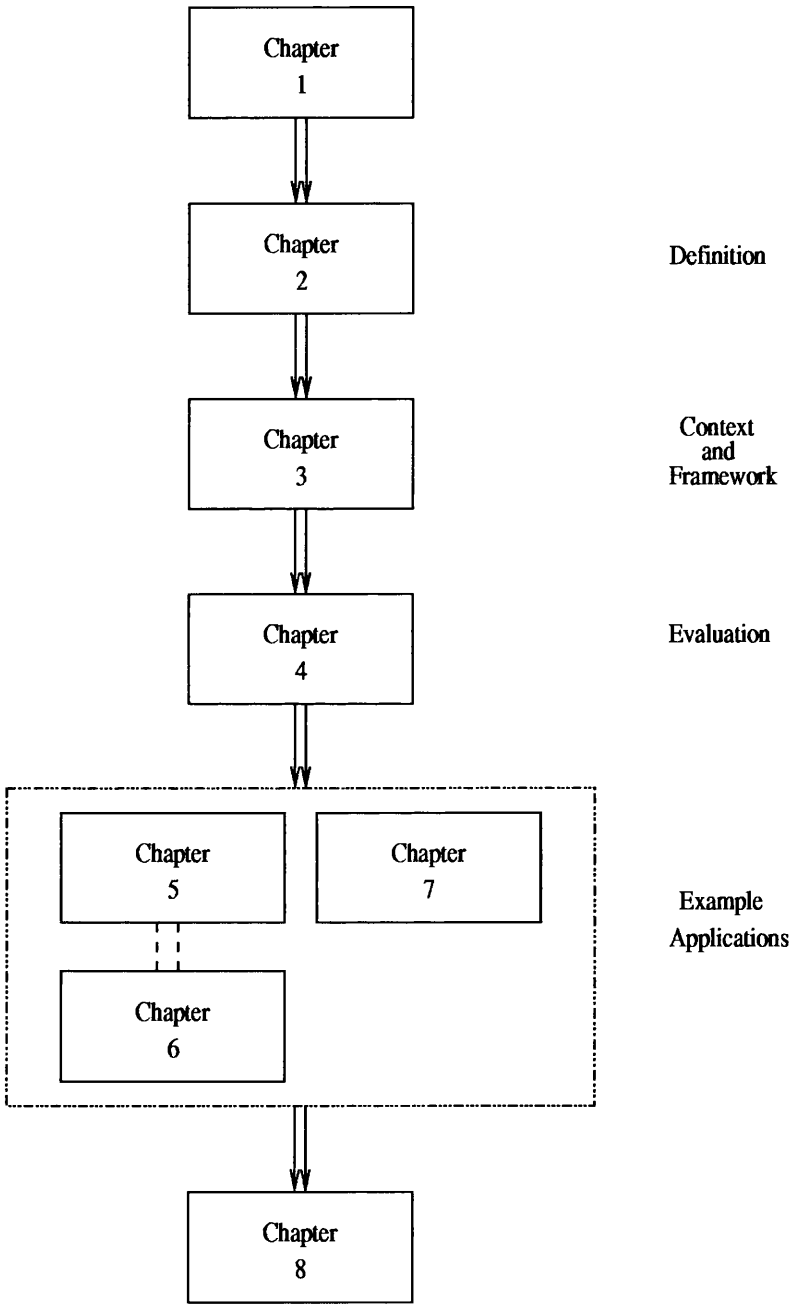


Fig. 1.1: Organisational view of this dissertation

properties. We define our formulation, *Value-Based Utility* (VBU), that incorporates both components of satisfaction. This work is covered in chapter two.

The second part establishes the context where VBU may be applied for traffic management. We outline features of router functions and mechanisms and develop a framework on how VBU may be used for management in chapter three.

The third part concentrates on presenting the evaluation process. We describe how we derive four additional measures from the computed utility. These additional measures are later used to evaluate the performance of management schemes and mechanisms. This work is carried out in chapter four.

The fourth area consists of the specific use of VBU either in tuning of operating parameters or as part of algorithms and policies. We specifically look at traffic management schemes that affect loss and delay. These examples are encapsulated by the work in chapters five and six for loss and seven for delay.

1.5 Summary

The second chapter introduces Value-Based Utility (VBU), our model of the state and degree of satisfaction. We begin by developing a linear utility function based on user performance requirements. We also define key terms and properties. We consider a simple example to highlight how VBU may be used to find near optimal operating conditions that would result in the highest number of satisfied users of the Ethernet.

The third chapter examines in abstract terms how VBU can be used to manage and control router resources and functions. We outline different mechanisms employed by a router and describe in general terms how they may be used along with VBU to offer service differentiation based on utility.

The fourth chapter defines the measures based on utility that will be used in the succeeding chapters. We begin by deriving the VBU given the Quality of Service (QoS). From this expression of satisfaction, we then adopt a management objective based on maximising the number of happy users but constrained by a variant of Jain et al.'s fairness index [44]. We use measures for *Overall Unhappiness, Class Unhappiness,*

Class Average Utilities and *Utility Fairness* to express this objective. As an example, we analyse the performance of a FIFO queue using these measures.

The fifth and sixth chapters contain a series of experiments that show the effects of using VBU in specific loss policies and algorithms. First, we evaluate the performance of Guerin et al.'s threshold-based scheme (*G+98*) [37] in terms of meeting user expectations and fairness. We show how, by adjusting the operating parameters of this scheme, different levels of performance can be achieved which provides for interesting tradeoffs and scenarios. We then develop VBU-aware schemes for FIFO and G+98 which improves on the performance of the base scheme.

Our seventh chapter looks at the issue of delay. We apply the same approach here as we did in chapter six. We first analyse the behaviour of three schemes in terms of our objectives and measures. We look at the delay utility characteristics of FIFO, Three Class Priority Queue (3CPQ) and the Deficit Round-Robin (DRR) [70]. We then incorporate VBU into both the 3CPQ and DRR resulting in two variants that have a wide range of performance capabilities. We conclude by highlighting the important observations and results.

Chapter eight provides an overview of our main conclusions. We also discuss possible research paths that have surfaced during our investigations.

Chapter 2

Value-Based Utility

2.1 Introduction

In terms of happiness with a given service, users[1] of a multiservice network can be in one of two *states*. They could either be happy or unhappy. When resources are low and demand for them is great, it is difficult to make every user happy but it is not impossible. Resources from satisfied users may be transferred to the dissatisfied users to try to make them less unhappy or even happy. This is possible because applications impose variable demands. Some applications may have high *expectations* of the service they need while others do not. This expectation is an indicator of how an application perceives a performance target or requirement. For an application who has expectations to be *satisfied*, the network must provide a service equal to its requirements. Any excess can only make it *happier* while failure to meet such requirements results in *dissatisfaction*. This notion of the *degree* (how much more or less) of *satisfaction* an application derives from a service has often been overlooked. This is because service concerns focus more on meeting quantitative demands than on qualitative attributes like satisfaction. This chapter develops a formulation called *Value-Based Utility* (VBU) to quantify both the state and degree of satisfaction. The formulation is simple and its construction is fairly straightforward. Value-Based Util-

[1]Note that we use *application* and *user* interchangeably unless otherwise specified.

ity uses the QoS requirements to define a ***utility function*** that associates a utility value with the service received by (or promised to) an application. Given this value, we can then characterise the application's state and degree of satisfaction with any given service.

We contrast our notion of utility with economic utility where utility functions are used to order resource or service bundles. In economics, the significance of the utility's value lies only in its inherent ability to rank preferences and choices. For example, consider a voice application with a utility function U, and two service bundles A and B with the following performance characteristics:

- A :<1000 ms delay @ 97% of the time, 20% loss, 32Kbps>
- B :<1000 ms delay @ 90% of the time, 20% loss, 16Kbps>

Economic utility states that the voice application prefers A to B if the application of U to A yields a higher value than the application of U to B; i.e., $U(A) > U(B)$. The problem with this proposition is that it does not tell us anything about the degree of satisfaction of the application. A may be a better service than B but the application may not be happy at all with a delay of 1000 ms. Similarly, A may be better than B but B may already be sufficient for the application. This would allow A to be allocated to some other user who needs it more.

For emerging network applications with strict QoS requirements, the use of utility functions to simply order and rank services is inadequate. It is incapable of capturing and modelling expectations of user requirements. In situations like these, it is more appropriate to use utility to represent user well-being. Information such as Value-Based Utility could be useful in managing resources, especially in resource-challenged environments or utilisation-conscious systems, because it identifies users who can possibly share some of their resources. In the succeeding sections, we develop these ideas.

2.2 Abstract Framework for Levels of Satisfaction

Quality of Service (QoS) requirements are often expressed as either a deterministic or a statistical bound [26, 27]. An example of a deterministic requirement is when the voice application in Section 2.1 requires that all packets should not be delayed by more than 1000 ms. Given the service choices A and B, neither would have been capable of delivering the desired service. A statistical bound is generally less restrictive. It is similar to a deterministic bound except that it has one additional parameter p, where p is the percentage of packets required to meet the bound. In a deterministic bound, this p is implied to be equal to one. For our voice example, instead of requiring all packets to meet the target, suppose we require that $p = 0.95$. This condition would result in service bundle A meeting the target while B still fails to meet expectations. Note that in both deterministic and statistical QoS representations, the service either succeeds in meeting the requirements or it does not. Unfortunately, questions like "How bad was the service for the deterministic case?" or "How good was the service for the statistical case?" cannot be answered.

To answer this, we first define the user expectation range to be some value between $happiness_{min}$ and $happiness_{max}$. These two points represent the level of user satisfaction given that the received or promised service is at least equal to the minimum requirements. A utility function U_i, which we formally define in Section 2.3, maps a user's received service to some value hopefully within the expected range. Whenever a user's utility $U_i = happiness_{max}$ then we say that the user has received the best possible service. If $U_i = happiness_{min}$ then the user's requirements have been minimally met. The worst that a user can be within this range is to be in a state of happiness. From a management perspective, it would be sufficient to operate the system at slightly above $happiness_{min}$ levels especially in times of high resource demands. There are no benefits for the network to expend resources that will not improve a user's state. This is because the user's expectation has already been achieved and the user is already happy. From the network's perspective, the users are indifferent to services evaluated within this happiness range.

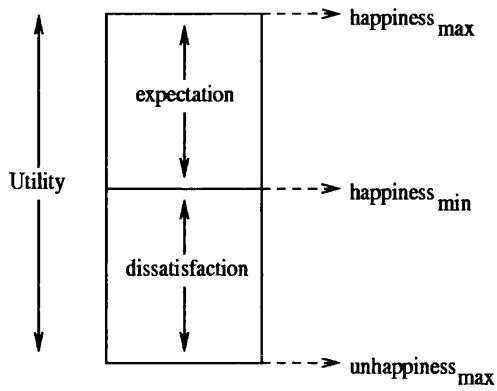


Fig. 2.1: The expectation range is delimited by the points $happiness_{max}$ and $happiness_{min}$ while the region of dissatisfaction is defined between the $happiness_{min}$ and $unhappiness_{max}$ values.

In cases where some services fail to meet user expectations, applications will become unhappy. Similarly, as with satisfaction, there are varying degrees of unhappiness. We represent unhappiness and its levels as a range called the dissatisfaction levels. This area lies just below the $happiness_{min}$ value and is delimited by the point called $unhappiness_{max}$. Notice that $happiness_{min}$ is a threshold value because it is where the state of utility changes (from satisfaction to dissatisfaction or vice-versa). Service that is evaluated below this value can only make a user dissatisfied. If a user flow's utility $U_i = unhappiness_{max}$, then the user is unhappy and is the recipient of the worst possible service. Both of these areas are illustrated in Figure 2.1. Note that the expectation range is equivalent to satisfaction levels. This is because a user is not expecting to be unhappy.

2.3 General Form of the Utility Function

In this section, we formally define Value-Based Utility and develop the function to express satisfaction. We also highlight the important characteristics of the utility function.

2.3.1 Formulation

Let us assume that some percentage p from a flow of packets belonging to application i must meet some QoS target bound b in an interval Δt. To find a utility function U, we first define the user expectation range to be in $[0, 1]$. This range gives us the two points $happiness_{max} = 1$ and $happiness_{min} = 0$. We shall later see that the definition of $unhappiness_{max}$ is dependent on these two points and is a function of p. We next partition all packets N transmitted in a time interval Δt into two sets; one set S that meets the requirements and another set Q that does not. We can then associate the following probabilities $P(S) = \frac{G}{N}$ and $P(Q) = \frac{N-G}{N}$ to these sets, where G is equal to the number of packets meeting the bound. The value $P(S) - P(Q)$ can be considered as the relative bias of a service either towards meeting targets when positive or to not meeting them when it is negative. However, this relation does not characterise how well performance has met expectation (p, b). We accomplish this by multiplying a factor α, which should be a function of p, to $P(Q)$ and then subtracting it from $P(S)$. Intuitively, we associate some benefit with $P(S)$ while $P(Q) * \alpha$ is the rate of how fast the benefit from $P(S)$ diminishes. Given the two points of $happiness_{max}$ and $happiness_{min}$, we find a suitable expression for α is given by $\frac{p}{1-p}$. We can also think of this ratio as the penalty factor for not meeting expectation p. Thus, $P(S) - P(Q) * \alpha$ gives us a utility function U for describing both user satisfaction and dissatisfaction within any specified time interval Δt.

Definition 1 *Value-Based Utility is an expression of user well-being. It uses a utility function to represent both the state and degree of user satisfaction (dissatisfaction). The expression for the Value-Based Utility function is given by:*

$$U_{i,QoS,m,\Delta t}(p, b) = \frac{G}{N} - \frac{N-G}{N} * \frac{p}{q} \tag{2.1}$$

where

$U_{i,QoS,m,\Delta t}$ is flow i's utility for the specified QoS at point m during the time interval Δt,

p is the target percentage of packets that should meet QoS requirement,

b is the target QoS bound,

G is the number of packets meeting flow i's requirements,

N is the total number of packets seen, and

q is equal to $1 - p$.

There are two problems when equation 2.1 is used to model utilities of deterministic requirements. In the first case, $\frac{p}{q}$ has no numerical meaning because $q = 0$. To avoid this we use the limit of α as p approaches one. This results in an expression for $\alpha = \lim_{p \to 1} \frac{p}{q} = \infty$. What this says is that when q becomes very small, α becomes very big and hence the penalty becomes larger. However, for operational purposes, q should not be allowed to become very small (except for zero). If no bound on q is set, managing resources would become impossible as the number of levels would infinitely increase. To avoid this, the number of different q's must be limited and the smallest of these values' passing point m must be known. Selecting a suitable α when $q = 0$ can then easily be obtained as long as it is bigger than the α of the known smallest q. To illustrate this, suppose point m allows the smallest q (largest p) to be equal to 0.01 ($p = 0.99$). The α of the smallest q is equal to 99 and therefore an α value of $\alpha_{smallestq} + step$ where $step > 0$ can be used when $q = 0$. To avoid confusion we shall use α' instead of α when dealing with deterministic requirements.

The other problem occurs when $G = N$. When this happens utility equals $happiness_{max}$. The utility, although it correctly models the state, does not represent the degree accurately. Since the expectation is $p = 1.0$, $G = N$ should be the minimum service to make the user happy. Therefore the resulting utility U must only be equal to $happiness_{min}$. To model this accurately, we modify $\frac{G}{N} - \frac{G-N}{N}$ of equation 2.1 to $-\frac{N-G}{N}$ when $G = N$. Combining these two special cases gives us the definition of the utility function for the deterministic case:

Definition 2 *The only level of happiness with a deterministic requirement (p=1.0) is* $happiness_{min}$. *The Value-Based Utility function is undefined for values greater*

than $happiness_{min}$. The expression for a deterministic Value-Based Utility function is defined as:

$$U_{i,QoS,m,\Delta t}(1,b) = -\frac{N-G}{N} * \alpha' \tag{2.2}$$

where

α' is some value greater than the α of the known smallest q at point m,

the others are defined as before.

2.3.2 Analysis

To verify that Equation 2.1 represents the state of user well-being, we consider the best, the minimal, and worst possible service. A similar analysis can also be used for the deterministic case (Equation 2.2). The best possible service occurs when $G = N$, which means that all the packets were serviced according to expectation (p, b). We see that the second term disappears and the equation simply evaluates to one or $happiness_{max}$. The second term also becomes zero when there is no expectation ($p = 0$ and a don't care bound b). In this case utility will always be greater or equal to $happiness_{min}$.

When the requirement is exactly achieved, that is $\frac{G}{N} = p$, utility is equal to zero or $happiness_{min}$. An application will be satisfied if the utility from the service is above or equal to this level. A service that performs less than the expectation will have a utility value less than $happiness_{min}$, which is a negative utility U. The range of unhappiness begins at a point below the $happiness_{min}$ level and is bounded by $unhappiness_{max}$. We find the expression $unhappiness_{max}$ is equal to $-\frac{p}{q}$ (see Figure 2.2). This occurs when service to all packets fail to meet objectives ($G = 0$). Note that $unhappiness_{max}$ is not assigned a fixed point because of its dependence on user expectation p. It is also interesting to see that $unhappiness_{max} = -\alpha$. This should not be surprising since α is the total penalty with the negative sign indicating dissatisfaction. We note that for the deterministic case, the best service $G = N$ is just equivalent to the required service $p = 1.0$.

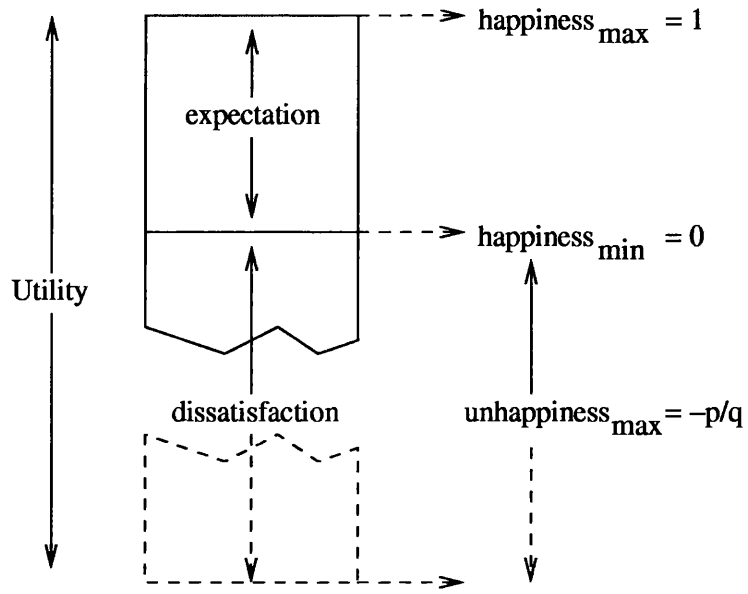


Fig. 2.2: The $happiness_{max}$, $happiness_{min}$ and $unhappiness_{max}$ are assigned the values 1, 0 and -p/q respectively.

2.3.3 Key Terms and Definitions

From the analysis of section 2.3.2 we can infer from utility the success or failure of the service in meeting requirements. More importantly, from utility we can deduce the level of user satisfaction (dissatisfaction). This allows us to determine how far above or below users are from the happiness threshold, a measure that can be used for management. We now summarise some of the key terms and definitions we used in the previous sections for future reference. These are given below

Definition 3 State of Satisfaction. *A user can either be in a State of Happiness or in a State of Unhappiness depending on whether utility is negative or not.*

Definition 4 State of Happiness. *A user is in a state of happiness or simply happy if utility is either positive or zero ($U \geq 0$). This implies user expectations were achieved.*

Definition 5 State of Unhappiness. *A user is in a state of unhappiness or simply unhappy if utility is less than zero ($U < 0$). This implies user expectations were not achieved.*

Definition 6 Degree of Satisfaction. *The degree of satisfaction describes the level of user happiness or unhappiness. It is dependent on the magnitude of utility.*

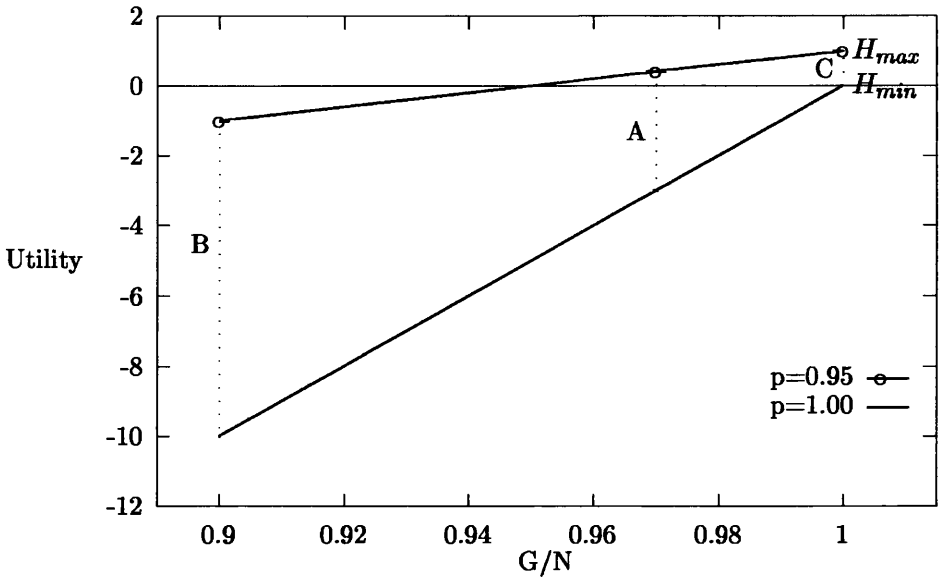


Fig. 2.3: Utility of service bundles A, B and C for both statistical and deterministic requirements.

Definition 7 Maximum Happiness. *$Happiness_{max}$ (H_{max}) is equal to one and occurs when G=N for $0 \leq p < 1$. It does not exist for p=1.0.*

Definition 8 Minimum Happiness. *$Happiness_{min}$ (H_{min}) is equal to zero and occurs when G/N=p for $0 \leq p < 1$ and G=N for p=1.0.*

Definition 9 Maximum Unhappiness. *$Unhappiness_{max}$ (UH_{max}) is equal to $-p/q$ when $0 < p \leq 1$. It does not exist for p=0.*

As an example, consider the state of satisfaction of the voice application in Section 2.1 with the service bundles A and B and another service C:

- C :<1000 ms @ 100% of the time, -, - >

Using $p = 0.95$ and applying Equation 2.1 on the delay parameter of each service offering, we get $U(A) = 0.4$, $U(B) = -1.0$ and $U(C) = 1.0$. Definition 4 tells us that service bundles A and C will provide a successful service because their utilities are non-negative while service bundle B will fail to meet expectations. Since $U(C) = 1.0$, this implies that the voice application will receive the best possible service (Definition 7).

Let us now assume that we have a deterministic requirement ($p = 1.0$). Using Equation 2.2 with $\alpha' = 100$ results in the following utilities: $U(A) = -3$, $U(B) = -10$ and $U(C) = 0$. In this example the voice application can only be happy if all of its packets receive the expected service. This means that only service bundle C could satisfy the application. The best possible service (100%) results in minimum happiness (Definition 8). The example utilities for both the statistical and deterministic requirements are shown in Figure 2.3. Observe that the two curves in this figure trace the possible values of U for different $\frac{G}{N}$'s.

Until now, we have only considered utility at the three reference values (0, 1 and $-p/q$). The values in between these points are equally important for signalling the level of well being. Hence, there is a need to characterise the utility in these areas as well. We accomplish this by observing how utility changes as both p and $\frac{G}{N}$ changes. However, before we do this, we present equivalent expressions for the Value-Based Utility functions.

Theorem 1 *An equivalent expression for Equation 2.1 is given by:*

$$U_{i,QoS,m,\Delta t}(p, b) = \frac{G}{Nq} - \frac{p}{q} \tag{2.3}$$

Proof. From Equation 2.1, we can expand the second term to get $\frac{G}{N} - \frac{Np-Gp}{Nq}$. We then rearrange the terms to give us $\frac{Gq-Np+Gp}{Nq}$. Grouping terms gives $\frac{G(q+p)}{Nq} - \frac{Np}{Nq}$ and simplifying gives us the result above.

Theorem 2 *An equivalent expression for Equation 2.2 is given by:*

$$U_{i,QoS,m,\Delta t}(1, b) = \frac{\alpha' G}{N} - \alpha' \tag{2.4}$$

Proof. This is just a simple rearrangement of terms.

Taking the derivative of theorems 1 and 2 with respect to $\frac{G}{N}$ yield $\frac{1}{q}$ and α' respectively; we see that both slopes are constant making it easy to compare utilities. We also know that as $p \to 1$, $\alpha \to \infty$ which means as requirements become more

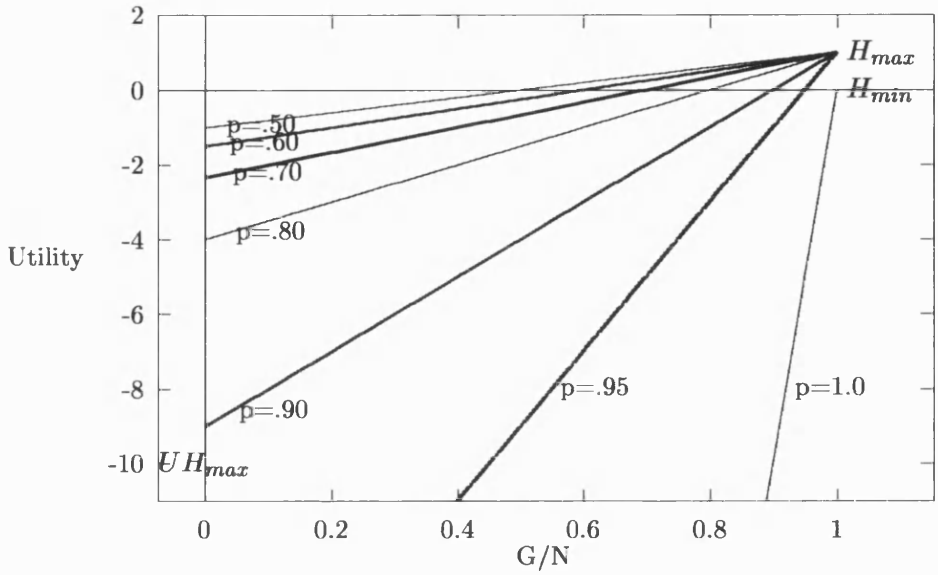


Fig. 2.4: This figure shows the utility's sensitivity to p. As p approaches one, the steeper the slope becomes.

strict, the more susceptible users are to missed targets. This behaviour is illustrated in Figure 2.4. In this figure, the lines represent the utilities of fixed p's as G/N is made to vary from zero to one. Observe that the utilities of larger p's are steeper than the ones with smaller p's. As an example, the line representing $p = 0.95$ is sloping at an angle of $\approx 87.13°$ while the line representing $p = 0.50$ is sloping at an angle of $\approx 63.43°$. Generalising gives us the following definition:

Definition 10 User Sensitivity. *Given two users A and B with p's p_1 and p_2 respectively, we say that user A: a) is more sensitive than B if $p_1 > p_2$; b) as sensitive as B if $p_1 = p_2$; and c) is less sensitive than B if $p_1 < p_2$. Generally, it is more difficult to satisfy a sensitive user than a less-sensitive user.*

2.4 Maximum Number of Satisfied Users - An Example

One of the most straightforward uses of VBU is to find the highest number of satisfied users. In an earlier work [30], VBU was used to find the maximum number of satisfied voice calls supported by a 10 Mbps Ethernet hub. Utilities were measured for throughput, delay and jitter. This work investigated the effects of packet size and

transmission rate on the measured VBU. It is well known that large frames maximise Ethernet throughput while small frames reduce utilisation [4, 1, 34, 21]. In this section we present some results from our earlier work [30] to show how VBU can be used to quantify the effects of frame sizes and transmission rates on the maximum number of voice calls supported.

2.4.1 Experimental Model

A model[2] was developed for an 10 Mbps Ethernet Hub using the CLOWN simulation environment [71][3]. In the model, each station sharing the Ethernet hub emulates a 64 Kbps voice source without silence suppression. Each source transmits a fixed size frame P bytes every ϕ ms to a randomly chosen receiver. Five different frame size-rate (P-ϕ) pairs were studied (see Table 2.1). For a given source N, each P-ϕ pair was used to load the Ethernet until 40000 frames were transmitted successfully across the Ethernet. This was replicated ten times for each source N. Two event times are logged: the time the frame was created at the source and the time it was received at the receiver. From these information statistics such as throughput, delay, jitter as well as VBU for these performance indices can be obtained. The number of happy users, V_{QoS}, can be determined by counting the number of users whose $U_{i,QoS}$ is greater or equal to zero. More formally, the number of satisfied users for a particular QoS is given by following definition :

Definition 11 Total Satisfied Users.

$$V_{QoS} = |X| \text{ where } X = \{x : x \in \{U_{i,QoS} \geq 0\}\} \tag{2.5}$$

2.4.2 Discussion

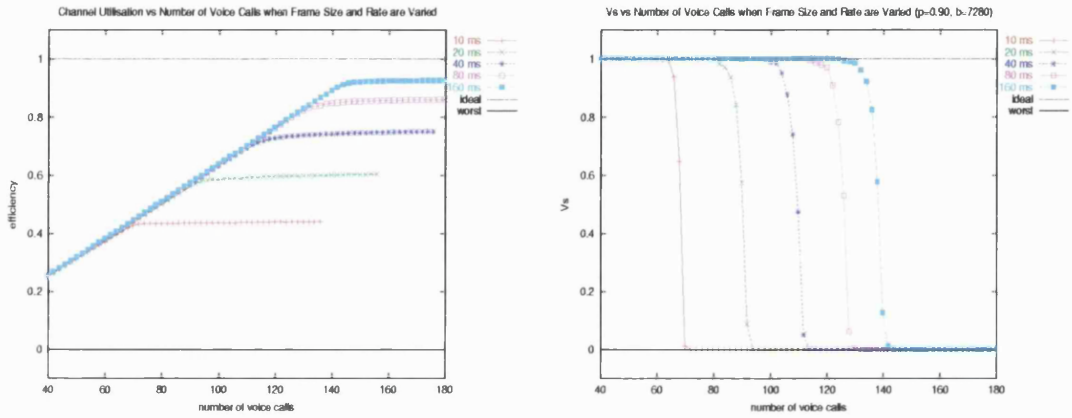
We now show selected results illustrating the relationship between channel utilisation and the maximum number of satisfied users for throughput (V_s), delay (V_d), and jitter (V_j). Figure 2.5(a) shows the channel efficiency as a function of the number of sources

[2]The model's channel utilisation was compared with the results obtained by [56] and [69].

[3]See Appendix A for details of the environment.

source	ϕ (ms)	P bytes
1	10	80
2	20	160
3	40	320
4	80	640
5	160	1280

Table 2.1: 64 Kbps Source Models with Different Frame Sizes and Rates



(a) Channel Efficiency vs Number of Voice Calls.

(b) V_s vs Number of Voice Calls

Fig. 2.5: Channel Utilisation and V_s vs Number of Voice Calls

for the different P-ϕ pairs[4]. In the figure it can be seen that with increasing frame sizes and slower transmission rates, the utilisation increases. In all ϕ's the utilisation reaches a point where it eventually levels off. We would suspect that the number of satisfied users will not be maximum above the knee point of utilisation. This intuition is confirmed by Figure 2.5(b) where the resulting V_s when $p = 0.90$ and $b = 7280$ bytes are shown. In all ϕ cases, the maximum number of satisfied users is below the knee point and when utilisation reaches the knee, the number of satisfied users is zero or near zero. This result is quite significant as it indicates that in terms of utility, operating the Ethernet at the maximum (near collapse) possible utilisation is not in the best interest of users.

Aside from varying the frame size and rate, there are two possible actions that can

[4]For simplicity of discussion we will from now on refer to this pair with the ϕ parameter.

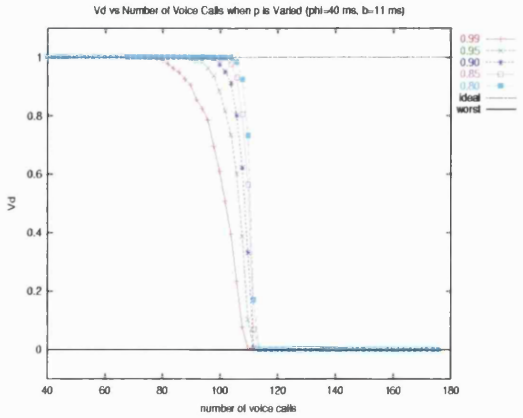
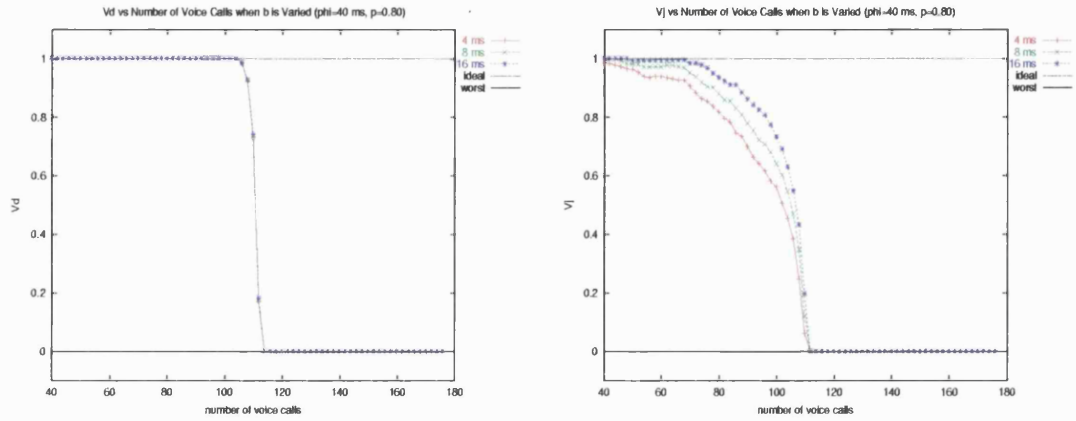


Fig. 2.6: V_d vs Number of Voice Calls with different p's ($\phi = 40$ ms and $b = 11$ ms)

affect the maximum number of satisfied users supported. The first way is by varying the value of p in the utilities. Definition 10 tells us that it is more difficult to satisfy sensitive users. Therefore, it is likely that we can make more of the less sensitive users happy than a group of sensitive users. This is confirmed by the result shown in Figure 2.6. The decline of the number of satisfied users, V_d, is smoother at higher p's. For lower p's, the decline is more sudden and abrupt. In terms of management, a slow decline is preferred because it allows for possible control actions to take place. These results are interesting because they show different possible trade-offs between control and meeting the objectives.

The other factor that can affect the value of the maximum number of satisfied users is the target bound b. Similarly to p, we would expect that a much more relaxed bound would allow for more users to be happy. However, it seems that this is not always the case. Figure 2.7(a) shows one example. In this figure, the curves representing the number of satisfied users for the three different delay bounds follow the same behaviour as the number of voice calls is increased. In this particular situation, no differences can be inferred. This not the case for jitter bounds. In Figure 2.7(b), we find that increasing the jitter bounds improves the V_j when the number of voice calls is increased. From both observations, it appears that the effects of b are QoS dependent and quite possibly environment dependent.



