THE PROVISION OF A LOW DELAY SERVICE WITHIN THE BEST-EFFORT INTERNET

THÈSE N^{O} 2516 (2002)

PRÉSENTÉE Á LA FACULTÉ IC SECTION SYSTÈMES DE COMMUNICATION

ÉCOLE POLYTECHNIQUE FÉDERALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES TECHNIQUES

PAR

Paul HURLEY

B.Sc. in Mathematical and Computer Science, University College, Galway, Irlande
et de nationalité irlandaise

Lausanne, EPFL
2002

For Sonja, Ted and Vi.

Acknowledgements

Firstly, I would like to thank my supervisor Prof. Jean-Yves Le Boudec for his guidance, perseverance and inspiration. He provided the basis on which to develop my own ideas and was always open for advice.

Without my eclectic and sociable colleagues, the lab would have been a dreary place. My gratitude goes out to those who had the dubious pleasure of sharing an office with me, namely Kestutis Patiejunas, Dr. Carmen Mas, Dr. Eric Gauthier and Gil Regev. I thank equally my other colleagues, past and present, from the "JYLB" group: Bozidar Radunovic, Catherine Boutremans, Dr. Chadi Barakat, Felix Farkas, Hans Einsiedler, Heiko Boch, Dr. Leena Chandran-Wadia, Dr. Ljubica Blazevic, Milan Vojnovic, Dr. Olivier Crochat, Olivier Dousse, Dr. Sam Manthorpe, Dr. Silvia Giordano, Thorsten Kurz and Dr. Werner Almesberger. It was a delight to share all the meetings, coffee breaks, lunches and the odd beer. Dr. D.C. Shark was also a source of inspiration at times.

A special note of thanks goes to Prof. Patrick Thiran for his invaluable discussions and help both at EPFL and Sprint. I would additionally like to thank Dr. Mourad Kara for his important contributions to the ABE project.

Appreciation is due for the generosity of Sprint ATL who provided funding and the opportunity of an internship. I had a great time there, and am more than grateful to Dr. Christophe Diot for his support and encouragement. I also thank Dr. Gianluca Iannaccone for proving to be the best cubicle-mate one can hope for. I gratefully acknowledge our initial funding partners, Swisscom and the European Space Agency, and in particular my contacts Pascal Meuret and Roberto Donadio.

I appreciate the great support provided by the secretaries Danielle Alvarez, Holly Cogliati and Angela Devenoge, and that of the system managers Jean-Pierre Dupertuis and Marc-André Lüthi.

I'd like thank all my friends, especially those who provided advice and went running with me, including Dr. Holger Reimerdes, Dr. Monique Calisti, Stefan Nahrgang, Dr. Stefan Ruile, and Dr. Steven Wilmott.

I am indebited to my dad Ted for ploughing through drafts of my dissertation and spotting the plethora of well hidden typos and badly phrased text. I thank my parents Ted and Vi for their support and love; for instilling a curious nature, giving the courage to persue a PhD, and far more besides.

Finally, I cannot thank Sonja enough for her love, encouragement, for the proverbial kick at approriate moments and for tolerating my whinges and absence.

Without these people you would not be reading this dissertation today.

Abstract

In this work, we define a novel Internet service, called ABE (Alternative Best-Effort), which allows interactive multimedia applications to receive low queueing delay within the existing best-effort Internet. We then develop and analyse a number of innovative scheduling algorithms that can implement ABE in a router.

ABE differs from other services that provide low queueing delay in that it does not rely on any reservation and requires no charging of those that avail of the low delay service. We retain the best-effort context by protecting traffic that does not require lower delay for individual packets so that the existence of ABE is transparent to them, namely their performance is at least as good as it would be in the current best-effort Internet.

The primary scheduler to implement ABE we developed is called DSD (Duplicate Scheduling with Deadlines). It guarantees that traffic marked as low-delay will spend no longer in the system of each router than some operator specified value. The performance of other traffic is protected by the use of a virtual queue, a packet deadline decision based mechanism and a controller. We prove important properties of DSD and show, by the tools of queueing analysis and simulation, that it succeeds in its goal: protecting the performance of other traffic and the best possible performance for low-delay traffic subject to the constraints of not exceeding the maximum permitted delay

The final part of this work addresses the question: *what is the resultant long-term distribution of rates amongst flows, with equal round-trip times in a general network, that update their rates by the additive increase/multiplicative decrease algorithm?* Contrary to what was previously believed, we find that the rate allocation is not proportionally fair, but is more closely represented by an allocation we derive.

Version abrégée

Dans ce travail, nous définissons un nouveau service pour l'Internet. Il permet aux applications multimedias interactives de bénéficier d'un faible délai d'attente dans le contexte actuel "best-effort" (meilleur service possible) de l'Internet. Nous développons et analysons plusieurs algorithmes d'ordonnancement et de service des paquets, capable d'implémenter ABE dans un routeur.

Contrairement aux autres services offrant un faible délai d'attente, ABE ne requiert ni réservation, ni facturation de frais supplémentaires à ses utilisateurs. Le contexte du best-effort est maintenu en protégeant, au niveau paquet, le trafic qui ne requiert pas un faible delai d'attente. La performance de cette classe de traffic est au moins aussi bonne avec ABE que dans l'Internet actuel. Ainsi, tout en procurant aux utilisateurs qui le souhaitent un faible délai d'attente, ABE est transparent aux autres utilisateurs qui souhaitent conserver le service actuel.

Le principal algorithme d'ordonnancement et service réalisant ABE que nous avons développé est appelé DSD (Duplicate Scheduling with Deadlines). Il garantit que la classe de trafic marquée pour le service à bas délai ne séjournera pas dans un routeur plus longtemps qu'une valeur spécifiée par le vendeur. Les performances de l'autre classe de trafic sont préservées à un niveau au moins aussi élevé que celui qui aurait été atteint si ABE n'était pas implémenté dans ce routeur, grâce à l'utilisation d'une file virtuelle, d'un serveur basant ses décisions sur les échéances auxquelles les paquets doivent avoir quitté le routeur, et d'un régulateur. Nous établissons des propriétés importantes du DSD. En utilisant des outils de théorie des files d'attente et de simulation, nous montrons que cet algorithme remplit son objectif: la meilleure perfomance possible pour le trafic à faible delai est atteinte, tout en garantissant un délai maximum fixé pour cette classe de trafic et un service au moins aussi bon que si ABE n'était pas déployé pour l'autre classe de trafic.

La dernire partie de ce travail répond à la question suivante: *dans un réseau dont les flots mettent à jour leurs taux de transmission par un algorithme de croissance additive/décroissance multiplicative, quelle est, à long terme, la distribution des taux alloués à chacun, dans le cas où leurs temps d'aller-retour dans le réseau sont identiques ?* Contrairement à l'opinion admise jusqu'à présent, nous trouvons que cette distribution n'est pas proportionellement équitable, mais se rapproche davantage d'un nouveau type d'allocation intermédiaire entre équité proportionnelle et équité max-min.

Contents

Chapter 1

Introduction

1.1 Goals and achievements

The primary goals of this dissertation are:

- the definition a low queueing delay service which enhances the standard best-effort Internet service;
- the derivation of router algorithms for the provision of this service;
- investigation of fairness objectives.

The primary achievements are:

- formal definition of a low queueing delay service for best-effort IP networks, the ABE service;
- creation of a new scheduling algorithm, DSD, based on a new idea of combining a virtual queue and service deadlines to packets;
- derivation of important properties of DSD, and analyse it using a queueing model and through in depth simulation testing;
- derivation, by modelling and analysis, of the fairness in the distribution of rates amongst sources when these are of the type additive increase/multiplicative decrease;
- design and simulation testing of a first generation ABE implementation, EDF.

1.2 Dissertation overview

1.2.1 Motivation

In best-effort IP networks, packets are forwarded independently of the nature of the different service requirements of the traffic. Many applications are of file transfer type and seek to minimise the overall transfer time (from source to destination) of a given amount of data. Others, such as interactive audio, are delay-sensitive, with real-time deadlines. Yet despite these varying requirements, both types of application are not treated differently by the network, in terms neither of the delay their individual packets receive nor in the chances that one of their packets may be dropped.

The performance parameters that affect the utility a delay-sensitive application receives from the network can be thought of as being at least two dimensional, as illustrated in Figure 1.1. In this case, an audio source receives very good delay performance when its packets take less than 150ms to reach their destination, while the quality starts to deteriorate more rapidly after this. A source receives higher benefit the higher its rate of transfer, but after reaching a sufficiently high rate any further increase provides negligible benefit.

Currently, even when a delay-sensitive application is allocated (by the network) *more than enough throughput*, it can be handicapped by the end-to-end delay. Moreover, it can be common to dimension large buffers in routers in order to keep packet loss low in times of congestion. While this may suit file transfer applications, it can result in unacceptable delays for more delay-sensitive applications.

Facilitating delay-sensitive flows is traditionally approached by providing a form of reservation with a level of guarantee on loss and delay. They then have priority over regular best-effort traffic. By treating this traffic quantifiably better than regular best-effort traffic, it can become necessary to charge for this service.

We approach the problem by a service which provide delay-sensitive flows with sufficiently low queueing delay within the *existing best-effort Internet*. This is motivated by the observation that there now exist interactive multimedia applications (such as those whose utility resembles that of Figure 1.1) that adapt their sending rate and encoding parameters in order to perform well across a wide range of loss and throughput conditions, but for which delay often remains the major impediment. It is desirable that such a service should:

- retain the operational simplicity of the current best-effort network;
- not have to police how much traffic uses the low-delay capability;

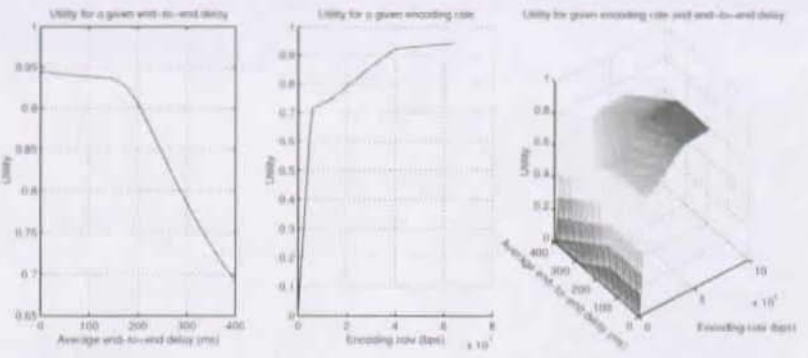


Figure 1.1: The utility an audio source receives for given performance under the E-model.

- be possible to introduce gradually;
- not require charging more to use the low-delay service.

1.2.2 The ABE service

One of the main achievements of this dissertation is the creation of ABE (Alternative Best-Effort), a novel service for IP networks which offers applications the choice between receiving a lower end-to-end delay and receiving more overall throughput. Best effort IP packets are either low delay packets, called *green* packets, or other best effort packets, called *blue* packets. The choice of the terms blue and green, two neutral colours, is to indicate that neither of the two has priority over the other. In order to distinguish one from the other, it may help the reader to note that green, which indicates low queueing delay, is the colour of the traffic light signal for *go*.

Green packets are guaranteed a low bounded delay at every hop. For the service to remain best-effort with no overall advantage to either traffic type, sources which choose not to avail of the lower delay must receive at least as good a service as they would as if all packets had been blue. The introduction of ABE must be transparent to them.

As such, ABE requires that *green does not hurt blue*. If some source decides to mark some of its packets green rather than blue, then the quality of service received by sources that mark all their packets blue must remain the same or become better. As a consequence, green packets are more likely to be dropped during periods of congestion than blue ones. All ABE packets belong to one single best effort class. If the total load is high, then every source may receive a small share of the bandwidth.

However, entirely blue sources would experience more throughput than entirely green sources sharing the same network resources.

Some applications will still require priority based schemes such as differentiated services. Our goal is not to replace these. Rather, we address a different market and offer no throughput guarantees. It offers a new degree of freedom to best-effort services, enabling a moderately loaded network to offer low delay to some applications (typically, adaptive multimedia applications), as long as such applications are satisfied with the throughput they receive. However, a highly loaded network offering ABE will give little throughput to all best effort flows, no matter whether green or blue.

The green does not hurt blue requirement divides into two types of transparency: *local* and *throughput*. Local transparency is satisfied if, for each blue packet in the ABE scenario, the delay is not larger than it would have been in flat best-effort and it is dropped only if it would have been in the flat best-effort scenario. Throughput transparency means that a blue flow receives no less throughput than if the network was flat best-effort, namely, where a node would treat all ABE packets as one single best effort class.

When rate adaptation is performed such that the application is TCP friendly, namely it does not receive more throughput than a TCP flow would, ABE provides low-delay at the expense of possibly less throughput. However, many multimedia flows are not TCP friendly and the stronger condition of local transparency enables protection for blue in the presence of non-adaptive traffic.

The first ABE implementation we created was ABE/EDF, which ensures throughput transparency. It, however, relies on flows being TCP friendly. The latest implementation, DSD, implements both local and throughput transparency, enabling the support of TCP, TCP friendly, and flows that do not adapt.

It is noteworthy that local transparency does not imply throughput transparency. Imagine some sources sending green that are rate-adaptive (TCP friendly) and greedy (take as much bandwidth as permitted). It is quite possible that, by becoming green, a TCP friendly source would achieve a higher data rate, due to the reduction in round-trip time. This is a result of the known bias in the TCP congestion control algorithm which favours flows with shorter round-trip times. In addition, unlike local transparency, throughput transparency seems impossible to implement exactly. It requires knowledge of the round-trip time for every flow, which is not practical and the rate adaptation algorithm implemented by a source may significantly deviate from strict TCP friendliness.

A router implementation of ABE is required to implement differential scheduling, and ensure those that do not choose lower delay, do not suffer in terms of overall throughput.

1.2.3 DSD implementation of ABE

The present state of the art ABE implementation we developed is Duplicate Scheduling with Deadlines (DSD). It provides the service through a novel use of packet deadlines and a virtual queue, and is designed to provide green with the best possible service while still ensuring protection for blue. DSD minimises the number of green losses subject to the constraints that green packets spends no longer in the system than a fixed time d, local transparency holds, the scheduling is work conserving and packet order is preserved between colour (i.e. no later blue packet is served before an earlier one and similarly for green).