(a) V_d vs Number of Voice Calls (b) V_j vs Number of Voice Calls

Fig. 2.7: V_d and V_j with Different Target Bounds b's ($\phi = 40$ ms, $p = 0.80$)

2.5 Summary

Our aim in this chapter was to represent both the state and degree of user satisfaction in a form that can be used for characterising network services and managing resources. We began by further motivating the need for such a representation of user satisfaction. In particular, we highlighted how existing views fail to encapsulate both aspects of satisfaction completely. We then introduced a notion of utility which we believe successfully models both state and degree of satisfaction. In a simple example, we demonstrated how our representation of satisfaction can be used as performance index which could potentially be used for resource management.

Chapter 3

Utility in Traffic Management

3.1 Overview

This chapter establishes the context in which utility can be used for traffic management. We begin by looking at how networks use various mechanisms to achieve service quality. These mechanisms are then abstracted and are described in a general framework on how utility may be applied to service differentiation.

3.2 Introduction

In an integrated service environment, the primary objective is the satisfaction of the application requirements. This is difficult to accomplish because of the diversity in the QoS requirements of applications. Two opposing views on how to achieve this objective have been the subject of numerous debates [11]. The first view is that the problem can be overcome by throwing bandwidth at it. It assumes that applications will always get the resources they need because bandwidth is infinite. The main argument for the viability of this approach is the steady fall of the cost of bandwidth. However, this method potentially wastes resources when it is available. In addition, it may not always be feasible because bandwidth cannot be expected to be available on an end-to-end basis. The second option makes use of traffic management in an attempt to use resources more efficiently. In order to achieve service commitments through traffic management, the routers may need to have more sophisticated scheduling and

buffering mechanisms. This means that the network elements become more complex and consequently require additional overhead. In addition, some form of signalling mechanism may be needed to allow for coordination between the different network elements inside the network.

We believe the choice between the two approaches is a tradeoff between cost constraints and performance gains. It is even reasonable to assume that future service differentiation would combine the merits of both approaches. In this dissertation we deal mainly with achieving efficiency through traffic management.

We now look at how some networks exercise control to satisfy the service requirements of applications using their facilities. There are several important mechanisms for both real-time and data traffic management that are used in existing network models. Some of these mechanisms include: admission control, resource reservation, traffic control, scheduling, discard mechanism and flow control.

3.2.1 Telephone Network

The telephone network is an example of a circuit-switched network. In this type of network, a fixed physical circuit is established between two communicating parties. The data exchanged between parties do not carry routing information. Resources are held throughout the duration of the call. Calls are admitted based on guarantees on the availability of resources. If no link with sufficient resources is available, calls are blocked. This is the primary QoS measure and is known as the blocking probability. A lower blocking probability indicates better service. This type of management where resources are reserved is possible because the traffic is continuous and is generated at a steady rate. Since the traffic is well-characterised, pre-allocating resources would be sufficient to guarantee QoS constraints. The major disadvantage of this approach is the inefficient use of resources when applied to applications exhibiting burstiness [52]. The circuit-switching approach is therefore similar to the notion of using infinite bandwidth to realise service commitments.

3.2.2 Internet

The Internet is the best example of a best-effort network. It has no explicit admission control procedure. Applications can generate data and start transmitting them anytime they want. Intermediate nodes transport variable-sized packets on a first-come first-serve basis. Traffic from a session do not necessarily follow the same route and may arrive at the destination out of order. Resources in the Internet are shared and are efficiently utilised. However, because of the high degree of sharing, congestion may occur. Congestion happens when the traffic entering a node momentarily exceeds its capability to forward packets to the next node. When this happens, buffers are filled and incoming packets may be discarded. The Internet uses three main strategies to alleviate this problem namely: routing, flow, and error control. Routers in the Internet can forward packet to other routes. This capability is ideal in finding alternative and less congested routes.

In times of congestion, sources are given feedback to decrease their transmission rate. The flow control of sources allows the network to recover. Error control takes advantage of the sequence numbers of packets to ensure their delivery to the destination in sequence without loss or duplication. These techniques provide adequate service for the transmission of data applications. However, since resources are shared and congestion occurs during periods of high demand, packets may experience excessive delays and no firm guarantees can be given for the transport of real-time traffic.

Currently, there are efforts to extend the Internet service model to include support for applications with time-varying and high bandwidth requirements. There are two such models, namely the Integrated Service Model (INTSERVE) [10] and the Differentiated Services (DIFFSERVE) Model [8]. In the INTSERVE model, applications can select between any of these three services: Best-Effort, Controlled-Load and Guaranteed Services. The Best-Effort is still the same as before while Controlled-Load service is a service that emulates a Best-Effort service operating under light conditions [81]. The Guaranteed service [68] promises a minimum sustained bandwidth and firm bounds on delay. In all cases, each flow is treated individually and

per-flow state information is maintained. Unlike INTSERVE, DIFFSERVE has a limited number of classes that can be predetermined administratively. Compared to per-flow information, the use of classes significantly reduces the overhead kept inside routers. The classes defined in DIFFSERVE are premium [61, 43] and assured or olympic services [38]. A premium service is characterised by low-delay and low-jitter. The assured (olympic) service on the other hand is an enhanced best-effort service that offers different levels in times of congestion. For example, three levels can be specified in decreasing quality: gold, silver and bronze. This can be implemented by having smaller input load for the gold service queue than the silver queue. Both the Integrated Services and Differentiated Services frameworks espouse a major shift from the best-effort paradigm to a less decentralised managed network involving admission control, resource reservation, scheduling and discard policies.

3.2.3 Asynchronous Transfer Mode

The Asynchronous Transfer Mode (ATM) uses the benefits of both circuit-switched networks and datagram networks in its model. A virtual circuit is established with small fixed-size cells exchanged between two communicating entities. ATM circuits are logical which allows them to take advantage of the statistical multiplexing gains from resource sharing. It handles congestion by reserving resources based on expected bandwidth use, traffic regulation and shaping. Unlike the Internet, ATM cells do not carry the complete routing information. The route where cells travel is set during admission. As with telephone networks, admission control is explicitly invoked but involves a complex signalling mechanism. A set of service classes is offered from which applications can choose. In admitting a request, the network requires applications to signal their QoS requirements and traffic characteristics. The network can only accept requests if sufficient resources are available to guarantee the application's requirements.

3.3 Using Utility for Traffic Management

Traffic management objectives are often varied. Some objectives may include improvement in efficiency, achievement of performance targets or revenue generation. It may also be the case that there is more than one objective. Throughout this dissertation, our objective is to satisfy performance expectations of as many applications while ensuring less demanding applications are not sacrificed. This section outlines how utility can be used with various traffic management mechanisms to accomplish this goal.

3.3.1 Admission Control

The function of Admission Control is important because it determines which flow is allowed inside the network. For best-effort networks no flow is ever denied access. However, for enhanced network services like ATM, DIFFSERVE and INTSERVE, admission must limit the number of flows accepted to maintain QoS requirements. The decision to admit a new flow is influenced by the traffic load and the QoS requirements of the current and new flows. In addition, it is also dependent on feedback information from mechanisms such as the scheduler and queue manager. The admission controller determines if acceptance of a new connection can be supported along with the existing connections. Most admission policies are pessimistic or assume worst-case traffic pattern scenarios. Although these policies can ensure QoS requirements, the network utilisation is often low. This is because existing flows may not always be transmitting at their declared maximum rates. An alternative is to use measurement-based approaches [45]. In these approaches, flows can be admitted based on average or even worst case data rates. The difference is flow descriptors of admitted flows are adjusted based on their bandwidth usage to allow for more connections to receive service.

The use of utility in admission is similar to that of measurement-based approaches. However, instead of measuring usage, we focus on the utility value of the service. There are two ways utility can be applied to admission control. The first is to admit new flows while $X\%$ of connections are still perceived to be happy. If the number of satisfied users falls below this percentage, no new flows are accepted. The admission

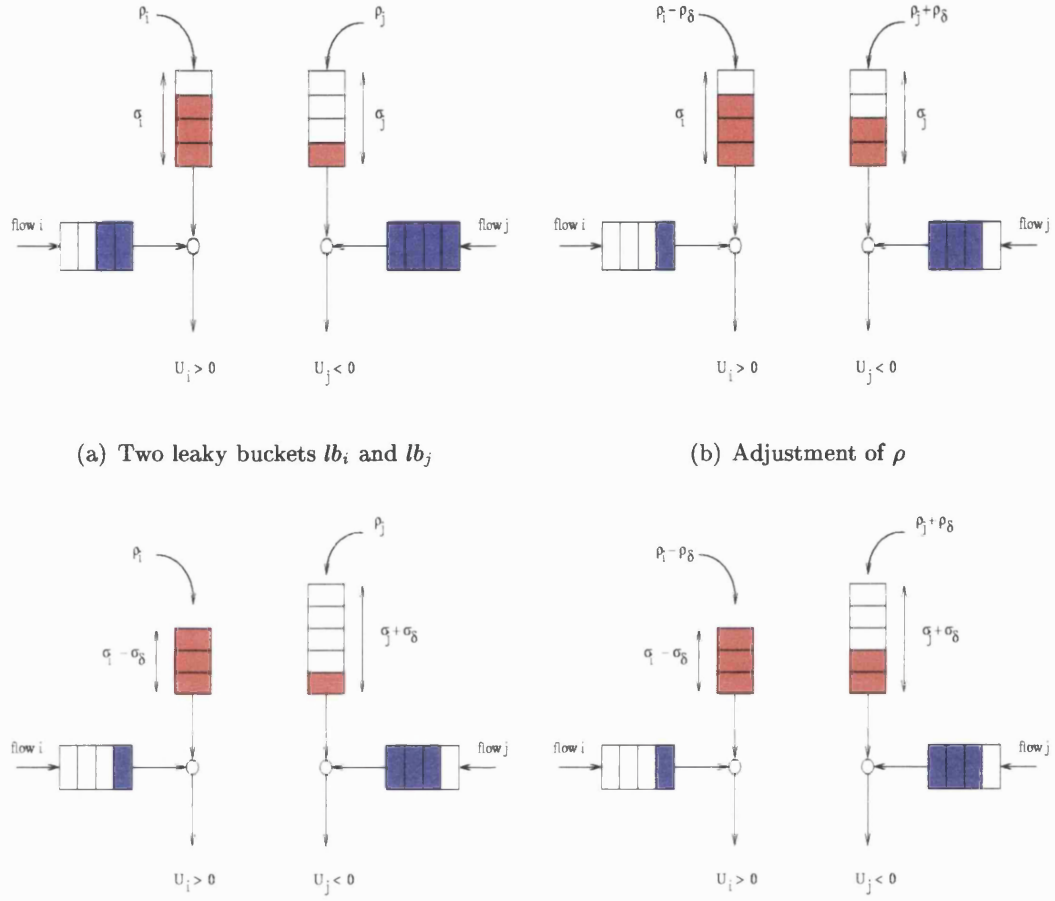
controller can also use utility information to identify connections that are extremely satisfied. Once identified, the admission controller can try pooling resources from these connections to support a new request. This action normally involves coordination with other management mechanisms.

3.3.2 Packet Classification

To effect service differentiation, it is important that each packet is distinguishable from others. Identification is usually accomplished using information in the packet headers. Based on this information, scheduling, queue management and other decisions can be made. Packet classifier algorithms must be simple and fast so that they can be implemented in hardware [51]. This avoids violations in QoS commitments due to packet pre-processing and classification. The requirement of fast packet processing prohibits the use of utility information for every packet because of the additional overhead. However, utility can still indirectly influence the mapping of packets to service classes either during set-up or renegotiation. The decision to admit, elevate or demote a flow is based on feedback from other mechanisms using utility inside the network. For example, after some time interval the scheduler may indicate to the packet classifier that packets belonging to a dissatisfied flow are to be promoted to the next higher service category. The packet classifier has nothing to do with reclassification decisions. It simply updates its packet mapping information according to feedback from other traffic management mechanisms.

3.3.3 Traffic Shaping and Conditioning

In network models like ATM and INTSERVE, a contract between the application and the network is agreed upon before traffic is allowed inside the network. Normally, the contract includes the traffic profile of flows and their QoS requirements. The traffic profile helps the network determine the necessary resources needed to achieve the QoS requirements of flows. The main task of a traffic shaping mechanism or a traffic conditioner is to regulate traffic and to ensure that flows entering the network keep to their specified profiles. Traffic shaping mechanisms often smooth out the burstiness



(a) Two leaky buckets lb_i and lb_j

(b) Adjustment of ρ

(c) Adjustment of σ

(d) Adjustment of both σ and ρ

Fig. 3.1: Leaky-Bucket Parameter Adjustments

of traffic streams. An example of a traffic shaping mechanism is the leaky-bucket regulator [76]. In a leaky-bucket regulator, tokens accumulate in a steady rate ρ inside a fixed-size token bucket. Each token represents the number of bytes allowed to enter the network. The total depth σ of the bucket is the maximum number of bytes allowed inside the network at any given time. Packets arriving at the regulator can only be transmitted if there are sufficient tokens inside the bucket. Otherwise, they wait for more tokens. When a packet is transmitted, the equivalent of the packet size in tokens is removed from the bucket.

Now consider two leaky-buckets lb_i and lb_j shown in Figure 3.1(a) which regulate two flows i and j. Assume that flow i is extremely happy with the QoS it is receiving while flow j is not happy. Furthermore, suppose that by moving and adjusting their leaky-bucket parameters, the utility of flow j can be improved. There are several ways this can be accomplished. The first alternative is to reduce the token generation rate of lb_i from ρ_i to ρ_i-ρ_δ and give ρ_δ to lb_j (Figure 3.1(b)). The second option is to decrease the token bucket depth of lb_i and increase the depth of lb_j by σ_δ (Figure 3.1(c)). The last alternative to improve utility is to combine the first two options. In this approach, both the depth σ and the rate ρ are adjusted (Figure 3.1(d)).

3.3.4 Packet Marking

Packet marking is one of the mechanisms used by the DIFFSERVE model to achieve service differentiation. The application, the edge routers or both have the task of marking packets. The classification and marking of packets are determined by service level agreements or contracts between customers and networks and between the networks themselves. The marking, which is placed in the DS field [60] of packets, determines the per-hop behaviour (PHB) the packets will receive [15, 38, 61, 43]. Normally, marking is based on whether packets are in-profile or out-of-profile of their leaky-bucket traffic characterisations. For example, packets exceeding the prescribed aggregate traffic profile can still be allowed to enter the network but it is marked as expendable or low priority. Thus during congestion, this packet is preferentially dropped before any non-expendable packets.

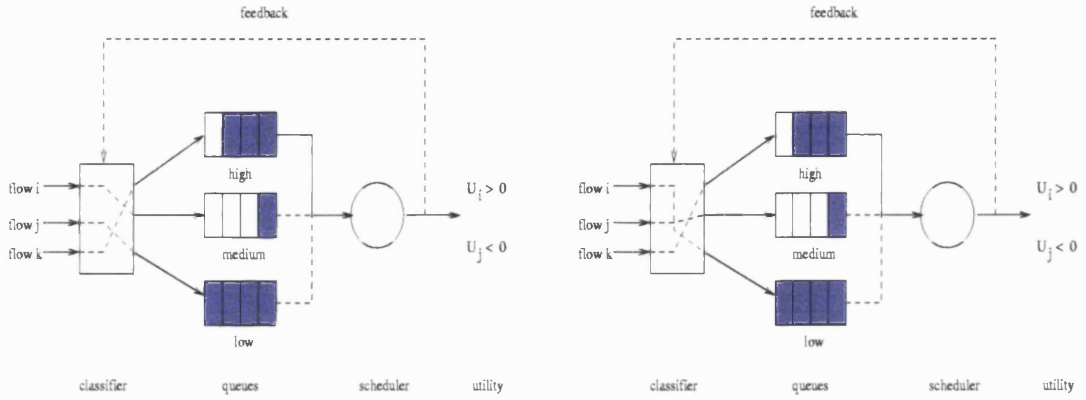
As with the earlier discussion in packet classification, utility can be used to reclassify the marking applied to flows. The decision to reclassify flows is again made by traffic management mechanisms inside the core network with the edge routers and applications performing the remarking. Another way to use utility is to change the leaky-bucket parameters associated with a class. This action is similar to how utility was used in Section 3.3.3.

3.3.5 Scheduling

The primary function of the scheduler inside the router is to determine the order by which packets are served. The choice influences the delays experienced by packets and the bandwidth allocated to certain connections. In some cases, guarantees on jitter are possible. The bounds on delay and guarantees on sustained bandwidth are important for real-time applications. In addition to providing performance bounds, some degree of protection and fairness can be offered to connections depending on the type of scheduler. Protection and fairness allow well-behaved applications to receive service in the presence of congestion due to malicious sources. The capability to give performance guarantees and ensure fairness and protection are desirable scheduler properties. A number of schedulers have been developed to support such properties. These schedulers and their properties are discussed in some detail in the reviews in [86, 3, 83, 84]. In this section, we focus on a range of scheduling disciplines and discuss the procedures on how utility may be used.

First-In First-Out

A First-In First-Out or FIFO queue schedules packets in the order they arrive. Flows passing this type of scheduler can experience unpredictable delays and cannot get rate guarantees. The resulting QoS is highly dependent on the load and arrival patterns of user traffic. Generally, it is almost impossible to support real-time traffic except perhaps when traffic load is light or the applications can adapt to service variation. The primary advantage of this scheduler is that it is easy to implement and requires minimum overhead. With FIFO operations, we only envisage using VBU to limit the active connections passing the scheduler. This requires the scheduler to keep VBU information for each flow or groups of flows. The information is then passed on to the admission controller who decides whether to accept or reject a new flow based on a defined operating threshold (e.g., percentage of satisfied flows).



(a) Flows i and j are initially assigned to medium and low priority queues respectively. Flow i is happy while flow j is not.

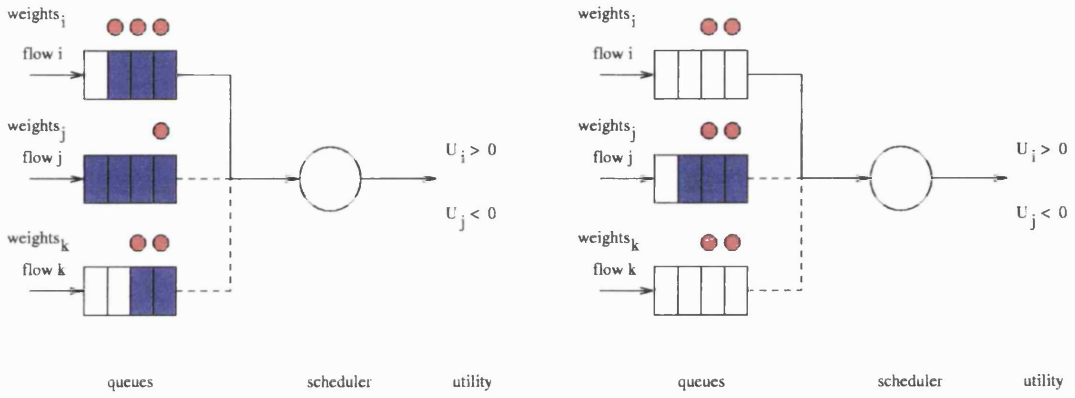
(b) Flow j is now assigned to the higher priority queue while flow i is placed in the lowest priority queue.

Fig. 3.2: Priority Queue Reclassification

Priority Scheduling

In Priority Scheduling, packets are classified according to service priorities. A queue is maintained for each priority level and is serviced in a FIFO manner. A lower priority queue is only serviced if all queues with a higher priority are empty. This gives high priority packets guarantees on performance albeit as a group. However, one main drawback is the likelihood of starving the lower tier queues. This service mechanism is the proposed scheduler for the DIFFSERVE model [61].

The use of VBU information in a scheduler is not restricted to informing the admission controller to limit the number of connections. The scheduler can also use VBU information to change the service category of flows. For example, if the scheduler recognises a dissatisfied low priority flow, it can instruct the packet classifier to reclassify the flow to the next higher level. To facilitate this reclassification, it may be necessary that a satisfied flow belonging to the higher priority queue replace the dissatisfied flow in the lower priority queue (Figure 3.2). For these actions to work, the scheduler must be able to keep sufficient information about the flows to determine who receives better service as well as those who will get degraded service. In addition, the scheduler must have a means to communicate the changes in priority to the packet



(a) Flows i and j have weights of 3 and 1 respectively. Flow i is happy and flow j is not.

(b) After reassignment, flow j is given a weight of 2 while flow i's weight is reduced to 2.

Fig. 3.3: Round-Robin Weight Reassignment

classifier.

Round-Robin

In Round-Robin schedulers, a queue is maintained for each flow class. As the name suggests, each queue is visited in a round-robin fashion. In each iteration, at least one packet is forwarded before the scheduler moves on to next queue. If the number of packets forwarded is different for each queue, then the selection is said to be weighted. The Round-Robin scheduler avoids the problems of starvation associated with a Priority Scheduler but the performance guarantees are looser.

A likely use of VBU in a Round-Robin scheduler is in its weighted variant (Figure 3.3). The service weights of queues are adjusted according to satisfaction levels. If there are dissatisfied flows, the scheduler can increase their weights. If required, the weights of satisfied flows may be decreased.

3.3.6 Discard Policy

The Discard Policy implemented inside the router determines when and which packets are dropped. Normally, packets are thrown away when queues build up or in anticipation of congestion. Depending on which policy is employed, some degree of

differentiation may be possible. In this section, we look at both Tail-Drop, which is used extensively by the Internet, and Threshold-Based approaches. In both cases, we describe how utility may be integrated into the schemes.

Tail-Drop

In Tail-Drop (Drop from Tail), packets arriving in a full queue are dropped. This type of policy can be used alongside any scheduler. This means that packets belonging to the same service category are dropped with equal likelihood. For example, in schedulers with multiple queues, Tail-Drop discards packets only from the overflowing queue. The type of scheduler used and the level of congestion may have a significant impact on some flows. When Tail-Drop is combined with the Priority-Scheduler, the lower level priorities queue could experience excessive losses as a result of starvation. The use of VBU is therefore constrained to the type of scheduler used alongside Tail-Drop. User expectation can be achieved either by: admission control, reclassification of packets, realignment of service weights or a combination of one or more of these techniques.

Threshold-Based

Threshold-Based discard mechanisms [32, 14, 37], throw packets away when the buffer occupancy exceeds a defined set of operating values. Normally, a threshold is associated with each group of flows supported by the buffer manager. Threshold-Based schemes avoid problems by dropping packets early. The threshold setting allows for various levels of service commitments to be offered. A buffer manager that uses utility could take advantage of the threshold settings by changing the value assignments. Dissatisfied flows get increased threshold values while the thresholds for satisfied flows are decreased. These actions result in lower losses for the dissatisfied flows and higher packet drops for satisfied flows.

3.4 Summary

Our goal in this chapter was to examine traffic management mechanisms at a functional level to abstract features that can be exploited for use in a utility managed environment. Firstly, for mechanisms that do not provide support for differentiation, we recommended defining a threshold below which flows are not admitted. This means that no new connections are allowed entry to the router or even a network if the threshold value is breached. The threshold is based on the number or percentage of satisfied users. Evaluation can simply involve checking against utilities associated to a flow or groups of flows.

Secondly, differentiation tools and resources such as access tokens, service weights, buffer thresholds and even buffer space, were shown to be amenable for utility management. Along with utility, the concept of sharing is easily implemented as these tools naturally discriminate between connections who have more and those who have less. Once the state and degree of satisfaction of flows are recognised, realignment of resources is straightforward.

Finally, traffic management is a complex issue that involves various mechanisms. The achievement of performance requirements is a collective goal between different mechanisms. Thus, the fulfilment of this goal depends on feedback from each other. We have identified, although in general terms, these relationships so that VBU information is used and communicated effectively between these mechanisms.

Chapter 4

Experimental Environment

4.1 Overview

This chapter describes the details on how utility is measured, the simulation environment and the evaluation process. Specifics about the algorithms used are not discussed here but are described in later chapters. We use the loss utility results from a FIFO router with different buffer sizes as an example to introduce the evaluation process.

4.2 Measuring QoS and the Resulting VBU

In Chapter 2, we presented an analysis of the resulting throughput, delay and jitter utilities. Specifically, we used these utilities and Equation 2.5 to find the maximum number of satisfied voice calls supported by a 10 Mbps Ethernet LAN with source loads with different transmission characteristics. Up to now we have omitted giving operational descriptions on how the QoS indices and their resulting VBU are measured. This was intentional because we want to discuss these issues separately in this section.

4.2.1 Preliminaries

To simplify the discussion, let us consider the system we want to measure or observe as a black-box. This black-box could represent an entire network, a domain or a single router. The specific details of what is inside the box are irrelevant for now. The important thing is that this representation gives an abstract view of how the QoS indices may be observed. This allows for the definition of some standard notations

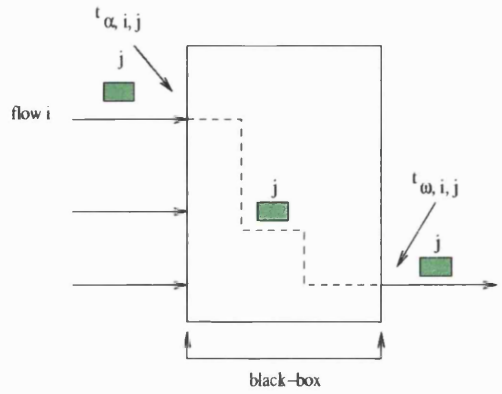


Fig. 4.1: Important Timings in a Black-Box Model

that can be used to describe the QoS indices.

Consider a packet j belonging to flow i passing through the black-box. We indicate the size of the packet as $s_{i,j}$. To measure a specific QoS, we must record the time the packet enters the black-box and the time it comes out of the box. We denote these times as $t_{\alpha,i,j}$ and $t_{\omega,i,j}$ respectively (Figure 4.1). As we shall see later, these values change depending on what QoS is measured and where it is measured.

The four QoS indices of interest are: throughput, delay, jitter and loss. The first three indices have been used to introduce the VBU concepts in Chapter 2. Loss and delay will be the focus of the experiments in Chapters 5 to 7.

4.2.2 Throughput

Throughput (T) is the average number of bytes exchanged between two users over a period of time. Using the black-box representation, throughput T_i is the number of bytes belonging to flow i coming out of the black-box in a time interval. For every packet j of flow i exiting the black-box, the throughput changes. A continuous measure for throughput $T_{i,j}$ is thus given by Equation 4.1. Note that throughput $T_{i,j}$ is averaged with respect to $t_{\omega,i,1}$, which is the time the first packet came out of the black-box.

$$T_{i,j} = \frac{s_{i,1} + \sum_{n=2}^{j} s_{i,n}}{t_{\omega,i,j} - t_{\omega,i,1}} \quad , \quad i \geq 1, j \geq 2 \tag{4.1}$$

In the experiments in Chapter 2, the black-box was the Ethernet LAN and $t_{\omega,i,j}$ was the time the last bit of packet j came out of the Ethernet. Throughput was measured at the receiver's end.

Given Equation 4.1, we can evaluate throughput utility by comparing $T_{i,j}$ with the throughput bound b of expectation $T(p,b)$. Recall that in Section 2.4.2, the throughput bound was $b = 7280$ bytes. For example, if $T_{i,j}$ is 7300 bytes, then the number of good packets G in Equation 2.1 is increased by 1 (since $T_{i,j} \geq 7280$ bytes). Once $T_{i,j}$ is known, the evaluation of throughput utility is straightforward.

4.2.3 Delay

Delay, like throughput, can be measured in different ways. The approach depends mainly on where delay is observed. For example if it is observed at the receiver, then the delay is end-to-end. If it is at the router then the delay is the time spent in the router. In this thesis, we will be concentrating on both types of delay. Regardless of where delay is measured, it is the difference between the finishing time and starting time from some observation period. Using a black-box abstraction, the delay $D_{i,j}$ a packet j of flow i experiences is given by Equation 4.2.

$$D_{i,j} = t_{\omega,i,j} - t_{\alpha,i,j} \quad , \quad i,j \geq 1 \tag{4.2}$$

In all our experiments involving end-to-end delay, $t_{\alpha,i,j}$ is the time the packet j was first created at the source. The time $t_{\omega,i,j}$ on the other hand is the time packet j arrived at the destination. For Ethernet experiments, the delay $D_{i,j}$ can be attributed to contention or access for the LAN. For experiments involving at least one router, the delay is the result of queuing inside the router as well as the retransmission of packets in the event of buffer overflow. In some cases, the delay associated with retransmission may be considerable. This is because the transmission order of the packets must be

preserved when received at the destination. Packets arriving out of sequence cannot be processed until earlier packets arrive. For this reason, we maintain packet sequence numbers.

For delay measurements involving routers, we refine the definitions of the start and finish times to include at what point m inside the network the measurements were taken. The start time $t_{\alpha,i,j,m}$ is the time the packet j of flow i is completely received at router m. The finish time $t_{\omega,i,j,m}$ on the other hand is the time packet j of flow i was completely transmitted from router m. The delay experienced by packet j of flow i at router m is given in Equation 4.3.

$$D_{i,j,m} = t_{\omega,i,j,m} - t_{\alpha,i,j,m} \quad , \quad i,j,m \geq 1 \tag{4.3}$$

For every packet arriving at the receiver, end-to-end delay utilities can easily be computed. Similar to the computation of throughput utility, the evaluation of delay utility involves the comparison of $D_{i,j}$ with a delay bound b of expectation $D(p,b)$. If $D_{i,j} \leq b$, then both G and N are incremented by 1. Otherwise only N is incremented by 1. After adjusting the values of G and N, computing the delay utility is simply a matter of substituting the values of G, N, p, and q in Equation 2.1. The evaluation of end-to-end utilities is important because it assesses the performance of the algorithm or control strategy with respect to a specific QoS index. However, in terms of management, the utilities measured inside the router are more significant. This is because the local information allows for possible adaptation and control inside the router which is the primary motivation of this work.

4.2.4 Jitter

Jitter is the variation in the overall delay experienced by data or multimedia traffic while traversing a network. It may be described as the non-regular and bursty arrival of traffic patterns at a receiver. Ferrari [26] and Wang and Crowcroft [79] define jitter as the difference from a base delay and the actual delay of a packet. While ElBatt et al. [23] and Figueira and Pasquale [31] use the difference between the maximum

and minimum inter-arrival time over the mean of all inter-arrival times. Still another definition [39] uses the sum of the underflow and overflow of the received media. For our purposes, we define an interval where the jitter experienced by a packet is acceptable [28, 29]. This interval is based on the delay bound b_D and has range of $b_D \pm b_J$, where b_J is the jitter bound.

If the measured delay $D_{i,j}$ of packet j of flow i is within the interval $b_D \pm b_J$, then the flow's G and N should be updated. If the $D_{i,j}$ is not within the range then only N is updated.

4.2.5 Loss

Like delay, packet loss is also measured both at the receiver and the intermediate node for similar reasons. The measurement at the receiver is mainly used as a reference measure for evaluating end-to-end performance. Measurements in a node are used for local management and control decisions. To measure loss, let us again consider the network as a black-box and assume that packets follow the same route and arrive in order with packet $(j-1)$ the last packet coming out of the black-box. Based on these assumptions, a packet j entering the black-box is considered lost if a later packet $(j+n)$, $n \geq 1$, belonging to the same flow i comes out before packet j appears[1]. The number of packets lost in the time interval $[t_{\omega,i,j+n}, t_{\omega,i,j-1}]$ is obtained by deducting from the packet number of the current packet, in this case $(j+n)$, the number of the previously received packet minus one. Since the last packet received was $j-1$, the number of lost packets is therefore equal to n. Formally, the number of packets lost between two successfully received packets is given by the expression in Equation 4.4 while the total number of packets lost is the sum of all these values.

$$L_{i,j+n,[t_{\omega,i,j+n},t_{\omega,i,j-1}]} = j+n-(j-1)-1 = n \tag{4.4}$$

To measure loss utility, there are two cases to consider. The first case is when Equation 4.4 yields a value greater than 0. This means the N of the utility function

[1] If this happens, packet j is not expected to come out.

type	Generation Rate			Size Char		λ
	pdf	scale	shape	pdf	mean	
normal	Weibull	0.0100	1.00	deterministic	40	4000
medium	Weibull	0.0050	0.50	deterministic	40	4000
high	Weibull	0.0001	0.20	deterministic	40	4000

Table 4.1: Traffic Types

is increased by this value and the corresponding utility is evaluated. Alternatively, if the value of Equation 4.4 is equal to 0, then both G and N in the utility function are increased by 1 and the corresponding utility is evaluated.

From hereon, we measure loss at the receiver in the manner described above. We could also measure loss at the routers in the same fashion, but for ease of implementation the router simply counts the number of dropped packets and updates the corresponding flow's G and N accordingly. This is a reasonable way to measure loss because there is no need to keep track of sequences and packet numbers inside the router.

4.3 Simulation Environment

In this section, we describe the different elements of the simulation model. These elements include the source models, expectation mixes, topology, measurement intervals and description of the data analysis.

4.3.1 Source Models

We currently use three types of sources in our experiments. They are classified as normal, medium and high which characterise the burstiness of their packet generation rate. The packet generation rate is taken from a Weibull distribution whose density function is given by Equation 4.5. The corresponding values of scale and shape parameters are shown in Table 4.1. The total offered load λ for each source type is 4000 bytes because the packet sizes are fixed (deterministic) at 40 bytes. In the sim-

ulations in this chapter, we use the normal type of source which is equivalent to an exponentially distributed source.

$$f(x) = \frac{shape * x^{shape-1}}{scale^{shape}} e^{-(x/scale)^{shape}} \tag{4.5}$$

4.3.2 Expectation Mixes

In addition to classifying sources in terms of their traffic characteristics, we also group the sources into classes representing the level of expectation or demand of users for a specific QoS. We group the sources into three classes namely *High Expectation Flows* (HEFS), *Medium Expectation Flows* (MEFS) and *Low Expectation Flows* (LEFS). The HEFS have a target of 99% of the packet meeting expectation, the MEFS 90% and the LEFS 80%. Although these groupings are not exhaustive, they do emulate a wide range of performance demands. The combination of the source models and expectations provide a rich set of scenarios to investigate. A sample of the mixes used in this thesis is presented in Table 4.2. For the FIFO example in this chapter, we use the *bmix* mix.

4.3.3 Single Node Topology

For the remainder of the dissertation, the results are based on a single router configuration. In this configuration, each traffic source is connected to the router by an infinite bandwidth link[2]. A specified host is assigned to be a sink or receiver where measurements are taken. Traffic flows in one direction, from the source to the router and then finally to the receiver.

Depending on the experiment, a specific buffer management scheme or scheduling discipline is implemented inside the router. The buffer size B_{total} is also varied depending on the type of QoS utility studied. For example, if loss issues are investigated, then a smaller buffer allocation is used to force packet loss. For delay, a larger buffer space is required to assess delays resulting from queuing and packet retransmissions.

[2]The resulting delay is essentially zero. A similar assumption was used in [45].