DSD sends a duplicate of each packet arrival to a virtual queue. A blue packet is only dropped if its duplicate was in the virtual queue. Otherwise it receives a deadline equal to the time its duplicate finishes service in the virtual queue. A green packet is accepted if it passes what is called the *green acceptance test* and is then assigned a deadline equal to its arrival time plus the maximum time it can spend in the system d. We subsequently reduce the complexity of this test by the use of what we call a *holding queue*.

Green and blue packets are queued separately, and the deadlines of the packets at the head of blue and green queues are used to determine which one is to be served next. Blue packets are served at the latest their deadline permits and green served in the meantime.

Throughput transparency is provided by two mechanisms. Firstly, we use a controller, which acts on the green bias parameter g to control the service received by green packets. Often both the blue and green packets at the head of their respective queues can wait, namely serving the other packet first still enables it to be served before its deadline. To decide which one is served first, the parameter g used, which is the probability of serving the green packet first. The delay and loss ratio are monitored, and the controller adjusts g to ensure throughput transparency is maintained. Secondly, we force green packets to be dropped if their duplicate was dropped in the virtual queue. This is done to avoid situations when green traffic has a lower loss ratio than blue traffic over a period of time.

We also developed serial DSD, a scheduler that is functionally equivalent to DSD but combines blue and green in the same queue and makes most decisions based on deadlines at packet arrival rather than at service. We then describe algorithms, which are extensions of serial DSD, that reduce the delay differential between blue and green traffic without increasing the green loss rate and while adhering to the maximum green delay d.

A Markov chain description of DSD was created and used to analyse the properties of the scheduler in steady-state under the simple case of Poisson arrivals and exponential packet sizes. This Markov chain description can be used as a basis for further queueing analysis of DSD.

1.2.4 EDF implementation of ABE

ABE/EDF is comprised of a Packet Admission Controller (PAC) and a scheduler. The PAC manages the queue by dropping packets whenever necessary or appropriate. Acceptance is biased in favour of blue packets, ensuring sufficient green packets are dropped to protect blue. Green packets that would otherwise experience a queueing delay greater than the specified guarantee d are dropped. It controls the dropping of green packets to ensure throughput transparency, while trying to keep these green losses to a minimum. A blue packet is dropped according to the queue management Random Early Detection (RED). A green packet is dropped if it cannot be served within the maximum queueing delay tolerated d or if it fails the RED acceptance test with a higher dropping probability determined by a specified value which is called the *drop bias*. This drop bias provides a drop disadvantage to green packets.

The scheduling is a form of Earliest Deadline First (EDF), where each packet is assigned a finishing service time deadline, and the packet currently having the lowest value is served first. Arriving green packets are assigned a finishing service time deadline equal to its arrival time. A blue packet is assigned a time equal to its arrival time plus the value of the *offset bound*, the goal of the offset bound being to limit the delay penalty imposed on blue. To preserve throughput transparency, we compensate for the delay penalty imposed on blues by artificially reducing their drop rate. This is done by adjusting the value of the drop bias. Minimising the green drop ratio is performed by means of a control loop which adjusts the offset bound and the drop bias.

1.2.5 Analysis of TCP fairness

As a separate part of this thesis, we analyse the distribution of the rates amongst flows when they are TCP or TCP friendly. We show that TCP compliant sources with equal round trip times competing for bandwidth do not, as was previously thought, end up with a distribution of rates in accordance with proportional fairness. Proportional fairness is a form of fairness which distributes bandwidth with a bias in favour of flows using a smaller number of hops; this is in contrast with max-min fairness, which gives absolute priority to small flows.

We show, on the other hand, that when feedback is rate dependent and negative feedback rare, the distribution agrees with what we define as F_A-fairness. This is confirmed by the fact that our results maintain consistency with the standard TCP throughput as a function of loss formula. It is known that TCP gives less throughput to connections with longer round trip times. Based on our analysis there are two possible reasons. F_A-fairness provides less to connections that use several hops or the fact that TCP maintains a sending *window* rather than a *sending* rate.

Modelling this mechanism is complex because it contains both a random feedback (under the form of packet loss) and a random delay (the round trip time, including time for destinations to give feedback). In our work, we consider all round trip times to be constant and equal. The method of the Ordinary Differential Equation (ODE) is used, which gives some insight into the convergence of the system. The result of the method is that the stochastic system converges, in some sense, towards an attractor of the ODE for which we identify a Lyapunov function. We find that for the case of small increments and constant round trip times, and in the regime of rare negative feedback, the proportional fairness result can only very approximately reflect the real rate allocation when we assume that the feedback received by sources is independent of their sending rates. In the case where sources receive feedback proportionally to their sending rates, and for sources with identical round trip times, this is no longer true and the fairness provided is different. We also show, by simulation on some examples, that even for larger increments, the average rate convergence is in agreement with our results.

1.3 Dissertation outline

The thesis is organised as follows. We define the low queueing delay service, ABE, in Chapter 2, outlining its motivations, and demonstrating the usefulness of the service.

Chapter 3 presents the DSD implementation of ABE. It motivates and describes its design, its properties, and the control loop used. Properties of the implementation are described and proved. In Chapter 4, we describe the serial DSD implementation of ABE.

Chapter 5 presents a Markov chain description of DSD. We then analyse it when the arrival process is Poisson and the packet size process is exponential. Chapter 6 presents results of tests, by simulation of DSD by simulation. Chapter 7 describes the EDF implementation of ABE and simulation tests of it.

The fairness of additive increase and multiplication decrease is examined in Chapter 7. Finally, in Chapter 8 we present conclusions and put forward possible further work.

Chapter 2

The ABE Service

In this chapter, we

- introduce and define the ABE service;
- describe its objectives and advantages;
- define and explain the important property, *green does not hurt blue*;
- demonstrate the application and usefulness of ABE in the context of best-effort services;
- describe other services that offer low-delay and how they differ from ABE.

2.1 Introduction and motivation

ABE (Alternative Best-Effort) is an enhancement to the IP best-effort service whose goal is to

- provide a low queueing delay service and
- operate in best-effort mode.

The first requirement is for applications with stringent real-time constraints, such as interactive audio. The second requirement is to retain the operational simplicity of the original Internet, with no requirement to regulate how much of the traffic can use the low delay capability.

2.1.1 Adaptive applications

ABE is designed primarily to support, in a best-effort service environment, multimedia applications who adapt their rate in order to cope with the current network conditions. They change their rate by adjusting the level of compression, altering the amount of error-correction to protect against packet losses, and dimensioning the play-out buffer to mitigate the effects of delay jitter. The sending rate is adjusted based on the level of congestion.

The feasibility of implementing adaptive multimedia applications which perform across a wide range of network conditions is now established [2, 7]. However, delay remains the major impediment for interactive applications in many circumstances [19].

The key idea of ABE is to provide low-delay at the expense of sometimes less throughput, a concept which is essential to ensure ABE does not require usage control.

2.2 Definition of ABE

ABE is defined as follows:

1. ABE packets are marked either green or blue.

2. Green packets receive a low, bounded delay at every hop.

3. *Green does not hurt blue:* If some source decides to mark some of its packets green rather than blue, then the quality of the service (delay and throughput) received by sources that mark all their packets blue either remains the same or improves. This definition is expounded on in Section 2.4 on page 13.

4. All ABE packets belong to one single best-effort class. If the total load is high, then every source may receive little throughput. However, entirely blue sources would experience more throughput than entirely green sources sharing the same network resources.

The terms blue and green, two primary colours of equal value, is to indicate that neither has priority over the other. Low queueing delay and green can be compared to the indication that a green traffic light means to go.

2.3 Discussion

ABE remains a best-effort service with no reservation assumed and no necessity to rely on per-flow information. As such, flat-rate based pricing may be retained.

The incentive to choose blue or green is based on the nature of the particular traffic with overall benefit for both traffic types. During a period when there is either no blue or green traffic, the other colour can make use of the whole bandwidth. The network-level quality of service (packet loss and queueing delay) received by blue or green cannot be classified as being better than the other. The appropriate matching between the service received and the application nature is a major advantage of this system. ABE also avoids making an unsatisfactory trade-off between the different buffer size requirements of real and non-real time traffic.

In essence, ABE can be thought of as allowing an application to trade delay for loss or less throughput, by marking some packets green. A key requirement of the service is green does not hurt blue, namely, if some sources send green rather than blue packets, there should be no negative impact on the throughput of those sources which remain blue. In particular, an entirely blue flow must receive as much average throughput as it would in a flat best-effort network, i.e. if all packets were blue. The requirement derives from the objective that the colour chosen by an application need not be policed. Indeed, if green does not hurt blue is enforced, then an application which decides to mark some packets green must do so because it values the low delay more than a potential increase in loss (or decrease in throughput). Otherwise, it would mark its packets blue.

In all cases, there is no penalty for other applications which might choose to mark all their packets blue. This requirement also plays a role in interworking and migration (Section 2.6 on page 16). As discussed in Section 2.3.3 on page 12, ABE supports traffic that may be solely TCP friendly traffic or non-TCP friendly, or a mixture of the two. Thus, it is not required to police the colour chosen by applications.

As is the case using current best-effort, a highly loaded network offering ABE will give little throughput to all best-effort flows, no matter whether green or blue. However, ABE enables a moderately loaded network to offer low delay to some applications, typically adaptive multimedia applications, as long as such applications have sufficient throughput to successfully operate.

2.3.1 Benefit for all

There is *benefit for all.* The addition of ABE is advantageous for both real-time and bulk-data transfer traffic. The decision of some flows to become green in order to have low queueing delay can only benefit those that remain blue. Delay sensitive traffic, such as voice, receives a low bounded delay, possibly at the expense of reduced throughput. This results in superior voice quality over moderately loaded network paths. If the path becomes highly loaded, such a source can always revert to sending blue packets only. Non-delay-sensitive traffic, such as image or text transfer, receives a low number of losses. If a source which was previously blue now decides to mark its packets green, it will receive lower queueing delay, but this will never be at the expense of other sources which continue to send blue packets.

2.3.2 Choice of blue or green

Green packets would usually be interactive traffic where packet transfer from end to end must be short and delay variation low. Examples of green traffic include Internet Telephony and videoconferencing traffic, where if the data does not reach the receiving application within a certain time it may well be too late to be useful to it.

Blue packets are typically non-interactive traffic whose end to end delay can be variable and the goal is minimisation of overall transfer time. Examples of blue traffic could include data traffic (e.g. TCP traffic) and delay adaptive stream-like applications (playback audio and video applications).

2.3.3 TCP friendliness

For reasons of fairness and to avoid congestion collapse, there is a movement to mandate that non-TCP sources be TCP friendly [14], namely, the source should not receive more throughput than a TCP flow would for the same conditions of loss.. However, it is still the case that many multimedia flows are *not* TCP friendly.

The requirement that green does not hurt blue applies thus even if green traffic originates from non-TCP friendly sources. Note that non-TCP friendly sources may, in some cases, severely hurt TCP friendly sources, and this is true with or without ABE. The requirement simply means that sources that send green traffic do not make the situation worse.

2.3.4 Network feedback

The amount of negative feedback (e.g. packet losses) received by green traffic is greater than that received by blue traffic. The admitted green packets are given a shorter queueing delay. In the Internet today, feedback is based on packet drop while in the future, binary feedback based on Explicit Congestion Notification (ECN) [11] may become widespread. ECN provides congestion feedback to the source by marking a bit in the packet header, thus enabling it to adjust to feedback without necessarily dropping its packets.

We focus on packet loss as the method of providing negative feedback. Nevertheless, ABE remains valid in the case of systems using some form of ECN (and indeed

ABE would work even better with it). If ECN is used, green packets are more likely than blue packets to be marked with a congestion bit.

2.3.5 Colour mixing

An ABE aware source would probably use a colour mixing strategy, where it would send some green packets and some blue. This would for example be used by the colour adaptation algorithm for monitoring purposes. This is perfectly permissible and considered normal practice; in fact, apart from possibly policing TCP Friendliness, the network supporting ABE does not need to analyse individual flows. Source strategies would typically be performed at the application level as expected by Application Layer Framing (ALF) [3].

We focus on the simpler case where a traffic source chooses to be either green or blue.

2.4 Green does not hurt blue

The service requirement that green does not hurt blue, is now defined more accurately. Intuitively, it can be expressed by considering two scenarios:

1. all sources are blue;

2. some sources decide to mark some packets green.

The quality of service received by those packets which have remained blue in the second scenario should be as good as in the first one. Firstly, the delay for any blue packet must not be any larger. Secondly, a packet which is not dropped (or marked with a congestion notification) in the first scenario would not be dropped in the second scenario either. As a consequence, the throughput of an entirely blue flow would be at least as good in the second scenario (if we assume that flows are TCP friendly).

A problem with that simple, intuitive definition is that sources are assumed to be adaptive (TCP friendly), and thus in the second scenario, the packets sent by the sources will not be the same as in the first one. Furthermore, the behaviour of sources is dependent on their rate adaptation algorithm which, while being TCP friendly, may have different incarnations.

Thus, we divide the requirement in two; the first addressing the case of non-TCP friendly sources, and the second accounting for the rate adaption.

2.4.1 Local transparency

Definition 2.4.1 *(Local Transparency) Consider the scenario, flat best-effort, in which a node would forget colour and treat all ABE packets as one single best effort class. The node satisfies* **local transparency** *if, for each packet that is blue in the original (ABE) scenario:*

1. *the delay is not larger in the real, ABE scenario than in the flat best-effort scenario;*
2. *if the blue packet is not dropped (or marked with congestion notification) in the flat best-effort scenario, then it is not dropped either in the real, ABE scenario.*

It means that if some packets are marked green it does not hurt blue packets, assuming one can ignore the effects due to rate adaptation in the sources. It is a necessary requirement to ensure green does not hurt blue. However, it may not be sufficient, since the rate adaptation algorithm at the source might produce a higher rate when the end-to-end delay is smaller.

2.4.2 Throughput transparency

The average rate of a TCP flow, for a given loss ratio and round-trip time, can be estimated by application of one of the many versions of the TCP loss-throughput formula given in [25, 31, 10, 37, 4]. A source can be considered to be TCP friendly if it produces a data rate not exceeding θ where θ is given by

$$\theta(p, R) = \frac{s}{R\sqrt{\frac{2p}{3}} + 3t_{\text{I}}\sqrt{\frac{3p}{8}}p(1 + 32p^2)} \tag{2.1}$$

and R is the round-trip time, p the rate of loss events, t_{I} the TCP retransmit time (roughly speaking, proportional to the round-trip time), and s is the packet size [37].

Thus, it is quite possible that, by becoming green, a TCP friendly source would be allowed a higher data rate, due to the reduction in round-trip time. Such a source would generate more packets than if it were blue, and there is the risk that, in some cases, it would hurt blue packets. This leads to the second part of the requirement:

Definition 2.4.2 *(Throughput Transparency) Assume that sources employ a rate adaptation algorithm which conforms to a loss-throughput formula such as Equation (2.1). An ABE node provides blue with* ***throughput transparency*** *if it ensures that an entirely green flow gets a lesser or equal throughput than if it were blue.*

Unlike local transparency, throughput transparency seems impossible to implement exactly. On the one hand, it requires knowing the round-trip time for every flow which is not practical. On the other hand, the rate adaptation algorithm implemented by a source may significantly deviate from a straight application of Equation (2.1). Indeed, the dependency of rate on round-trip time in Equation (2.1) is not necessarily a desirable feature of a rate adaptation algorithm. It should not be confused with the fact that a source using many hops should receive less throughput, which is desirable and caused by a having a higher loss ratio [43].

As we will later describe in Section 3.4 on page 39, local transparency may hold while, over a time window of many round-trip times, green traffic has a *lower* loss ratio than blue traffic. This situation needs to be avoided since adaptive flows of short duration may see a lower loss probability while green, and thus receive less throughput if blue. An ABE implementation can ensure this is not possible, such as by how it is done in Section 3.4.1 on page 40.

Fixes have been suggested to rate adaptation algorithms that would remove the dependency of rate on round-trip time [18]. If such fixes were to become widespread, and a method employed to ensure the green loss ratio is not lower than the blue loss ratio for certain period of time, then throughput transparency would be an automatic consequence of local transparency. However, the current definition of TCP friendliness does imply a dependency of rate on round-trip time and in this context, it is necessary to compensate for the delay decrease obtained by green traffic.

2.5 Requirements for a router implementation

A router implementing ABE must:

1. Provide low, bounded delay to green packets; the delay bound is fixed by network management.
2. Provide local transparency (Definition 2.4.1).
3. Enforce throughput transparency (Definition 2.4.2).
4. Keep green packet loss as low as possible, while adhering to the above requirements.

The first three requirements directly derive from the previous discussion. The fourth requirement is because an implementation should try to make using green as attractive as possible.

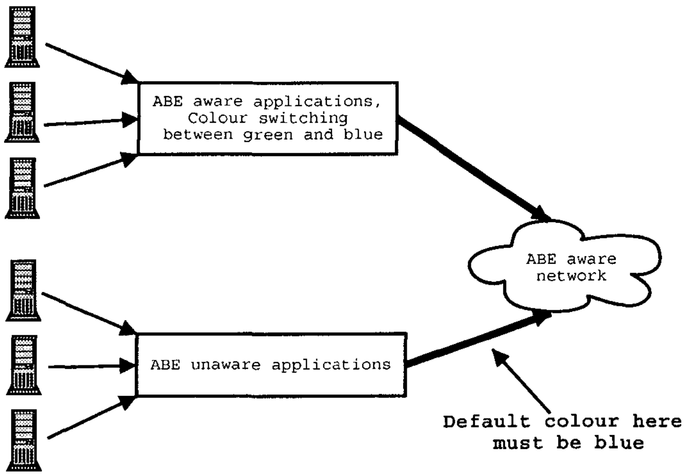

ABE aware applications, Colour switching between green and blue
ABE aware network
ABE unaware applications
Default colour here must be blue

Figure 2.1: Gradual deployment of ABE

In today's Internet, it is considered desirable to preserve packet ordering, though this is not always enforced. Similarly, an ABE node is expected to preserve packet order as much as possible. However, the delay preference given to green may result in a green packet overtaking a blue one.

2.6 Internetworking and migration

ABE could be used by an operator in two distinct ways; either as a separate service, or as a replacement to the flat (existing) best-effort IP service; we focus on the latter. Indeed, as mentioned earlier, the initial thinking behind ABE was to provide support for interactive, adaptive Internet applications, while retaining the simplicity of the original Internet service model.

An operator might introduce ABE and let customers and other carriers gradually move over to ABE without any specific change to charging or control policies. ABE may be employed on some links in the network only where needed. As such, an ABE aware source may use a concatenation of networks, some ABE, some flat best-effort, and sources that are oblivious to the existence of ABE must not suffer adversely due to the introduction of ABE.

Replacing flat best-effort by ABE requires a rule for assigning a colour to packets that do not have one (such packets come from a non-ABE source or network).

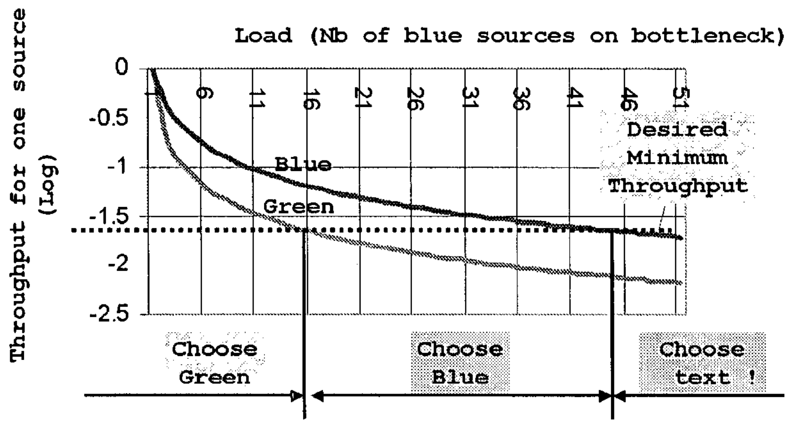


Figure 2.2: A possible strategy for a multimedia source using the ABE service.

Imagine, for example, the scenario as in Figure 2.1. The network has added ABE, but there are some sources (in the lower part of the diagram) which would benefit from being green in that they want low delay, but their applications do not exploit ABE. The default traffic colour is blue. Local Transparency ensures no increase in delay or loss for blue packets. Thus ABE unaware sources do not suffer as a result of the deployment of ABE.

We have mentioned earlier that an ABE aware source must probably implement a colour adaptation algorithm. Now, depending on traffic conditions, the ABE source might see small or large delays even for green traffic. This implies that the colour adaptation algorithm should not make any quantitative assumption about the value of end-to-end delay guaranteed for green traffic.

2.7 An example

A flow that receives sufficient throughput for application operation, while being green, albeit a smaller share than if it were blue, can operate with reduced end-to-end packet delays. This can be seen in a simple simulation shown in Figure 2.2. An interactive, adaptive audio source has a minimum rate it needs in order to operate. It competes with n background sources for one bottleneck. The source has the choice of marking packets blue or green. All other sources are blue.

Assume the source has, for a given loss pattern in the network, a required minimum rate R_0 in order to function properly. The graph shows the throughput the source would receive if blue or green for varying numbers of blue sources. The rate R_0 is given by the horizontal dashed line. Let us also assume that the source is able to forward-correct packet losses, as long as the minimum rate is achieved (see [2] for such an application example. Note that this would not be needed if ECN was used). The choice of whether to be green or blue is made by the audio application.

The quality the source obtains from the network conditions depends on its utility function $u(R, D)$, for a given throughput R and end-to-end network delay D. On this simplified example, we assume that the utility function for our source satisfies

- $u(R, D) = 0$ for $R < R_0$ and
- $u(R, D)$ is a decreasing function of D only for $R \geq R_0$.

In other words, once the minimum rate R_0 is achieved delay becomes the major impediment. Note that in general more complex utility functions of delay and rate will be used, such as the one shown in Figure 1.1.

While the minimum throughput is attained, the source is better off being green since its queueing delay will be much shorter. At the point the minimum rate is no longer attainable while green, the source must then become blue provided it can operate with highly variable end-to-end packet delays. The application cannot operate when the minimum throughput R_0 cannot be achieved even while blue. Thus, it must either cease sending for a while or as shown resort to just sending text as opposed to audio.

This example illustrates how ABE opens up a new region of operation for the best-effort network. In low load scenarios, a source may decide to obtain less throughput at the benefit of low delay. In a flat best-effort network, a network without the ABE service, there is no such option. Indeed, by refraining from sending at a higher rate, there is in general no impact on queueing delay, due to external sources.

Note that the region in which the source is currently operating must be detected *automatically* by the source itself, using a colour adaptation algorithm. This could, for example, take the form of sending probe packets of either colour in order to determine which region the source is currently operating in. Unlike the multimedia source above, a source using TCP is probably more interested in its throughput and should thus mark all its packets blue.

2.8 ABE is better than destination drop

Destination drop might appear to be an alternative to ABE that would require no support from the network. This would consist of the destination dropping all

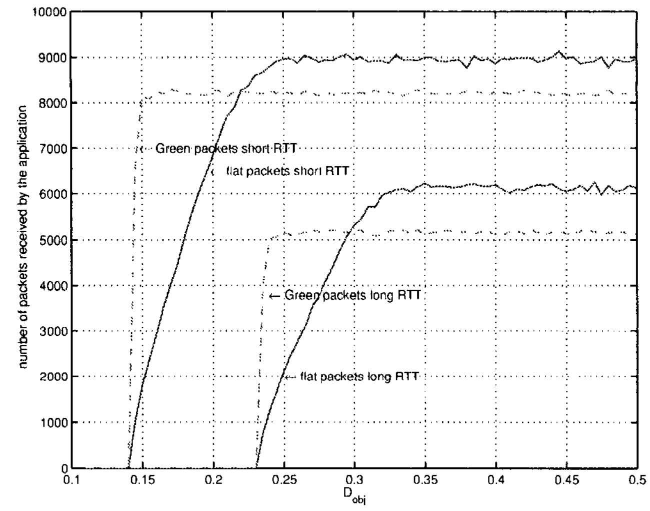


Figure 2.3: The amount of useful packets received by an application as a function of the end-to-end delay tolerance D_{obj}.

---

packets that arrive too late, after say some maximum tolerable end-to-end delay D_{obj}. However, as we now show, it wastes network resources, since packets are dropped after being carried by the network, and the overall performance of such a scheme can become very poor.

Consider the simulation results in Figure 2.3. The network consists of flows with long and short round-trip times which are TCP friendly, and each compete on the same best-effort network. The number of useful packets received by the application as a function of the tolerated deadline D_{obj} is plotted.

If D_{obj} is less than the end-to-end delay when there is no queueing, no useful packets can be received by the application. Then, the amount of useful packets received by the application is larger while green, up to point that the delay tolerance is sufficiently high that there is no need for a low delay service. The throughput reduction due to being green is less than the throughput reduction due to dropping late packets. When the delay objective D_{obj} is large, then of course no packet is dropped at the destination, and it is better to mark the packets blue.

There are thus explicit cases, using ABE, when it is better for this type of application to use green rather than blue packets. In addition, this behaviour frees up network resources, benefitting other users of the network. In a flat best-effort network all late packets are discarded at the destination, wasting network resources.

ABE results in more packet drops in the network but less at the destination, thereby resulting in a less congested network.

2.9 Other services

As the Internet evolves towards a global communication infrastructure, a number of propositions exist for providing QoS (Quality of Service) architectures. These aim to support more sophisticated services than those provided by flat best-effort services. Currently there are two broad families for QoS provision and both are based on some form of priority and service differentiation.

The first family of solutions, Integrated Services, (IntServ) uses reservations (admission control) and requires routers to manage per flow states and perform per flow operations. It also requires per flow accounting and charging. The second family of solutions, Differentiated services (DiffServ), is based on a coarser notion of QoS, focussing on aggregates of flows in the core routers and intending to differentiate between service classes rather than provide absolute per flow QoS measures.

Integrated services have been shown to exhibit much higher flexibility and assurance level than those provided by Differentiated services. However the main disadvantages of these services are that they are less scalable and robust than differentiated services. Hence, these latter services have been the focus of attention lately mainly because they move the complexity of QoS provision from the core to the edges of the network where it may be feasible to maintain a restricted amount of per-flow state. Often Integrated services are identified as being supported by network architectures with a lot of state (because of the per-flow management), while differentiated services are identified as underpinned by stateless network architectures.

An example of such an architecture is SCORE (Scalable Core) proposed by Stoica and Zhang [41] with the aim of providing guaranteed services without per-flow state management. They proposed the Dynamic packet State (DPS) technique to estimate the aggregate reservation rate and use that estimate to perform admission control. To achieve this, they perform Core-Stateless fair queueing (CSFQ) using DPS to encode dynamic per flow state in the context of approximating the Fair queueing algorithm. Nandagopal et al. [35] also proposed a core stateless QoS architecture (called Corelite) which offers per-hop per-class relative average delay differentiation and end-to-end delay adaptation.

There are several proposals for supporting QoS through differentiated services. Crowcroft [6] proposed a low delay service, analysed by May et al [32], coded with a single bit. Turning on this bit ensures that the packet receives serving priority while constrained to a smaller buffer size. Depending on the input traffic and the buffer sizes of both types of traffic, this typically would result in the low delay traffic also having more throughput. Similarly, Expedited Forwarding [20] (EF) aims to provide extremely low loss and low queueing delay guarantees. SIMA [22] offers applications the choice of a level (0-7) of how "real-time" its traffic is, with each level having relatively lower delay and loss ratio than the previous one.

Dovrolis et al [8] described a proportional differentiation model where the quality between classes of traffic is proportional and thus can be performed independently of the load within each class. Central to their work was the utilisation of two packet schedulers BPR (Backlog Proportional Rate) and WTP (Waiting-Time Priority) to approximate the behaviour of the proportional differentiation model. Moret and Fdida [34] also described a two-class proportional differentiation model called Proportional Queue Control Mechanism (PQCM). Both studies propose controlling the relative queueing delays between classes.

All of these proposals couple low delay with improved throughput, and are some form of priority. They can be used to support adaptive and non-adaptive interactive applications, provided that some form of admission control is performed. They can provide a premium service, at a price that has to be higher than the best effort service (otherwise all traffic would use the better service). In contrast, ABE green packets cannot be said to receive a better treatment than blue ones and ABE may be introduced as a replacement for the existing best effort service. On the other hand, ABE is not suited to support multimedia applications which require hard guarantees and cannot adapt.