Mix	Expectation Group			Traffic Type
	HEFS	MEFS	LEFS	
bmix	12	12	12	normal
emix-201	24	0	12	normal
emix-210	24	12	0	normal
emix-300	36	0	0	normal
tmix-B10	11	11	11	normal
	1	1	1	medium
tmix-A20	10	10	10	normal
	2	2	2	medium
tmix-A02	10	10	10	normal
	2	2	2	high
tmix-840	8	8	8	normal
	4	4	4	medium
tmix-804	8	8	8	normal
	4	4	4	high
tmix-822	8	8	8	normal
	2	2	2	medium
	2	2	2	high
tmix-642	6	6	6	normal
	4	4	4	medium
	2	2	2	high
tmix-624	6	6	6	normal
	2	2	2	medium
	4	4	4	high
tmix-444	4	4	4	normal
	4	4	4	medium
	4	4	4	high

Table 4.2: Traffic Characteristic and Expectation Mixes

The service rate S *Bps* (bytes per second) at the router is also varied depending on the desired utilisation ρ. However ρ is normally set at 90%. The reason for this simple topology is to be able to prove if it is possible to perform local control decisions using utilities.

4.3.4 Measurement Process

Previously (Chapter 2), only one utility value was obtained for each flow during the entire simulation. This single value was sufficient for the goals of these experiments and for introducing utility. From the individual flow utilities, the number of satisfied users was obtained simply by checking whether the final values were greater than or equal to zero. However, in order to perform management at the router level, we need to have a system of measuring utilities at discrete intervals. Choosing the size of these intervals is non-trivial. If the chosen interval is too large, the schemes may not be able react to short term fluctuations in the traffic load. Alternatively, control schemes based on too short intervals may become over-sensitive and react hastily. Another issue is whether past information should be included in the measurements or to simply divide time by fixed and non-overlapping intervals.

Although finding the *right* interval to perform the management is an important issue, it is not the focus of this dissertation. To show that we can use utilities for local control, we have taken a heuristic approach when selecting the interval where we want to perform management. The interval, which is measured in terms of packets rather than time, is fixed at 250 packets. This means that the resulting utilities for that flow have $N = 250$. In addition to defining the size of the interval, we smooth the data by restricting the interval to include 200 old packets and at most 50 new packets[3].

To illustrate how this measurement interval operates, time is first divided in terms of events called *Packet Successfully Forwarded* (PSF) events. Let $PSF_{i,j}$[4] be the event when packet j from flow i was successfully forwarded by the router. Each

[3]Except for the first interval where all packets are new.

[4]For measurements at the receiver, we rename PSF to *Packet Successfully Received* (PSR) event. The use of PSR will still be the same as PSF except that the notation is changed.

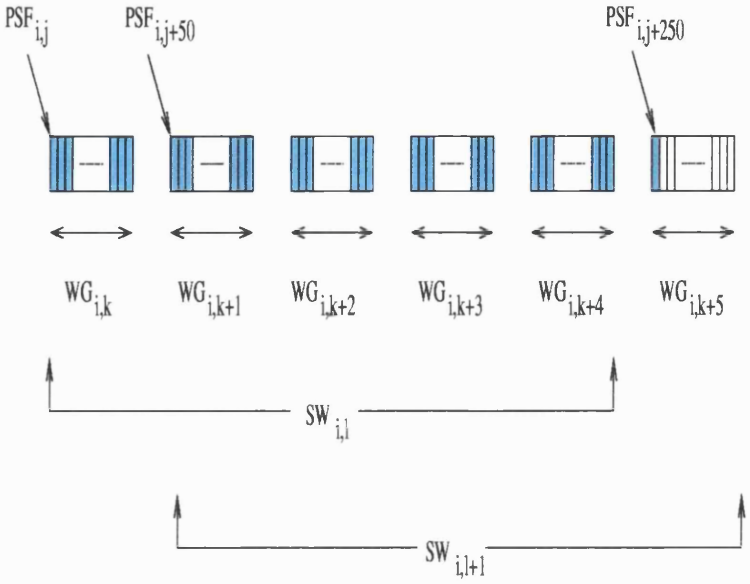


Fig. 4.2: Measurement Window

$PSF_{i,j}$ belongs to a logical grouping of 50 PSF events. We refer to such a grouping as a window group $WG_{i,k}$, where k is the group index. The 50th PSF $PSF_{i,j+49}$, will be associated to $WG_{i,k}$. Five consecutive window groups, for example $WG_{i,k}$ to $WG_{i,k+4}$ define the measurement interval $SW_{i,l}$ where utilities are measured for each flow. On the event $PSF_{i,j+250}$, the sliding window will be moved to $SW_{i,l+1}$. This interval now includes 200 old packet events represented by 4 window groups $WG_{i,k+1}$ to $WG_{i,k+4}$ and at most 50 new packet events represented by $WG_{i,k+5}$. This hierarchy is illustrated in Figure 4.2.

4.3.5 Simulation Notes

All of our experiments lasted for 700 seconds with the first 400 seconds considered as the transient period. For each model and parameter setting, the experiment was replicated at least five times. A 95% confidence interval was used for the average utilities in all experiments.

4.4 Evaluation

The performance of a FIFO server will be used to introduce how we evaluate results. This will also provide baseline information against which the performance of various schemes can be compared. In this example, the size of the router's buffer is varied from 280 to 480 bytes. The router's service rate S is equal to 160000 *Bps*. The *bmix* traffic and expectation mix as described in Table 4.2 is used. This means that there will be 36 sources with a total mean load of 144000 *Bps*. With a utilisation ρ of 0.90 and the router's small buffer, we would expect packets to be lost. The effects of such losses will be manifested differently by the three flow classes. If all the expectations and traffic characteristics were the same, we could expect that all the flows should get fair treatment from a FIFO server because bandwidth is equally shared. However, since the expectations are different, the FIFO server would not be able to differentiate between flows and therefore cannot give preferential treatment. This leads to inefficient use of resources and unfairness. In the succeeding sections, we present the different ways of evaluating the results and introduce our notion of fairness.

4.4.1 Total Number of Satisfied Users and QoS Unhappiness

In Chapter 2, we presented results in terms of the number of satisfied users. In this section and the subsequent chapters, we modify the way we present our index. Instead of using the total number of satisfied users as our index, we use the ratio or percentage of unhappy users. We call graphs of the ratio of unhappy users *Unhappiness* plots. As an example, we show in Figure 4.3 the Loss Unhappiness plot for a FIFO server with a range of buffer sizes. Each point in the graph is the average ratio over a 10 second interval. Intuitively, we should expect that with more buffer space there would be less loss. This is consistent with the trend we see in Figure 4.3 where a 40 byte increase significantly minimises the percentage of unhappy users. However, if we compare the results of using a 280 byte and 320 byte buffer, we find that we cannot really differentiate which one provides better performance.

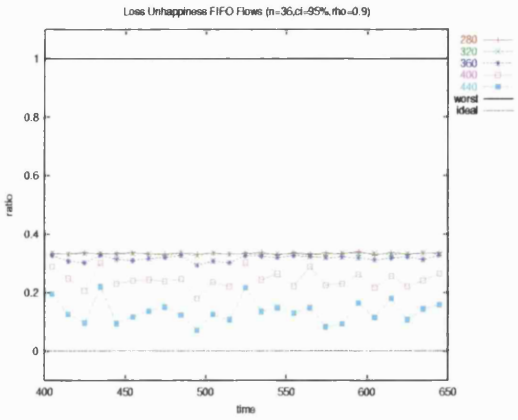
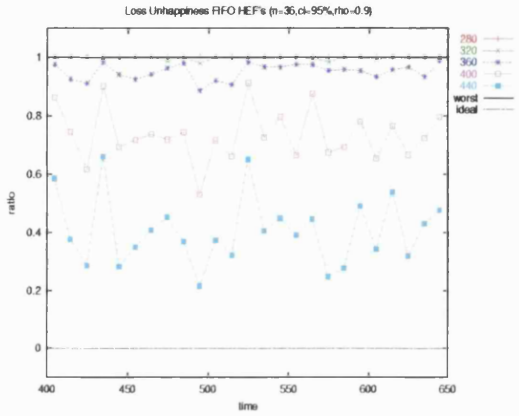


Fig. 4.3: Loss Unhappiness (WG)

4.4.2 Loss Unhappiness and Average Utility Per Class

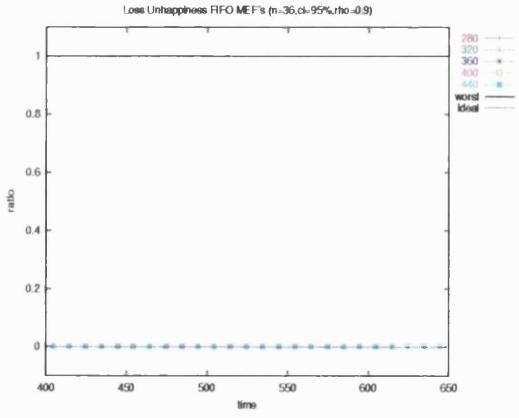
The problem with simply measuring the overall loss unhappiness is that it cannot convey what level of happiness is achieved by the different groups of flows. It is therefore important to have loss unhappiness plots for each of the three expectation mixes. Figure 4.4 shows three related subfigures that plot the loss unhappiness for each class of flows. Figures 4.4(a), 4.4(b) and 4.4(c) give the loss unhappiness for the high, medium and low expectation groups respectively. Note that in overall loss unhappiness (Figure 4.3) only the loss unhappiness of the HEFS (Figure 4.4(a)) is reflected because the MEFS and LEFS loss unhappiness plots indicate that no user from those groups is unhappy.

In order to further assess the performance of any scheme or control algorithm, the average utility is also measured. However we do not use the average utility for all flows because of the different range of values of the different expectation types. More sensitive users like HEFS have a larger range of values than the LEFS, especially in the unhappiness range. In fact, even inside a class we cannot simply take the average utility for all flows belonging to that class. If we do so, the resulting average value may be skewed towards a value in or near the range of unhappiness levels. Consider the LEFS, which have an expectation of $p = 0.80$. The happiness interval of these, which ranges from 0 to 1, is less than the unhappiness interval which ranges from a

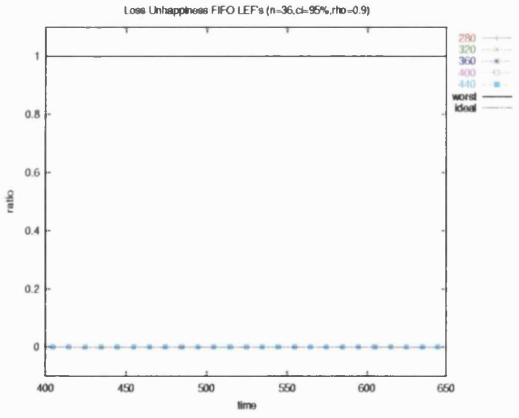


(a) HEFS

Fig. 4.4: The loss unhappiness of the three classes of flows under a FIFO router with different buffer sizes.



(b) MEFS



(c) LEFS

value less than 0 to -4. To address this problem, we use two different average utilities for each class, one for the satisfied users and another for the dissatisfied users.

Figures 4.5(a), 4.5(b) and 4.5(c) show both the satisfied and dissatisfied average utilities for the HEFS, MEFS and LEFS groups respectively for the FIFO experiments. Values above or equal to $happiness_{min}$ are average utilities for the satisfied users while those below are average utilities for the dissatisfied users. Note that these indices cannot be interpreted in isolation. The corresponding loss unhappiness must also be taken into account as both plots give complementary information about performance and behaviour. For example, we do not have a clear indication that using 320 byte buffer is better than using a 280 byte buffer when looking at the Figure 4.4(a). However if we look at the average utilities in Figure 4.5(a), we find that HEFS are generally less unhappy with a 320 byte buffer than with a 280 byte buffer.

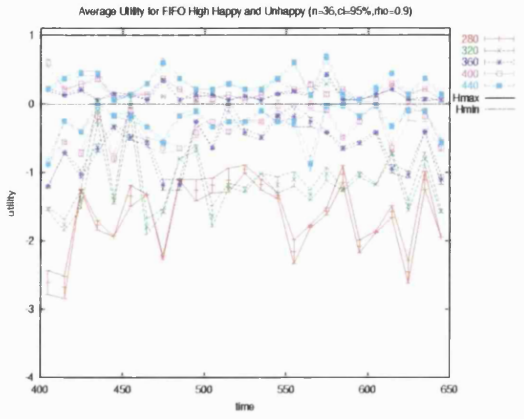
In circumstances where class loss unhappiness cannot differentiate between the performance of two schemes, the average utility proved useful in finding the better scheme. Additionally, the average utilities tell us the degree to which users are happy or unhappy. Looking at the average utilities of the MEFS (Figure 4.5(b)) and LEFS (Figure 4.5(c)), we find that all their flows are extremely happy. Potentially, this knowledge can be useful in traffic management because the utilities of the MEFS and LEFS users could be brought down to improve the utilities of the HEFS. This action may even make some HEFS happier.

4.4.3 Utility Fairness

This section introduces the notion of utility fairness. In this dissertation we define two levels of utility fairness namely, inter-class utility fairness and intra-class utility fairness.

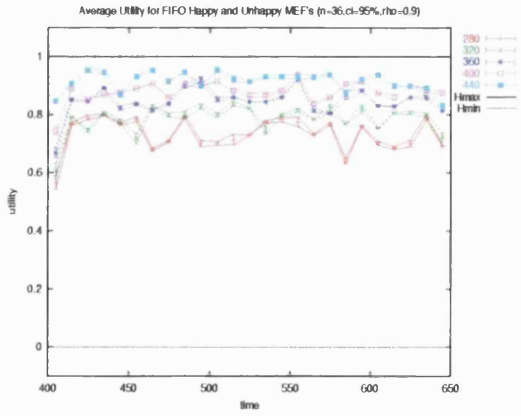
Inter-Class Utility Fairness

Inter-class utility fairness adopts the principle that no higher expectation flow must become happy at the expense of the unhappiness of lower expectation flows. This means that at the very least, fewer lower expectation flows must be unhappy as com-

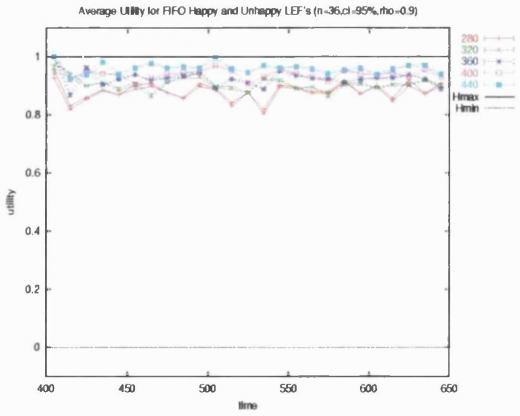


(a) HEFS

Fig. 4.5: The average loss utility of the three classes of flows under a FIFO router with different buffer sizes.



(b) MEFS



(c) LEFS

pared to the number of unhappy higher expectation flows. The idea here is that in terms of requirements, it is easier to provide for the less sensitive flows than the more demanding flows. This philosophy is quite similar to max-min fairness where resources are shared according to an allocation based on maximising the small demands first. A related work by Cao and Zegura [12] has extended max-min fairness to include utilities. Their approach is to maximise the minimum utilities of flows. We differ with their approach in that they do not distinguish between satisfaction and dissatisfaction. In addition to this, we also recognise that class differences affect the values of utilities. A third difference is in the way utilities are formulated. In their formulation, the utilities are functions of QoS while in our case utilities are functions of target QoS and bound (expectation).

For inter-class utility fairness, the goal is to provide and maintain a service equal to $happiness_{min}$ for the lower expectation flows. This is consistent with the overall theme of sharing because it is likely that flows belonging to this group will be the ones sharing their resources. If all flows are satisfied, then inter-class utility fairness is a non-issue. The loss unhappiness results of the earlier section indicate the FIFO server, at least for *bmix* conditions, is inter-class utility fair. This is because all lower expectation flows, MEFS and LEFS, are happy.

Intra-Class Utility Fairness

To evaluate the fairness within a class, we shall adapt the fairness index proposed in [44]. The index $fair(x)$, which has been used in several studies [12, 25, 54, 78] is given by Equation 4.6. This measure is independent of population size, scale and metric. In addition, it is bounded and continuous and can therefore be used for a wide range of application domains.

$$fair(x) = \frac{(\sum x_i)^2}{n \sum (x_i{}^2)} \tag{4.6}$$

We have one utility fairness index for each of the three classes. Since the range of utility values include both positive and negative values, a straightforward use of

Equation 4.6 would yield an inaccurate result. To correct this, we need to transform the ranges to become either positive or negative. We select the latter[5] and transform the utilities u to a new utility u' by subtracting 1. This transformation is given by Equation 4.7.

$$u'_{i,l} = u_{i,l} - 1 \tag{4.7}$$

The new utility $u'_{i,l}$ is a transformation of the utilities measured using the sliding window $SW_{i,l}$ of flow i. Using these $u'_{i,l}$ values and plugging them into Equation 4.6 gives the intra-class fairness over time. Figures 4.6(a), 4.6(b) and 4.6(c) show the resulting utility fairness for the three classes of flows under a FIFO server. We notice that for all cases a smaller buffer space is more fair than a larger allocation. For example, around 92% of the flows are given fair treatment using a 440 byte buffer. This is around 5% less than the fairness achieved by the 280 byte buffer. This not surprising because with extra space, it is likely that in times of bursts, one flow may hold on to the extra space thereby preventing others from using it. This increases the flow's utility while decreasing the utilities of other flows.

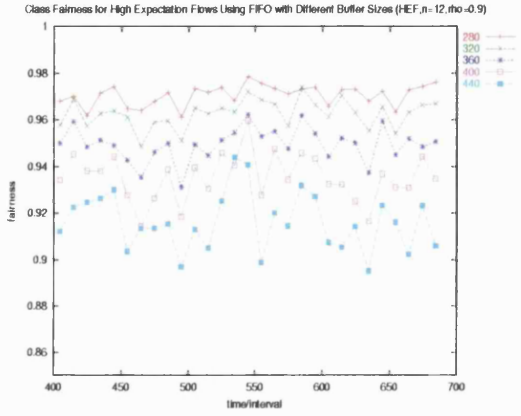
Like the average utility graphs, these three figures should be viewed in the context of the loss unhappiness plots. Since both MEFS and LEFS are all happy, the intra-class fairness results for these two groups are not that important. The important result is that regardless of buffer sizes, the fairness of the HEFS is reasonably high for the FIFO server (between 91% and 97%).

4.5 Summary

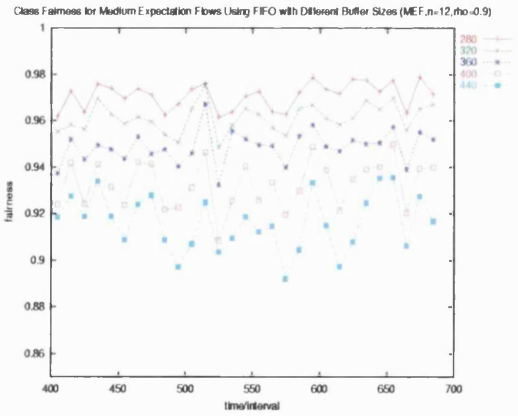
In this chapter, we showed how to obtain the resulting utilities given a measured QoS index. For each flow's utility, the number of steps of the procedure is constant as it only involves simple operations. For all cases, one comparison between the measured QoS and target expectation and at most two additions are made before the utility is

[5]The index would have given the same results had we added p/q to the utilities to make the range positive.

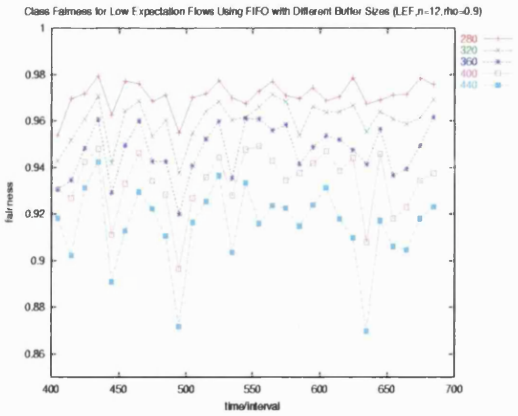
Fig. 4.6: The measured fairness of the three classes of flows under a FIFO router with different buffer sizes.



(a) HEFS



(b) MEFS



(c) LEFS

evaluated by substituting the resulting G and N parameters into the utility function. This is an important characteristic of the operation because in traffic management, the ability to scale is desirable.

From the resulting utilities we also derived measures for evaluating the performance of management schemes. We presented four measures and discussed the conditions when each type is applicable. The first type *Overall Unhappiness*, measures the total percentage of dissatisfied users. This measure is ideal for giving an overall view of a management scheme's performance. However, because of the measure's generality, class specific performances cannot be evaluated. For this level of detail, measures of *Class Unhappiness* and *Class Average Utility* are used. The *Class Unhappiness* gives the percentage of dissatisfied users of a class. The *Class Average* measures the mean utilities of satisfied and dissatisfied users in a class.

The ability of a scheme to achieve the goal of making less demanding users[6] happy first is measured by *Inter-Class Utility Fairness.* This measure can be evaluated by checking and comparing the levels of the *Class Unhappiness* measures. Another utility fairness measure is *Intra-Class Utility Fairness. Intra-Class Utility Fairness* measures fairness within a class. This index is suitable for evaluating fairness when traffic characteristics of flows belonging to a group are varied.

[6]Level of demand is in terms of expectations.

Chapter 5

Loss Management Schemes

5.1 Overview

In this chapter, we evaluate the buffer management scheme described in [37]; we call this the *G+98* scheme. We also propose an alternative scheme that makes use of Value-Based Utility (VBU) in allocating buffer space. We use the metrics and indices defined in Section 4.4, namely: loss unhappiness, average utility, inter- and intra-class fairness, as the basis for our comparison of the two schemes. Our objective is to assess the capability of these schemes to keep all flows happy or failing that, establish the conditions for the closest approximation to overall happiness.

5.2 A FIFO Scheme Using Buffer Thresholds

We describe the buffer management scheme we call *G+98* [27]. We then evaluate its performance for different Quality of Service (QoS) demands; we use the expectation mixes defined in Section 4.3.2. Our first objective is to find which buffer threshold assignments allows the G+98 scheme to satisfy all users. We also seek to understand and classify the effects of threshold settings on G+98's effectiveness in allocating buffer space.

5.2.1 Background

In the G+98 [37] scheme, a FIFO buffer was partitioned into logical units. These units were then allocated to flows; the assignment of units to a flow depends on the flow's

threshold value. This value represents the maximum allowed buffer usage for each flow. If a flow has exceeded this threshold, a router rejects incoming packets of that flow. The allocation of buffer thresholds for every flow is expressed by the following:

$$B_i = B_{total} * \lambda_i / S + \sigma_i \tag{5.1}$$

where

B_i = the logical buffer threshold assigned to flow i

B_{total} = the total physical buffer space

λ_i = the mean arrival rate (bytes/sec) of flow i

S = the service rate of node (bytes/sec)

σ_i = the burstiness of flow i

Buffer space can only be guaranteed if the sum of all logical buffer allocations (B_i) is less than or equal to the physical buffer space:

$$\sum_{i=1}^{n} B_i \leq B_{total} \tag{5.2}$$

The packet admission algorithm of the G+98 scheme is illustrated in Figure 5.1. In this scheme, a flow is said to be within characterisation if its incoming packets can be admitted without exceeding its threshold. Since the FIFO buffer allocation is based on logical units, the total logical buffer space allocation may not be the same as the actual available space. Hence, a flow's incoming packet will only be accepted if the flow is within characterisation and physical buffer space is available.

5.2.2 Buffer Thresholds

The G+98 scheme was originally devised to provide rate guarantees using simple buffer management. Guerin et al. [37] showed that they could assure different levels of guarantees by trading off efficiency against complexity. However, their scheme

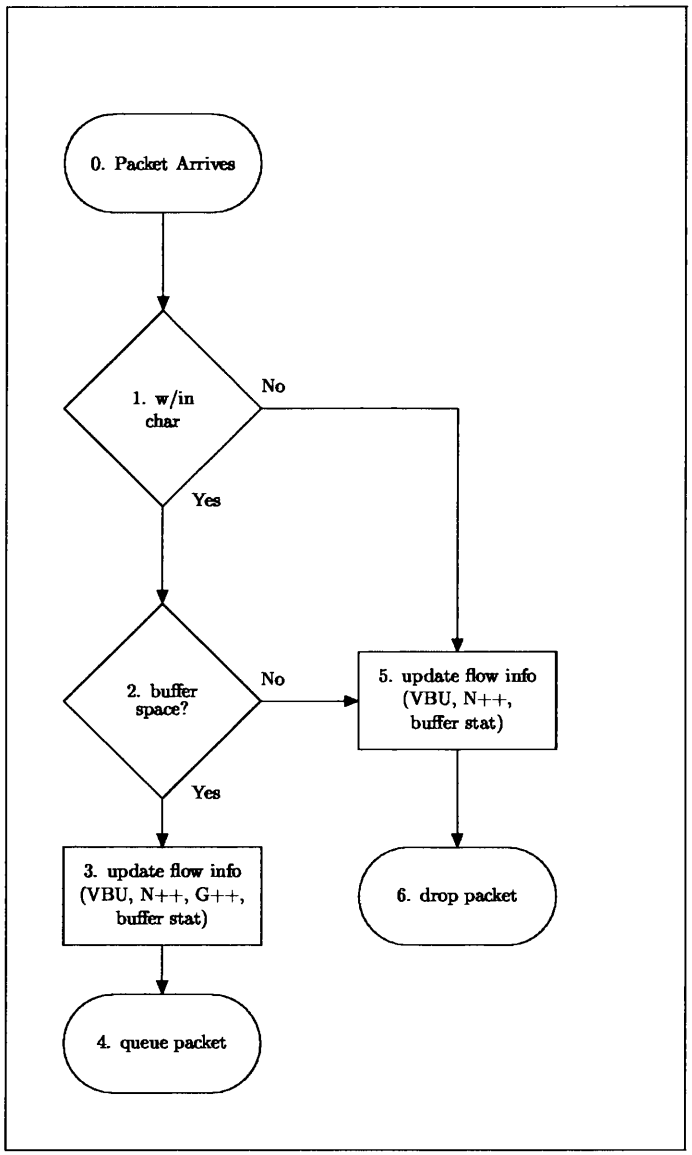


Fig. 5.1: G+98 Packet Dropping Algorithm. We discuss variants of this scheme in a later chapter.

requires complete information. All variables in the right hand side of Equation 5.1 must be known. This presents a problem when it is not possible to obtain all of these values beforehand. For example, the burstiness factor σ can be correctly estimated only if the sources are leaky-bucket constrained.

We consider how the G+98 scheme performs under a different objective and with lesser initial information. Instead of rate guarantees, we use G+98 to satisfy loss expectations of flows. We do not assume that sources are leaky-bucket constrained. We also replace the use of a burstiness factor; instead, σ denotes the differences in flow expectations. For example, a higher σ value may be allocated to MEFS than LEFS to allow the MEFS more space in the buffer.

For our examination, we assume a router with: $B_{total} = 400$ bytes and $S = 160,000$ *Bps*. We also use the *bmix* base conditions in Table 4.2. This means the traffic is normal and each of the three expectation (HEFS, MEFS and LEFS) groups has 12 flows.

5.2.3 Performance of G+98 Under Different $\sigma_H\sigma_M\sigma_L$ Tuples

We want to find the combination of σ values for high expectation flows σ_H, medium expectation flows σ_M and low expectation flows σ_L that would result in the best G+98 performance. We also want to understand the effects of different tuple $\sigma_H\sigma_M\sigma_L$ combinations on each flow's performance.

The tuple $\sigma_H\sigma_M\sigma_L$ combinations we consider and their relationship to each other are illustrated by the experimental design tree in Figure 5.2. Our initial condition is described by the root of the tree ($\sigma_H \geq \sigma_M \geq \sigma_L$). This states that the highly sensitive flows (HEFS) must be given at least equal amount of resources (σ) as less sensitive flows (MEFS and LEFS). This allocation policy is possible because we assumed all flows have similar traffic characteristics. The root of the tree branches out into two subgroups and the second subgroup has two further branches. The leaf nodes show the hexadecimal values of the tuples that we used in the simulations. For example, a threshold setting of *321* means that $\sigma_H = 3$, $\sigma_M = 2$ and σ_L=1. In the succeeding

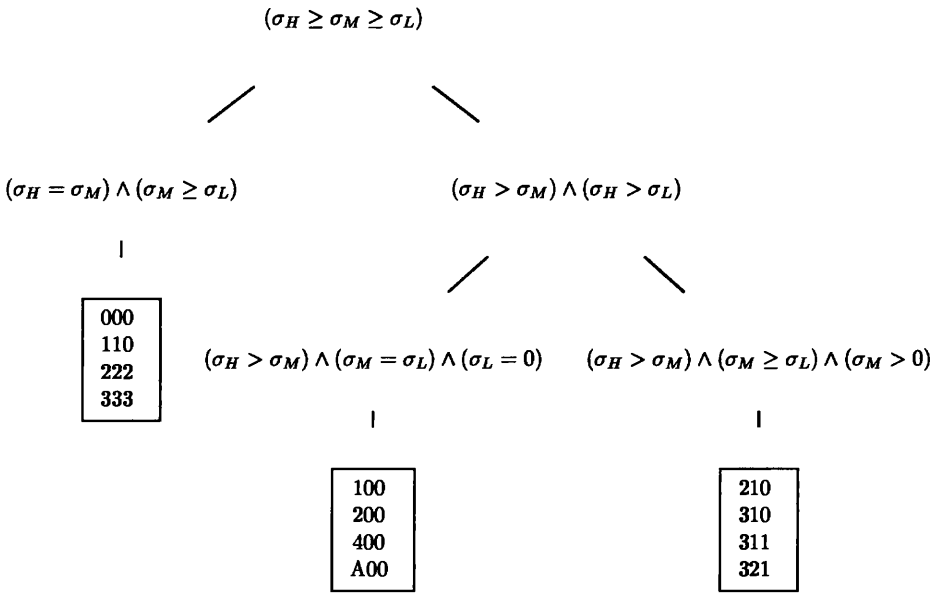


Fig. 5.2: Experimental Design Tree for G+98

sections, we discuss each of these scenarios in more detail.

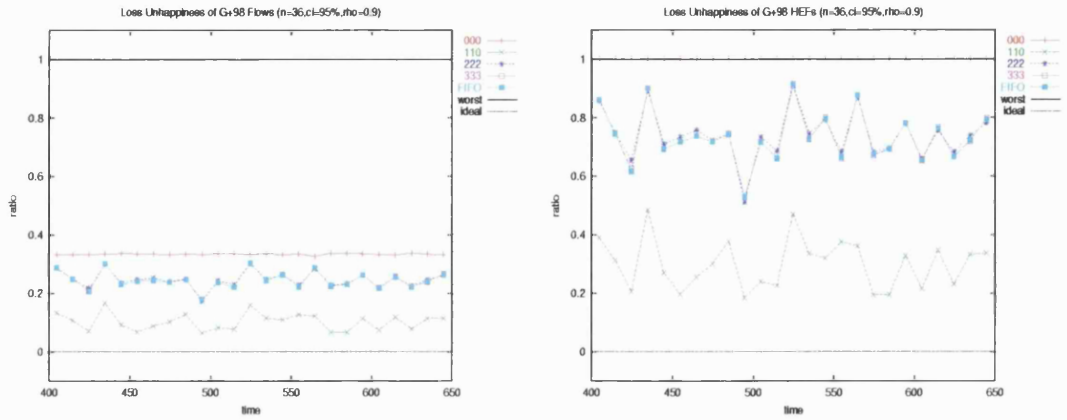
5.2.4 $(\sigma_H = \sigma_M) \wedge (\sigma_M \geq \sigma_L)$

In this category, we used the tuples values *000, 110, 222* and *333* for the G+98 scheme. For every tuple, we evaluated the performance of G+98, in terms of loss unhappiness, average utility and intra-class fairness. We used the performance of the FIFO buffer under similar conditions as a benchmark and we compared the plots of G+98 with the FIFO policy.

Loss Unhappiness

We show the loss unhappiness for the evaluated tuples for all flows in Figure 5.3(a) and for the high expectation flows (HEFS) only in Figure 5.3(b). From Figure 5.3(a), we see that the best performing G+98 configuration is the tuple *110* which has the lowest number of unhappy flows. The worst configuration is the tuple *000* which resulted in roughly 33% unhappy flows. The shape and level of unhappiness of the remaining two tuples, *222* and *333*, approximate the behaviour of the FIFO baseline.

Similar observations were drawn for the HEFS (Figure 5.3(b)), with *110* the best



(a) Overall (b) HEFS

Fig. 5.3: G+98 Loss Unhappiness : $(\sigma_H = \sigma_M) \wedge (\sigma_M \geq \sigma_L)$

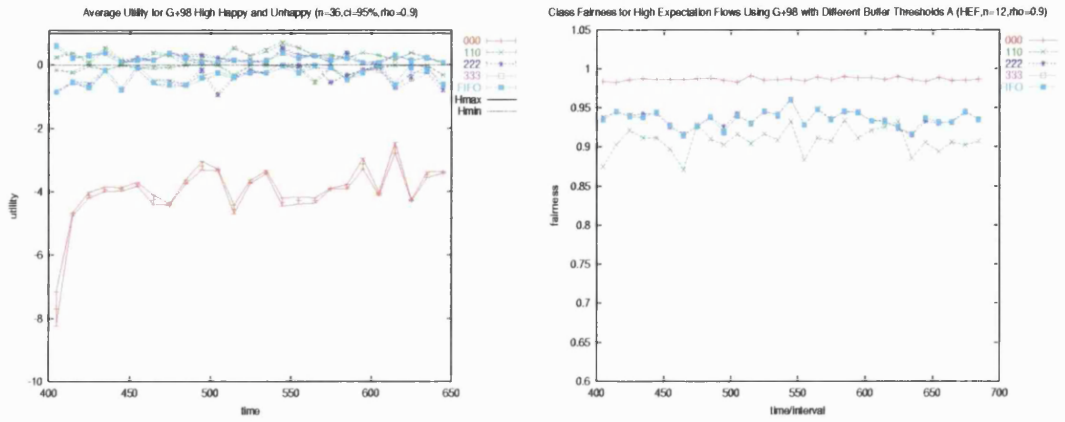
tuple configuration, *000* the worst, and *222* and *333* in-between. Additionally, all the unhappy flows in Figure 5.3(a) can be attributed to the HEFS because all the MEFS and LEFS are happy. Since the MEFS and LEFS are happy then we say that the G+98 configurations are inter-class fair.

Average Loss Utility and Intra-Class Fairness of the HEFS

Since the configurations are inter-class fair, we concentrate on the HEFS and examine their average utilities and intra-class fairness. We wish to highlight the difference between the G+98 configuration with the FIFO baseline for the HEFS.

We show the performance of the HEFS in terms of average utility in Figure 5.4(a) and intra-class fairness in Figure 5.4(b). From Figure 5.4(a), we see that the *110* tuple produced the best G+98 performance. Aside from having the most satisfied HEFS (see Figure 5.3(b)), the dissatisfied HEFS in the *110* configuration were the least unhappy. Similar to loss unhappiness, the worst average utility was from the *000* configuration. The other two configurations (*222* and *333*) closely approximated the behaviour of the FIFO baseline.