Another differentiated service is Assured Forwarding (AF) [17]. It divides AF traffic into classes within each there are distinct levels of drop precedence and offers an assurance that IP packets are forwarded with high probability as long as the aggregate input traffic within a class does not exceed an agreed profile. The authors also suggest that an AF class could be used to implement a low delay service where low loss is not an objective, by allocating an AF class with a low buffer space (call it the low delay AF class). Such a service is in principle different from ABE, which views all blue and green packets as one class; the service received by green packets is dependent on the amount of green *and* blue traffic. In contrast, the performance of an AF low delay class is not expected to be affected by the amount of best effort traffic. In that sense, the low delay AF class is a differentiated service which requires differentiated charging, contrary to ABE. Hence, ABE can be viewed as being positioned between the flat best-effort service and AF.

Chapter 3

ABE Implementation: DSD

Duplicate Scheduling with Deadlines (DSD), is a novel scheduling algorithm to implement ABE. It optimises green traffic performance while satisfying the constraint that blue traffic must not be adversely affected.

In this chapter, we:

- show how a quick and dirty scheduler which satisfies the requirements of ABE can provide extremely poor performance to green (Section 3.1, page 24);
- motivate and describe the DSD scheduler (Section 3.2, page 25);
- define and prove properties of DSD (Section 3.3, page 32);
- discuss the provision of throughput transparency in DSD and describe two methods for its enforcement: an additional acceptance criterion for green and a control loop (Section 3.4, page 39);
- describe how DSD can be implemented in conjunction with RED (Random Early Detection) queue management (Section 3.5, page 43);
- describe a *holding queue* which is used to reduce the complexity of the green packet acceptance test in DSD.

In Chapter 4, we describe serial DSD, a scheduling algorithm that is functionally equivalent to DSD in that it provides the same output for the same input, but uses one queue to store both green and blue packets. In cases of potential ambiguity, we refer to the original DSD implementation, described in this chapter, as *vanilla* DSD.

In that chapter we also describe modifications to serial DSD called *green at the back* and *green at the back, blue pushed up*. The latter algorithm reduces the complexity of the scheduler, while both algorithms increase the average green delay without affecting the number of greens accepted.

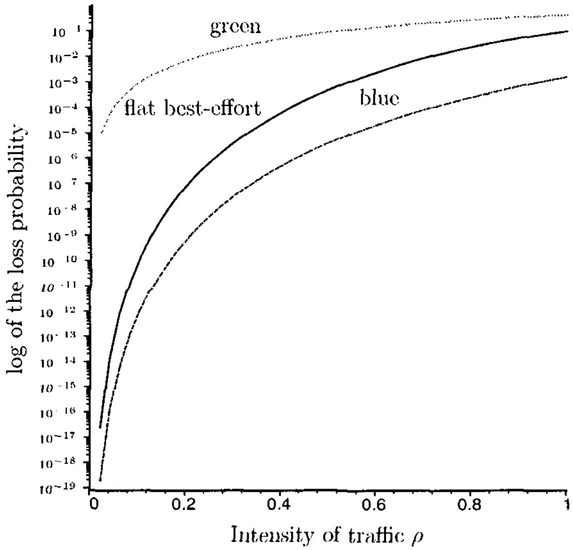


Figure 3.1: The loss probability (log scale) for green, blue and flat best-effort in the "rough and ready" notional implementation. The total intensity varies but the proportions of green and blue traffic remains the same (parameters were $\rho_b = 0.5\rho, K = 10, K_g = 3$).

In Chapter 5, we present a Markov chain description of DSD that lends itself to queueing analysis and verification, and we then analyse this model under the relatively simple but tractable case of Poisson distributed arrivals and service.

3.1 Rough and ready implementation?

An implementation that may spring to mind at first glance at the requirements is a first-come first served scheduling discipline with a threshold drop policy which filters green packets. In such a scheme, blue packets would be accepted whenever there is space in the buffer, while green packets would be accepted only if they can be served within some maximum delay d of arriving. However, it exhibits very poor performance, and most of the time there would be little or no incentive to be green.

Consider Figures 3.1 and 3.2. A description of the model and method used in the attainment of these results is deferred until Section 5.2 on page 62, given that a feeling for the potential futility of this scheme is all that is needed at this stage. The loss rates for green are magnitudes higher than for the blue. The green only see a significantly reduced delay over blue when the load is high, so they pay an

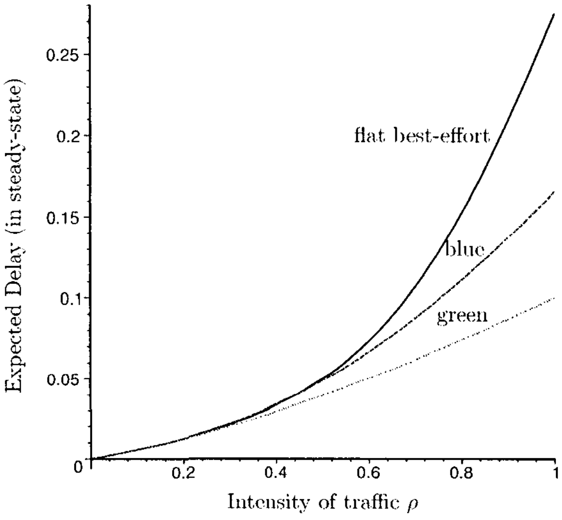


Figure 3.2: The long-term expected delay for green, blue and flat best-effort in the "rough and ready" notional implementation. The total intensity varies but the proportions of green and blue traffic remains the same (parameters were $\rho_b = 0.5\rho, K = 10, K_g = 3, \lambda_g = 20$).

exorbitantly high price in the tradeoff of lower delay for higher loss.[1] Note how the blue benefits from the greens' poor performance, having a lower loss-rate and delay than in flat best-effort.

Obviously such a scheme is not practical. The desire is to provide green with the best service possible while still ensuring green does not hurt blue. Any significant extra gain by blue packets is at the expense of green ones, so it should be kept to a minimum such that there is still an incentive to use green packets whenever appropriate. The way is now paved for presenting the motivation behind DSD.

3.2 DSD

3.2.1 Introduction

The overall design goal of DSD is to provide green with the best possible service while still ensuring protection for blue, namely that green does not hurt blue. It is a solution to the optimisation problem, which minimises the number of green losses

[1]The results shown are for fixed levels of traffic. With TCP or TCP friendly sources, green would react to losses by drastically reducing their rate and they would be even worse off!

subject to the following constraints:

- Green packets remain in the system no longer than d, namely that they finish service within d seconds of arriving (thus satisfying the low delay requirement).
- Local transparency to blue holds.
- The scheduling is work conserving[2].
- No reordering within colour: Blue (respectively green) packets are served in the order of arrival.

DSD supports any mixture of TCP, TCP friendly and non-TCP friendly traffic. It is based on the concept of *duplicates* and the use of a virtual queue. Deadlines are assigned to packets upon arrival, and green and blue packets are queued separately. For both green and blue packets, the deadline represents the latest time the packet can remain in the system (it must finish service at the latest by its deadline). At service time, the deadlines of the packets at the head of the blue and green queues are used to determine which one to serve next.

DSD sends a duplicate of each packet arrival to a virtual queue. A blue packet is dropped if its duplicate was dropped in the virtual queue. Otherwise it receives a *deadline* equivalent to the service time its duplicate has in the virtual queue. A green packet is accepted if it passes what is called the *green acceptance test*, and then assigned a deadline equal to its arrival time plus the maximum time it can spend in the system d.

Green and blue packets are queued separately, and the deadlines of the packets at the head of blue and green queues are used to determine which one is to be served next; blue packets are served at the latest their deadline permits and green packets are served in the meantime.

The virtual queue is not restricted to drop-tail queueing. An active queue management scheme such as RED [12] can be applied to the virtual queue, which changes the assigned losses and deadlines appropriately. More detail on this is given in Section 3.5 on page 43.

Some of the building blocks in DSD are similar to those in other scheduling techniques. The calculation and tagging of deadlines to each arriving packet is also performed by Earliest Deadline First (EDF)[44] schedulers and its variants. However, EDF sorts packets according to deadlines, whereas DSD serves each of its two queues in a first come, first served manner, and the deadlines are used at service to determine whether the head of the green or the head of the blue queue should

[2]work conservation means the scheduler never goes idle whenever there is a packet to serve.

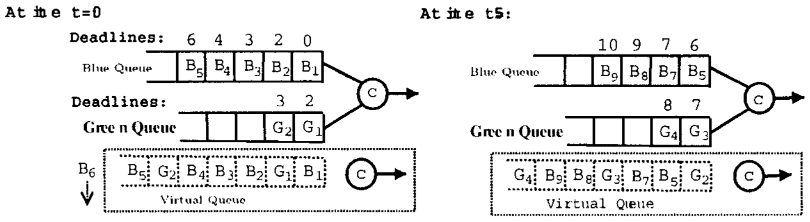


Figure 3.3: Two snapshots as an example of DSD, at time $t = 0$ (left) and $t = 5$ (right). For this example, all packets have the same length and "packet" time is used. To facilitate understanding, we consider first the case where green packets do not undergo the green acceptance test and where $g = 1$. The maximal buffer size is $Buff = 7$ packets. The maximum green queue wait is $d = 3$ packets. B and G denote blue and green packets respectively. In the first snapshot, B_1 is served at time $t = 0$ in order to meet its deadline, followed by G_1, B_2, B_3, B_4. However, G_2 has to be dropped from the green queue because it has to wait for more than $d = 3$, whereas B_6 had to be dropped because the virtual queue length was equal to $Buff$ when it arrived. At time $t = 5$, we reach the situation of the second snapshot. As no blue packet has reached its deadline yet, G_3 can be served, followed by B_5, B_7, G_4, B_8, and B_9.

be served. The use of a virtual queue has been used many times, for example in an admission control context [9].

To provide throughput transparency in the DSD scheduler, a controller, as described in Section 3.4.2 on page 41, acts upon a parameter g to control the service received by green packets. g is the probability of serving the green queue first in the event that the deadlines of the packets at the head of each queue can both be met if the other queue was served beforehand. The delay and loss ratio are monitored, and the controller adjusts g to ensure throughput transparency is maintained.

3.2.2 Description

The architecture of DSD is outlined in Figure 3.4, while Figure 3.5 provides an overview of the actions at arrival and service. Pseudocode for DSD is shown in Table 3.1. An example of how DSD works is shown in Figure 3.3.

Virtual Queue

Duplicates of all incoming packets are sent to a virtual queue with a buffer size $Buff$. A duplicate is admitted if the virtual buffer has space. Packets in the virtual queue are served according to first come, first served at rate c, as they would be in flat best-effort. The times at which duplicates will finish service are used to assign blue

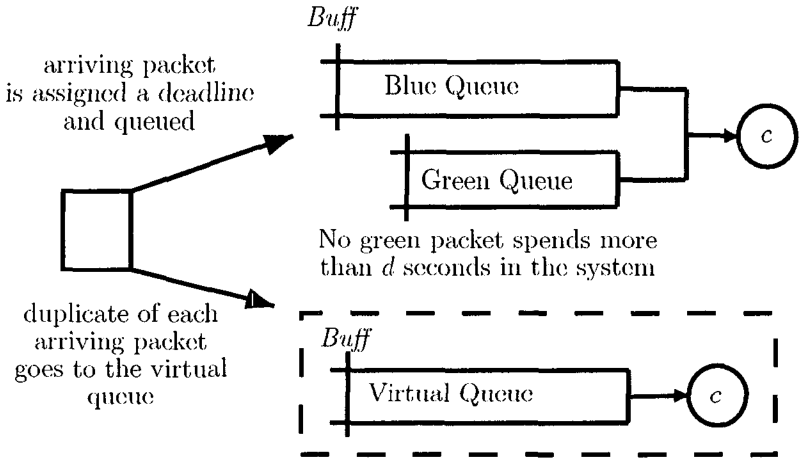

Buff
arriving packet is assigned a deadline and queued
Blue Queue
Green Queue
c
No green packet spends more than d seconds in the system
duplicate of each arriving packet goes to the virtual queue
Buff
Virtual Queue
c

Figure 3.4: Overview of DSD.

packets *deadlines* at which they would have been served in flat best-effort.

The original arriving packets are fed according to their colour into a green and a blue queue. Blue packets are always served at the latest their deadline permits subject to work conservation. Green packets are served in the meantime if they can finish service within d seconds of arriving, and are dropped otherwise.

Blue Packet Acceptance

A blue packet is dropped if its duplicate was not accepted in the virtual queue. This rule we call the *virtual queue test*. Otherwise, it is tagged with a deadline, given by the time at which its duplicate will be served in the virtual queue, and placed at the back of the blue queue.

Green Packet Acceptance

Given the constraints of transparency, the goal is to squeeze in as many green as possible. One way to do this would be to accept all green packets initially, serve those that could make it in d seconds, and drop all those that do not (indeed our draft design did exactly this). This method turns out to be a high drain on resources and instead we use a test as follows.

A green packet is accepted if it passes what is called the *green acceptance test* and dropped otherwise. Consider a green packet arriving at time t. It fails the test if the sum of the length of the green queue at time t (including this packet), and of

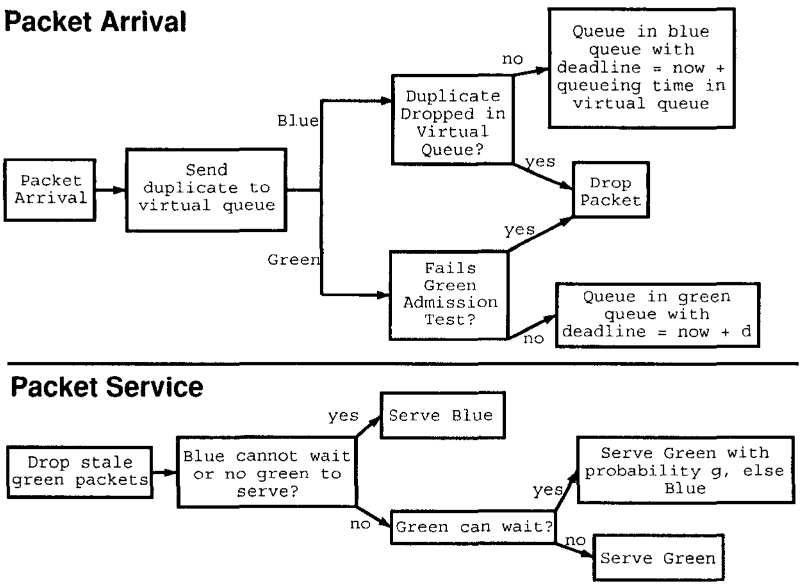

Packet Arrival
Packet Arrival
Send duplicate to virtual queue
Blue
Green
Duplicate Dropped in Virtual Queue?
no
yes
Queue in blue queue with deadline = now + queueing time in virtual queue
Drop Packet
yes
Fails Green Admission Test?
no
Queue in green queue with deadline = now + d
Packet Service
Drop stale green packets
Blue cannot wait or no green to serve?
yes
Serve Blue
no
Green can wait?
yes
Serve Green with probability g, else Blue
no
Serve Green

Figure 3.5: DSD queueing and serving.

the length of the first part of the blue queue, that contains packets tagged with a deadline which is less than or equal to $t + d$, is more than cd, and passes otherwise.

We prove in Section 3.3 on page 32 that the use of the test ensures the total buffer occupancy, namely the sum of the green and blue queue lengths does not exceed *Buff*. An accepted green packet is assigned a deadline which is the sum of its arrival time plus the maximum time it can spend in the system d, and placed at the back of the green queue.

Consider again the example in Figure 3.3, except green packets are now queued only if they pass the green acceptance test. This amounts here to accepting a green packet at time t if the number of green packets in the queue at time t, plus the number of blue packets in the queue with a deadline between $[t, t + 4]$ is no more than 4. The only difference from Figure 3.3 is that G_2 is no longer queued. Indeed, when it arrived, the green queue already contained packet G_1, and the blue queue contained packets B_1, B_2 and B_3. The total queue length at the time $t = 0$ was 5 packets (including G_2), and so G_2 fails the test.

Note that green are *not* subject to the virtual queue test, so it may be that a green is accepted while its duplicate is not. This does not violate local transparency, but is a point worth returning to, which we shall do in Section 3.4.1 on page 40. A green that is not accepted but whose duplicate is accepted in the virtual queue causes what we refer to as a *discriminatory drop*.

Serving Algorithm

At each service time, a decision is made as to which queue to serve. The serving mechanism's primary function is to ensure that blue packets are always served no later than their deadlines. The best performance green could receive would be to then serve the green queue as much as possible, subject to this restriction. However, as previously discussed in Section 2.4.2 on page 14, in addition to local transparency, throughput transparency is needed to ensure green adaptive applications do not benefit too much from lower delay.

As an aid to what follows, it useful to say precisely what we mean by a packet's ability to wait.

Definition 3.2.1 *A packet in a DSD queue* ***can possibly wait*** *if its deadline is less than the time it finishes service.*

By itself, all that can be said is that a packet can *possibly* wait. When the transmission time of another packet is known, we can define a term *can wait* given this other packet.

Definition 3.2.2 *A packet p in a DSD queue* ***can wait*** *given a packet p' if p still finishes service by its deadline if p' was served before it.*

Mathematical definitions of these concepts "can possibly wait" and "can wait" are given later in Section 3.3.1 on page 33.

There are service instances when both blue and green packets at the head of their respective queues are able to wait given the other, as letting the other packet go first would still allow it to be served within its deadline. When this situation arises, the packet serving algorithm uses the current value of the *green bias* g, a value in the range $[0, 1]$, to determine the extent to which green is favoured over blue. More precisely, when both blue and green packets *can wait* given each other, g is the probability that the green packet is served first.

The value $g = 1$ corresponds to the case where green is always favoured. Conversely, the value $g = 0$ corresponds to the systematic favouring of blue packets e.g. in Figure 3.3 the order of packet service would have been B_1, B_2, G_1, B_3, B_4, B_5, B_7, G_3, G_4, B_8 and B_9.

A value of g less than 1 causes the delay for green traffic to be increased. This increase in delay for green TCP friendly traffic reduces their throughput, thereby

Packet Queueing Algorithm

```
packet $p$ arrives at the output port
$dup \leftarrow p$
Add $dup$ to the virtual queue

if $p$ is blue
{
  if $dup$ was dropped from virtual queue
  {
    drop $p$
  }
  else
  {
    $vl$ = length of the virtual queue (in bits)
    $p$.deadline $= now + vl/c$
    add $p$ to blue queue
  }
}
else // $p$ is green
{
  if $p$ fails "green acceptance test"
  {
    drop $p$
  }
  else
  {
    $p$.deadline $\leftarrow now + d$
    add $p$ to green queue
  }
}
```

"Green acceptance Test"

```
$l_p$   length of packet $p$
$l_g$   length of green queue
$l_b$   length of packets in blue queue with...
        ...deadlines $\leq now + d$
if $l_g + l_b + l_p > cd$
  return "$p$ fails test"
else
  return "$p$ passes test"
```

Packet Serving Algorithm

```
headGreen ← packet at head of green queue
headBlue ← packet at head of blue queue

if headGreen $=$ 0 // no green to serve
{
  if headBlue $\neq 0$
    serve headBlue
}
else if headBlue $=$ 0 // no blue to serve
{
  serve headGreen
}
else // both queues contain packets
{
  $p_g$ = headGreen.transDelay
  $dead_g \leftarrow$ headGreen.deadline
  $p_b \leftarrow$ headBlue.transDelay
  $dead_b \leftarrow$ headBlue.deadline

  // if headBlue *cannot wait* given headGreen
  if $now + p_g > dead_b - p_b$
    serve headBlue
  // else if headGreen *cannot wait* given headBlue
  else if $now + p_b > dead_g - p_g$
    serve headGreen
  else with probability $g$ // both *can wait*
    serve headGreen
  else
  {
    serve headBlue
    clean up any possible stale green packets
  }
}
```

Table 3.1: *Pseudocode of DSD.* now is the current time, p.deadline denotes the latest time a packet p can remain in the queue (whose value is tagged onto packet p), and p.transDelay denotes its transmission delay. Throughout, we use the notation that 0 denotes no packet.

enabling blue traffic to increase its throughput. Increasing the delay of non-TCP friendly traffic may not reduce their throughput, but blue flows are, in the worst case, as equally protected from this type of traffic as they would have been in a flat best-effort service. The value of g chosen is made according to a control loop which is described in Section 3.4.2 on page 41.

At service time, the possible events that arise and packets served by DSD are summarised in Table 3.2.

All green packets that miss their deadline, by spending more than d seconds in the system, are said to have become *stale*, and are removed from the green queue. Stale packets can occur if $g < 1$ and a blue packet is favoured over a green packet when both can wait.

Event	**What is Served?**
Both queues empty	Nothing
Green queue empty, blue queue not empty	Head of blue queue
Blue queue empty, green queue not empty	Head of green queue
Head of blue queue *cannot wait*	Head of blue queue
Head of blue queue *can wait*, head of green queue cannot	Head of green queue
Head of green queue and of blue queue *can wait*	With probability g, head of green queue, else head of blue queue

Table 3.2: Service Decisions.

3.3 Properties of DSD

We now describe some of the most important properties of DSD.

1. All accepted blue packets will be served by their deadlines. Accepted blues are thus served at the same time as, or earlier than, they would have been in flat best-effort.

2. All green packets finish service within d seconds of arrival, or are otherwise dropped. Low bounded (per hop) delay for the green packets is enforced by dropping a green packet that waits or would have to wait d seconds in the queue.

3. The green acceptance test does not cause the dropping of any green packet that could have otherwise been served within its deadline.

4. If $g = 1$, the green acceptance test accepts exactly those green packets that will be served within d seconds.

5. Buffer space constraint: the total buffer occupancy for real packets (green and blue counted together) at any given time is less than *Buff*, the maximum buffer size of the virtual queue. Moreover, it is always less than current virtual queue size at any given time.

Items 1 and 2 are direct consequences of the DSD algorithm. Items 3 and 4 are proved in Theorem 3.3.1 on page 34, and item 5 in Theorem 3.3.2 on page 35 and Theorem 3.3.3 on page 37.

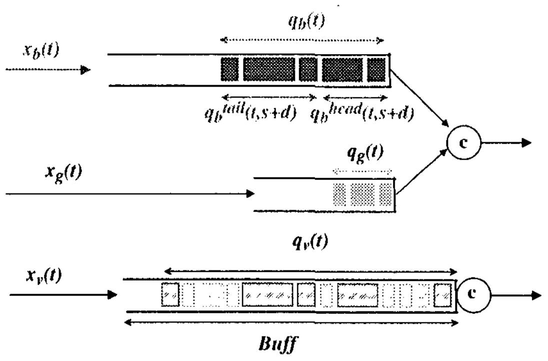


Figure 3.6: Notation used in proofs of properties of DSD.

3.3.1 Proofs of properties

The definitions of "can possibly wait" (Definition 3.2.1), and "can wait" (Definition 3.2.2) can be expressed mathematically in the following way.

Definition 3.3.1 *Let t be the current time. Consider a packet p with length l_p and with deadline $d_p \geq t$ in the DSD queue serving a link of capacity c. Let $q(t)$ be the number of bits that currently will be served ahead of p. We say p* ***can possibly wait*** *if and only if*

$$d_p - \frac{l_p}{c} \geq t + \frac{q(t)}{c}.$$

Definition 3.3.2 *Let t be the current time. Consider a packet p with length l_p and with deadline $d_p \geq t$ in the DSD queue serving a link of capacity c. Let p' be another packet of length $l_{p'}$. Let $q(t)$ be the number of bits that currently will be served ahead of p. We say p* ***can wait*** *given p' if and only if*

$$d_p - \frac{l_p}{c} \geq t + \frac{q(t) + l_{p'}}{c}.$$

Some of the notation used in the proofs is illustrated in Figure 3.6. Let $q_g(t)$ be the length of the green queue at time t. Let $q_b(t)$ be the total length of the blue queue and $q_v(t)$ be the length of the virtual queue at time t.

Let $q_b^{\text{head}}(t_1, t_2)$ be the sum of the length of the packets in the blue queue whose deadlines are in $[t_1, t_2]$.[3]

[3]In the DSD pseudocode of Table 3.1, $l_g = q_g(t)$ and $l_b = q_b^{\text{head}}(t, t+d)$

Theorem 3.3.1 (Green Acceptance Test) *Let $g = 1$. Consider a green packet G of length l_G which arrives at time t. Let $q_g(t-)$ be the number of green bits in the queue just before the green packet arrived. Let local transparency hold. Let the queue be served at a rate c.*

Then, if the green packet is placed in the green queue, it will finish service by time $t + d$ if and only if

$$q_g(t-) + q_b^{\text{head}}(t, t + d) + l_G \leq cd. \tag{3.1}$$

Proof: G can spend a maximum of d seconds in the queue, thus the deadline for G is $t + d$.

Provided G can finish service by $t + d$, no packet which arrives after G will be served before G according to the DSD serving algorithm when $g = 1$. (This is proved in Theorem 3.3.4 on page 38)

(i) Consider first the case when there are no blue packets in the queue. Then G can be served if and only if the time it takes to drain the green queue plus serve G is less than d. Thus G can be served if and only if Equation (3.1) holds.

(ii) Consider the case where all packets in the blue queue *can wait* given G. Now G will then be accepted if and only if

$$q_g(t-) + l_G \leq cd \tag{3.2}$$

since, when $g = 1$, the green will go in front of the first packet in the blue queue but behind all greens still in the queue. It is clear that Equation (3.1) implies Equation (3.2).

Now imagine that G is accepted, and consider the last packet in the blue queue with deadline in $[t, t + d]$. Let its deadline be d_b. Since this packet can wait given G (all blue *can wait*),

$$t + d \geq d_b \geq t + \frac{q_b^{\text{head}}(t, t + d) + q_g(t-) + l_G}{c}$$

and thus Equation (3.1) follows.

(iii) Now consider the case where there is at least one packet in the blue queue that *cannot wait* given G, and let W denote the last one. Let its length be l_W and its deadline be d_W. We have, by Definition 3.3.2, that

$$d_W < t + \frac{q_g(t-) + q_b^{\text{head}}(t, t + d_W) + l_G}{c}. \tag{3.3}$$

Now G will finish service by its deadline $t + d$ if and only if G *can wait* given packet W, i.e. by Definition 3.3.2, if and only if

$$t + d - \frac{l_G}{c} \geq t + \frac{q_g(t-) + q_b^{\text{head}}(t, t + d_W)}{c} \tag{3.4}$$

or equivalently, if and only if

$$q_g(t-) + q_b^{\text{head}}(t, t + d_W) + l_G \leq cd. \tag{3.5}$$

What we wish to show is that Equation (3.1) holds if and only if Equation (3.5) holds.

Note that Equations (3.3) and (3.4) holding implies

$$d_W < t + d. \tag{3.6}$$

$\Rightarrow$: First the "if" part. Let Equation (3.1) hold. Equation (3.6) implies that

$$q_b^{\text{head}}(t, t + d_W) \leq q_b^{\text{head}}(t, t + d)$$

and thus Equation (3.1) implies Equation (3.5).
$\Leftarrow$: Now assume that the green can finish service within its deadline $t + d$, namely that Equation (3.5) holds.

Call L the last packet in the blue whose deadline d_L is less than $t+d$. This exists since, by Equation (3.6), at least W has a deadline less than $t + d$.

If $L = W$, then $d_L = d_W$ and

$$q_b^{\text{head}}(t, t + d_W) = q_b^{\text{head}}(t, t + d_L) = q_b^{\text{head}}(t, t + d),$$

and the result follows. Otherwise, L is behind W in the queue since, by Equation (3.6), W's deadline is less than $t + d$.

L can necessarily wait given G since W is the last blue packet that *cannot wait*. Thus,

$$t + d \geq d_L \geq t + \frac{q_b^{\text{head}}(t, t + d) + l_G}{c}$$

which implies that

$$q_g(t) + q_b^{\text{head}}(t, t + d) + l_G \leq cd.$$

□

Theorem 3.3.2 (Buffer space constraint) *The sum of the blue and green queue* *lengths does not exceed the maximum virtual queue size Buff, i.e. at any time t,*

$$q_b(t) + q_g(t) \leq \mathit{Buff}.$$

Proof: Consider the system at any time $t \geq 0$.

(i) Consider first the case where there are no green packets in the queue at time t, i.e. $q_g(t) = 0$.

A blue packet is present in the blue queue only if its duplicate is present in the virtual queue. This is because a blue packet is accepted if and only if its corresponding duplicate is also admitted, and since it is always served no later than its duplicate. Therefore $q_b(t) \leq q_v(t)$.