In terms of intra-class fairness for the HEFS, Figure 5.4(b) depicts opposite results. The *000* tuple was the fairest of the four configurations while *110* tuple was the least



(a) Average Utility (b) Intra-Class Fairness

Fig. 5.4: G+98 HEFS Performance : $(\sigma_H = \sigma_M) \wedge (\sigma_M \geq \sigma_L)$

equitable. These results are consistent with the earlier experiments on the FIFO server with different buffer sizes (Section 4.4.3). In those experiments, we discovered that the FIFO server with the least buffer space was the most fair. The allocation of $\sigma = 0$ for HEFS in the current G+98 scheme, is similar to configuring a smaller FIFO buffer. Both of these prevented unfair access. The tuple configurations *222* and *333* were more fair than the *110* tuple because they allow flows from different classes to compete equally for buffer space.

Remarks

Our findings indicate that for the G+98 scheme, if the σ allocations to all flows are equal to each other (e.g., *000*, *222* and *333*), then their loss unhappiness, average utilities and intra-class fairness would generally approximate but never surpass the performance of the FIFO baseline. This result is not surprising because the G+98 scheme is simply a FIFO server with large and equal σ allocations.

A further finding indicates that a $\sigma = 0$ allocation for both MEFS and LEFS was sufficient to satisfy these flows. Thus, more σ space can be allocated to HEFS to improve their utilities, as we shall see later in Section 5.2.5. In addition, we note that varying the σ allocations for all flows (e.g., the tuple *110*) could potentially produce

significantly better performance than the FIFO scheme. However, this improvement is limited only to loss unhappiness and average utilities.

5.2.5 $(\sigma_H > \sigma_M) \wedge (\sigma_M = \sigma_L) \wedge (\sigma_L = 0)$

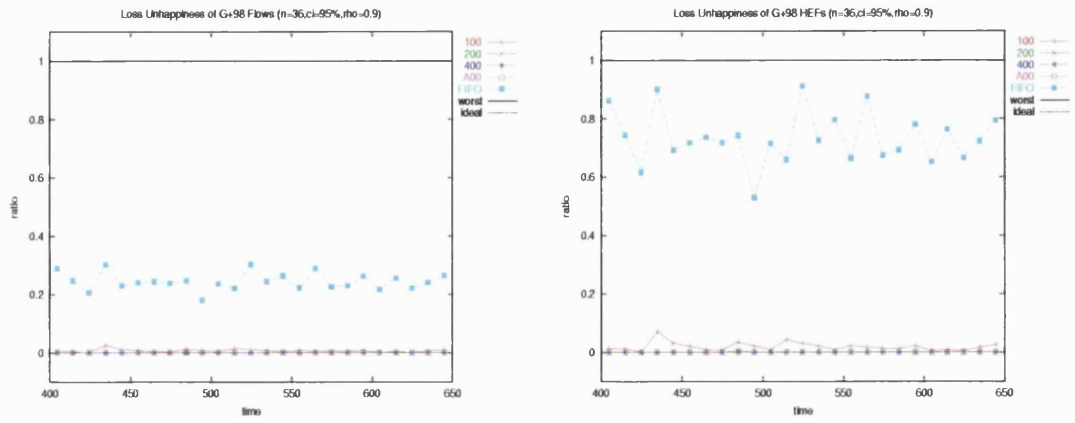
In this section we discuss the performance of the G+98 scheme when σ_H is greater than zero and σ_M and σ_L are both zero. In particular, we look at the behaviour of tuples *100*, *200*, *400* and *A00*. For these configurations, the MEFS and LEFS were satisfied. Hence, we only show the results for the HEFS in terms of loss unhappiness, average utility and intra-class fairness.

HEFS Performance

Figure 5.5(a) shows the overall loss unhappiness while Figure 5.5(b) singles out the HEFS loss unhappiness. In both figures, we see that the G+98 configurations performed significantly better than the FIFO baseline. In fact these settings were able to provide complete satisfaction to the HEFS in most cases (except for *100*). The results show that increasing σ_H while keeping both σ_M and σ_L zero improved the utilities of the HEFS. This adjustment eventually decreased the number of dissatisfied HEFS. Figure 5.6(a) which gives the average utility of the HEFS, shows that this trend appears to be bounded. The increase in average utility resulting from a change of tuple values from *100* to *200* was quite significant. However, this increase became smaller when we changed from *200* to *400*. Furthermore, no change was discernible as a result of the move from *400* to *A00*. The HEFS intra-class fairness shown in Figure 5.6(b), like the HEFS fairness in Section 5.2.4, also supports the notion that G+98 becomes less fair with increased buffer space. For both loss unhappiness and average utilities, the *A00* tuple is the best although it is one of the least intra-class fair.

5.2.6 $(\sigma_H > \sigma_M) \wedge (\sigma_M \geq \sigma_L) \wedge (\sigma_M > 0)$

Figure 5.7 shows the overall loss unhappiness for all the configurations in this category. Like the previous cases we have looked at, only the HEFS contributed to the overall loss unhappiness. In this figure, we can see that there are two groups of curves. The



(a) Overall (b) HEFS

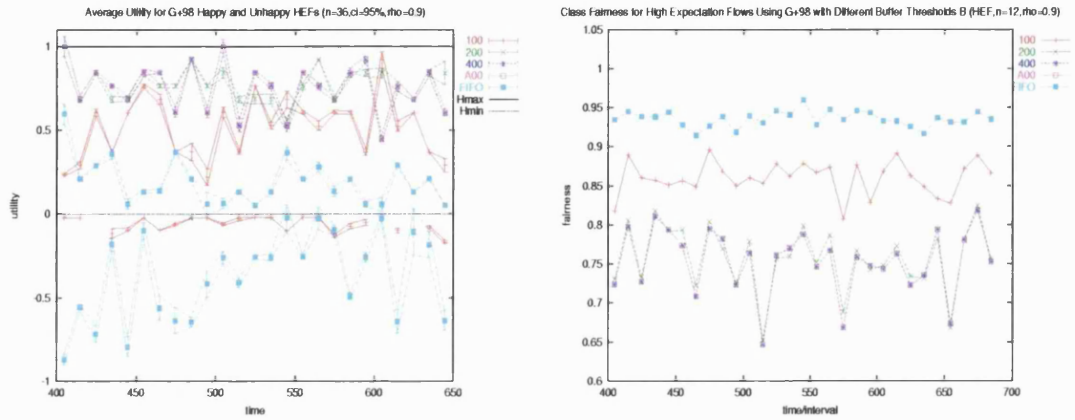
Fig. 5.5: G+98 Loss Unhappiness : $(\sigma_H > \sigma_M) \wedge (\sigma_M = \sigma_L) \wedge (\sigma_L = 0)$

first group consists of the FIFO baseline and tuples with non-zero σ_L. This suggests that once a certain level of σ assignments is reached, the behaviour of the G+98 mimics that of the FIFO. The second group consists of tuples with zero σ_L values. In this group, we find the two best configurations. Similar to our previous results in this chapter, we find that the best configurations in terms of loss unhappiness *310, 210* are the least intra-class fair.

5.2.7 Best Performing G+98 Configuration

In all the G+98 cases we examined, we found HEFS to be the most sensitive group. In Section 5.2.4, we saw two possible G+98 characteristics which could be used to address the sensitivity of the HEFS to yield optimum performance. First, a σ assignment that differentiates between flows could decrease loss unhappiness values. This was further supported by the results from Sections 5.2.5 and 5.2.6. However, the improvements were bounded and depended on the combination of the σ values.

The second G+98 behaviour of interest in Section 5.2.4 was that of setting both σ_M and σ_L to zero. We saw that this assignment was sufficient to keep the medium and low expectation flows satisfied. This implied that more σ space could be allocated to the HEFS to minimise their sensitivity. In Section 5.2.5, we found a class of settings



(a) Average Utility (b) Intra-Class Fairness

Fig. 5.6: G+98 HEFS Average Loss Utility and Intra-Class Fairness: $(\sigma_H > \sigma_M) \wedge (\sigma_M = \sigma_L) \wedge (\sigma_L = 0)$

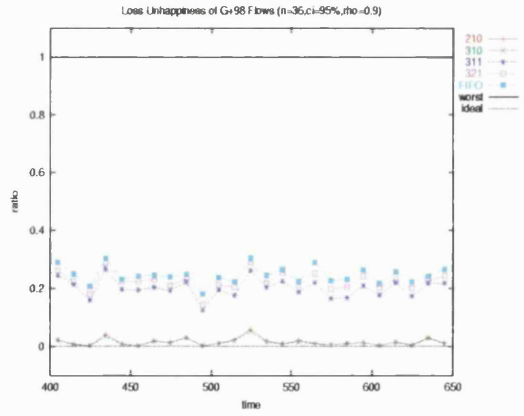


Fig. 5.7: G+98 Overall Loss Unhappiness : $(\sigma_H > \sigma_M) \wedge (\sigma_M \geq \sigma_L) \wedge (\sigma_M > 0)$

that yielded the best results in terms of happiness. Under this category, an increasing σ_H coupled with a setting of zero to σ_M and σ_L allowed, in almost all values of σ_H, to make all flows satisfied.

5.3 A FIFO Scheme Using Value-Based Utility

In this section, we present a FIFO scheme using Value-Based Utility (VBU) to manage loss inside a router.

5.3.1 Proposed Scheme

In this scheme, a flow[1] is assigned a utility threshold based on its sensitivity. Higher expectation flows were assigned larger utility thresholds than lower expectation flows. When utility congestion occurs, the router attempts to keep the flow's level of satisfaction below this threshold value. Utility congestion is the condition where some flows are satisfied while others are not. This scheme prevents utility congestion from deteriorating by dropping packets from flows who have exceeded their threshold. Usually the packets belonging to a flow with lower expectations are the first to be dropped because they are considered less sensitive and given lower thresholds. We hope that with this sacrifice, buffer space will become available for packets associated with unhappy flows when they are arrive at the router. Normally, when all flows are satisfied, this scheme does not drop packets. We note that like G+98, this scheme does not require per flow queuing and the number of operations is constant. Checking if all the flows are satisfied can be done in O(1) complexity. The algorithm for this scheme is given in Figure 5.8.

5.3.2 Performance of VBU Under Different Utility Thresholds

The performance of this scheme is similar to that of FIFO when the utility thresholds for all the flows are set to one. This is because a utility threshold of one means that no packet should ever be dropped, unless there is no physical space available. As with G+98, it is essential that the scheme performs better than the FIFO baseline. This section examines how much the performance can be improved when the thresholds are varied under the *bmix* base conditions in Table 4.2 using a router with 400 bytes worth of buffer and a service rate of 160,000 *Bps*. These parameters are similar with those used in the previous sections in this chapter.

We now present some results from our experiments on VBU. Figure 5.9 shows the results of a select group of thresholds in terms of HEFS loss unhappiness (Figure 5.9(a)) and average utilities for the three different flow groups (Figures 5.9(b),

[1]This may be extended to classes by considering a group of flows.

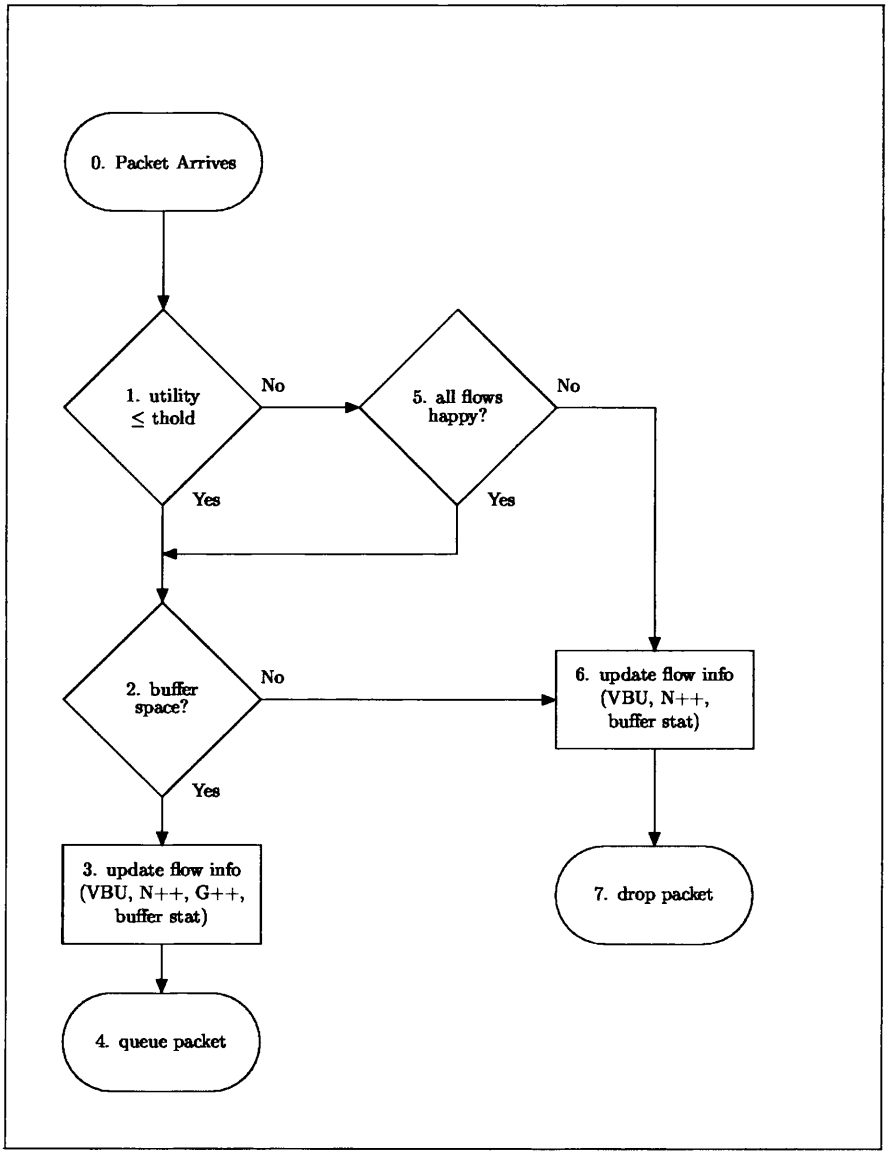
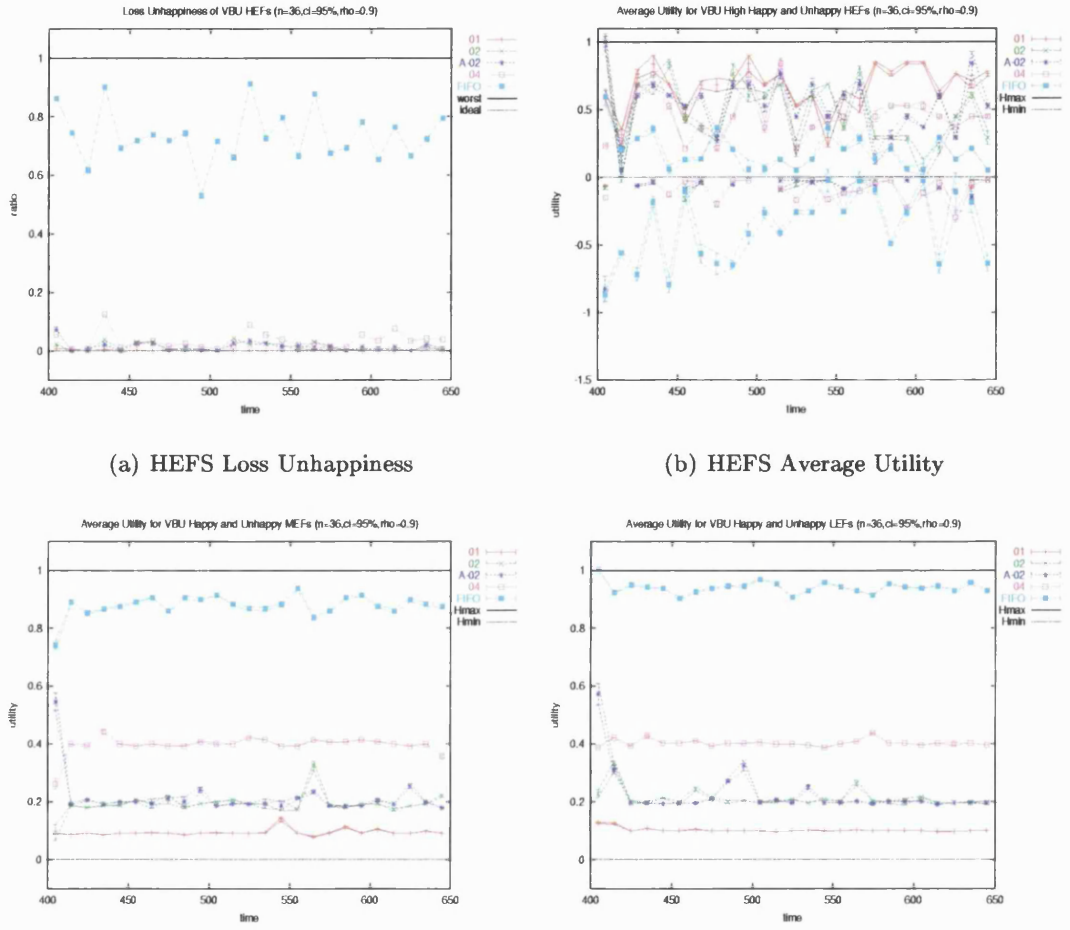


Fig. 5.8: VBU Packet Dropping Algorithm



(a) HEFS Loss Unhappiness

(b) HEFS Average Utility

(c) MEFS Average Utility

(d) LEFS Average Utility

Fig. 5.9: VBU HEFS Loss Unhappiness and Average Loss Utility

5.9(c) and 5.9(d)). The group shown in this section used only two thresholds, one for the high expectation flows and another for both the medium and low expectation flows. The HEFS are assigned a threshold of 1.0. The sensitivity of the HEFS is the reason that we were unable to use threshold values lower than 1.0. The MEFS and LEFS were either assigned *0.10, 0.20* or *0.40* utility thresholds.

In Figure 5.9(a), we see that the combination of protecting the HEFS and decreasing the thresholds associated with the MEFS and LEFS lowers the number of unhappy HEFS. In the case of a *0.10* threshold, all the HEFS were satisfied. The

same trends can also be seen in Figure 5.9(b) where the average utilities increased as the thresholds of the MEFS and LEFS were decreased. The MEFS and LEFS utilities were kept almost constant at their assigned thresholds as shown in Figures 5.9(c) and 5.9(d). The occasional values rising above their assigned threshold, for example the MEFS with *0.20* threshold at time 560 seconds, can be attributed to the scheme finding that all flows are satisfied. Under this condition, the scheme allows flows to go above their threshold levels.

In terms of fairness, we see that HEFS in the VBU scheme seem to behave like the FIFO scheme with large buffer sizes. This can be seen in Figure 5.10(a) where the fairness among the HEFS decreased as the thresholds of the other flows are decreased. However, if we look at the fairness of the MEFS (Figure 5.10(b)) and LEFS (Figure 5.10(c)), we find that they have almost identical fairness. The reason for this is that the utilities of these flows were always kept at or near their assigned thresholds which minimised the variation and increased fairness. If all flows were happy, it is possible for fairness to go down because these flows will not be limited to their thresholds.

The results in this section have clearly shown that keeping less sensitive flows at acceptable and satisfactory levels can benefit the more demanding users such as the HEFS. We have successfully shown that under this policy, the overall happiness can be increased by selectively dropping packets. This indicates that VBU is a viable option for managing loss because it directly exploits knowledge of flow utilities.

5.4 Summary

In this chapter, we have described two FIFO-based buffer management schemes, G+98 and VBU. We evaluated the performance of both schemes for normal traffic patterns for the three flow expectation groups: HEFS, MEFS and LEFS. Our common objective was to make all flows happy, or at the very least, to maximise the number of satisfied flows without sacrificing fairness. Hence, we evaluated both schemes in terms of: loss unhappiness, average utility, and inter- and intra-class fairness.

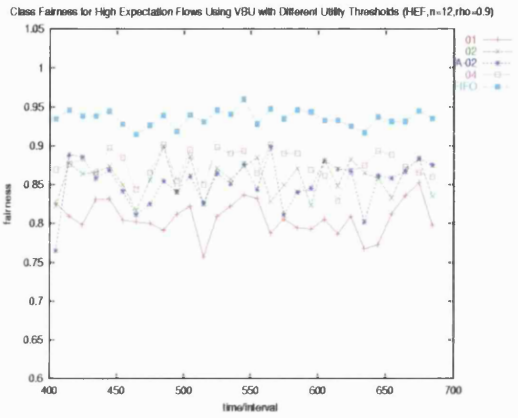
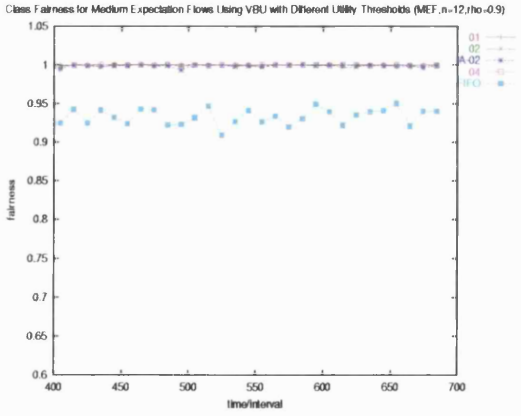
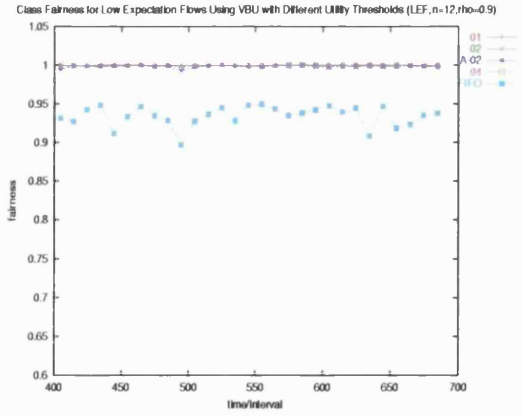


Fig. 5.10: VBU Class Fairness

(a) HEFS



(b) MEFS



(c) LEFS

For the G+98 scheme, we compared the performance of the three flow groups based on their utility measurements. We learned that it is possible to fulfil the requirements of all flows by adjusting the σ allocations to address the sensitivity of each flow group. This means that the demands of highly sensitive HEFS could be met without rendering the MEFS and LEFS unhappy. However, our use of utility for the G+98 scheme was limited to performance measurements and comparisons. It did not influence the initial σ allocations or when a packet must be dropped.

For the VBU scheme, we assigned utility threshold values to each flow group commensurate to each group's expectations. HEFS were assigned the highest possible threshold while we considered several threshold values for MEFS and LEFS. As with the G+98 scheme, we used the utility measurements to compare performance. However, we also relied on our measurements to decide when to drop packets during congestion. We managed to find a utility threshold value for MEFS and LEFS where all flows were satisfied.

The configurations for both schemes where all flows were satisfied also improved on the performance of the FIFO baseline. This indicates that, for normal traffic patterns, utility can be used for buffer management.

Chapter 6

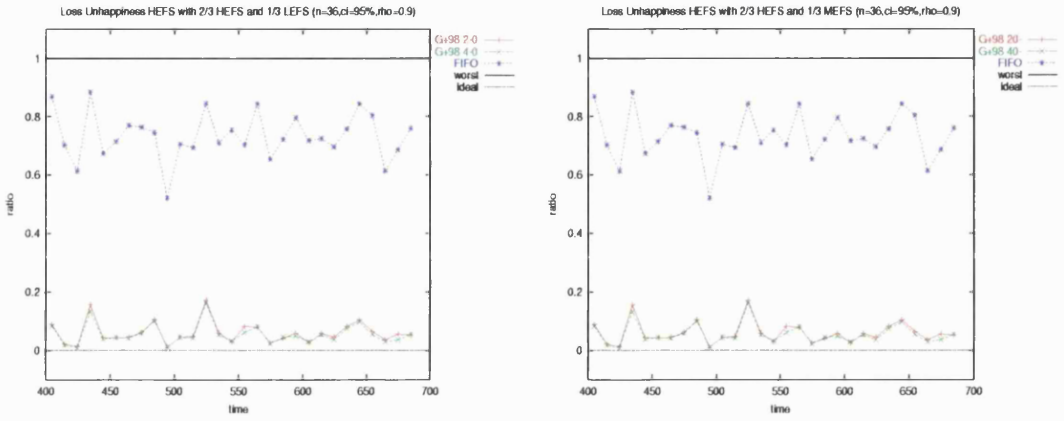
Loss Sensitivity

6.1 Overview

This chapter explores the robustness of G+98 and VBU under a range of conditions. We look at their performance when the expectation mixes are more demanding and the traffic characteristics more bursty. In addition to this, we investigate how utility can be used to enhance the performance of the G+98 scheme under these conditions. Three variants of the G+98 scheme are presented and analysed.

6.2 Expectation Mixes

In Chapter 5 we used the *bmix* loading conditions of Table 4.2 to observe and study the behaviour of both G+98 and VBU. In the *bmix* conditions, each of the 36 sources was of normal traffic type (Table 4.1) and evenly divided among the different expectation groups. In this section, we use the *emix-201* and *emix-210* conditions of Table 4.1; each represents different groupings of flows that impose a range of unique loading and allocation issues. The *emix-201* has a grouping of two-thirds high expectation and one-third low expectation flows while the *emix-210* has two-thirds high expectation and one-third medium expectation flows. Other factors such as source models, source topology and router parameters remain the same.



(a) two-thirds HEFS, one-third LEFS (b) two-thirds HEFS, one-third MEFS

Fig. 6.1: G+98 HEFS Loss Unhappiness for Two Different Expectation Mixes.

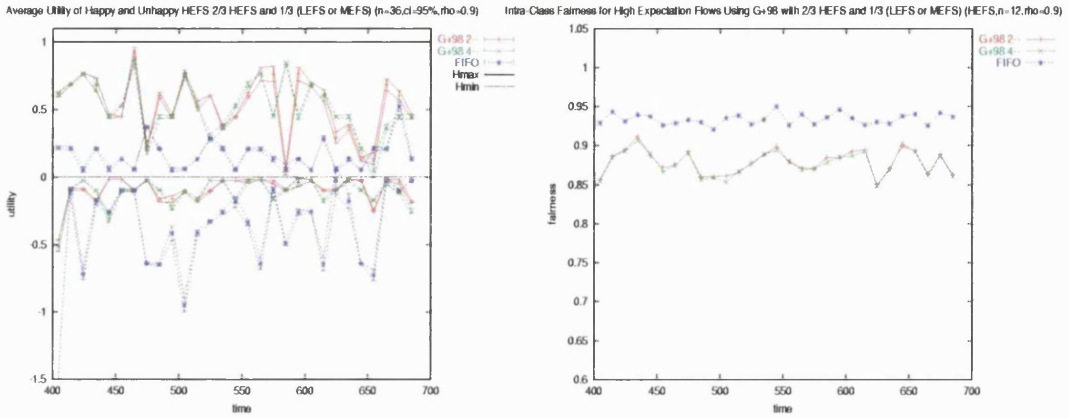
6.3 G+98 With Different Expectation Mixes

Earlier, we found that the set of G+98 σ tuples with a positive σ_H value and a zero setting for both σ_M and σ_L allowed for the most number of satisfied flows for typical (*bmix*) conditions. From this set, we now take two G+98 tuples and evaluate their performance when the distribution and groupings of expectation mixes are unequal and varied. We use G+98 tuples whose σ_H values are either 4 or 2.

6.3.1 Loss Unhappiness

The results[1] for the HEFS in terms of loss unhappiness for *emix-201* scenario are shown in Figure 6.1(a) while the results for *emix-210* case are given in Figure 6.1(b). The first thing we notice in both figures is that the G+98 tuples were not able to satisfy all flows. However, the level of satisfaction is significantly better than the FIFO baseline. In addition to this, two other important points can be made. The first is that an increase in the σ_H allocation, such as from 2 to 4, does not always decrease the number of unhappy users. In fact the satisfaction levels are almost identical (see Figures 6.1(a) and 6.1(b)) suggesting the values of σ_H (4 and 2) are near optimal.

[1] Similar to the previous chapter, both the MEFS and LEFS are all happy and hence our discussion for this section revolves primarily around the HEFS.



(a) Average Utility (b) Intra-Class Fairness

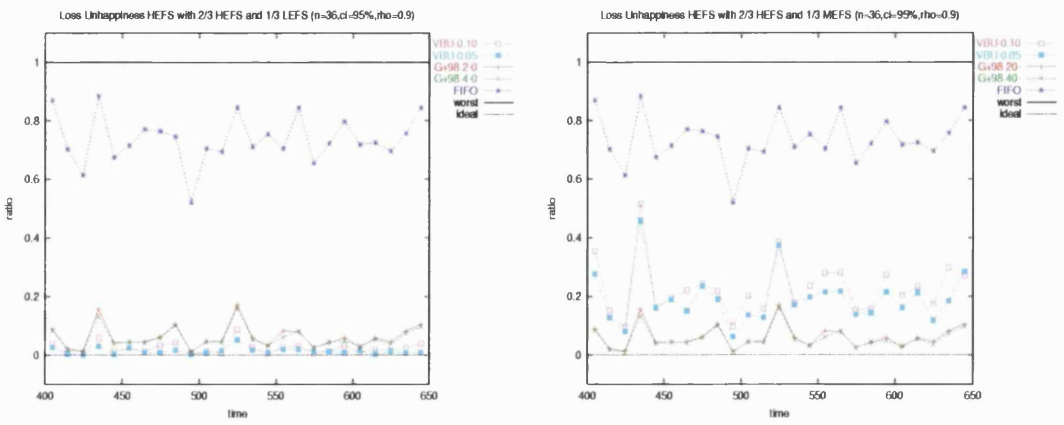
Fig. 6.2: G+98 HEFS Average Utilities and Intra-Class Fairness for Two Different Expectation Mixes.

The second point is that the loss unhappiness of the HEFS is unaffected by whether MEFS or LEFS are grouped with HEFS. This is true as long as the proportion of flows is kept the same (two-thirds HEFS to one-third MEFS or LEFS). This result suggests that G+98 has the ability to separate flows based on their expectation groupings and treats them accordingly. The implication is that the dissatisfaction of the HEFS is partly attributed to the competition for buffer space among themselves.

6.3.2 Average Utilities and Intra-Class Fairness

Figures 6.2(a) and 6.2(b) display the average utility and intra-class fairness of the G+98 HEFS applicable to both *emix* scenarios. Unlike the case of loss unhappiness, the effects of the σ_H allocation are recognisable in Figure 6.2(a). It is not clear, however, which one of the σ_H allocations is better. In some cases, one could have a higher average utility for the satisfied HEFS while at the same time have a lower average utility for the dissatisfied HEFS. Examples of these situations can be seen during the periods between 570-590 seconds for σ_H equal to 4 and 510-530 seconds for σ_H equal to 2.

Consider the interval 570-590 seconds. Since both configurations of G+98 have very similar loss unhappiness, we can assume the happiness of some of the satisfied



(a) two-thirds HEFS, one-third LEFS (b) two-thirds HEFS, one-third MEFS

Fig. 6.3: VBU HEFS Loss Unhappiness for Two Different Expectation Mixes.

flows with σ_H equal to 4 has increased while it has decreased for the dissatisfied flows. It is even possible that some of the happier flows became less happy. This means that, during these intervals, there are more variations in individual utilities for this configuration than when σ_H was equal to 2. These variations should lead to more unfairness. This is confirmed by the results of the HEFS intra-class fairness shown in Figure 6.2(b). Although the differences are small, the σ_H allocation of 2 is more fair during these times than the allocation of 4.

6.4 VBU with Different Expectation Mixes

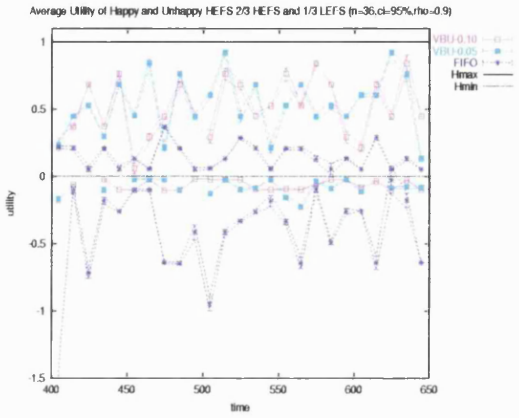
In Section 5.3 we investigated the performance of VBU when used in a scenario similar to *bmix* conditions and found that the overall happiness can be increased by allocating suitable utility thresholds for less demanding flows while protecting the more sensitive flows. We now evaluate the performance of this scheme in a situation where the proportions of expectation mixes are unequal and skewed towards more demanding flows.

6.4.1 Loss Unhappiness

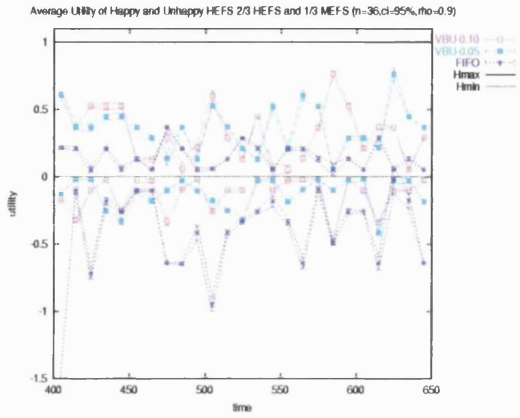
Figure 6.3 shows the HEFS loss unhappiness of two VBU configurations with utility threshold levels 0.10 and 0.05 which we call *VBU-0.10* and *VBU-0.05* respectively. Although the VBU schemes are not able to make all flows happy for both the *emix-201* (Figure 6.3(a)) and *emix-210* (Figure 6.3(b)) scenarios, we observe that by lowering the utility threshold from 0.10 to 0.05 the number of unhappy users is decreased. This result is consistent with what we found in Section 5.3.2. As expected, there are more dissatisfied flows when the *emix-210* flows are used than with *emix-201*. This is because the MEFS required more packets to go through to maintain the utility threshold of 0.10 (or 0.05) as compared to the LEFS. The resulting increase in the demand for buffer space has in turn affected the HEFS. In contrast, the HEFS of G+98 maintained the level of loss unhappiness regardless of the scenario used. The G+98 scheme is able to achieve this consistency by employing rate guarantees. This ensures that flows, especially the HEFS, have access to their buffer space when they are within their threshold. In the VBU scheme, this feature is deployed for the MEFS and LEFS while the HEFS are allowed to get as much space as they can get.