Since the virtual queue length is always less than *Buff*,

$$q_b(t) + q_g(t) = q_b(t) \leq q_v(t) \leq \mathit{Buff}.$$

(ii) Consider now the case where $q_g(t) > 0$. Let s be the last time prior to t that an incoming green packet arrived and was admitted.

$q_b^{\text{head}}(t, s+d)$ is the length of the portion of the blue queue whose packets have deadlines in $[t, s+d]$. It contains the bits that are counted in $q_b^{\text{head}}(s, s+d)$.

Similarly, $q_g(t)$ contains the bits that are counted in $q_g(s)$ and that have not yet been served, since there are no new green arrivals after time s.

The green queue is never empty in $[s, t]$ because $q_g(t) > 0$, and therefore the server is never idle during $[s, t]$. Thus,

$$q_b^{\text{head}}(t, s+d) + q_g(t) = q_b^{\text{head}}(s, s+d) + q_g(s) - (t-s)c.$$

At time s, since the last green packet to arrive passed the green acceptance test,

$$q_b^{\text{head}}(s, s+d) + q_g(s) \leq cd$$

which combined with the previous equation yields

$$q_b^{\text{head}}(t, s+d) + q_g(t) \leq cd - (t-s)c. \tag{3.7}$$

On the other hand, let $q_b^{\text{tail}}(t, s+d) = q_b(t) - q_b^{\text{head}}(t, s+d)$ denote the length of the portion of the blue queue at time t with packets having deadlines larger than $s+d$.

The deadline of a queued blue packet is larger than $s+d$ if there are at least $(d+s-t)c$ bits to be served by the virtual server of rate c, before the corresponding duplicate finishes service.

The maximum number of bits in the virtual queue at time t that can belong to duplicates of blue packets with deadline greater than $s+d$ is at most $\mathit{Buff}-(d+s-t)c$.

The length of the portion of the blue queue with packets having a deadline larger than $s+d$ satisfies

$$q_b^{\text{tail}}(t, s+d) \leq \mathit{Buff} - (d+s-t)c,$$

since a blue packet is present in the blue queue only if its duplicate is present in the virtual queue. Combining this inequality with Equation (3.7) yields

$$\begin{aligned} q_b(t) + q_g(t) &= q_b^{\text{tail}}(t, s+d) + q_b^{\text{head}}(t, s+d) + q_g(t) \\ &\leq \mathit{Buff} - (d+s-t)c \\ &+ cd - (t-s)c = \mathit{Buff}. \end{aligned}$$

□

Theorem 3.3.2 can be refined and the following established using the fact that the amount of bits admitted in the virtual queue is the same as the incoming fresh traffic as long as $q_v < \mathit{Buff}$. The proof follows the approach of the proof of Lemma 3, p. 20 in [26].

Theorem 3.3.3 (Virtual Queue Bounds Actual Queue) *At any time t,*

$$q_b(t) + q_g(t) \leq q_v(t).$$

Proof: Let $a(t)$ be the total amount of traffic (in bits) that arrived in the router in $[0,t]$. Let $x(t)$ (respectively $x_v(t)$) be the total number of bits corresponding to packets (resp. duplicates) that have been admitted in the router (resp. in the virtual queue) in $[0,t]$. The sum of the backlogs in the blue and green queues at time t can then be expressed by

$$q_g(t) + q_b(t) = \sup_{s\in[0,t]} \{x(t) - x(s) - (t-s)c\}$$

whereas the virtual queue length at time t is

$$q_v(t) = \sup_{s\in[0,t]} \{x_v(t) - x_v(s) - (t-s)c\}.$$

Let t be a given time.

(i) Consider first the case where the virtual queue was never full in $[0,t]$. The backlogged data in the actual system at time t is then given by

$$\begin{aligned} q_g(t) + q_b(t) &= \sup_{s\in[0,t]} \{x(t) - x(s) - (t-s)c\} \le \sup_{s\in[0,t]} \{a(t) - a(s) - (t-s)c\} \\ &= \sup_{s\in[0,t]} \{x_v(t) - x_v(s) - (t-s)c\} = q_v(t). \end{aligned}$$

(ii) Consider now the case where the virtual queue was full at least once in $[0,t]$. Let $u \in [0,t]$ be the last time the virtual queue was full. Thus $q_v(u) = \mathit{Buff}$. The traffic entering the virtual system during $[u,t]$ is identical to the traffic that arrived during $[u,t]$ i.e. $x_v(t) - x_v(u) = a(t) - a(u)$. Then using Theorem 3.3.2,

$$\begin{aligned} q_g(t) + q_b(t) &= \sup_{s\in[0,t]} \{x(t) - x(s) - (t-s)c\} \\ &= \sup_{s\in[0,u]} \{x(t) - x(s) - (t-s)c\} \vee \sup_{s\in[u,t]} \{x(t) - x(s) - (t-s)c\} \\ &= \sup_{s\in[0,u]} \{x(t) - x(u) - (t-u)c + x(u) - x(s) - (u-s)c\} \\ &\qquad \vee \sup_{s\in[u,t]} \{x(t) - x(s) - (t-s)c\} \\ &= \{x(t) - x(u) - (t-u)c + \sup_{s\in[0,u]} \{x(u) - x(s) - (u-s)c\}\} \\ &\qquad \vee \sup_{s\in[u,t]} \{x(t) - x(s) - (t-s)c\} \\ &= \{x(t) - x(u) - (t-u)c + q_g(u) + q_b(u)\} \vee \sup_{s\in[u,t]} \{x(u) - x(s) - (t-s)c\} \\ &\le \{x(t) - x(u) - (t-u)c + \mathit{Buff}\} \vee \sup_{s\in[u,t]} \{x(t) - x(s) - (t-s)c\} \\ &\le \{a(t) - a(u) - (t-u)c + \mathit{Buff}\} \vee \sup_{s\in[u,t]} \{a(t) - a(s) - (t-s)c\} \\ &= \{x_v(t) - x_v(u) - (t-u)c + \mathit{Buff}\} \vee \sup_{s\in[u,t]} \{x_v(t) - x_v(s) - (t-s)c\} \\ &= \{x_v(t) - x_v(u) - (t-u)c + q_v(u)\} \vee \sup_{s\in[u,t]} \{x_v(t) - x_v(s) - (t-s)c\} \\ &= \{x_v(t) - x_v(u) - (t-u)c + \sup_{s\in[0,u]} \{x_v(u) - x_v(s) - (u-s)c\}\} \\ &\qquad \vee \sup_{s\in[u,t]} \{x_v(t) - x_v(s) - (t-s)c\} \\ &= \sup_{s\in[0,u]} \{x_v(t) - x_v(s) - (t-s)c\} \vee \sup_{s\in[u,t]} \{x_v(t) - x_v(s) - (t-s)c\} \\ &= \sup_{s\in[0,t]} \{x_v(t) - x_v(s) - (t-s)c\} = q_v(t). \end{aligned}$$

□

Theorem 3.3.4 *Let $g = 1$. Then an accepted blue packet that arrives after an accepted green packet will be served after the green packet.*

Proof: Suppose a green packet arrives at time t. It be must checked that any accepted blue packet that arrives after t will be served after a green packet that arrived at time t.

Let u be the smallest time larger than or equal to t such that $q_v(u) > cd$ (if no such time exists, let $u = \infty$). Then $q_v(s) \leq cd$ for all $s \in [t, u)$.

The sum of the blue and green queue lengths is less than cd, due to Theorem 3.3.3, so

$$q_g(u) + q_b(u) \leq q_v(u) \leq cd.$$

This means all packets which arrived at any time $s \in [t, u)$, including the green packet that arrived at time t, will finish their service within d seconds from their arrival time in a first come, first served manner.

On the other hand, for all $s \geq u$,

$$q_v(s) \geq q_v(u) - (s-u)c > cd - (s-u)c$$

and thus any blue packet which arrived at any time $v \geq u$ will have a deadline such that

$$\begin{aligned} s + q_v(s)/c &> s + d - (s-u) \\ &= u + d \geq t + d, \end{aligned}$$

i.e. after the green packet under consideration will have completed its service. □

3.4 Providing throughput transparency

As previously mentioned, local transparency is not necessarily sufficient to ensure green do not hurt blue as sources sending green traffic that are rate-adaptive and greedy may obtain more throughput than they would if they were blue. There are two reasons for this:

1. TCP causes flows with shorter round-trip times to receive more bandwidth. With local transparency, the decrease in delay may more than compensate for the increase in loss.

2. As mentioned in Section 3.2.2 on page 28, plain DSD does allow green packets to be accepted when their duplicate is not accepted in the virtual queue. This

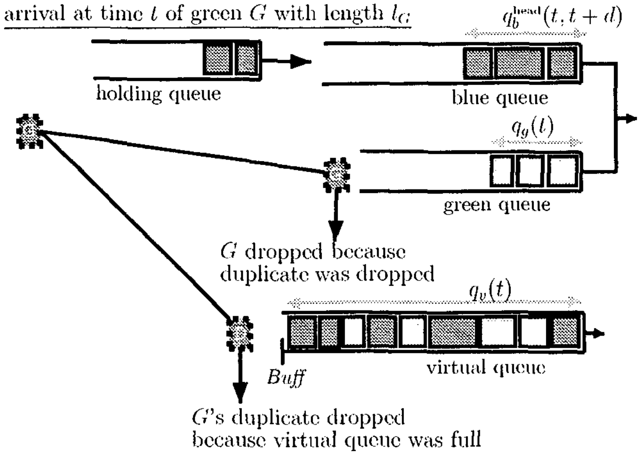

arrival at time t of green G with length l_G
$q_b^{\text{head}}(t, t+d)$
holding queue
blue queue
$q_g(t)$
green queue
G dropped because duplicate was dropped
$q_v(t)$
$Buff$
virtual queue
G's duplicate dropped because virtual queue was full

Figure 3.7: Illustration of *green undergo virtual queue test*. The green packet G is dropped despite sufficient space ($q_b^{\text{head}}(t, t+d) + q_g(t) + l_G \leq cd$) because its duplicate was dropped ($q_v(t) + l_G \geq Buff$).

leads to periods of time when green packets are accepted and not blue, enabling green flows to increase their rate in periods when blue flows are reducing theirs.

We solve Item 1 by the use of a controller, described in Section 3.4.2. It acts upon the green bias parameter g. By measurement of the delay and loss ratio, and using the TCP loss-throughput formula (Equation (2.1)), the controller adjusts g to ensure throughput transparency is maintained.

The controller solves the issue of evaluating the round-trip time of flows by the observation that an under-evaluation of green round-trip times is more protective of blue flows. Thus, the controller assumes that all flows are greedy and have a total round-trip time equal to the queueing time at this node plus a fixed, virtual base value. This value is taken to be small so that it is unlikely that any real value could be below it.

Item 2 is solved by the *green undergo virtual queue test*, namely that whenever a green's duplicate is not accepted in the virtual queue, the green is not accepted in the real queue. In Section 3.4.1 we describe the method and the reasoning behind it.

3.4.1 Green undergo virtual queue test

This requirement, as illustrated in Figure 3.7, means that a green is not accepted into the real system if its duplicate was dropped in the virtual queue *even if* it could finish service within d seconds of arrival and not violate local transparency.

This extra requirement on basic DSD is used to preserve throughput transparency. Without it, in a very busy period, blues can be dropped while greens are accepted because of green drops that occurred at the beginning of the busy period which can be many round-trip times ago.

The test ensures that, over the time window of the order of a round-trip time, the rate of green drops is not lower than the rate of blue drops.

3.4.2 Control loop

As described in Section 2.4.2 on page 14, unlike local transparency, maintaining throughput transparency is by its nature approximate. The green bias g is used as a control parameter to balance the throughputs of green and blue, which are estimated by the loss-throughput formula in Equation (2.1) on page 14.

A fixed value τ_e is used to represent the non-queueing delay portion of the round-trip time of a flow. This value is chosen to be small, since this favours blue traffic, which we show below.

For the purposes of the control, flows are assumed to be greedy, since this also increases the protection to blue flows.

Estimates for the delay and loss ratio for both green and blue traffic are monitored, and throughput estimated by Equation (2.1). Let $\theta_b(t)$ and $\theta_g(t)$ be these estimates for the blue and green throughput respectively at time t. The value of g is chosen so that their ratio is close to a desired value γ, which is slightly larger than 1 to provide blues with a small advantage in throughput and to offer a safety margin for protection from errors in throughput estimation.

g is updated every T seconds according to the control law,

$$g(t+T) = (1-\alpha)g(t) + \frac{\alpha}{1 + (\gamma\theta_g(t)/\theta_b(t))^K}, \tag{3.8}$$

where $\alpha \in (0, 1)$ and $K > 0$ are two control parameters. T is a chosen parameter of the system which determines the rate of update of g. The initial value of g upon commencement of control can be chosen to be 1, namely $g(0) = 1$.

We do not claim this control loop is optimal. The use and control of the green bias g is only one possible scheme, and its performance can be improved, for example, by taking into account the deadlines of all the packets in the queues, and not just those at the head.

Let us briefly explain the rationale behind this choice of control law. In the ideal case where $\theta_b = \gamma\theta_g$, there should not *a priori* be any bias against blue or green, and the value of g should be 1/2. If θ_b is larger than $\gamma\theta_g$, then g must be increased, and vice versa if θ_b is smaller than $\gamma\theta_g$.

We wish to maintain symmetry in the amount by which we increase or reduce g. The amount by which g is increased if $\theta_b/\gamma\theta_g$ is multiplied by some factor A should be the same amount by which g is decreased if $\theta_b/\gamma\theta_g$ is divided by the same factor A.

Denoting by $\xi = \ln(\theta_b/\gamma\theta_g)$, the targeted g should therefore be an increasing function F of ξ with central symmetry around 0, and such that $F(0) = 1/2$, $F(\xi) = 0$ for $\xi \to -\infty$ and $F(\xi) = 1$ for $\xi \to \infty$. The sigmoid

$$F(\xi) = \frac{1}{1 + \exp(-K\xi)}$$

is such a function. K is the slope of F at the origin. The larger K is, the closer the sigmoid is to the step (Heavyside) function

$$\overline{F}(\xi) = \begin{cases} 1 & \text{if} \quad \xi > 0 \\ 1/2 & \text{if} \quad \xi = 0 \\ 0 & \text{if} \quad \xi < 0 \end{cases} .$$

The control law

$$g(t + T) = g(t) + \alpha(F(\xi) - g(t))$$

where α is the adaptation gain, will therefore bring g to the targeted value. If $\alpha \in [0, 1]$, this control law keeps $g(t)$ between 0 and 1 at all times t. Replacing ξ by $\ln(\theta_b/\gamma\theta_g)$ in this equation, we get the control loop equation for the green bias as given in Equation (3.8).

Best protection for blue is smallest round-trip time

We show here that if $\theta(q_b, \tau_e + d_b) \geq \theta(q_g, \tau_e + d_g)$ then for all $h \geq 0$,

$$\theta(q_b, \tau_e + d_b + h) \geq \theta(q_g, \tau_e + d_g + h).$$

We first note that Equation (2.1) on page 14 can be written in the form $\theta(p, R) = \theta_l(p)/R$ and thus,

$$\frac{\tau_e + d_g}{\tau_e + d_b} \geq \frac{\theta_l(q_g)}{\theta_l(q_b)}.$$

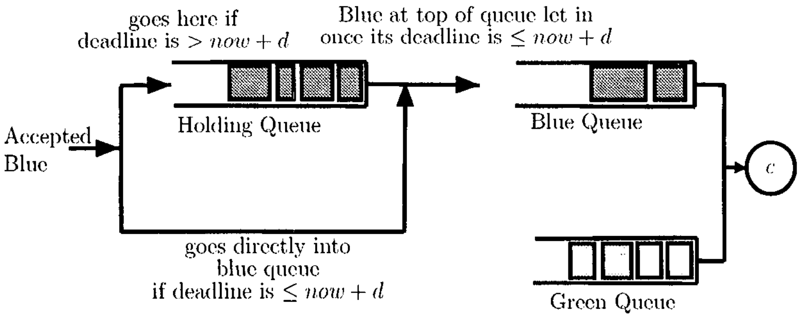


Figure 3.8: Holding Queue for DSD. The Blue queue is partitioned into those packets with deadlines $> now + d$ and those with deadline $\leq now + d$.

Now $(\tau_e + d_b)h \geq (\tau_e + d_g)h$ for all $h \geq 0$, since blue traffic has a higher average queueing delay $(d_b \geq d_g)$, which implies that

$$\frac{\tau_e + d_g + h}{\tau_e + d_b + h} \geq \frac{\tau_e + d_g}{\tau_e + d_b} \geq \frac{\theta_l(q_g)}{\theta_l(g_b)}.$$

3.5 Supporting RED

When the queue management on the virtual queue is RED the following is the alteration to DSD.

An arriving green packet is queued if it can be served within d seconds *and* its duplicate was accepted in the virtual queue (i.e. it also undergoes the green virtual queue test). An arriving blue packet is, as before, accepted in the blue queue if its duplicate is accepted in the virtual queue, and if accepted, assigned a deadline equal to the time it will finish service in the virtual queue.

The virtual queue accepts a duplicate if the buffer is not full and the RED dropping algorithm decides the packet should be accepted. The green must undergo the virtual queue test. Otherwise, the sum of the blue and green queues could potentially be larger than the size of the virtual queue, causing blues to be possibly unable to meet the deadlines they are assigned.

3.6 Holding queue

The main bottleneck in the green acceptance test is the calculation of the number of blue bits with deadline less than or equal to $now + d$, a value that changes continuously. When a green packet arrives, calculating this value would involve a search from the front of the blue queue until the end is reached, or a packet with deadline greater than $now + d$ is found.

We overcome this difficulty by introducing the concept of a *holding queue*, as in Figure 3.8. The blue queue is partitioned into those that have a deadline greater than $now + d$ and those that do not, enabling us to know straight away if a green can be accepted. Blue packets do not enter the blue queue while their deadline exceeds $now + d$, and are instead kept in the holding queue until their deadline is less than or equal to $now + d$.

The sum of the lengths of blue packets with deadline less than $now + d$ is then simply the total length in bits of the blue queue. A green packet is then accepted if its length plus the total number of bits in the green and the blue queues does not exceed cd.

The pseudocode for the algorithm is given in Table 3.3. The service algorithm must be modified slightly also. It can happen that there are blue packets in the holding queue, but none in the blue queue[1]. In this case, the first packet in the holding queue is under consideration for service.

[1]e.g. a burst of greens were dropped followed by blues that were accepted. These blue may have higher deadlines than $now + d$ because the duplicates of the green burst were accepted in the virtual queue.

DSD with Holding Queue

Packet Queueing Algorithm:

```
packet $p$ arrives at the output port
$dup = p$
Add $dup$ to the virtual queue

if $p$ is blue
{
  if $dup$ was dropped from virtual queue
  {
    drop $p$
  }
  else
  {
    $vl$ = length of the virtual queue (in bits)
    $p$.deadline = $now + vl/c$
    if $p$.deadline > $now + d$
    {
      add $p$ to holding queue
      if $p$ is the only packet in holding queue
        setHoldingQueueTimer()
    }
    else // $p$.deadline $\leq now + d$
    {
      add $p$ to blue queue
    }
  }
}
else // $p$ is green
{
  $l_g$ = length of green queue (in bits)
  $l_b$ = length of blue queue (in bits)
  if $l_g + l_b$ + p.length > $cd$
  {
    drop $p$
  }
  else
  {
    $p$.deadline = $now + d$
    add $p$ to green queue
  }
}
```

setHoldingQueueTimer()

```
{
  $q$ = head of holding queue
  set timer to expire at $q$.deadline − ($d + now$)
}
```

timerExpires()

```
{
  $q$ = head of holding queue
  transfer $q$ to back of blue queue
  if holding queue non-empty
    setHoldingQueueTimer()
}
```

Packet Serving Algorithm:

```
headGreen = packet at head of green queue
headBlue = packet at head of blue queue
headHolding = packet at head of holding queue
if headBlue == 0
  headBlue = headHolding

if headGreen == 0 // no green to serve
{
  if headBlue ≠ 0
    serveBlue()
}
else if headBlue == 0 // no blue to serve
{
  serve headGreen
}
else // both queues contain packets
{
  $p_g$ = headGreen.transDelay
  $dead_g$ = headGreen.deadline
  $p_b$ = headBlue.transDelay
  $dead_b$ = headBlue.deadline

  // if headBlue cannot wait given headGreen
  if $dead_b - p_b < now + p_g$
    serveBlue()
  // else if headGreen cannot wait given headBlue
  else if $dead_g - p_g < now + p_b$
    serve headGreen
  else with probability $g$ // both can wait
    serve headGreen
  else
  {
    serveBlue()
    drop any possible stale green packets
  }
}
```

serveBlue()

```
{
  serve headBlue

  if headBlue == headHolding
  {
    cancel previous timer
    if holding queue non-empty
      setHoldingQueueTimer()
  }
}
```

Table 3.3: *Holding queue version of DSD* (pseudocode). now is the current time, p.deadline denotes the latest time a packet p can remain in the queue, p.transDelay its transmission delay, and p.length its length in bits.

Chapter 4

Serial DSD

In this chapter, we:

- describe (in Section 4.1) the serial DSD algorithm; this algorithm is functionally equivalent to vanilla DSD (described in Chapter 3) in that it gives, for the same input, the same output. However, it has different complexity properties than vanilla DSD.
- discuss the possibility of and reasons for increasing the average green delay, while still ensuring the maximum green delay d is maintained and that there are no additional green losses (Section 4.2 on page 53);
- describe *green at back*; a change to serial DSD which reduces the complexity of the scheduler while increasing the average green delay with no change in loss rate (Section 4.2.1 on page 55);
- present *blue pushed up*; an additional mechanism to green at the back, which further increases the average green delay with no effect on the loss rate (Section 4.2.2, page 56).

4.1 Serial DSD

We present an algorithm, referred to as *serial DSD*, which is equivalent to vanilla DSD but uses a single queue instead of separate blue and green queues. More precisely, we present three different algorithms:

1. *serial DSD, simple version* which is functionally equivalent to vanilla DSD when $g = 1$.
2. *serial DSD, optimised version*: The simple version is refined to allow for a holding queue plus the maintenance of the last blue that can possibly wait.

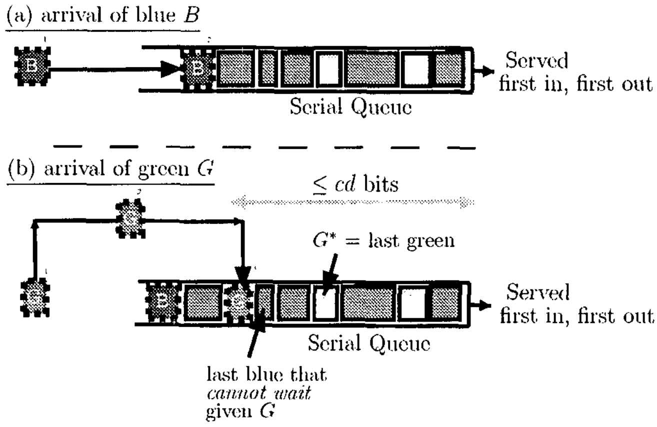


Figure 4.1: Illustration of *serial DSD, simple version*. An arriving blue B (shown first) goes at the back of the serial queue since its duplicate is accepted in the virtual queue. Then an arriving green G (shown second) can finish service in d seconds and is thus accepted to go just behind the last blue that *cannot wait* given G.

3. *serial DSD, variable g* which is functionally equivalent to vanilla DSD and can be used with the control loop as described in holding queue.

By layering the algorithmic complexity in this manner we endeavour to facilitate initial understanding without the clutter of all the bells and whistles. As well as warranting study in its own right, serial DSD enables extensions such as those in Section 4.2 on page 53, and proves additionally applicable in the modelling of Chapter 5.

We note that for the first two implementations, deadline checking at service time is completely avoided, and for the last one the amount of checking at service is sizeably reduced. However, deadline checking and decisions at the arrival of a packet is required.

Anyone who implements the scheduler will have to make decisions based on hardware, software limitations and the traffic parameters. We cannot say *a priori* which algorithm is best in all circumstances and instead strive to facilitate ease of implementation by providing many potential optimisations. Indeed, it is conceivable to implement a scheduler which combines features of serial and vanilla DSD.

```
Serial DSD, simple version

$G^*$ = last green accepted and still in serial queue
(let $G^* = 0$ if no green in the serial queue)
$t^*_G$ = time $G^*$ will start service

Packet Queueing Algorithm:
packet $p$ arrives at the output port
$dup = p$
Add $dup$ to the virtual queue

if $p$ is blue
{
  if $dup$ was dropped from virtual queue
    drop $p$
  else
  {
    $vl$ = length of the virtual queue (in bits)
    $p$.deadline = $now + vl/c$
    add $p$ to the back of serial queue
  }
}
else // $p$ is green
{
  $p$.deadline = $now + d$
  if no packet in serial queue awaits service
  {
    $t$ = time packet in service finishes transmission
    ($t = now$ if there is no packet in service)
    if $t > now + d - p$.transDelay
    {
      drop $p$
      return // we are finished
    }
    $b = 0$
  }
  else if $G^* \neq 0$ // there is a green in serial queue
  {
    $b = G^*$.next
    $t = t^*_G + G^*$.transDelay
  }
  else // only blue in serial queue
  {
    $b$ = head of serial queue
    $t$ = time $b$ is due to start service
  }

  while $b \neq 0$ // while not reached back of serial queue
  {
    // if $p$ cannot wait given $b$
    if $p$.deadline $- p$.transDelay $< t + b$.transDelay
    {
      drop packet
      exit while loop
    }
    // if $b$ can wait given $p$
    if $b$.deadline $- b$.transDelay $\geq t + p$.transDelay
    {
      insert $p$ in serial queue just in front of $b$
      exit while loop
    }
    $t = t + b$.transDelay
    $b = b.next$ // try next blue in the queue
  } // end of while loop

  if $p$ was not dropped
  {
    if $b = 0$ // no blue could wait, $p$ can go at back
      add $p$ to the back of serial queue
    $G^* = p$
    $t^*_G = t$
  }
}

Packet Serving Algorithm:
$p$ = first packet in queue (serve FIFO)
if $p = G^*$ // there is no longer any green in queue
  $G^* = 0$
serve $p$
```

Table 4.1: Pseudocode of Serial DSD, simple version. now is the current time, p.deadline denotes the latest time a packet p can remain in the queue, p.next denotes the next packet behind p in the queue (which will be served after p), and p.transDelay denotes its transmission delay.

4.1.1 Simple version

Figure 4.1 outlines the general technique of the simple version and the pseudocode is shown in Table 4.1. Advantage is taken of the fact that, when $g = 1$, future arrivals neither affect an arriving green packets acceptance nor its service time (shown in Theorem 3.3.1 on page 34 and Theorem 3.3.4 on page 38).

As in vanilla DSD, provided a blue's duplicate is accepted by the virtual queue, it is accepted in the real system and assigned a deadline corresponding to the time its duplicate finishes service.

The current location of the last green in the serial queue G^* as well as the time it is due to begin service t^*_G is stored. When a green arrives a search is done from the packet behind G^* to the end of the serial queue (where only blue reside) until one of the following occurs:

1. The green *cannot wait* given some blue. The green is then dropped.
2. A blue is found such that it *can wait* given the green arrival. The green is then inserted so as to receive service before this blue.
3. The length of the serial queue, including the length of the green packet, is smaller than cd bits. It is then placed at the back of the serial queue.

Essentially, the decision as to which packet should go first is made at arrival time instead of waiting until service time, as in the vanilla DSD case. The green acceptance test is not used since we determine explicitly whether the packet can be accepted and where it should go.

Serial DSD is the same as vanilla DSD when $g = 1$ for the following reasons:

- Local transparency holds. Exactly those blues that are accepted in vanilla DSD are accepted in serial DSD. In addition, no blue's deadline is allowed to be violated. A green G is only inserted in front of a blue that can wait given G. All blues that come behind this blue can also wait given G thanks to Theorem 3.3.4 on page 38.
- It is clear from the algorithm that no reordering has occurred. A blue goes at the back of the queue while a green goes behind the last accepted green (if any).
- Optimality for green holds. The same greens are accepted as in vanilla DSD. They are served the earliest possible subject to local transparency, work conservation, and the prohibition of reordering within colour.