6.4.2 Average Utility and Intra-Class Fairness

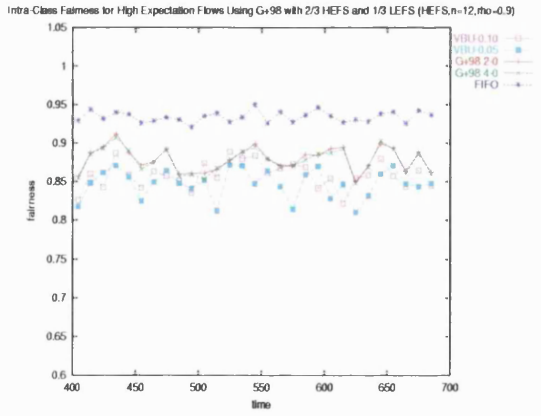
Unlike the HEFS average utilities for the G+98 configurations, the shape and behaviour of the 0.10 and 0.05 utility thresholds are quite dissimilar. If we look at the upper half of Figures 6.4(a) and 6.4(b) where the average utilities of the satisfied HEFS are displayed, it appears that neither VBU-0.05 nor VBU-0.10 can be considered better. However, as noted in Section 4.4, the average utility plots should be used in conjunction with other evaluation indices to get a clearer picture. If we consider the times when the satisfied average utilities of VBU-0.10 are higher than those for VBU-0.05, we will find that the loss unhappiness for VBU-0.10 is generally higher (which means more flows are dissatisfied) than for VBU-0.05. Examples of this situation can be seen in Figure 6.4(a) at times 425, 445, 475, 525, 545, 555 and 585 seconds. At these times, a few HEFS in VBU-0.10 managed to get very happy which resulted



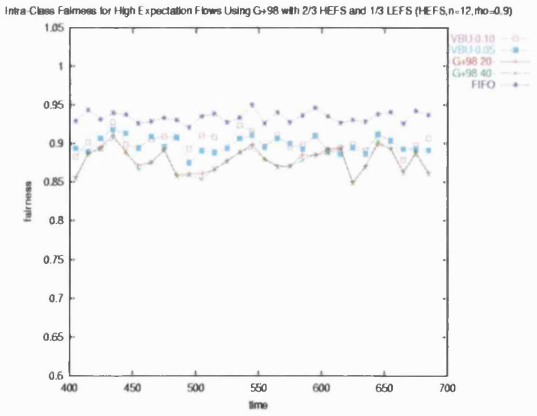
(a) Average Utility: two-thirds HEFS, one-third LEFS



(b) Average Utility : two-thirds HEFS, one-third MEFS



(c) Intra-Class Fairness : two-thirds HEFS, one-third LEFS



(d) Intra-Class Fairness : two-thirds HEFS, one-third MEFS

Fig. 6.4: VBU HEFS Average Utilities and Intra-Class Fairness for Two Different Expectation Mixes.

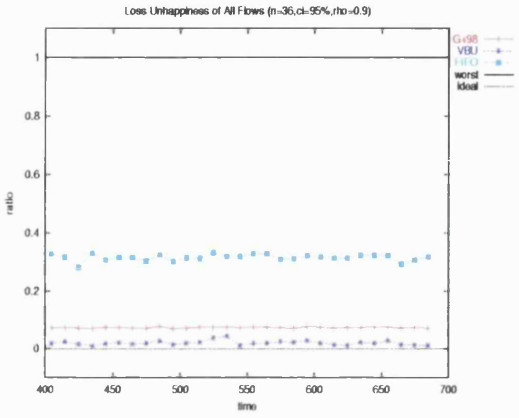
in the mean utility of the satisfied flows to go up. However, this did not appear to affect the overall fairness as the VBU-0.10 was generally more fair than VBU-0.05 as Figures 6.4(c) and 6.4(d) indicate. Moreover, since the HEFS average utilities for the satisfied and dissatisfied flows in the *emix-210* scenario for both VBU configurations are closer together than their counterparts in the *emix-201* scenario, the intra-class fairness is higher. This also supports the idea that with increased demands – which lessens the chances of a few flows from getting extreme utility values – fairness is improved.

6.5 Effects of Bursty Sources

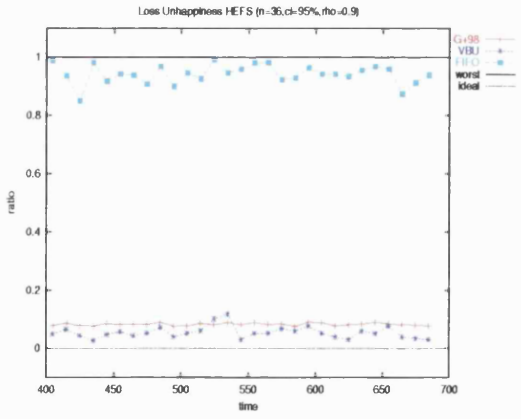
In this section, we analyse the loss behaviour of G+98 and VBU under more bursty conditions. Three scenarios ranging from a small number of bursty sources to a third of the total number of flows being bursty are investigated. Each bursty source is modelled as a medium type of source whose parameters are given in Table 5.1. In all three scenarios, the number of HEFS, MEFS and LEFS are equal.

6.5.1 The Performance of G+98 and VBU

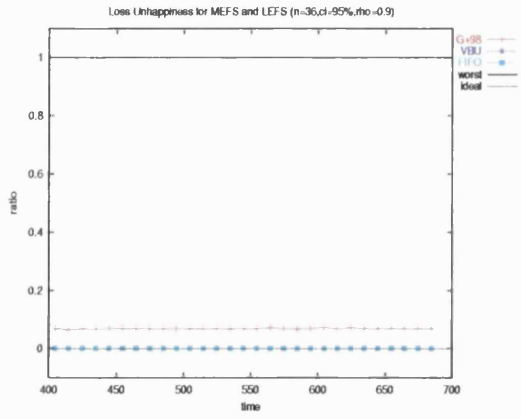
In Figure 6.5, we show the loss unhappiness behaviour of G+98 and VBU schemes when three bursty sources, one for each flow group, are mixed with normal sources. The G+98 and VBU configurations used are the ones that were able to keep all flows satisfied under the *bmix* conditions. The overall loss unhappiness (Figure 6.5(a)) shows a significant improvement by both G+98 and VBU over the FIFO scheme, with the VBU having fewer dissatisfied flows than G+98. The improvement of G+98 and VBU over the FIFO scheme was mainly achieved through the management of high expectation flows (Figure 6.5(b)). In the FIFO scheme, almost all the HEFS were dissatisfied. Only the bursty source from the HEFS in G+98 was dissatisfied. The G+98 scheme specifically controlled the HEFS bursty source because it has constantly exceeded its B_i buffer allocation. This action prevented the buffer pool from being exhausted, protecting flows operating within their characterisation. However, the



(a) All



(b) HEFS



(c) MEFS and LEFS

Fig. 6.5: Loss Unhappiness of the G+98 and VBU Schemes. The traffic characteristic and expectation mix used in these results is the *tmix-B10*.

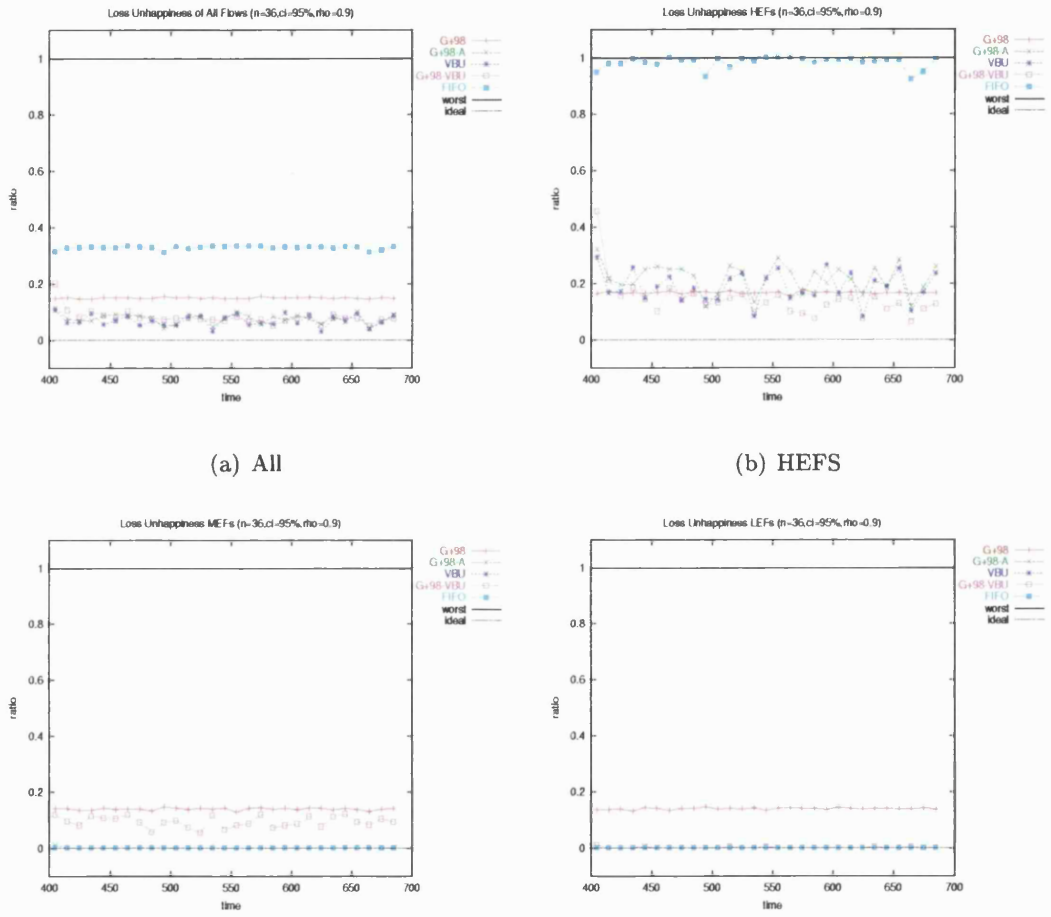
control as shown in Figure 6.5(c)[2], was also used on the bursty sources of the MEFS and LEFS. The result of this is action is undesirable because, for this occasion the G+98 configuration is inter-class unfair. For this measure, even the FIFO scheme performs better than the G+98 scheme. Under the VBU scheme, the HEFS are allowed access to the buffer pool as long as the MEFS and LEFS are all satisfied. This policy, along with threshold setting of 0.10, proved successful in satisfying most of the flows.

6.5.2 G+98 Scheme with VBU Adjustments

The performance of the G+98 scheme in Section 6.5.1 can be improved by adjusting the σ's to account for the burstiness of the flows. An example of this improvement, with even more bursty sources, can be seen in the loss unhappiness of *G+98-A* in Figure 6.6. The G+98-A scheme shown in this figure is a G+98 scheme with the σ values for bursty sources set at higher values (e.g., $\sigma_H\sigma_M\sigma_L$= 531 as compared to 200). In Figure 6.6(a), where the overall loss unhappiness is shown, the G+98-A configuration has improved on the performance of the G+98 configuration. In fact, it even approximates the levels of the VBU scheme. Although the HEFS loss unhappiness level (Figure 6.6(b)) of the G+98-A scheme is generally poorer than that of the G+98 scheme, it is still the better choice because it is able to keep the MEFS and LEFS satisfied (Figures 6.6(c) and 6.6(d)).

Also shown in Figure 6.6 is the loss unhappiness of the *G+98-VBU* scheme which is a combination of G+98 with VBU. Rather than preallocating a suitable σ value for each flow and keeping it static, the G+98-VBU scheme attempts to match the σ values to the conditions seen by the router (Figure 6.7). The primary operation of the G+98-VBU scheme still remains the same as G+98. The only difference is that the σ values can change during execution. Initially, each flow's σ is initially set to zero (Figure 6.7(a)). The σ of a flow can only be incremented if its utility is below $happiness_{min}$ levels and the number of satisfied flows are above a predefined threshold

[2]The loss unhappiness for MEFS and LEFS are similar so we just present the result of one.



(a) All

(b) HEFS

(c) MEFS

(d) LEFS

Fig. 6.6: Loss Unhappiness of G+98-VBU Scheme in Relation to the Basic G+98 and VBU Schemes. The traffic characteristic and expectation mix used in these results is the *tmix-A20*.

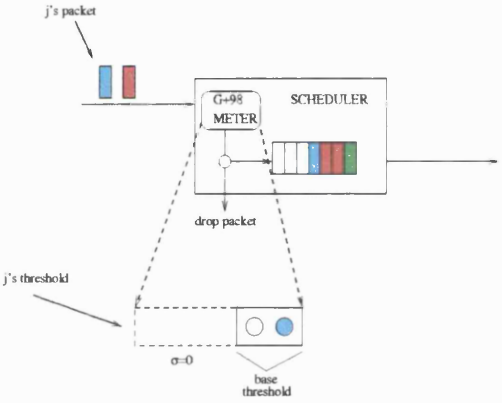
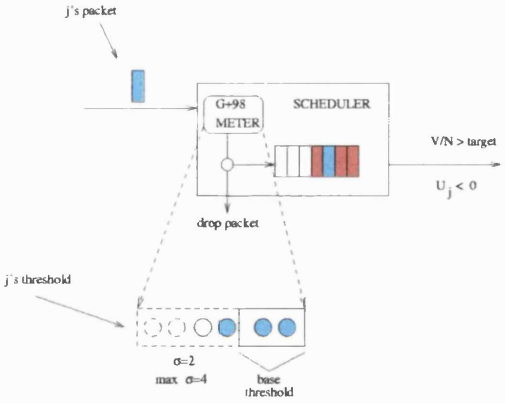
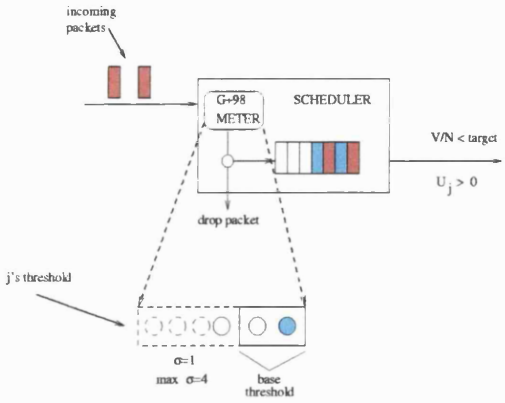


Fig. 6.7: G+98-VBU Scheme

(a) The σ values of all flows are initially set to 0.



(b) The σ can only be increased when flow is dissatisfied and the total number of satisfied flows are above a defined target.



(c) The σ can be decreased if the flow is satisfied and the number of satisfied flows are below a defined target.

(Figure 6.7(b)).

The first condition restricts the increase to those who are dissatisfied. The second condition ensures that the σ's do not increase at the same time. If the σ values are allowed to increase simultaneously, the G+98-VBU will simply mimic a FIFO router. A limit on how big σ can grow is also required. This is to ensure that no one flow grabs a large chunk of the buffer space.

In addition, there is a provision to decrease the σ value of a flow. This occurs when the flow is satisfied and the number of satisfied flows is below a predefined threshold (Figure 6.7(c)). The logic behind this action is that the flow is already satisfied and thus can afford to lose some packets. This will then improve the buffer availability for other flows.

The performance of the G+98-VBU scheme in terms of the overall loss unhappiness is as good as that of G+98-A and VBU (Figure 6.6(a)). The HEFS under this scheme were even better than G+98 scheme (Figure 6.6(b)). However, they failed the inter-class fairness criterion because some of the MEFS were dissatisfied. Since the HEFS are more sensitive, they are likely to be the first ones to get their σ's adjusted. This sensitivity of the HEFS may have prevented the G+98-VBU scheme from finding suitable σ values for these MEFS. This may be because the predefined threshold for the number of satisfied users may already have been exceeded by the time the MEFS or other lower expectation flows become dissatisfied.

6.5.3 G+98 Sharing Scheme and G+98-VBU Sharing Scheme

A variation of the G+98 scheme that allows for sharing of unused buffers even when flows are above their B_i threshold was presented in [37]. We shall call this algorithm the *G+98-Sharing* scheme. In this approach, buffer space is first partitioned into two parts, the *headroom* and the *holes*. Access to the headroom partition is reserved for flows that have not exceeded their thresholds while the holes partition can be used by anyone. The operation of this algorithm is presented in Figure 6.8. Packets belonging to flows that are within their thresholds are immediately given access to the holes as

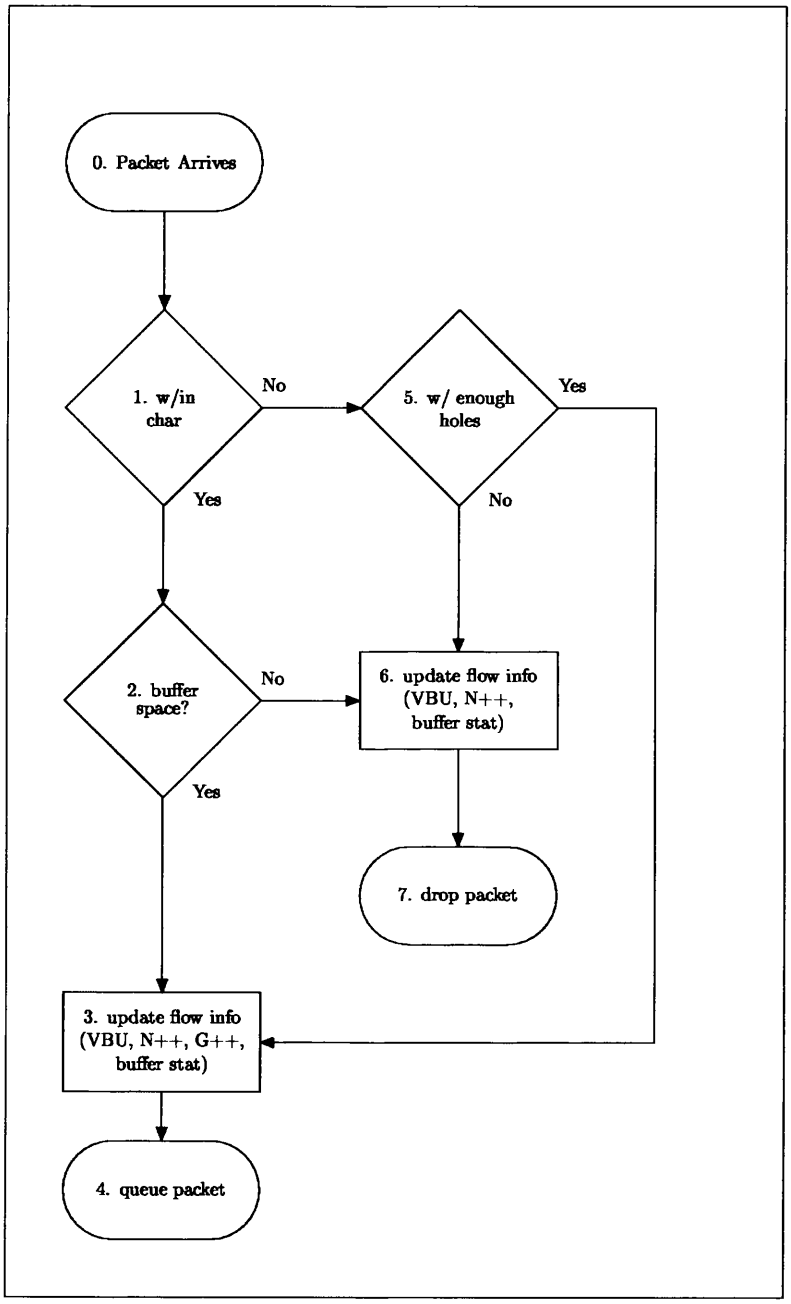
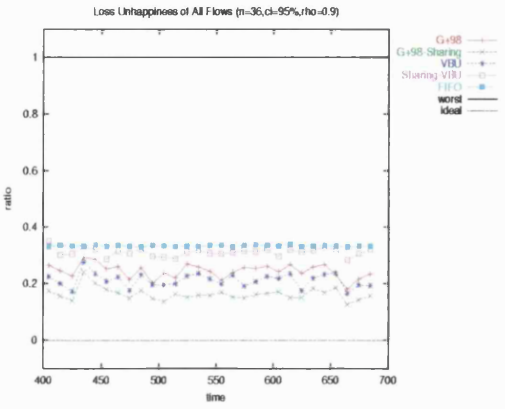
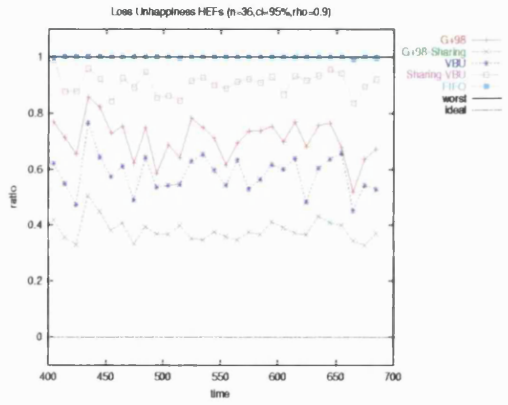


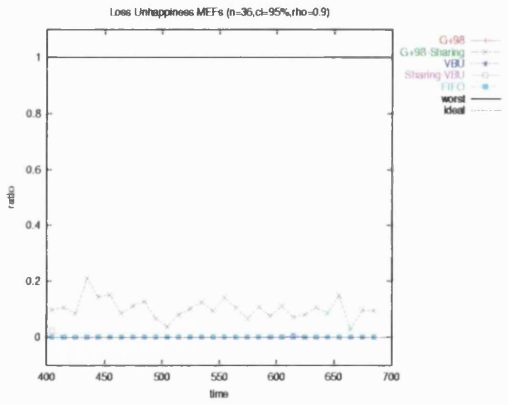
Fig. 6.8: G+98-Sharing Packet Dropping Algorithm.



(a) All



(b) HEFS



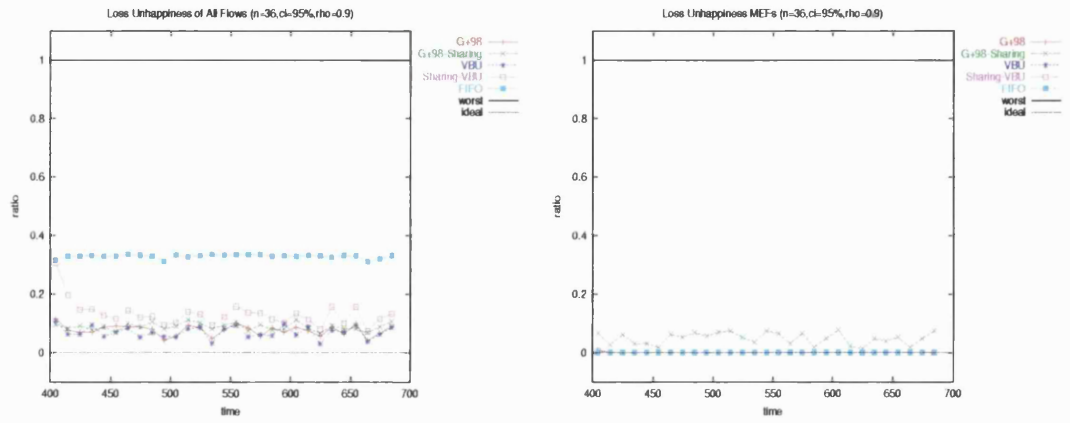
(c) MEFS

Fig. 6.9: Loss Unhappiness of G+98-Sharing and G+98-VBU Sharing Schemes in Relation to the Basic G+98 and VBU Schemes. The traffic characteristic and expectation mix used in these results is the *tmix-840*.

long as there is space. If there is no space, an attempt is made to access the headroom. Flows that are above their threshold can use buffer space from the holes provided the extra space[3] they will receive above their threshold is not more than what will be left in the holes. Otherwise their packets are dropped. Freed buffers are always returned first to the headroom up to the total headroom allocation. Only when the maximum space for the headroom is regained, can space be allocated to the holes.

The loss unhappiness of this scheme for the *tmix-840* scenario is compared with other schemes in Figure 6.9. Of all the schemes shown, the G+98-Sharing has the

[3]This extra space also includes the buffers exceeding the threshold a flow is currently using.



(a) All (b) MEFS

Fig. 6.10: Loss Unhappiness of G+98-Sharing and G+98-VBU Sharing Schemes in Relation to the Basic G+98 and VBU Schemes. The traffic characteristic and expectation mix used in these results is the *tmix-A20*.

fewest number of dissatisfied flows (Figure 6.9(a)). The improvement over the G+98 has mainly been achieved because the scheme allows fewer HEFS to become less dissatisfied (Figure 6.9(b)). However, the improvement is at the expense of some MEFS (Figure 6.9(c)). In terms of inter-class fairness, the result is not acceptable. This unfairness can be removed by adjusting the σ values during operation which is what *Sharing-VBU* does. The enhancement done by the Sharing-VBU scheme is similar to the adjustments made by G+98-VBU to the basic G+98 scheme. As Figure 6.9(c) shows, the MEFS are all satisfied. However, the performance of Sharing-VBU in terms of overall loss unhappiness is similar to that of the FIFO scheme. For the *tmix-840* scenario, it appears that Sharing-VBU is unable to take advantage of the utility information and make the flows satisfied. On the other hand at less bursty conditions such as *tmix-A20*, we find that the scheme nearly approximates the performance of G+98-Sharing in terms of overall loss unhappiness (Figure 6.10(a)). In addition, the scheme minimises the inter-class fairness of G+98-Sharing (Figure 6.10(b)).

6.6 Summary

In this chapter as well as in Chapter 5, we investigated how utility can be used to manage loss under different conditions. We considered scenarios where expectation mixes are more demanding and traffic characteristics more bursty. We evaluated the performance of the VBU scheme, the G+98 scheme and G+98 variants under this range of difficult situations. We were able to show different levels of success in using utility to satisfy flows in terms of their expectations. The VBU scheme seems to be more adept in meeting expectations across a range of conditions. This result is not surprising as VBU was designed to manage flows based on expectations. In the case of the G+98 scheme, we were able to show that reorienting the purpose of the burstiness factor σ to model expectations has potential. In the first variant, G+98-VBU, this factor was dynamically adjusted to find the appropriate value for conditions seen inside the router. In terms of overall loss unhappiness, the scheme proved to be generally outstanding. However, it failed in one of the most important criteria which was inter-class fairness. The second variant evaluated was the G+98-Sharing [37]. It performed well in terms of overall loss unhappiness especially during bursty conditions, but it also was inter-class unfair. The third variant, Sharing-VBU, addressed this problem of inter-class unfairness. It even managed to approximate the overall loss unhappiness of the G+98-Sharing scheme. However, this result was only limited to traffic conditions ranging from low to medium burstiness.

Although we believe these results are promising, we feel that more research is needed to improve on the use utility in managing flows and buffer usage. For example, we have not really considered issues of stability and time scales for management. We have only used simple methods for management. This was intentional because we wanted to elucidate the potential of utility.

Chapter 7

Delay Management Schemes

7.1 Overview

In this chapter we present a scheduler that uses VBU to manage delay. We compare its performance with several non-VBU aware schemes in terms of delay unhappiness and inter-class fairness under different delay scenarios and burst conditions.

We begin by establishing baseline numbers for a FIFO server. The performance of a per-flow based scheme, *Deficit Round-Robin (DRR)* [70], and a class-based priority scheme, *Three Class Priority Queue (3CPQ)*, are then evaluated against the FIFO reference values. Subsequently, two schemes incorporating features from both DRR and 3CPQ are described and their performance evaluated. The first of these schemes, *Three Class Priority Scheme with Per-Flow DRR (3CPQ-DRR)*, involves running a DRR service that treats each flow individually within a class. The second variant, *Three Queue Value-Based Utility with Class DRR (3QV-DRR/3QV-DRR-R)*, runs a VBU-aware DRR service where flows are aggregated into a class.

7.2 Delay Scenarios

The delay inside the network can be viewed as having a fixed and a variable component. The fixed part is due to physical transmission constraints and limitations. The variable component, on the other hand, can be caused by queuing delays and packet retransmission in the event of packet loss. In this chapter, we investigate three delay

scenarios involving the variable component of the network delay. We classify these scenarios as follows:

Sequential (*S-type*): This scenario is based on selective retransmission. Packets must arrive in the order they were transmitted, and the measured delay will include a contribution from out of sequence packets waiting for missing packets to arrive. Packet loss will consequently have a severe impact on the utility of a flow.

Retransmission (*R-type*): Although the throughput is maintained, there is no distinction between the original and retransmitted packets. The delay associated with any packet is simply the time difference between transmission and arrival. Utilities will be affected because retransmission contributes to the traffic load.

Loss (*L-type*): Lost packets are never retransmitted and throughput is thus not maintained. The utility characteristics seen from this case are expected to be lower than that of the *S-* and *R-type* schemes when packet loss is substantial.

7.3 Burst Conditions and Other Parameters

In this section, we briefly give an overview of the experimental conditions and parameters used in this chapter.

7.3.1 Burst Conditions

In the experiments discussed in previous chapters, we used insufficient buffer space to force packet loss. To create a suitable environment to investigate delay management, we increase the buffer pool at the router by a factor of 50 (to 500 packets) compared to the experiments on loss. We also use traffic mixes with very bursty sources to increase buffer occupancy and force queue overflow and packet retransmission. To avoid excessive packet loss, the load on the router is reduced to 80%. This is accomplished by increasing the speed of the router's output link from 160,000 *Bps* to 180,000 *Bps*.

The interarrival times from sources are based on the Weibull distributions listed in Table 4.1 and the traffic mixes (*tmix*) are those listed in Table 4.2.

7.3.2 Expectation Mixes

We used the same flow classification defined for the experiments in loss utilities: HEFS, MEFS, and LEFS. The 99%, 90% and 80% expectation levels associated with each group are retained, but instead of monitoring loss, we measure the packet delay and compare it to a target delay bound. Packets arriving before this delay bound are considered successful and will therefore increase utility. Experiments in this chapter, adopt a target delay bound of 9 *ms* for each flow for schemes that manage on a per-flow basis. Schemes that operate on aggregated flows use the same delay bound per class. The value was chosen from a range of values because it provides a suitable baseline for the FIFO scheme.

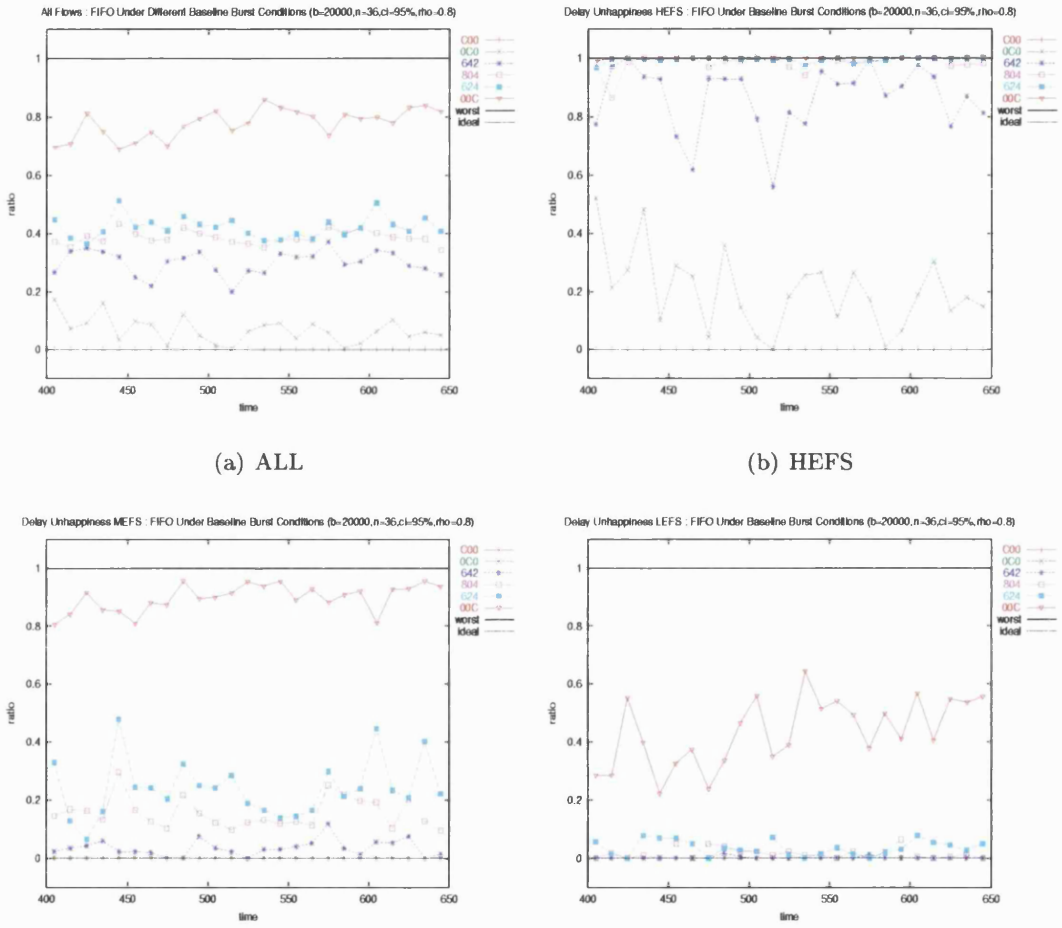
In some instances, we need to make sure any packet loss is detected. In these experiments, we set the loss expectation to 90% for all flows regardless of their delay expectation classification. A measured average loss utility of less than 1.0 for a flow would indicate that packets have been dropped.

7.4 Performance of the FIFO Scheme

In this section, we examine the FIFO scheme's performance for three traffic loads representing baseline burst conditions. The results of these baseline experiments will be used as a comparison point for subsequent experiments.

7.4.1 Burst Conditions

Figure 7.1 displays the delay unhappiness of the FIFO scheme under different burst conditions for the *S-type* scenario. We first consider the results of *tmix-C00, tmix-0C0* and *tmix-00C*, three simple, homogenous traffic loads with increased burstiness. These experiments represent baseline conditions for the FIFO scheme and will serve as references against which other experiments will be compared. The *tmix-C00* line as shown in Figure 7.1(a) tells us that when all flows have low burstiness, the FIFO scheme is able to achieve an ideal performance where all flows are satisfied. The second reference curve represents the worst possible situation. This occurs when all the flows



(a) ALL

(b) HEFS

(c) MEFS

(d) LEFS

Fig. 7.1: FIFO Delay Unhappiness Under Baseline Burst Conditions

are very bursty (*tmix-00C*). In this experiment, at least 70% of the total flows are dissatisfied in any given point in time. The dissatisfied flow population is composed of all the HEFS (Figure 7.1(b)), at least 80% of the MEFS (Figure 7.1(c)) and at most 60% of the LEFS (Figure 7.1(d)). The last baseline result is the performance of the FIFO scheme when all the flows use the medium burstiness *tmix-0C0* sources. Under these load conditions, the FIFO is able to keep the MEFS (Figure 7.1(c)) and LEFS (Figure 7.1(d)) happy. However, it is unable to satisfy all the HEFS (Figure 7.1(b)). In all baseline curves, we see that the FIFO scheme is inter-class fair. Since all flows in the *tmix-C00* experiments are happy, the FIFO is, by default, inter-class fair. In the case of the *tmix-0C0* and *tmix-00C*, the FIFO is also inter-class fair because the fraction of dissatisfied lower expectation flows is always less than the fraction of flows that belong to the expectation level above. This follows from the definition of inter-class fairness in Section 4.4.3.

Also shown in Figure 7.1 are the delay unhappiness values of traffic mixes that combine low, medium and high bursty sources. Three traffic mixes *tmix-642, tmix-804* and *tmix-624*, have been chosen because their performance was expected to be between *tmix-00C* and *tmix-0C0*. The results from these and the baseline figures suggest that the delay increases as the traffic becomes increasingly bursty. This is consistent with the known effects of bursts on a FIFO queue [85].

7.4.2 Delay Scenarios

Figure 7.2 shows the overall delay unhappiness of the FIFO scheme for the three delay scenarios when *tmix-624* is used. Notice that, in all cases the unhappiness levels are identical. This suggests that no loss occurred and the delay experienced by the packets is therefore based on queuing delays. Had there been any loss, the average loss utility of the *L-type* scenario[1] would have at some point dropped below 1.0. This is clearly not the case as the average loss utility of the *L-type* scenario - labelled *loss* - is 1.0 throughout the observation interval.

[1]This is the only scenario that loss can be seen, since in the other two cases lost packets are retransmitted.

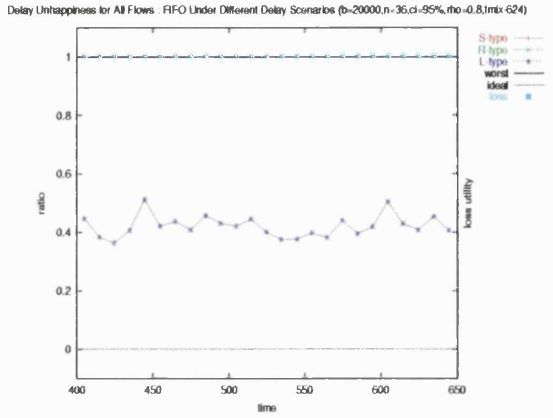


Fig. 7.2: FIFO Delay Unhappiness Under Different Delay Scenarios

7.5 Deficit Round Robin and Three Class Priority Queue

In this section we describe Deficit-Round Robin (DRR) [70] and the Three Class Priority Queue (3CPQ) schemes. We evaluate their performance in terms of delay unhappiness and inter-class fairness under different burstiness conditions and delay scenarios. We also compare their performance to those achieved by the FIFO scheme under similar circumstances.

7.5.1 Deficit Round Robin

In a DRR [70] router, each flow i is assigned to an individual queue. For the experiments in this section, the size of each queue B_i is given below:

$$B_i = B_{total} * \lambda_i / S \tag{7.1}$$

where

B_{total} is the total buffer space

λ_i is the mean arrival rate in bytes/sec

$S =$ is the service rate of node in bytes/sec

Queues are visited in a round-robin fashion and served based on the values of their *Quantum* and *DeficitCounter*. The *Quantum* represents the bandwidth share a flow

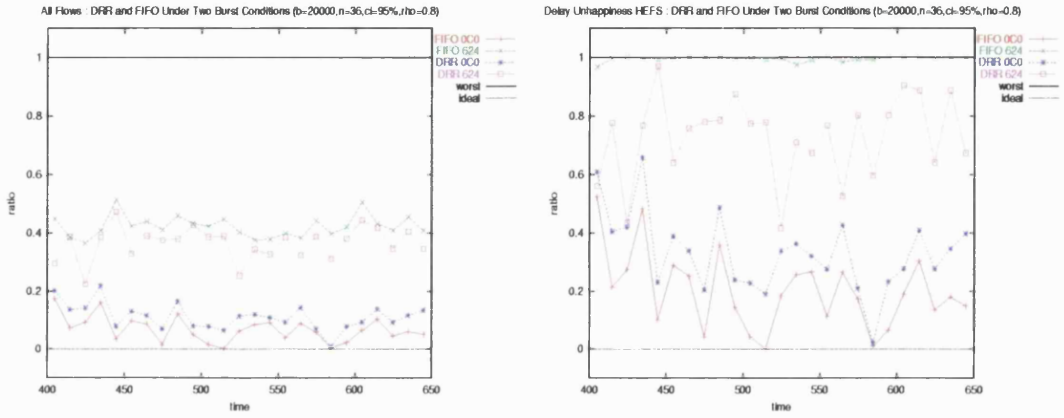
is given during one round of server visits of non-empty queues. The *DeficitCounter*, initialised to zero, is the flow's accumulated *Quantum* which is used to determine the number of bytes to be served. During each visit, the *DeficitCounter* of the corresponding flow is incremented by the *Quantum* of the flow. All packets of this flow up to a total less than or equal to the value of the *DeficitCounter* are then served. The difference between the *DeficitCounter* and the total packets served becomes the new value of the counter only if there still are packets of the flow not served. If there are no more packets to be served, the value of *DeficitCounter* is set to zero.

Burst Conditions

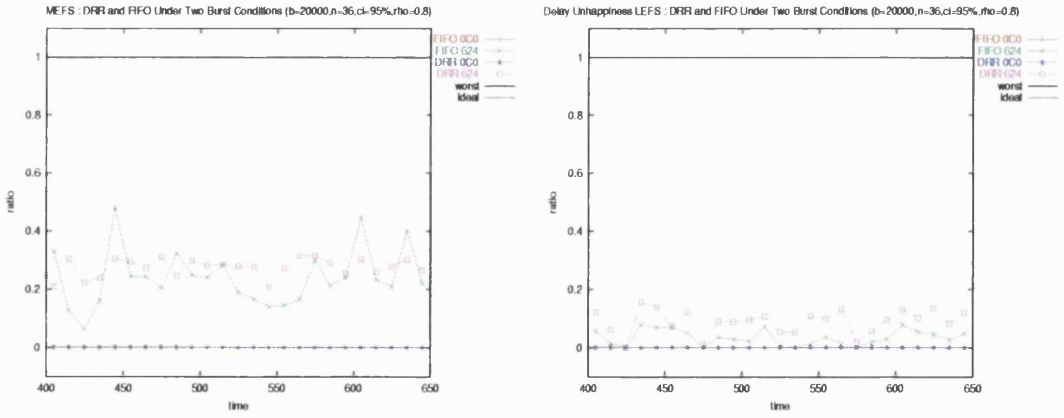
Figure 7.3 compares the delay unhappiness of flows under the DRR scheme to that of the FIFO scheme. The experiments use *tmix-0C0* and *tmix-624* with an *S-type* delay scenario. From Figure 7.3(a), we see that when the burstiness is low (*tmix-0C0*), more users are satisfied under FIFO than under the DRR scheme. However, as the burstiness increases (*tmix-624*) the situation is reversed. The lower total number of dissatisfied flows of the DRR is to a large extent due to HEFS as Figure 7.3(b) indicates. This is despite the greater satisfaction achieved by the MEFS (Figure 7.3(c)) and LEFS (Figure 7.3(d)) under the FIFO scheme. The results suggest that the flow isolation provided by DRR benefits the sensitive flows during bursty conditions. Under FIFO, the bandwidth available to MEFS and LEFS is partly protected from the demands of the HEFS.

Delay Scenarios

Figure 7.4 shows the delay unhappiness of DRR for the three delay scenarios when *tmix-624* is used. Unlike the FIFO scheme, the DRR's overall delay unhappiness is not the same for all three cases (Figure 7.4(a)). There are clear differences in the performance of the DRR between different delay scenarios. The DRR performed less favourably in the *S-type* scenario than in any other. The delay of the retransmission and the load contributed to the DRR's performance in this scenario. The additional delay associated with retransmission is reflected in the discrepancy in the overall

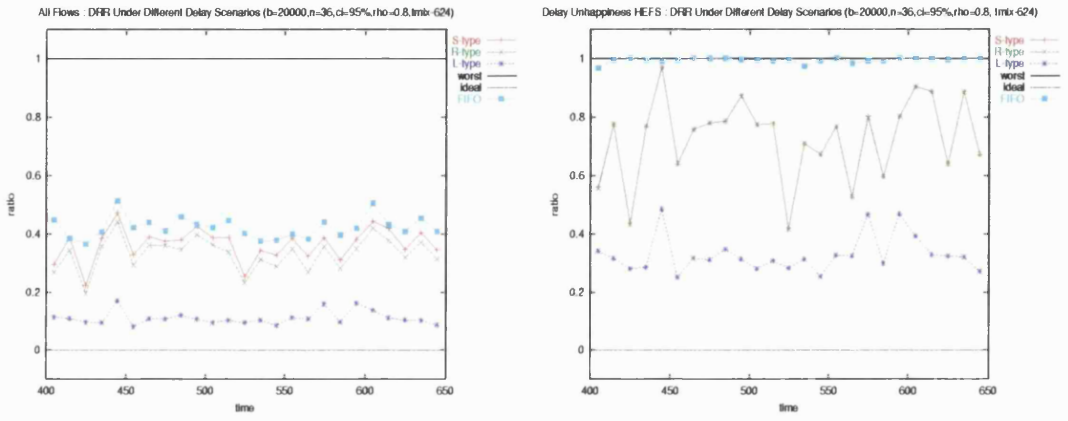


(a) ALL (b) HEFS

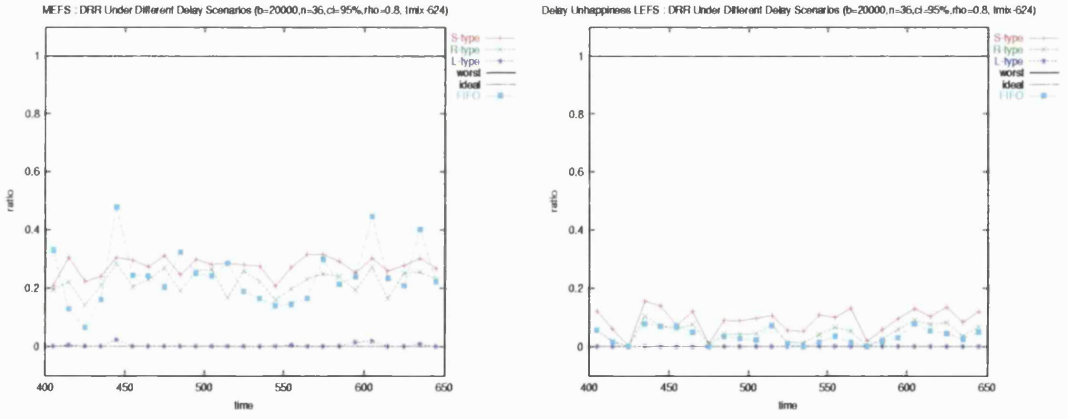


(c) MEFS (d) LEFS

Fig. 7.3: DRR Delay Unhappiness Under Baseline Burst Conditions



(a) ALL (b) HEFS



(c) MEFS (d) LEFS

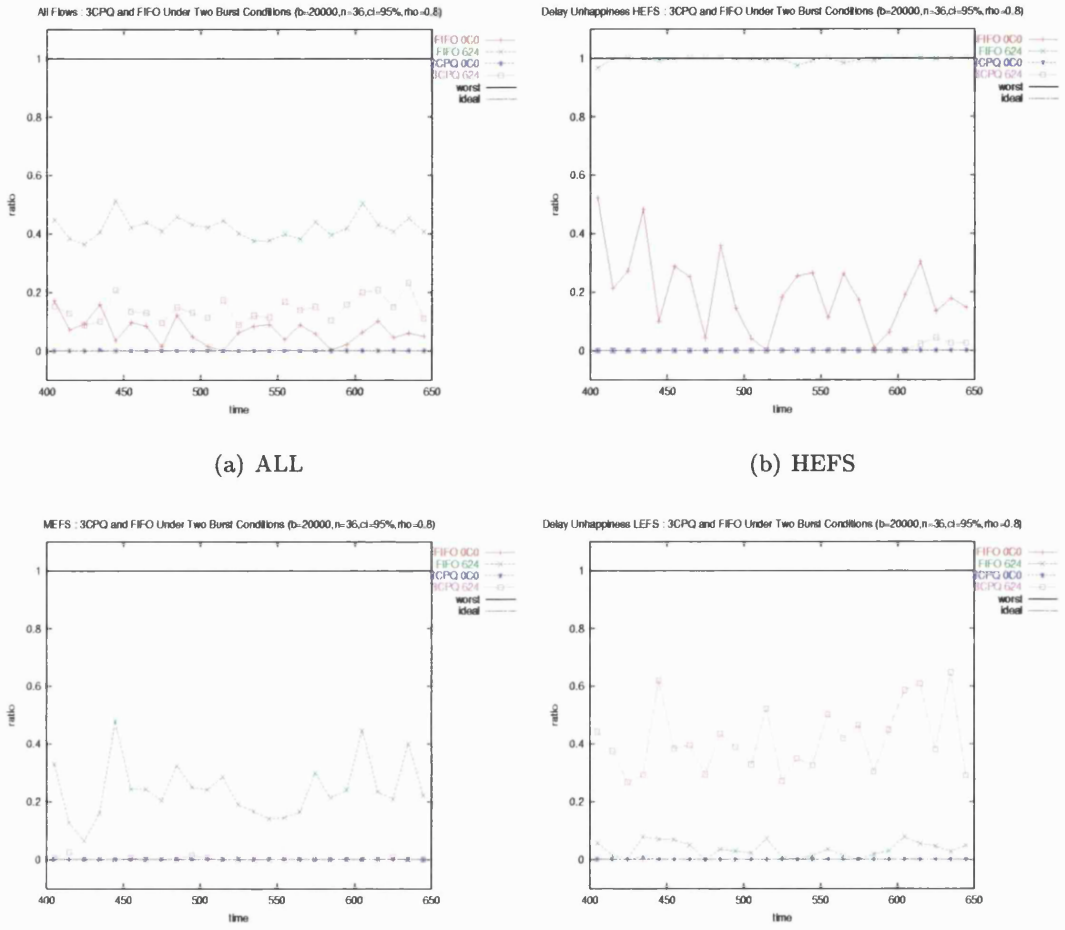
Fig. 7.4: DRR Delay Unhappiness Under Different Delay Scenarios : *tmix-624*

(Figure 7.4(a)), MEFS (Figure 7.4(c)) and LEFS (Figure 7.4(d)) delay unhappiness between the *S-type* and *R-type* experiments. The HEFS (Figure 7.4(b)) on the other hand show similar performance characteristics for DRR in both scenarios. The performance of DRR in *L-type* scenario shows a substantial improvement for all flows over the *S-type* and *R-type* scenarios. This improvement stems from the absence of retransmission which in turn lowers the throughput and congestion, especially when the traffic is bursty.

It is clear from these results that, if some level of packet loss is acceptable, the DRR can provide satisfactory service that is inter-class fair and capable of maximising the number of flows satisfied. When the retransmission of lost packets is necessary, the advantages of DRR over a FIFO scheme are not significant. In fact, for lower expectation flows, the FIFO provides a better service than the DRR.

7.5.2 Three Class Priority Queue

In this scheme, the flows are classified into three classes namely the high, medium and low priority class. We denote these classes as *HPC, MPC* and *LPC* respectively. For consistency, the HEFS are assigned to HPC, the MEFS to MPC and the LEFS to LPC. Each of these three classes operates as a FIFO queue and is allocated one-third of the total buffer space. The priority scheme is realised by serving the high priority queue until it becomes empty. When this happens, MPC packets can be served. LPC packets are only served when the queues associated with the other two classes are empty. To avoid starvation and high packet loss of the MPC and LPC, pre-emption is not allowed. Instead, the lower expectation classes (MPC and LPC) are assigned a fixed number of bytes or packets (the 3CPQ service quanta) that the class is allowed to service before HPC service is resumed. A *3CPQ* service quantum of four assigned to the MPC thus allows for the forwarding of four MEFS packets before the HPC service is restored. Generally, a 3CPQ-$X_m X_l$ scheme implies that the service quanta for the MPC and LPC are X_m and X_l respectively. While it is possible to fine tune the performance through the quanta settings, this is not the object of this report and



(a) ALL (b) HEFS

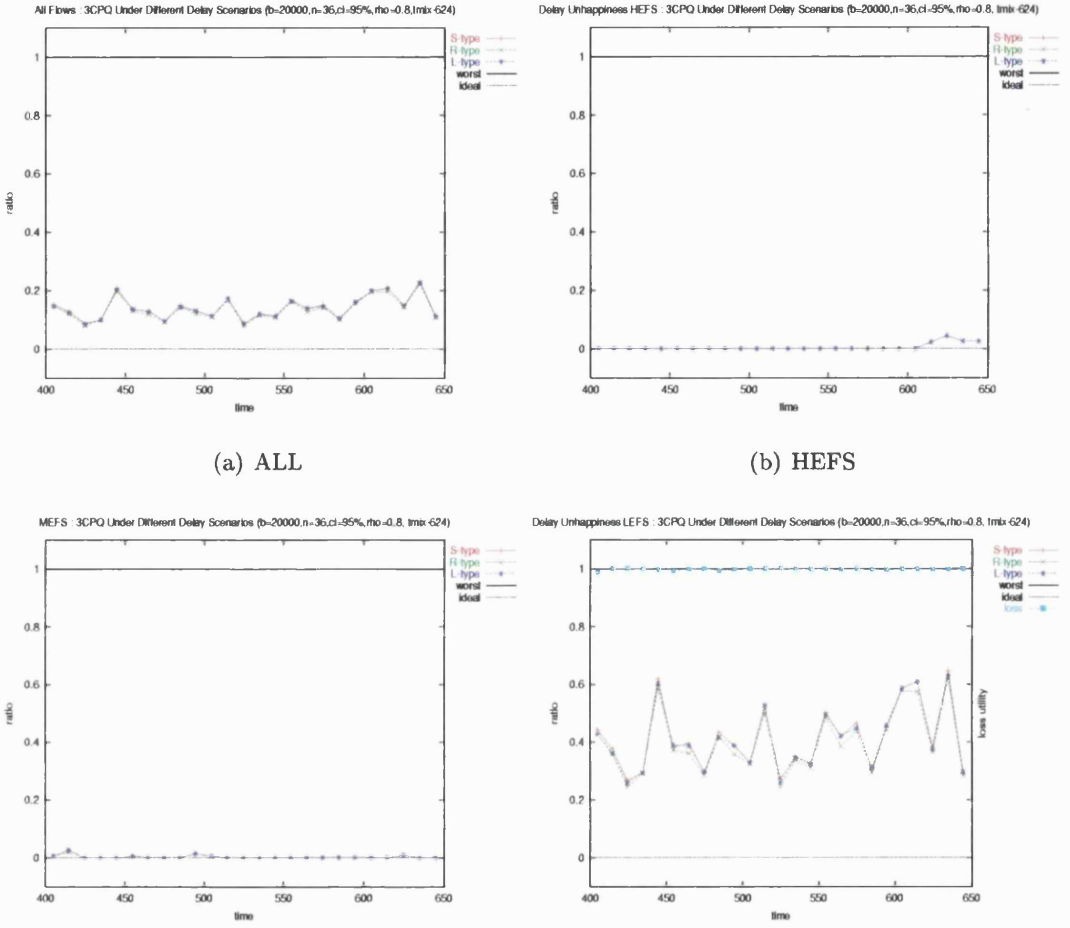
(c) MEFS (d) LEFS

Fig. 7.5: 3CPQ Delay Unhappiness Under Baseline Burst Conditions

the experiments in this section all use $X_m = 1$ and $X_l = 1$.

Burst Conditions

Figure 7.5 compares the delay unhappiness under the 3CPQ scheme with those observed when the FIFO scheme is used. Both set of experiments use identical traffic mixes and the *S-type* delay scenario. Notice that, in Figure 7.5(a), the overall delay unhappiness of the 3CPQ scheme for *tmix-624* is significantly lower than for the FIFO scheme. In addition, the 3CPQ scheme is able to achieve complete satisfaction for all flows when *tmix-0C0* is employed. The priority given by the 3CPQ scheme to the



(a) ALL (b) HEFS

(c) MEFS (d) LEFS

Fig. 7.6: 3CPQ Delay Unhappiness Under Different Delay Scenarios : *tmix-624*

sensitive flows ensures that the HEFS and to some degree the MEFS, are satisfied. However, this is at the expense of the lowest expectation group, where we find a significant number of dissatisfied flows. In contrast, the combination of *tmix-624* and the FIFO scheme exhibits no adverse effect on the LEFS. In addition, we observe that the 3CPQ is not inter-class fair as the LEFS have more dissatisfied constituents than the MEFS and HEFS combined.

Delay Scenarios

Figure 7.6 shows the delay unhappiness under 3CPQ for all three delay scenarios with *tmix-624*. We find (Figure 7.6(a)) that all scenarios exhibit virtually identical delay unhappiness. This suggests similar behaviour to the FIFO scheme with few, if any, packets lost. This conclusion is certainly true for the HEFS (Figure 7.6(b)) and MEFS (Figure 7.6(c)) observations where all the flows are satisfied. The LEFS show a slight difference among the three scenarios. This difference is the result of LPC losses as Figure 7.6(d) confirms. A further indication of dropped LEFS packets under 3CPQ is an average utility of less than $happiness_{max}$ (defined as 1.0) for the *L-type* environment.

The effect of loss is different for each scenario. As expected, the worst performance is associated with the *S-type* scenario where retransmission contributes to the total packet delay. Curiously, the *R-type* delay unhappiness is slightly better than that achieved by the *L-type* scenario. This marginal improvement may be attributed to retransmitted packets, which are considered to be independent under the *R-type* scenario. The delay they experienced may have been within the delay target and they may consequently increase the average delay utility sufficiently to make a flow happy.

7.6 Variants of Three Class Priority Queue and Deficit Round Robin

We have seen that the 3CPQ scheme, with its classification of the flows into groups and prioritisation of certain classes, managed to keep the number of dissatisfied users low. However, this was at the expense of the lower expectation flows and poor inter-class fairness. In DRR we saw that inter-class fairness was possible, but the total number of dissatisfied flows was high despite that the scheme operated on a per-flow basis. In this section, we present and evaluate two scheduling schemes that combine some features of the 3CPQ and DRR schemes. Like 3CPQ, both variants classify and group flows according to expectations, the difference lies in how the schemes serve packets. One of these schemes uses VBU information to make forwarding decisions.

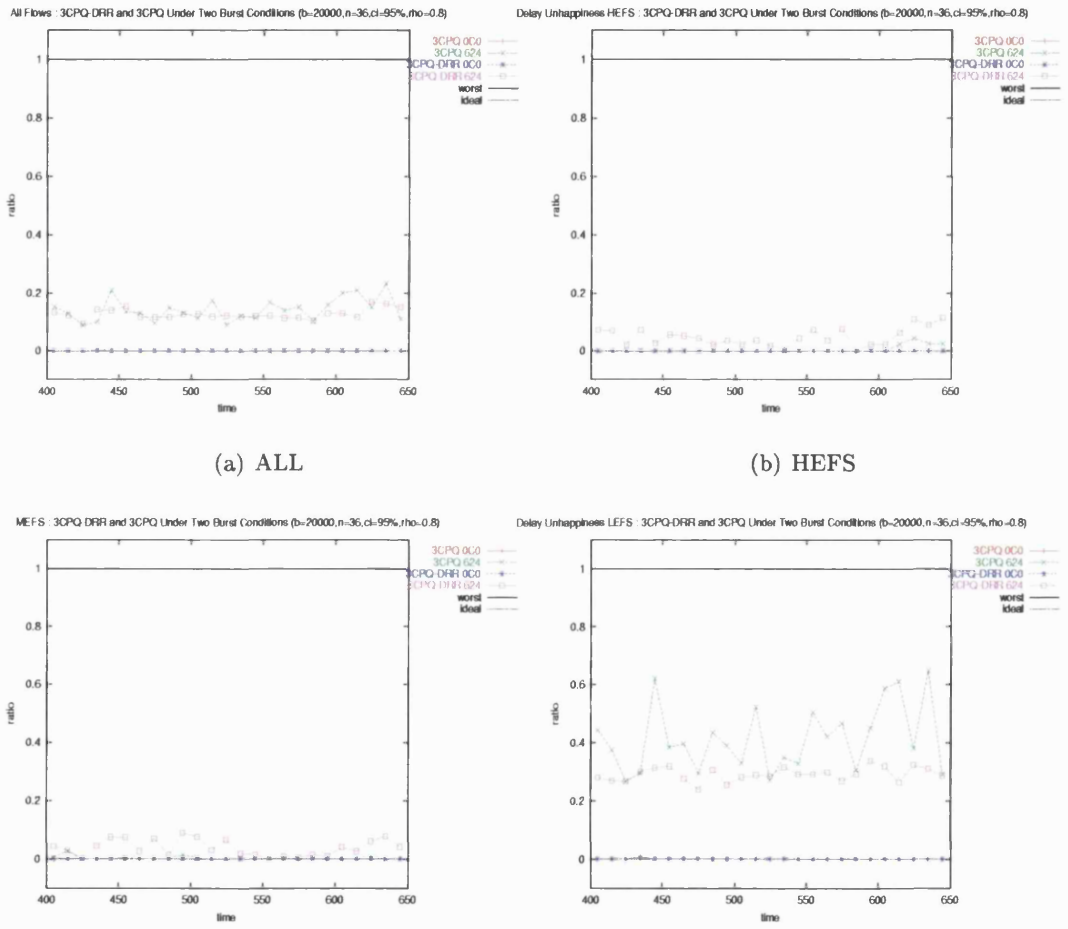
7.6.1 Three Class Priority Scheme with Per-Flow DRR

In the first variant, *Three Class Priority Scheme with Per-Flow DRR (3CPQ-DRR-X_mX_l)*, priorities are maintained and higher expectation flows are served first until there are no more packets in HPC. However the flows in all classes are serviced according to a DRR- rather than a FIFO scheme. There are two modes of operation[2] in a 3CPQ-DRR-X_mX_l server. The first mode is for the HPC, which is just like an ordinary DRR service. A queue is maintained for each flow and serviced in a DRR fashion depending on the *Quantum* and *DeficitCounter* associated with the flow. When the server has no more HPC packets to serve, it moves on to the MPC or – if the MPC is empty – the LPC. The flows in the lower classes are still served in a DRR manner. However, the server will return to the HPC when it has serviced X_m (X_l) flows. If the HPC is empty, the server will proceed to the next priority class with waiting traffic. Because the service of the lower priority classes stops after X_m (X_l) flows, the server must remember the flow it last serviced during the previous visit. All experiments in this section use X_m=1 and X_l=1 and for ease of notation, we denote such a configuration as 3CPQ-DRR.

Burst Conditions

The unhappiness levels under the 3CPQ-DRR and 3CPQ schemes for the *S-type* scenario are compared in Figure 7.7. For *tmix-624*, the total number of dissatisfied flow is slightly less with the 3CPQ-DRR than the 3CPQ (Figure 8.7(a)). Although the performance for the HEFS (Figure 7.7(b)) and MEFS (Figure 7.7(c)) were poorer in the 3CPQ-DRR experiments, nearly all the flows from these groups were satisfied. However, the delay unhappiness under 3CPQ-DRR was less variable than under 3CPQ. The advantage over 3CPQ stems mainly from its ability to ensure that flows from the lower expectation classes are protected at the very least from themselves. This is clearly illustrated in Figure 7.7(d) where the LEFS utilities reflect their protection

[2]Strictly speaking, there should be three modes because there is a one-to-one correspondence between the number of operations and classes. However, the operation of this scheme for MEF and LEF are similar and can be considered as one. The MPC and LPC differ only in the values of service quanta and priority order of server visits.



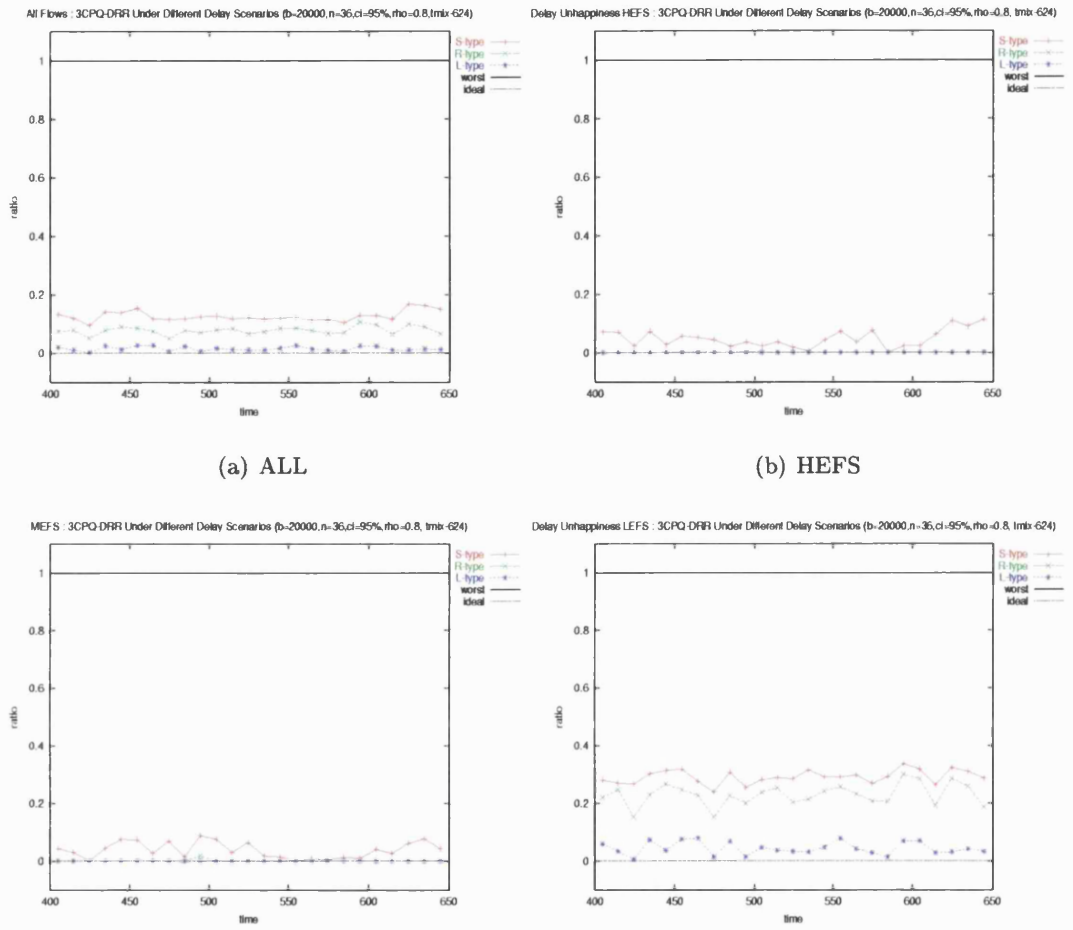
(a) ALL

(b) HEFS

(c) MEFS

(d) LEFS

Fig. 7.7: 3CPQ-DRR Delay Unhappiness Under Baseline Burst Conditions



(a) ALL (b) HEFS

(c) MEFS (d) LEFS

Fig. 7.8: 3CPQ-DRR Delay Unhappiness Under Different Delay Scenarios : *tmix-624*

under the 3CPQ-DRR scheme.

Delay Scenarios

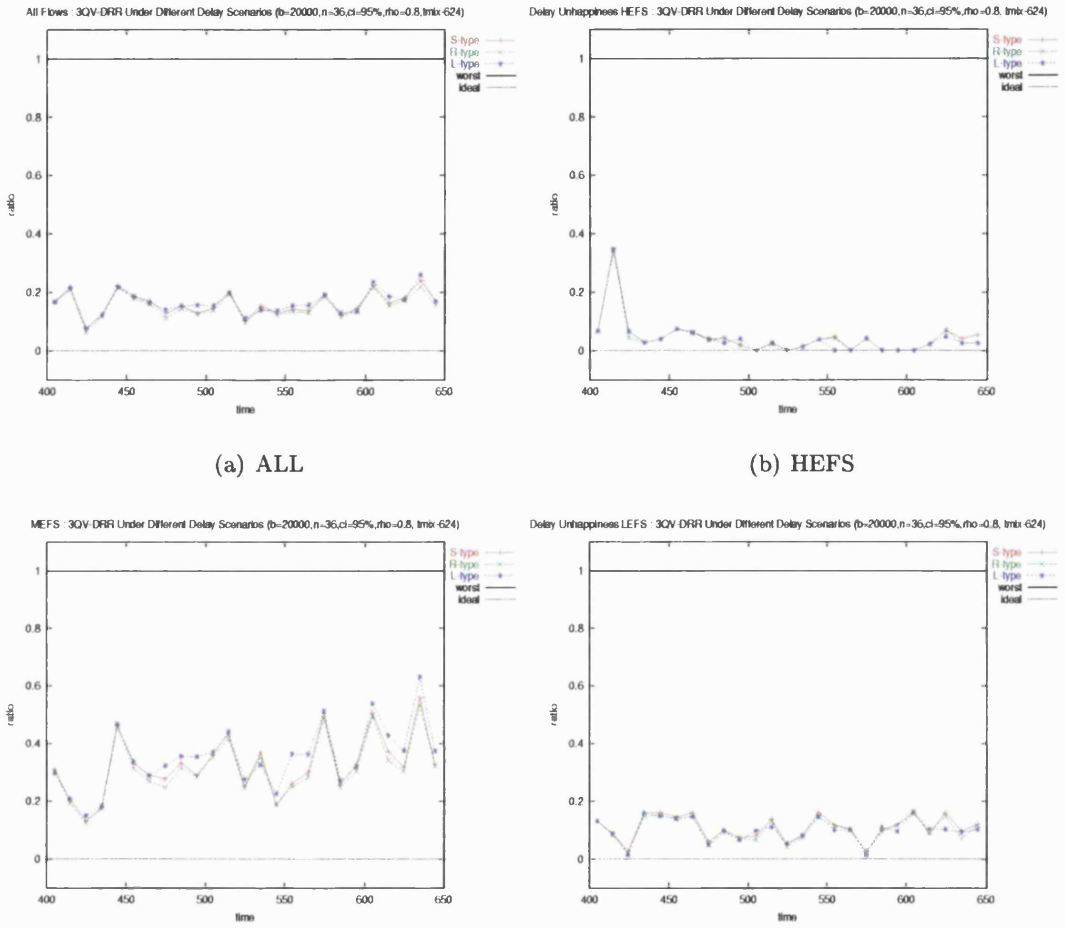
The delay unhappiness of the 3CPQ-DRR scheme for the three delay scenarios is shown in Figure 7.8. As was seen earlier with the DRR scheme, there are small but distinctive differences in the delay unhappiness of each scenario under 3CPQ-DRR. As expected, DRR-like behaviour is observed in the performance of 3CPQ-DRR. This is because the 3CPQ-DRR operates a round-robin service inside a class. All flows are allocated individual queues which means that loss will influence the resulting

delay utilities. This can be clearly seen in the delay unhappiness of 3CPQ-DRR for all the different expectation groups. In Figures 7.8(a), 7.8(b), 7.8(c) and 7.8(d), we find the *S-type* scenario gives the worst delay unhappiness. In the case of the HEFS (Figure 7.8(b)) and MEFS (Figure 7.8(c)), the *R-type* scenario's performance is nearly identical to that of the *L-type* experiments. This is again expected because these two groups have high priority service resulting in lower delays and in turn leading to the satisfaction of all HEFS and MEFS. In any priority scheme, one or more groups will receive degraded service; for this particular scheme, the LEFS are at the lowest end of the priority ladder. The difference between the *R-type* and *L-type* experiments can therefore easily be discerned. The retransmission of dropped packets as new packets may occur more than once which explains the difference of at least 10% more dissatisfied users between the *R-type* and *L-type* operation.

7.6.2 Active Use of Value-Based Utility to Schedule Packets

In this section, consider the performance of a scheme that uses utility information to schedule packets. The *Three Queue Value-Based Utility Deficit Round-Robin* (3QV-DRR) scheme partitions the buffer into three FIFO queues HPC, MPC and LPC, each of which represents a class associated to one of the expectation groups. Packets belonging to the HEFS are placed in the HPC while the MEFS and LEFS are queued in the MPC and LPC respectively. Instead of individual utilities, the router will assign delay expectations for a class and measure utilities accordingly. Like the basic DRR scheme, *Quanta* and *DeficitCounter* values are associated with each of the queues. The operations of the scheme are similar to that of DRR except that no more than three queues can be served per round.

The main difference between this scheme and DRR is that the value of the *Quanta* associated with a class is allowed to change during operation. The change is based on the perceived utility of the class. Depending on the utility of the classes, a class can either share quanta, receive quanta or retain the value of its quanta. Sharing and receiving quanta is a complementary action. A class will only allow its excess *Quanta*



(a) ALL

(b) HEFS

(c) MEFS

(d) LEFS

Fig. 7.9: 3QV-DRR Delay Unhappiness Under Different Delay Scenarios : *tmix-624*

to be shared if it is satisfied. The recipient of the excess *Quanta* is always a class that is dissatisfied. This implies that the total number of *Quanta* moving around the three classes is constant. If all classes are happy, then no quanta will be shared or received. As class states change, we could expect to see the *Quanta* values of the classes adapt. This adaptation will then hopefully result in fewer dissatisfied flows while retaining inter-class fairness.

Delay Scenarios

Figure 7.9 displays the delay scenarios for the 3QV-DRR under *tmix-624* conditions. Unlike some schemes we have seen earlier (i.e., DRR, 3CPQ-DRR), the total number of dissatisfied flows is slightly higher under the *L-type* scenario than any of the others (Figure 7.9(a)). Looking at the unhappiness of individual classes, we see that this unexpected result was largely due to the poor performance of the MEFS (Figure 7.9(c)). Although the MPC for all delay scenarios over-committed themselves by giving some quanta away much to their detriment, the *L-type* suffered most. It appears that the MPC in the *L-type* scenario gave more quanta than its counterparts. Unfortunately, the quanta it shared did little to improve on the performance of the delay unhappiness of either the HEFS (Figure 7.9(b)) or LEFS (Figure 7.9(d)) with respect to other scenarios. The poor performance in the *L-type* experiment can be attributed to lost packets which can be inferred from the lower throughput as compared with the *S-type* scenario (Figure 7.10). Unlike in other scenarios, the losses under *L-type* experiment would decrease the delay unhappiness because lost packets are never retransmitted. The chances are that, with a smaller number of packets going through the router, a class experiencing loss can become more satisfied. This makes the class a candidate as a donor of quanta. By giving up its quanta early on, the MPC under the *L-type* scenario performed slightly worse than its counterparts in the *S-type* and *R-type* scenarios. Had the traffic mixes been more bursty or the load higher, the loss rates would have been higher and the delays experienced by retransmitted packets would have increased. This would have subsequently increased the unhappiness of flows under *S-type* and *R-type*. We could therefore expect their unhappiness to be higher than those experienced by flows in the *L-type* case.

Shared Quanta

To confirm that the MPC under the *L-type* scenario shared more quanta than its counterparts in the other scenarios, we show the class quanta of the three classes for the *S-type* and *L-type* scenarios in Figure 7.11. The initial quantum values at time 400

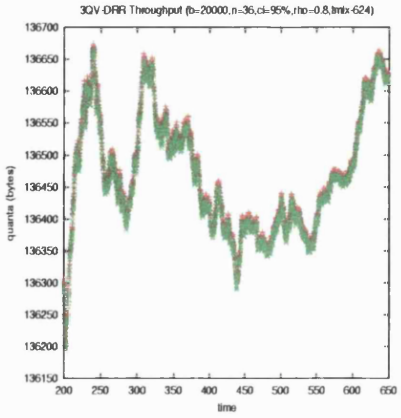


Fig. 7.10: 3QV-DRR Throughput

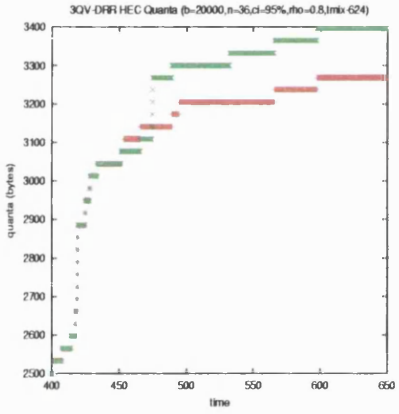
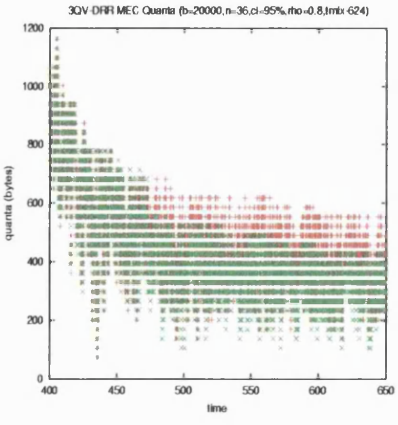
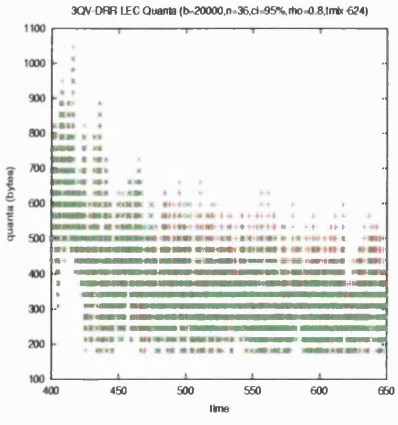


Fig. 7.11: 3QV-DRR Class Quanta Under Different Delay Scenarios : *tmix-624*

(a) HPC



(b) MPC

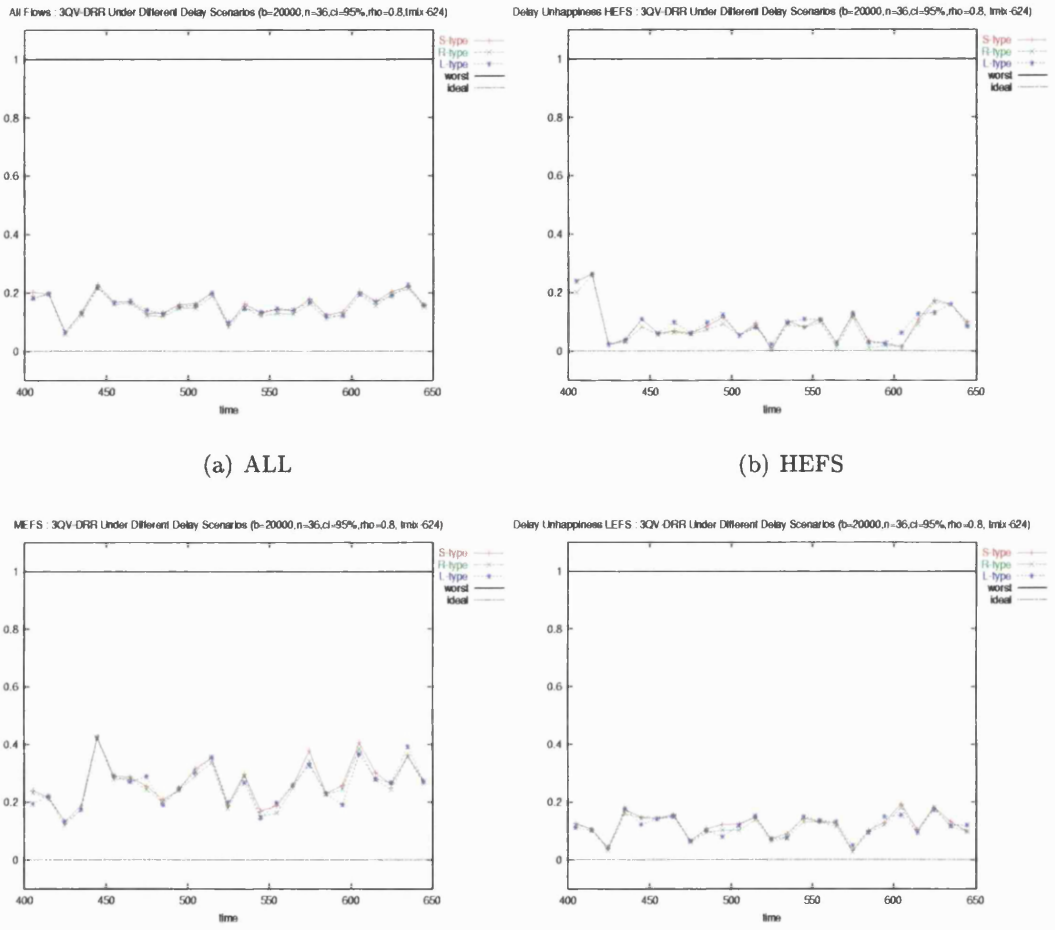


(c) LPC

seconds is set to 2500 (Figure 7.11(a)), 1000 (Figure 7.11(b)) and 500 (Figure 7.11(c)) bytes for the HPC, MPC and LPC respectively. In Figure 7.11(a), we find that between 450-500 seconds the quanta of HPC of the *L-type scenario* becomes larger than the HPC of the *S-type* scenario. At the same time, the quanta of the *L-type*'s MPC became smaller with respect to the MPC of the *S-type* scenario. What happened here was that the router assumed the flows in the MPC were satisfied enough for some of their quanta to be shared with other classes. Although this was the correct action at the time, it was not beneficial in the long term (Figure 7.9(c)). This is because the HPC is the most sensitive group and is often in trouble. It is therefore unable to return the quanta to the MPC. This limits the source of shared quanta to the MPC and LPC. To make matters worse, it appears that the HPC is able to obtain some more quanta from either the MPC or the LPC thus reducing further the number of potential excess quanta. In the case of the LPC, this was less of a problem because it is less sensitive than the MPC. The inability of HPC to return quanta it borrowed because it is still dissatisfied lead to inter-class unfairness. This is true regardless of what scenario is considered under *tmix-624* conditions.

3QV-DRR with Retrieval

An obvious enhancement to the 3QV-DRR scheme is to allow a class to retrieve its shared quanta when it comes into trouble. In this scheme, designated as *3QV-DRR with Retrieval (3QV-DRR-R)*, the basic operation is similar to 3QV-DRR except for one aspect. The main difference from the basic 3QV-DRR scheme is the ability of a class in this scheme to forcefully reacquire quanta it has shared, up to its initial allocation, regardless of the state of the class it will come from. This means that even if a class is dissatisfied, some of the quanta it is currently using can be returned to the original owner. The performance of this scheme is shown in Figure 7.12. As we can see, the level of dissatisfaction is not as high among the MEFS (Figure 7.12(c)) as we observed under 3QV-DRR scheme (Figure 7.9(c)). This was achieved mainly because the HPC was forced to return the *Quanta* it borrowed. This action prevented



(a) ALL (b) HEFS

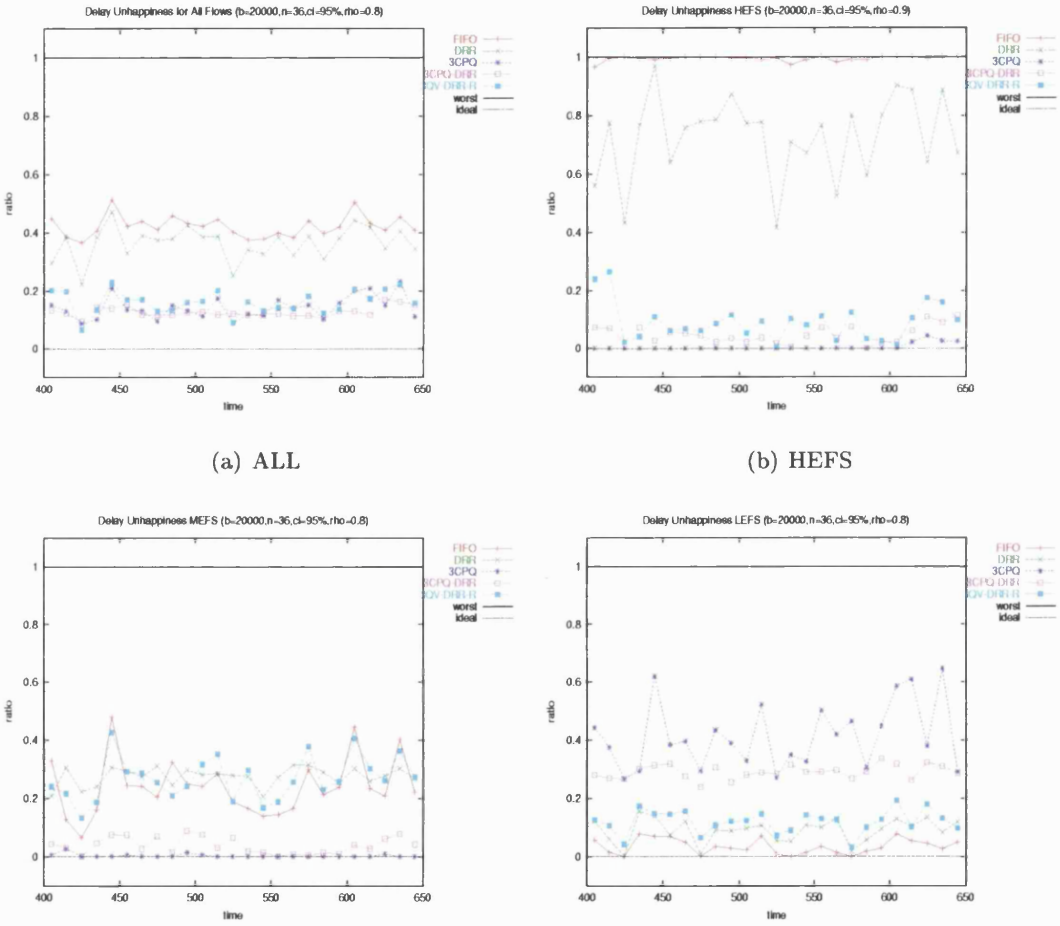
(c) MEFS (d) LEFS

Fig. 7.12: 3QV-DRR with Retrieval Under Different Delay Scenarios : *tmix-624*

the other flows in the other classes from suffering and also lowered the inter-class unfairness seen in the basic 3QV-DRR scheme.

Burst Conditions

Figure 7.13 shows the delay unhappiness of the 3QV-DRR-R alongside the performance of the other schemes for the traffic mix *tmix-624* under the *S-type* delay scenario. In Figure 7.13(a), where the overall delay unhappiness is shown, two groups of curves based on performance can be seen. The first group consist of the FIFO and DRR schemes while the second is composed of the 3CPQ, 3CPQ-DRR and 3QV-DRR-



(a) ALL

(b) HEFS

(c) MEFS

(d) LEFS

Fig. 7.13: Delay Unhappiness of All Schemes : *tmix-624*

R schemes. Focusing on the results of the 3QV-DRR-R scheme, we find that a large improvement over the FIFO was achieved. The 3QV-DRR-R even approximates the results of the 3CPQ scheme and to some lesser degree, the 3CPQ-DRR scheme. This result can also be observed in the delay unhappiness of the HEFS in Figure 7.13(b). This performance was made possible because the 3QV-DRR-R allowed the HPC to receive a share of the service quanta of the MPC and LPC. Unlike in the 3CPQ and 3CPQ-DRR schemes, 3CPQ-DRR-R did not allow the low expectation flows to become excessively dissatisfied. This was because of the ability of classes in this scheme to share quanta and retrieve the quanta it shared. This feature allowed 3CPQ-DRR-R to approximate the performance of the FIFO for the MEFS and LEFS. Of the three schemes with the lowest overall number dissatisfied flows, the 3QV-DRR-R is the closest to being inter-class fair.

7.7 Summary

In this chapter we showed that in a FIFO scheme for all delay scenarios, sensitive flows suffer delays exceeding their target bounds during bursty conditions leading to flow dissatisfaction. We also saw that lower expectation flows are less likely to be affected by such conditions. One redeeming feature of the FIFO scheme is that, for the traffic mixes we have looked at, it is inter-class fair.

After establishing a reference baseline, we evaluated several schemes representing different levels of complexity and service granularity. The first scheme we looked at was DRR. In DRR each flow is allocated a portion of the available buffer and guaranteed minimum bandwidth through the service quanta. In the three delay scenarios, the DRR proved effective when a certain level of packet loss can be tolerated even during bursty conditions. If loss is deemed important, the combination of small individual buffers and the burstiness of sources can overwhelm the DRR and lead to deterioration in service. Although the DRR performs slightly poorly for the MEFS and the LEFS during these scenarios, it still managed to significantly improve on the performance of the HEFS of the FIFO while retaining inter-class fairness.

The next scheme we looked at was the 3CPQ scheme in which flows were explicitly mapped to a service class. This scheme operates on a priority basis. The scheme proved extremely effective for the higher expectation flows but it penalises the lowest expectation flows. Although 3CPQ improved on the FIFO and the DRR in terms of overall dissatisfaction, this scheme was not inter-class fair.

The third scheme we evaluated was the 3CPQ-DRR scheme. This scheme retained the class distinction and priorities of the 3CPQ. However, instead of aggregating flows and scheduling them in a FIFO manner, a DRR service for each class was used. Although, the performance of the HEFS and MEFS was not as good as 3CPQ, it managed to reduce the number of dissatisfied LEFS in relation to 3CPQ. In terms of fairness, the 3CPQ-DRR was less unfair than 3CPQ.

In the last scheme, 3QV-DRR, we retained the aggregation of flows into classes. Instead of prioritising higher expectation flows, we treated each class as a FIFO queue and operated DRR on top of the three classes. In addition, VBU was used to adjust, during the operation of the scheme, control values that could increase or decrease the delay a class of flows will experience. In terms of performance, this scheme nearly reached the same overall delay unhappiness levels of 3CPQ-DRR. However, this was at the expense of slightly poorer performance of its lower expectation flows as compared to the FIFO scheme. An enhancement of this scheme was also presented that prevents classes from holding on to the shared quanta indefinitely. In the 3QV-DRR-R scheme, where classes are allowed to retrieve quanta it has given away, the inter-class unfairness seen in the basic 3QV-DRR scheme was minimised.

The main conclusion we draw in this chapter is that utility can be used to find operating points of algorithms and schemes that yields the least number of dissatisfied flows in terms of delay while maintaining inter-class fairness. Results from the FIFO, DRR, 3CPQ and 3CPQ-DRR suggest that the different operating points, which include scheme specific parameters and other router parameters, can potentially be adapted based on utilities to manage traffic. In 3QV-DRR and 3QV-DRR-R, we were able to some degree successfully use utility in this manner. However, our success was

limited to simple adaptation and further analysis of improving the use of other scheme specific parameters is needed.

Chapter 8

Conclusions

8.1 Overview

In this chapter we present the conclusions of this dissertation and an overview of the future directions that this work may follow.

8.2 Conclusions

The potential of using the state and degree of user satisfaction in effectively managing router resources and services has been largely unrealised. In this dissertation we claim that both types of information could be used locally inside the router for the purposes of traffic management. We demonstrate this by using the state and degree of satisfaction for buffer management and scheduling.

We have defined a formulation called *Value-Based Utility* (VBU) which is capable of expressing both the state and degree of user satisfaction. In Sections 2.1 and 2.2, we used a simple example to reveal the deficiencies of only using either the state or degree to represent user satisfaction. We assert that not only can the utility function defined in Section 2.3 provide a measure of user satisfaction given a resource allocation or service, but that it also can be used as a tool for management. We showed in the experiments in Section 2.4 that this utility function is indeed capable of expressing user satisfaction. Additionally, the results indicate that utilities can be used to find the ideal operating point of the Ethernet (i.e., channel utilisation versus frame size

and rate).

We have established a framework wherein we showed how utility can be used to manage router functions and services. In Chapter 3, we explored the notion of using VBU for management. We first examined how network environments exercise control to offer service differentiation. We identified specific features and characteristics of some of these control mechanisms that can be manipulated. We then described in general terms how these features can be used to effect a policy based on sharing. The approach we suggest does not require changing how the schemes operate, we do however need them to base their resource management decisions on VBU. We also defined four main measures namely, *Overall Unhappiness*, *Class Unhappiness*, *Class Average Utility* and *Utility Fairness*, and described the conditions where each type is appropriate in Section 4.4. We use these measures to assess the performance of the schemes in Chapters 5 through 7.

We have demonstrated the use of VBU for buffer management by developing extensions for a specific buffering mechanism and proposing a new scheme that manages buffers using VBU. In Section 5.2, we identified a specific parameter of the *G+98* [37] scheme that when varied, provides different levels of performance. We offer two variants of the G+98 scheme that exploit this feature in Sections 6.5.2 and 6.5.3. We showed how VBU, in both derivatives, can be used to adjust the parameter and provide improved service over the basic G+98 scheme. In Section 5.3, we used the notion of individual utility and *Loss Unhappiness* to determine which packet to accept and drop. We examined this scheme's performance in a range of conditions in Sections 6.4 and 6.5 wherein the results showed it to be the one of the more robust.

We have provided additional evidence that VBU can be used for management by proposing a scheduling scheme that makes decisions based on utility. We first analysed the performance of three benchmark scheduling mechanisms namely, *FIFO*, *3CPQ* and *DRR* in Sections 7.4 and 7.5 under different delay scenarios and traffic conditions. We then addressed some of the problems associated with these policies by developing the VBU-based scheduling scheme, *3QV-DRR* in Section 7.6.2. We

demonstrated experimentally that this scheme minimises the unfairness of the 3CPQ while improving on the loss unhappiness of DRR.

These results indicate that VBU is a flexible framework that can be adapted into existing management mechanisms. Its adoption is further motivated by the improved performance of some of the VBU-based mechanisms over their non-VBU aware counterparts. The uniqueness of our framework, however, is that it offers a new perspective on performance management. By combining both the state and degree of user satisfaction, we revise the definition of acceptable resource allocations.

8.3 Future Work

The VBU schemes we have presented have managed expectations based on information from a heuristically defined time interval. As such, the validity of our results is limited to this period. A more detailed study on using VBU that considers the issue of time scale of management [35] that is reflective of real situations is required. Additionally, the conditions (high utilisation, small buffer and limited number of flows) under which we have examined our schemes do not necessarily scale. We do however suspect that when utilisation is lower, the proposed schemes may in fact prove effective as there is more room to adapt. More experiments will be needed to resolve this issue.

The schemes we have proposed focused on obvious parameters which when adjusted, yielded the desired effects. However, the effects of other scheme-specific parameters to determine if they, like the parameters we have considered can be used in the same manner, must be evaluated. A related issue that must also be considered is the effect of interaction between parameters. In line with this, the feasibility of investigating the application of utility to other schemes is naturally forthcoming.

In a more general sense, we would like to see VBU applied in the context of end-to-end QoS provisioning such as in [5, 57, 58]. The issues of how different VBU mechanisms interact and deliver the necessary satisfactory levels is an interesting question to pose. Equally important, the translation of real user requirements to the form we specified is worth investigating. A parallel issue to investigate is to determine

whether a single utility function can encapsulate different QoS expectations of users.

Appendix A

CLOWN Modelling

A.1 Overview

The models in this dissertation were developed using the Concatenated LOcal-area and Wide-area Network (**CLOWN**) simulator [72, 71]. CLOWN is an object-oriented simulation environment where models are built using a library of predefined and user-defined network objects. We begin by describing the system architecture of CLOWN, focusing on the functionality provided by the different system components. We then illustrate building models in CLOWN using examples from our experiments.

A.2 CLOWN System Architecture

Before modelling and simulation, it is necessary to understand the basic features available in CLOWN and how they all fit together. The system architecture of CLOWN's simulation engine is shown in Figure A.1. There are seven components each with their own function. At the heart of this engine is the *Experimentation Manager*. This component acts as the central coordinator of activities for other components. Users build CLOWN objects using high-level descriptions of the model which requires translation before simulation can commence. The *Model Builder* performs this functionality. It takes in as input a file with the high-level description of models and parses them into a form that is passed on to the Experimentation Manager. The Experimentation Manager checks whether the model descriptions are valid and consistent with

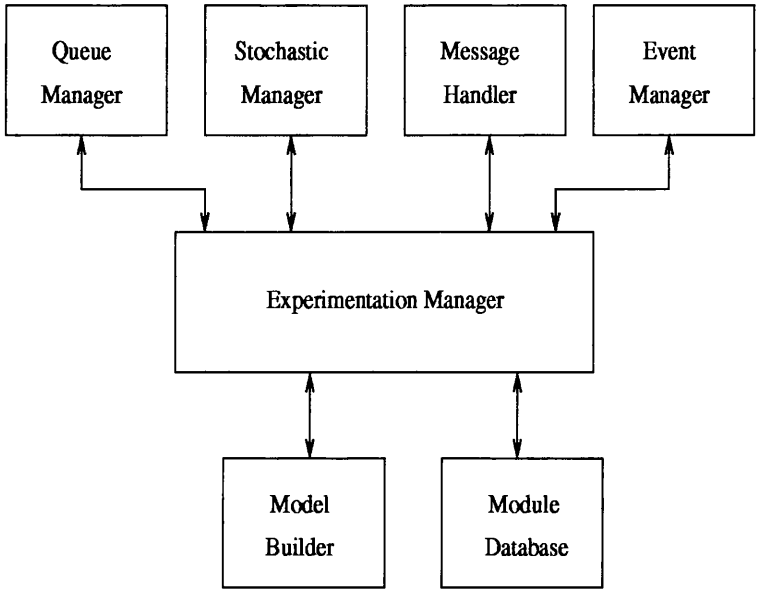


Fig. A.1: CLOWN System Architecture

the entries in the *Module Database.* In particular, it verifies that events exchanged between models are correct. Once these relationships are established, the simulation can begin. The module definitions in the Module Database are extensible and can be created or modified using C-constructs and structures. The *Queue Manager* maintains a small set of scheduling disciplines with the necessary data handling facilities such as create, insert and delete. The Queue Manager is reconfigurable to allow for a wider choice of schedulers. The function of the *Message Handler* is to provide support for message exchange between network objects. It maintains a data structure and a set of primitive operations to access and communicate information. The *Event Manager* maintains the simulation event list. It also provides the necessary facilities to select, order, insert and remove events in the event queue. The *Stochastic Manager* provides a library of random number generators to support event driven simulation.

A.3 Model Building

A single node network topology is used in most of our experiments. The goal of these experiments is to evaluate the performance of different buffer management and

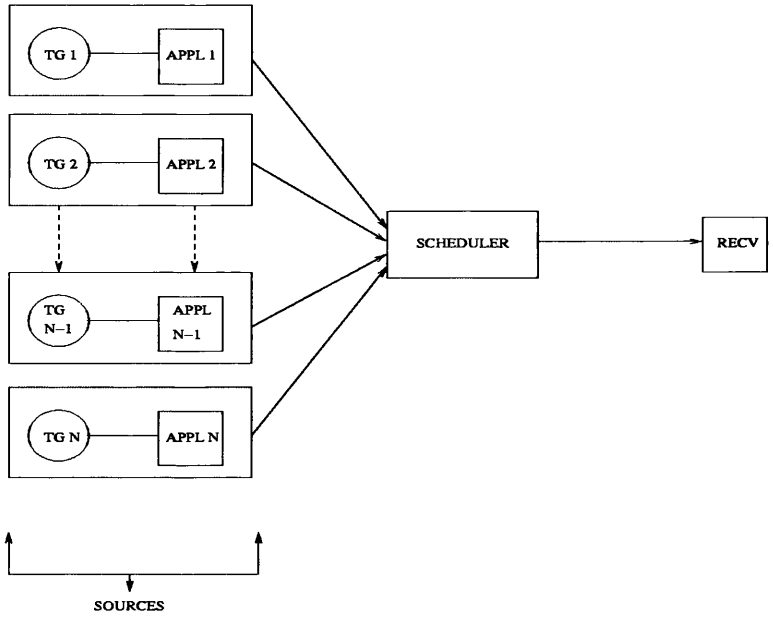


Fig. A.2: CLOWN Model

scheduling algorithms under a wide range of conditions. Building CLOWN objects to represent the network model and conditions is straightforward once the modules in the Module Database have been defined and developed. The basic CLOWN model built for the experiments is shown in Figure A.2. There are N applications with different requirements and expectations. Each application has a traffic generator associated with it. The traffic generator produces packets according to predefined parameters. The packets generated by the traffic generator are then sent to the application. After some processing, it is then forwarded to the scheduler. The scheduler is an abstraction for the functions of buffering and scheduling. The model for the scheduler will change depending on which algorithm is evaluated and used. The scheduler will then buffer incoming packets when it is busy and forward them to the receiver or sink when possible. At the receiver, the utility and other appropriate measures are then evaluated. These models are built from the Module Database. In the next section, we describe how to build modules.

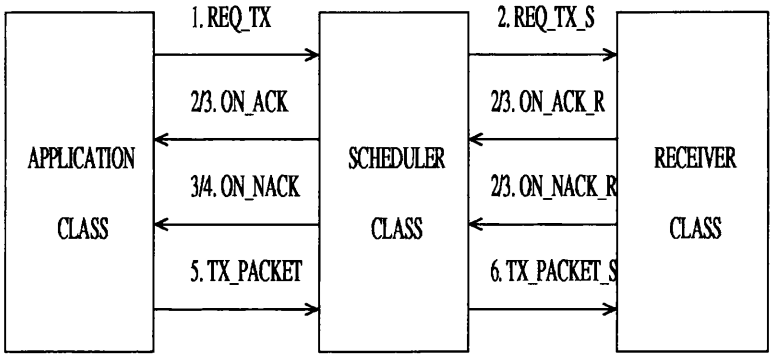


Fig. A.3: Exchanged Messages Between Classes

A.4 CLOWN Object Modelling

As in object-oriented modelling, the central theme in CLOWN is the concept of class. A class defines a template of behaviour for a group of objects. An object is an instance of a class that has both a structure and a set of procedures for initialising and using it. An OBJECT in CLOWN is composed of four parts: state variables, associated methods, set of incoming events, and set of exported events. The state variables and associated methods are local to the class and define its properties. This means that no other object can directly access these variables and methods. The behaviour of a CLOWN object can only be influenced by a set of incoming events which is the exported events of other CLOWN objects. These sets of events provide the objects an interface with other objects that it is allowed to communicate with. The exchanged events between the sources, the scheduler and the receiver are shown in Figure A.3. Therefore, it is ideal to represent object behaviour using state-transition diagrams.

A.4.1 Source Transition Diagram

A simple three-state transition diagram shown in Figure A.4 can be used to model the high-level behaviour of the source (Traffic Generator and Application). The source initially starts on the *Begin Transmission* state. From this state, the source seeks permission from the network to transmit packets. Once it has transmitted the request

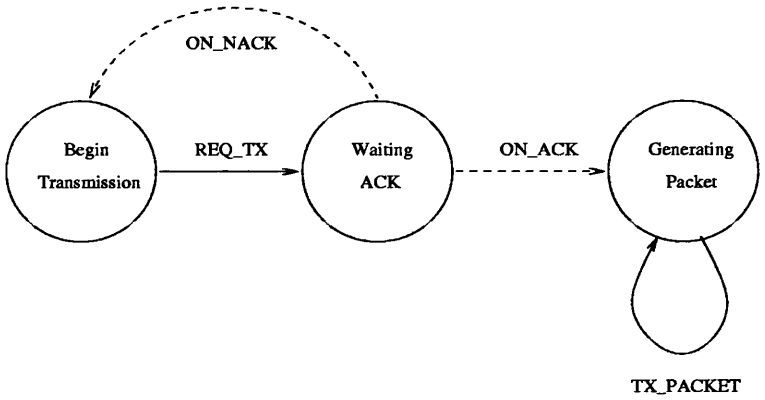


Fig. A.4: A Three-State Transition Diagram of Source Behaviour

REQ_TX, the source will go to the *Waiting ACK* state. On this event REQ_TX, the source also communicates information about its traffic characteristics and expectations of the new connection. This information will allow the network to decide whether to accept the new connection or not. On a non-acknowledgement (ON_NACK) from the network, the source goes back to the first state. On an acknowledgement (ON_ACK), the source moves on to the *Generating Packet* state. Once here, the source will continuously create and transmit packets. Note that events represented by solid arrows are local events (events the object creates) while dashed arrows are external events (events other objects create). For example, REQ_TX and TX_PACKET are local while ON_ACK and ON_NACK are external events.

From the initial state-transition diagram and after some refinements, a two-class representation of the source is used. Each of the classes has its state and properties. The first class is the GENERATOR class while the second is the APPLICATION class. The GENERATOR class specifies the traffic characteristics and produces packets. The request for admission and actual transmission of the packets are the responsibilities of the APPLICATION class. The partitioning of the source in this manner allows for future extendibility of the model. Future extensions may include a leaky bucket regulator or the provision for the negotiation of new requirements and expectations by the source. Before a CLOWN model representing the classes can be built, a more detailed transition-diagram must be prepared. The detailed state-transition

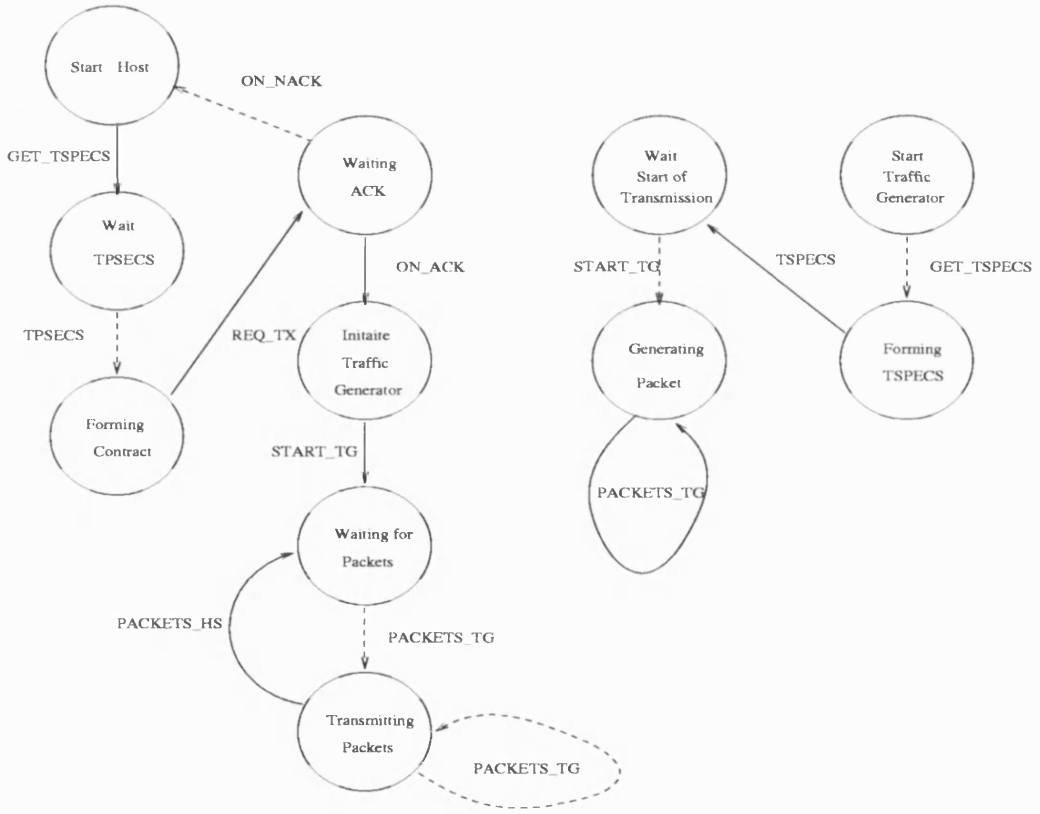


Fig. A.5: Detailed State Transition Diagram of Source Behaviour

```
typedef struct Appl  {
    int SourceID;
    int DestinationID;
  CLtime StartTime;
  double BandwidthShare;
  double OutBandwidth;
  double PropagationDelay;

   /*
      The following parameters represent the traffic specification. These parameters will be used
      in computing utility at the management nodes as well as reporting performance of system.
   */

   double LossTarget;              /* The target loss probability.                          */
   double DelayTarget;             /* The target delay probability meeting delay bound.     */
   double DelayBound;              /* The expected delay.                                   */
   double JitterTarget;            /* The target jitter probability meeting jitter bound.   */
   double JitterBase;              /* This could be used instead of delay bound in computation.  */
   double JitterUpperBound;        /* The expected upperbound jitter offset from delay bound.    */
   double JitterLowerBound;        /* The expected lowerbound jitter offset from delay bound.    */

   /* The following variables represent the traffic characterisation. */

   double MeanRate;
   double PeakRate;
   double MeanPacketSize;
          int Sigma;               /* The third parameter. Others can also be included.     */

   CLaddr TransmitBuffer;          /* Packets are stored in a transmit buffer if it cannot be
                                      transmitted and is used in conjunction with a leaky bucket.  */
      int TransmitBufferSize;      /* The size of the source buffer.                        */
      int TransmitBufferSizeLeft;

} APPL_DATA, *APPL_DATA_P;

  void APPL_Module( int );
CLaddr APPL_DataSetup();
  void APPL_Edit( FILE *, int, int, CLaddr );
  void APPL_Startup();

  void APPL_RetryRequest();
  void APPL_FinishTransmitting();
  void APPL_InitiateRequest();
  void APPL_NewPacketFormed();
  void APPL_RetryRequest();
  void APPL_ReceivingToken();
  void APPL_StartGenerator();

#define SERVER_INTERFACE 1
#define GENERATOR_INTERFACE 2
#define LEAKYBUCKET_INTERFACE 3
```

Fig. A.6: Application Class Declaration

diagrams for the APPLICATION (left) and GENERATOR (right) CLASS are shown in Figure A.5. The next step is to define the class.

A.4.2 CLOWN Class Declarations

In defining a class, the declarations must have a section for the state variables, the function headers and the interface definitions. A declaration for the APPLICATION class with this parts is shown in Figure A.6. The variables of the class APPLICATION is defined as a structure of type *Appl.* Each object in the class has its characteristics that are based on the values of the structure. For example, the application object is uniquely identified by the *SourceID* field. Other variables also include the time when the application started transmitting, the traffic characteristics, the expectations and the transmission buffer properties. The function headers following the class declaration

```
.
.
.
.
.
void CLOWN_RegisterUserModules()
{
     TG_Module( LIBRARY_NUMBER+1 );
     LB_Module( LIBRARY_NUMBER+2 );
     APPL_Module( LIBRARY_NUMBER+3 );
     SERVER_Module( LIBRARY_NUMBER+4 );
     RECEIVER_Module( LIBRARY_NUMBER+5 );

} /* END CLOWN_RegisterUserModules */
.
.
.
.
.
```

Fig. A.7: Registering Classes

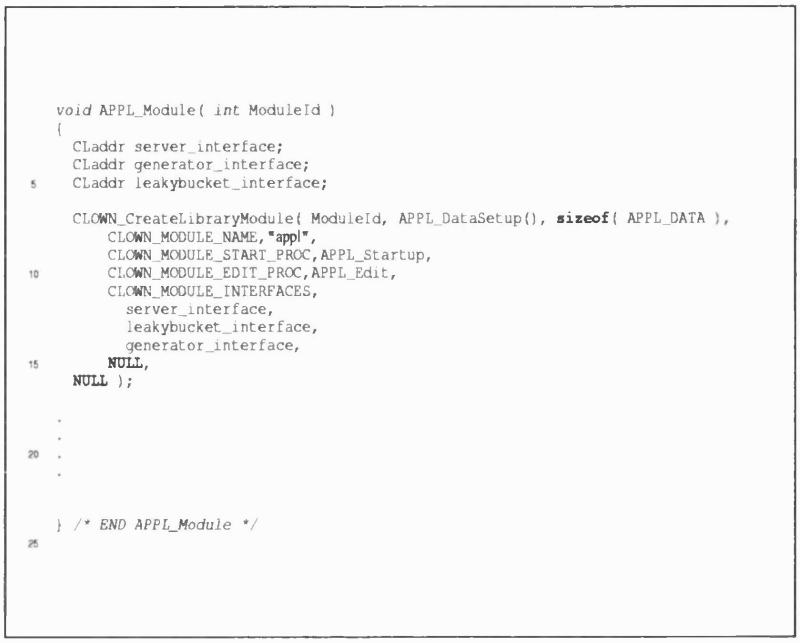
defines the methods associated with this class. In CLOWN, there are two groups of methods: the basic and the user defined methods. The basic methods are the required functions for any class. The user-defined methods on the other hand, define the functions that will describe the objects behaviour in addition to the required functions. The possible objects that are allowed to communicate with an object in the APPLICATION class are defined as interfaces. There are three of these namely: the SERVER, the GENERATOR and the LEAKYBUCKET. We shall discuss these interfaces later in the section.

A.4.3 Creating and Registering Class Modules

In order to report a module to the experimentation manager it is necessary to run an initialisation function. This is accomplished through the function

CLOWN_RegisterUserModules()

The class library for a simulation is constructed with this function. In order for a class to be registered, its initialisation function must be called from this function. For



```
void APPL_Module( int ModuleId )
{
  CLaddr server_interface;
  CLaddr generator_interface;
  CLaddr leakybucket_interface;

  CLOWN_CreateLibraryModule( ModuleId, APPL_DataSetup(), sizeof( APPL_DATA ),
      CLOWN_MODULE_NAME,"appl",
      CLOWN_MODULE_START_PROC,APPL_Startup,
      CLOWN_MODULE_EDIT_PROC,APPL_Edit,
      CLOWN_MODULE_INTERFACES,
        server_interface,
        leakybucket_interface,
        generator_interface,
      NULL,
  NULL );

  .
  .
  .
  .

} /* END APPL_Module */
```

Fig. A.8: Registering APPLICATION Module

the experiments in this dissertation, five classes were built. The registration process is shown in Figure A.7.

The details of initialisation function APPL_Module(int ModuleID) of the APPLICATION class is shown in Figure A.8. APPL_Module calls the function

CLOWN_CreateLibraryModule(*type, data, datasize, argument_list*)

The CLOWN_CreateLibraryModule function performs the actual registration of the class. The first argument *type* of this function is the module ID that identifies the class in the class library. The second argument *data* is a pointer to objects state variables. The third argument *datasize* is the size of the *Appl* data structure in bytes. The fourth argument is a variable argument list. It recognises the parameters for: the module name, a pointer to a start-up function, a pointer to a function that edits variables of the object and a list of pointers to associated interfaces of the class.

```
CLaddr APPL_DataSetup()
{
  APPL_DATA_P data;

  /* CLOWN's version of the memory allocation function malloc. */
  data = (APPL_DATA_P) CLOWN_Malloc( sizeof( APPL_DATA ) );

  data->SourceID               = 0;
  data->DestinationID          = -1;
  data->StartTime              = 0.;
  data->MeanRate               = 5.;
  data->PeakRate               = 5.;
  data->Sigma                  = 5;
  data->OutBandwidth           = 0.;
  data->PropagationDelay       = 0.;

  data->LossTarget             = -1.0;
  data->DelayTarget            = -1.0;
  data->DelayBound             = -1.0;
  data->JitterTarget           = -1.0;
  data->JitterBase             = -1.0;
  data->JitterUpperBound       = -1.0;
  data->JitterLowerBound       = -1.0;

  data->TransmitBuffer         = Null(CLaddr);
  data->TransmitBufferSize     = 1000;
  data->TransmitBufferSizeLeft = data->TransmitBufferSize;

  return( (CLaddr) data );

} /* END APPL_DataSetup */
```

Fig. A.9: Initialising Object State Variables

A.4.4 Initialising Object State Variables

In the function CLOWN_CreateLibraryModule, a call to a function APPL_DataSetup was made. The function APPL_DataSetup is used to initialise the variables of the object and to return a pointer to the created state variables. The definition of this function for the APPLICATION class is given in Figure A.9.

A.4.5 Creating a Class Startup Function

The call to a startup procedure

CLOWN_MODULE_START_PROC()

in CLOWN_CreateLibraryModule is used to prepare the objects of the class and to commence simulation. The function that is registered as the startup function for the APPLICATION class is APPL_Startup. This is shown in Figure A.10. The function is called once for each instance of the module class when the simulation goes through its initialisation procedure. The function is here used to initiate a request to get the traffic characteristics from the generator.

```
void APPL_Startup()
{

    APPL_DATA_P data;

    data = (APPL_DATA_P) CLOWN_GetModuleData();
    data->State = INIT;
    data->TransmitBufferSizeLeft = data->TransmitBufferSize;
    data->TransmitBuffer = CLOWN_CreateQueue( FIFO_QUEUE, data->TransmitBufferSize );
    /* Call request admission module. This is to randomise to start of transmission. */
#ifdef RANDOM_START
    CLOWN_EventByText( data->StartTime + CLOWN_Random(), "Request contract appl =>tg", GENERATOR_INTERFACE
, 0, Null(Claddr) );
#else
    CLOWN_EventByText( data->StartTime, "Request contract appl =>tg", GENERATOR_INTERFACE, 0, Null(Claddr)
 );
#endif

} /* END APPL_Startup */
```

Fig. A.10: Creating a Startup Function

A.4.6 Creating an Instance of a Class

The function APPL_Edit is registered as the edit function. It is activated every time a new instance of the APPL class is created to allow a user to manipulate the factors. The input to this function is a data file where the object parameters are specified. An "End of definition" call in the data file will terminate the factor update session.

A.4.7 Creating Class Interfaces

The other function called by APPL_Module (Figure A.11) is

CLOWN_CreateInterface(*interface_id, num, argument_list*)

CLOWN_CreateInterface defines the associated interfaces of the class. This function is invoked three times in the APPLICATION class because there are three interfaces declared in CLOWN_CreateLibraryModule. The first argument to this function is the interface label defined in the class definition. The second argument is the number of objects that can connect to the interface. The variable argument list of CLOWN_CreateInterface recognises a list of text strings representing events exported by the interface (CLOWN_ REQUESTS) and a list of actions supported by the interface (CLOWN_ACTIONS).

```
void APPL_Module( int ModuleId )
{

 .
 .
 .
 .

 generator_interface =
    CLOWN_CreateInterface( GENERATOR_INTERFACE, /* Copies */ 1,
      CLOWN_ACTIONS,
        CLOWN_CreateAction( "Contract tg => appl", (void *) APPL_InitiateRequest ),
        CLOWN_CreateAction( "Packet tg => appl", (void *) APPL_NewPacketFormed ),
      NULL,
      CLOWN_REQUESTS,
    "Request contract appl => tg",
    "Begin transmission appl => tg",
      NULL,
  NULL );

  leakybucket_interface =
    CLOWN_CreateInterface( LEAKYBUCKET_INTERFACE, /* Copies */ 1,
      CLOWN_ACTIONS,
        CLOWN_CreateAction( "Token lb => appl", (void *) APPL_ReceivingToken ),
      NULL,
      CLOWN_REQUESTS,
    "Request token appl => lb",
      NULL,
  NULL );

  server_interface =
    CLOWN_CreateInterface( SERVER_INTERFACE, /* Copies */ 1,
      CLOWN_ACTIONS,
        CLOWN_CreateAction( "ACK Transmission server => appl", (void *) APPL_StartGenerator ),
        CLOWN_CreateAction( "NACK Transmission server => appl", (void *) APPL_RetryRequest ),
      NULL,
      CLOWN_REQUESTS,
"Request Transmission appl => server",
        "Packet Transmitted appl => server",
      NULL,
  NULL );

} /* END APPL_Module */
```

Fig. A.11: Creating Class Interfaces

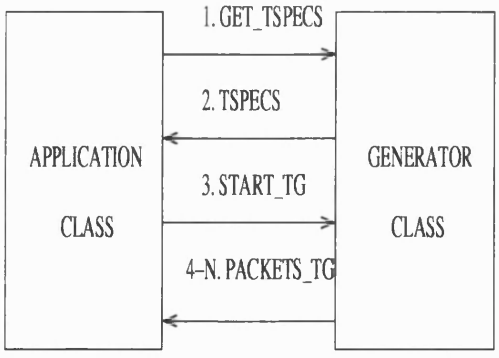


Fig. A.12: Events Between APPLICATION and GENERATOR Classes

A.4.8 APPLICATION and GENERATOR Interface

Consider now the events between the APPLICATION and GENERATOR class. From their state-transition diagrams we see that there are four events - two from each class (Figure A.12) - exchanged between them. The interface defined by both classes must be able to recognise and handle these events. From the APPLICATION's perspective, the events TSPECS and PACKETS_TG are incoming events while GET_TSPECS and START_TG are exported events. The APPLICATION class associates an action to each incoming event. This is supported by the function

CLOWN_CreateAction(*Name, Action*)

For example, in

CLOWN_CreateAction("Contract *tg* => *appl*", (void *) APPL_InitiateRequest)

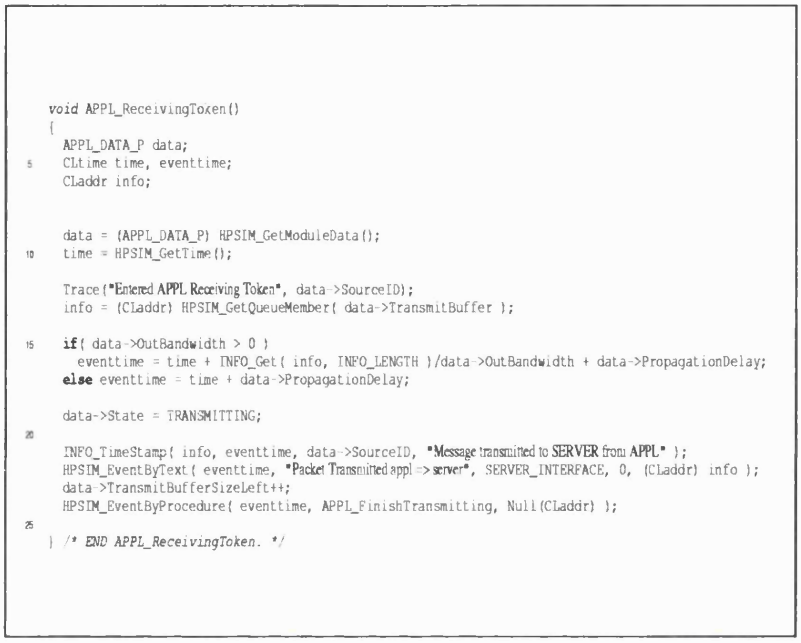
the response to an incoming event "Contract tg => appl" is to execute the function APPL_InitiateRequest. This transition models the action when the APPLICATION receives the TSPECS events from the GENERATOR. The list of exported events is specified by listing them as CLOWN_REQUESTS. From the GENERATOR's perspective, the classification of the events is reversed.

A.4.9 Event Handling

Once the interfaces between two classes are defined, events can now be exchanged using one of two functions. The choice of the function depends on whether the event is internal to the object or destined for another object or an incident of the same object. If the event is internal, both object state and action procedure are known. The user can therefore use the function

CLOWN_EventByProcedure(*time, action, info*)

Here *time* is the time the event will mature, *action* is a pointer to the action itself and *info* is a pointer to further information associated with the event.



```
void APPL_ReceivingToken()
{
  APPL_DATA_P data;
  CLtime time, eventtime;
  CLaddr info;

  data = (APPL_DATA_P) HPSIM_GetModuleData();
  time = HPSIM_GetTime();

  Trace("Entered APPL Receiving Token", data->SourceID);
  info = (CLaddr) HPSIM_GetQueueMember( data->TransmitBuffer );

  if( data->OutBandwidth > 0 )
    eventtime = time + INFO_Get( info, INFO_LENGTH )/data->OutBandwidth + data->PropagationDelay;
  else eventtime = time + data->PropagationDelay;

  data->State = TRANSMITTING;

  INFO_TimeStamp( info, eventtime, data->SourceID, "Message transmitted to SERVER from APPL" );
  HPSIM_EventByText( eventtime, "Packet Transmitted appl => server", SERVER_INTERFACE, 0, (CLaddr) info );
  data->TransmitBufferSizeLeft++;
  HPSIM_EventByProcedure( eventtime, APPL_FinishTransmitting, Null(CLaddr) );

} /* END APPL_ReceivingToken. */
```

Fig. A.13: Event Handling

If the event is not local, the EventByProcedure call cannot be used because the object lacks knowledge of the address of the receiver object. That information is only available to the Experimentation Manager. All an object knows is the user addresses of its ports. Furthermore, the object will not control the actions associated with the receiver object. The object must send events through a port using the function

CLOWN_EventByText(*time, name, port_type, port_number, info*)

The argument *time* is the time the event will mature, *name* is the name of the event, *port_type* is the interface type of the port, and *port_number* is the incidence number of the port. Ports are numbered consecutively within each type. The argument *info* is a pointer to further information associated with the event. An example of how both these events handling procedures are used is shown in Figure A.13. In this sequence, the APPLICATION object exported the event "Packet Transmitted appl => server" to the SERVER_INTERFACE along with associated messages with the event. After updating its buffer count, it now invokes a local procedure so it can change it state. No messages are passed in this local event.

```
void APPL_InitiateRequest()
{
  APPL_DATA_P data;
  CLaddr contract;
  CLtime time;
  double newdelaybound;

  time = CLOWN_GetTime();
  data = (APPL_DATA_P) CLOWN_GetModuleData();

  contract = CLOWN_GetEventInfo();

.
.
.
} /* End APPL_InitiateRequest */
```

Fig. A.14: Message Passing

A.4.10 Message Passing

When an action is activated the user may want access to the state of the object and the information associated with the event. This is obtained by calling

CLOWN_GetEventInfo()

which returns a the pointer to the event information, and

CLOWN_GetModuleData()

which returns a pointer to the data structure associated with the receiver object. An example of how these functions are used is shown in Figure A.14. When the action APPL_InitiateRequest is activated, calls to both CLOWN_GetModuleData and CLOWN_GetEventInfo are invoked. The information from both functions can now be used. It may be updated depending on the subsequent actions in APPL_InitiateRequest.

A.4.11 Queue Management

CLOWN offers a range of queue management facilities. Creating queues is done by using

CLOWN_CreateQueue(*type, size*)

where *type* is the queuing discipline and *size* is the maximum number of elements that can be placed in the queue. The function returns a reference pointer which should be used in all subsequent communications with the queue. There are currently five types of queuing disciplines supported namely:

- FIFO_QUEUE (first in first out)
- LIFO_QUEUE (last in first out)
- PRIORITY_QUEUE (message priority)
- SHORTEST_DELAY_QUEUE (shortest message first)
- SERIAL_QUEUE (message serial number)

For example, in Figure A.5, a FIFO_QUEUE was created to hold the packet coming from a GENERATOR object.

Once the queue is defined, information can be placed in the queue using

CLOWN_AddQueueMember(*queue, info*)

where *queue* is reference pointer to the queue and *info* is a pointer to information to be placed on the queue. The function returns *TRUE* if the message has been queued and *FALSE* if it has been refused because the maximum queue length was exceeded. Removing information from the queue is done using

CLOWN_GetQueueMember(*queue*)

This function will return the information reference pointer, or a null pointer if the queue is empty. Other queue management features include emptying all contents of the queue, getting the size of the queue and getting information about elements in the queue.

A.4.12 Random Numbers

To use CLOWN's library of random number generators, the function

CLOWN_RandGen(*type, param1, param2*)

is used where *type* is the distribution type. There are currently five distributions supported namely:

- DETERMINISTIC
- UNIFORM_DISTRIBUTION
- EXPONENTIAL_DISTRIBUTION
- NORMAL_DISTRIBUTION
- WEIBULL_DISTRIBUTION

The library can easily be extended like most of the modules in CLOWN. The function returns a single random number from a distribution according to the type-request. The parameters *param1* and *param2* have different meaning according to the selected distribution. The GENERATOR class uses this random number generator library. The size and the time when the packets are transmitted are based on these generators.

Bibliography

[1] B. W. Abeysundara and A. E. Kamal. High-Speed Local Area Networks and Their Performance : A Survey. *ACM Computing Surveys*, 23(2):221–264, June 1991.

[2] R. Apteker, J. Fisher, V. Kisimov, and H. Neishlos. Video Acceptability and Frame Rate. *IEEE Multimedia*, 2(3):32–40, Fall 1995.

[3] C. Aras, J. Kurose, D. Reeves, and H. Schulzrinne. Real-Time Communication in Packet-Switched Networks. In *Proceedings of the IEEE*, volume 82, pages 122–139, January 1994.

[4] J. M. Arco, B. Alarcos, A. M. Helln, and D. Meziat. Quality of Service over Ethernet for Telelearning Applications. In *Proceedings of the 4th Annual SIGCSE/SIGCUE on Innovation and Technology in Computer Science Education*, 1999.

[5] C. Aurrecoechea, A. Campbell, and L. Hauw. A Survey of QoS Architectures. *ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture*, 6(3):138–151, 1998.

[6] T. Berners-Lee, R. Cailliau, J.-F. Groff, and B. Pollermann. World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, 1(2), Spring 1992.

[7] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The World Wide Web. *Communications of the ACM*, 37(8):76–82, August 1994.

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

[9] A. Bouch, M. A. Sasse, and H. G. DeMeer. Of Packets and People: A User-Centred Approach to Quality of Service. In *Proceedings of 8th International Conference on Quality of Service (IWQoS'00)*, June 2000.

[10] R. Braden, D. Clark, and S. Shenker. Intgrated Services in the Internet Architecture: An Overview. RFC 1633, June 1994.

[11] L. Breslau and S. Shenker. Best-Effort Versus Reservations: A Simple Comparative Analysis. *ACM Computer Communication Review*, 28(4):3–16, 1998.

[12] Z. Cao and E. Zegura. Utility Max-Min: An Application Oriented Bandwidth Allocation Scheme. In *Proceedings of IEEE INFOCOM 99*, March 1999.

[13] S. Casner and S. Deering. First IETF Internet Audiocast. In *ACM SIGCOMM'92 Proceedings*, pages 92–97, September 1992.

[14] I. Cidon, R. Guerin, and A. Khamisy. On Protective Buffer Policies. *IEEE/ACM Transactions on Networking*, 2(3):240–246, June 1994.

[15] D. Clark and W. Fang. Explicit Allocation of Best Effort Packet Delivery Service. *IEEE/ACM Transactions on Networking*, 6(4):362–373, August 1998.

[16] M. Claypool and J. Tanner. The Effects of Jitter on the Perceptual Quality of Video. In *Proc. ACM Multimedia '99*, pages 115–118, 1999.

[17] R. Cocchi, S. Shenker, D. Estrin, and L. Zhang. Pricing in Computer Networks: Motivation, Formulation, and Example. *IEEE/ACM Transactions on Networking*, 1(6):614–627, December 1993.

[18] C. Cowan, S. Cen, J. Walpole, and C. Pu. Adaptive Methods for Distributed Video Presentations. *ACM Computing Surveys*, 27(4):580–583, December 1995.

[19] R. V. Cox and P. Kroon. Low Bit-Rate Speech Coders for Multimedia Communication. *IEEE Communications Magazine*, 12(34):34–41, December 1996.

[20] L. DaSilva, D. Petr, and N. Akar. Equilibrium Pricing in Multiservice Priority Based Networks. In *IEEE GLOBECOM 97*, pages 232–239, November 1997.

[21] J. D. DeTreville. A Simulation-Based Comparison of Voice Transmission on CSMA/CD Networks and on Token Buses. *AT & T Bell Laboratories Technical Journal*, 63(1):33–54, January 1984.

[22] C. Diot and L. Gautier. A Distributed Architecture for Multiplayer Interactive Applications on the Internet. *IEEE Network*, 13(4):6–15, July/August 1999.

[23] T. A. ElBatt, S. El-Henaoui, and S. Shaheen. Jitter Recovery Strategies for Multimedia Traffic in ATM Networks. In *GLOBECOM '96*, 1996.

[24] H. Eriksson. MBONE : The Multicast Backbone. *Communications of the ACM*, 37(8):55–60, August 1994.

[25] W. Fang. *Differentiated Services: Architecture, Mechanisms and an Evaluation.* PhD thesis, Princeton University, 2000.

[26] D. Ferrari. Client Requirements for Real-Time Communication Services. RFC 1193, November 1990.

[27] D. Ferrari and D. Verma. A Scheme for Real-Time Channel Establishment in Wide-Area Networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.

[28] C. M. Festin, S. Clayman, and S. Sørensen. Jitter Analysis. In *Fourth Communication Networks Symposium*, 1997.

[29] C. M. Festin and S. Sørensen. Measurement and Analysis of Delay Jitter. In *Thirteenth UK Computer and Telecommunications Performance Engineering Workshop*, pages 1–6, 1997.

[30] C. M. Festin and S. Sørensen. A Framework for User Expectation of QoS Requirements. In *Fourteenth UK Computer and Telecommunications Performance Engineering Workshop*, pages 94–101, 1998.

[31] N. Figueira and J. Pasquale. Leave-in-Time: A New Service Discipline for Real-Time Communications in a Packet-Switching Network. *Computer Communication Review*, 25(4):207–218, October 1995.

[32] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[33] G. Ghineas and J. Thomas. QoS Impact on User Perception and Understanding of Multimedia Video Clips. In *Proceedings of ACM Multimedia Conference*, pages 49–54, September 1998.

[34] T. A. Gonsalves and F. A. Tobagi. Comparative Performance of Voice/Data Local Area Networks. *IEEE Journal on Selected Areas in Communications*, pages 657–669, June 1989.

[35] M. Grossglauser, S. Keshav, and D. Tse. RCBR: A Simple and Efficient Service for Multiple Time-Scale Traffic. *IEEE/ACM Transactions on Networking*, 5(6):741–755, December 1997.

[36] J. Grudin. Computer-Supported Cooperative Work: History and Focus. *IEEE Computer*, pages 19–26, May 1994.

[37] R. Guerin, S. Kamat, V. Peris, and R. Rajan. Scalable QoS Provision Through Buffer Management. *Proceeding of ACM SIGCOMM*, pages 29–40, October 1998.

[38] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. RFC 2597, June 1999.

[39] J. Hui, E. Karasan, J. Li, and J. Zhang. Client-Server Synchronization and Buffering for Variable Rate Multimedia Retrievals. *IEEE Journal on Selected Areas in Communications*, 14(1):226–237, January 1996.

[40] D. Hutchison, G. Coulson, A. Campbell, and G. S. Blair. Tech. Rep. MPG-94-02: Quality of Service Management in Distributed Systems. Technical report, Lancaster University, 1994.

[41] B-ISDN Service Aspects. ITU-T Recommendation I.211, March 1993.

[42] Information Technology - Open Distributed Processing - Reference Model: Foundations. ITU-T Recommendation X.902, November 1995.

[43] V. Jacobson, K. Nichols, and K. Poduri. An Expedited Forwarding PHB. RFC 2598, June 1999.

[44] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems. Technical report, Eastern Research Lab Digital Equipment Corporation, 1984.

[45] S. Jamin, P. Danzig, S. Shenker, and L. Zhang. A Measurement-Based Admission Control Algorithm for Integrated Service Packet Networks. *IEEE/ACM Transactions in Networking*, 5(1):56–70, February 1997.

[46] J. I. Jung. Quality of Service in Telecommunications Part I: Proposition of a QoS Framework and Its Applications to B-ISDN. *IEEE Communications Magazine*, pages 112–117, August 1996.

[47] G. Karlsson. Asynchronous Transfer of Video. *IEEE Communications Magazine*, pages 118–126, August 1996.

[48] S. Keshav. Report on Workshop on Quality of Service Issues in High Speed Networks. *ACM SIGCOMM Computer Communication Review*, 22(5):74–85, October 1992.

[49] I. Kouvelas, O. Hodson, V. Hardman, and J. Crowcroft. Redundancy Control in Real-Time Internet Audio Conferencing. In *Proceedings of AVSPN'97*, 1997.

[50] M. Krunz. Bandwidth Allocation Strategies Transporting Variable-Bit-Rate Video Traffic. *IEEE Communications Magazine*, 37(1):40–46, January 1999.

[51] V. P. Kumar, T. V. Lakshman, and D. Stiliadis. Beyond Best Effort: Routers Architectures for the Differentiated Services of Tomorrow's Internet. *IEEE Communications Magazine*, 36(5):152–64, May 1998.

[52] J. Kurose. Open Issues and Challenges in Providing Quality of Service Guarantees in High-Speed Networks. *ACM Computer Communication Review*, 23(1):6–15, January 1993.

[53] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff. A Brief History of the Internet. http://www.isoc.org/internet/history/brief.html, August 2000.

[54] H. Liu, W. Ng, and E. Lim. Improving the Fairness of Timely Refresh of Web Views. In *7th International Conference on Database Systems for Advanced Applications*, April 2001.

[55] J. K. Mackie-Mason and H. R. Varian. Pricing Congestible Network Resources. *IEEE Journal on Selected Areas in Communications*, 13(7):1141–1149, September 1995.

[56] R. Metcalfe and D. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks. *Communication of the ACM*, 19(7):395–404, July 1976.

[57] K. Nahrstedt. End-to-End QoS Guarantees in Networked Multimedia Systems. *ACM Computing Surveys*, 27(4):613–616, Dec. 1995.

[58] K. Nahrstedt and J. M. Smith. The QOS Broker. *IEEE Multimedia*, 2(1):53–67, Spring 1995.

[59] K. Nahrstedt and J. M. Smith. Design, Implementation, and Experiences of the OMEGA End-Point Architecture. *IEEE Journal on Selected Areas in Communications*, 14(7):1263–1279, September 1996.

[60] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, December 1998.

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

[62] J.-P. Nussbaumer, B. V. Patel, F. Schaffa, and J. P. G. Sterbenz. Networking Requirements for Interactive Video on Demand. *IEEE Journal of Selected Areas in Communications*, 13(5):779–787, 1995.

[63] J. F. Patterson and C. Egido. Three Keys to the Broadband Future: A View of Applications. *IEEE Network Magazine*, 4(2):41–47, March 1990.

[64] C. A. Polyzois, K. H. Purdy, P. F. Yang, D. Shrader, H. Sinnreich, F. Mnard, and H. Schulzrinne. From POTS to PANS - A Commentary on the Evolution to Internet Telephony. *IEEE Network*, 13(3):58–64, May/June 1999.

[65] H. G. Schulzrinne. Conferencing and Collaborative Computing. In *Proceedings of the Dagstuhl Seminar on Fundamentals and Perspectives of Multimedia Systems*, 1994.

[66] S. Shenker. Fundamental Design Issues for the Future Internet. *IEEE Journal on Selected Areas in Communications*, 13(7):1176–1188, September 1995.

[67] S. Shenker. Service Models and Pricing Policies for an Integrated Services Internet. In B. Kahin and J. Keller, editors, *Public Access to the Internet*, pages 315–337. MIT Press, 1995.

[68] S. Shenker, C. Partridge, and R. Guerin. Specification of Guaranteed Quality of Service. RFC 2212.

[69] J. Shoch and J. Hupp. Measured Performance of an Ethernet Local Network. *Communications of the ACM*, 23(12):711–721, December 1980.

[70] M. Shreedhar and G. Varghese. Efficient Fair Queueing Using Deficit Round Robin. *IEEE/ACM Transactions on Networking*, 4(3):375–385, June 1996.

[71] S. Sørensen. CLOWN: An Object Oriented Simulation Environment. In *Proceedings Intern. AMSE Conference Modelling, Simulation & Control*, pages 2018–2024, October 1992. vol. 3.

[72] S. Sørensen and M. G. W. Jones. The CLOWN Network Simulator. In J. Hillston, P. King, and R. Pooley, editors, *Computer and Telecommunications Performance Engineering*, pages 123–130. Springer-Verlag, 1991.

[73] R. Steinmetz. Human Perception of Jitter and Media Synchronization. *IEEE Journal on Selected Areas in Communications*, 14(1):61–72, January 1996.

[74] C. Sunshine. Interconnection of Computer Networks. *Computer Networks*, 1:175–195, 1977.

[75] D. Towsley. Providing Quality of Service in Packet Switched Networks. In *Performance Evaluation of Computer and Communications Systems*, pages 560–586. Springer Verlag, 1993.

[76] J. S. Turner. New Directions in Communications (or Which Way to the Information Age?). *IEEE Communication Magazine*, 25(10):8–15, October 1986.

[77] A. Vogel, B. Kerherve, G. Bochmann, and J. Gecsei. Distributed Multimedia and QoS: A Survey. *IEEE Multimedia*, 2(2):10–19, Summer 1995.

[78] R. Wade, M. Kara, and P. M. Dew. Modelling and Simulation of STTP, a Proactive Transport Protocol. In C. Tham and L. Zhang, editors, *Proceedings of the IEEE International Conference on Networks (ICON'2000)*, pages 485–486, September 2000.

[79] Z. Wang and J. Crowcroft. Analysis of Burstiness and Jitter in Multimedia Communications. *GLOBECOM'93 Proceedings*, 3:1496–1500, 1993.

[80] D. J. Wright and M. To. Telecommunication Applications of the 1990s and their Transport Requirements. *IEEE Network Magazine*, pages 34–40, March 1990.

[81] J. Wroclawski. Specification of the Controlled-Load Network Element Service. RFC 2211, September 1997.

[82] R. H. Zakon. Hobbes' Internet Timeline. RFC 2235, November 1997.

[83] H. Zhang. Providing End-to-End Performance Guarantees Using Non-Work-Conserving Disciplines. *Computer Communications : Special Issue on System Support for Multimedia Computing*, 18(10), October 1995.

[84] H. Zhang. Sevice Disciplines for Guaranteed Performance Service in Packet-Switching Networks. *Proceedings of the IEEE*, 83(10):1374–1396, October 1995.

[85] H. Zhang and D. Ferrari. Improving Utilization for Deterministic Service In Multimedia Communication. In *International Conference on Multimedia Computing and Systems*, pages 295–304, 1994.

[86] H. Zhang and S. Keshav. Comparison of Rate-Based Service Disciplines. In *ACM SIGCOMM'91*, pages 113–121, August 1991.