4.1.2 Optimised version

The optimised version adds a holding queue plus the extra state of a blue packet B^* and the corresponding time it is currently due to start service t_B^*. If there are green packets in the serial queue, then B^* is the first blue, behind the last green packet G^*, that can possibly wait. If there is no green packet currently in the queue, then B^* is simply the first blue that can possibly wait.

An example is sketched in Figure 4.2, and we present pseudocode in Table 4.2. Knowledge of B^* is used to speed up checking at arrival time. We now, in effect, have a quadruple partition of the queue:

1. the holding queue of containing blues with deadline greater than $now + d$;

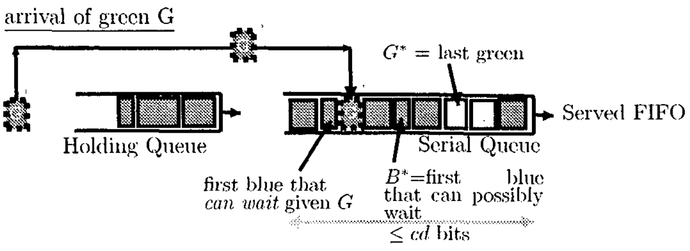


Figure 4.2: Illustration of *serial DSD, optimised version*. An arriving green G can finish service in d seconds (because the serial queue had less than cd bits once G was added) and is thus accepted. Its place in the queue is just behind the last blue that *cannot wait* given G, which is found by going from the last blue behind G^* that can possibly wait until the first blue that *can wait* given G is found.

2. the blues, behind the last green G^* in the queue, that can possibly wait;
3. the blues behind G^* that cannot possibly wait;
4. G^* plus the greens and blues ahead of it.

4.1.3 The variable g algorithm

We can make adjustments to the serial DSD, optimised version, in order to make it work with a variable g so that it is compatible with the control loop as described in Section 3.4.2 on page 41. Figure 4.3 provides an example of how it works and the changes are as follows:

- *arrival of green G*: As in the serial DSD, optimised version, G is dropped if the length of the serial queue plus the length of G is more than cd. Otherwise, B_1, the first blue that *can wait* given G, is found. Now G goes ahead of B_1 with probability g. Otherwise, it goes ahead of B_2, the next packet behind B_1, with probability g. This continues until either it goes ahead of some blue or goes at the back of the serial queue.

- *placing of blue B into the serial queue* (either from holding queue or directly upon arrival): A search from the first packet after the last blue in the serial queue to the back of the queue proceeds as follows. If a packet *can wait* given B, the packet B goes in front of this packet with probability $1-g$ or otherwise the search continues.

- *service time*: If there are blue packets in the serial queue, serve the head of the serial queue. Otherwise, consider the head of the serial queue G and the

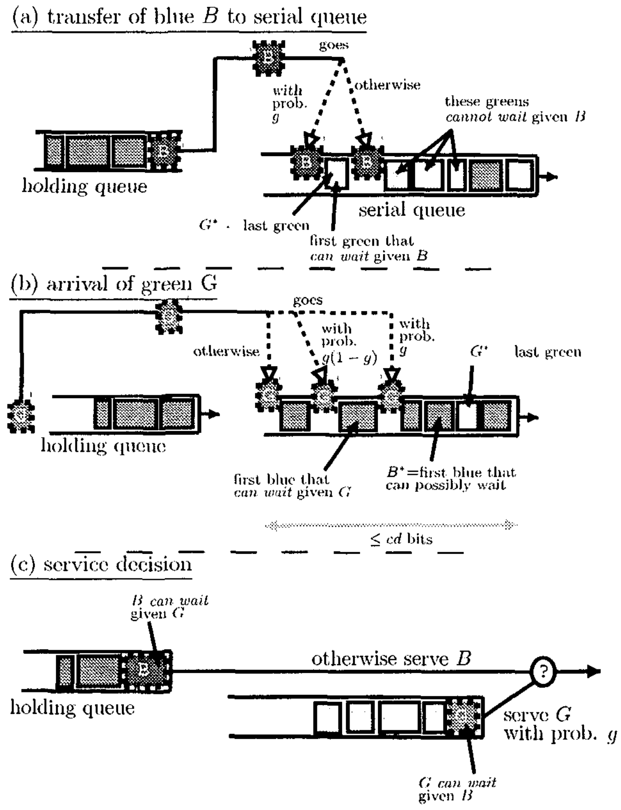


Figure 4.3: Illustration of *serial DSD with variable* g. (a) A blue B leaves the holding queue. The last green that *can wait* given B is G^*, so B goes behind G^* with probability g (based on a random draw). If it succeeds in overtaking G^* then it cannot overtake the next packet (that cannot wait) and thus must go just behind it. (b) A green G arrives and can be accepted. It is put, depending how it fares in the random biased coin tosses (biased by g), at worst behind the last blue in the queue, at best in front of the first blue that *can wait* given G. (c) There are no blue packets in the serial queue. Both B *can wait* given G and G *can wait* given B, so G goes first with probability g.

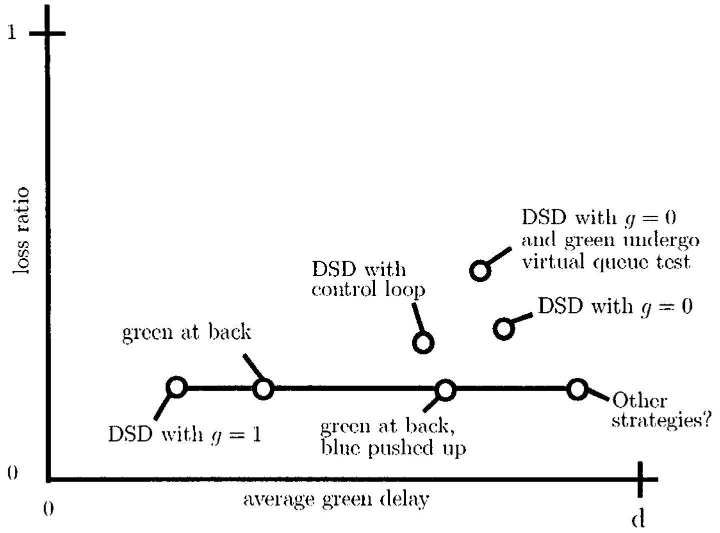


Figure 4.4: *Effect of various algorithms on green delay and green loss* (note the figure is illustrative and does not represent any particular calculated values).

head of the holding queue B. If G *cannot wait* given B then G is served. Otherwise, either *can wait* given the other, so serve G with probability g, and B otherwise.

With the exception of the direct holding queue service, all confrontations that occur in vanilla DSD are decided in advance.

4.2 Maximising green delay

We now discuss strategies that, while preserving the maximum tolerated time spent in the system d, can increase the delay of green packets without increasing the number of green losses.

As stated in Section 3.2.1, page 25, DSD is a solution to the optimisation problem to minimise the number of green losses subject to local transparency, work conservation, the maximum green delay d and no reordering within classes. It also minimises the green delay given the minimum number of green losses.

There is, in a sense, a two-dimensional optimisation problem in green loss and delay given all the above mentioned constraints. Look at Figure 4.4. It illustrates the trade-off of green loss and average green delay for a host of different scheduling

```
Serial DSD, optimised version

$G^*$ = last green accepted and still in serial queue
(let $G^* = 0$ if no green in the serial queue)
$t^*_G$ = time $G^*$ will start service
$B^*$ = first blue behind $G^*$ in serial queue...
        that can possibly wait
(similarly $B^* = 0$ if none exists)
$t^*_B$ = time $B^*$ will start service
```

```
Packet Queueing Algorithm:
packet $p$ arrives at the output port
$dup = p$
Add $dup$ to the virtual queue

if $p$ is blue
{
  if $dup$ was dropped from virtual queue
    drop $p$
  else
  {
    $vl$ = length of the virtual queue
    $p$.deadline = $now + vl/c$
    if $p$.deadline > $now + d$
    {
      add $p$ to holding queue
      if $p$ is the only packet in holding queue
        setHoldingQueueTimer()
    }
    else // $p$.deadline $\leq now + d$
    {
      addBlueToSerial(p)
    }
  }
}
else // $p$ is green
{
  $l_s$ = length of serial queue (in bits)
  if $l_s$+p.$length > cd$
  {
    drop packet
  }
  else
  {
  $p$.deadline = $now + d$
  addGreenToSerial()
  }
}
```

```
Packet Serving Algorithm:
$p$ = first packet in serial queue (serve FIFO)
if $p$ = 0
{
  $p$ = first packet in holding queue
  cancel previous timer
  if holding queue non-empty (after $p$'s removal)
    setHoldingQueueTimer()
}
if $p == G^*$
  $G^*$ = 0 // no longer any green in serial queue
else if $p == B^*$
{
  $t^*_B = t^*_B + B^*$.transDelay
  $B^*$ = $B^*$.next
}
serve $p$
```

```
addGreenToSerial()
{
  if $B^* \neq 0$ // a blue behind $G^*$ that can possibly wait
  {
    $b$ = $B^*$
    $t = t^*_B$
  }
  else if $G^* \neq 0$ // there is a green in serial queue
  {
    $b = G^*$.next
    $t$ = $t^*_G + G^*$.transDelay
  }
  else
  {
    $b$ = 0
    $t$ = $now$
  }
  while $b \neq 0$ // while not reached back of serial queue
  {
    // if $b$ can wait given $p$
    if $b$.deadline − $b$.transDelay $\geq t +$ $p$.transDelay
    {
      insert $p$ in serial queue just in front of $b$
      recalcLastThatCanPossiblyWait()
      exit while loop
    }
    t = t + $b$.transDelay
    $b$ = $b$.next // try next blue in the queue
  } // end of while loop

  $G^* = p$
  $t^*_G$ = $t$
  if $b == 0$ // no blue could wait, $p$ can go at back
    add $p$ to the back of serial queue
}
```

```
addBlueToSerial(q)
{
  add $q$ to the back of serial queue
  if $B^* == 0$ and $q$ can possibly wait
  {
    $B^* = q$
    $l_s$ = length of serial queue (in bits)
    $t^*_B = now + l_s/c$
  }
}
```

```
timerExpires()
{
  $q$ = head of holding queue
  addBlueToSerial($q$)
  if holding queue non-empty
    setHoldingQueueTimer()
}
```

```
recalcLastThatCanPossiblyWait()
{
  $t^*_B$ = $t + p$.transDelay
  $B^* = b$
  // while not reached the end of serial queue
  // and haven't found a packet that can possibly wait
  while $B^* \neq 0$ AND $B^*$ cannot possibly wait
  {
    $t^*_B$ = $t^*_B + B^*$.transDelay
    $B^* = B^*$.next
  }
}
```

Table 4.2: Pseudocode of *Serial DSD, optimised version*. now is the current time, p.deadline denotes the latest time a packet p can remain in the queue, p.next denotes the next packet behind p in the serial queue (which will be served after p), and p.transDelay denotes its transmission delay. setHoldingQueueTimer() is the same as in Table 3.3 on page 45.

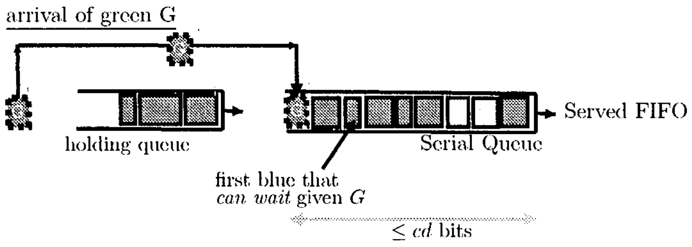


Figure 4.5: Illustration of *green at back* strategy. An arriving green G can finish service in d seconds (because the serial queue had less than cd bits once G was added) and is thus accepted. It is placed at the back of the serial queue. This is in contrast to Figure 4.2 where it is placed just behind the last blue that *cannot wait* given G. Only the green delay is increased, with no increase in the chances of dropping a future green. Note how the algorithm completely avoids queue searches.

algorithms. As stated, DSD is optimal in green delay given the minimum in green losses[1].

When dealing with TCP or TCP friendly sources, increasing average green delay without increasing green losses (at all/much) is desirable for reasons of throughput transparency and stability. Increasing average green delay has the effect of decreasing the rate at which green adaptive flows send, while decreasing blue delay (a direct consequence) has the opposite effect. We stress that increasing the green delay can *increase* green throughput in certain circumstances as it can have the effect of causing green flows to induce *less* losses from the constraints of local transparency.

Shortly, we present two algorithms, *green at back* and *green at back, blue pushed up* where the minimum of green loss ratio is achieved. In Figure 4.4 these are shown along the line of minimal loss. We then answer the question as to whether the second algorithm is optimal in the sense that we cannot increase the green delay more without potentially increasing green losses.

4.2.1 Green at back

Consider the serial DSD, optimised version from Section 4.1.2 on page 50. The *green at back* strategy simply involves placing an accepted green G at the back of the serial queue. This is as opposed to finding the best position by placing G behind the last blue that *can wait* given G.

The strategy is shown in Figure 4.5. The chances of a future green packet being

[1]the optimal in green delay unconstrained by losses is to drop all green packets that cannot immediately begin service

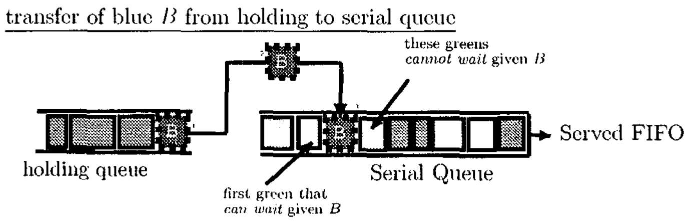

transfer of blue B from holding to serial queue
these greens *cannot wait* given B
Served FIFO
holding queue
Serial Queue
first green that *can wait* given B

Figure 4.6: Illustration of *green at back with blue pushed up* strategy. A blue B is transferred from the holding queue because its deadline is now equal to $now + d$. It is placed ahead of two greens that *can wait* given B, behind the last green that *cannot wait* given B. This is in contrast to Figure 4.2 where it is placed at the back of the green queue. Note that an arriving blue which bypasses the holding queue (its deadline was less than or equal to $now + d$ at arrival) is also placed in the serial queue ahead of the last green that *can wait* given this packet. Only the green delay is increased, with no increase in the chances of dropping a future green.

dropped is not increased. This is because a green is accepted if the number of bits in the serial queue plus its length is less than cd, which is independent of the order of packets in the serial queue.

This first strategy to maximising green delay is doubly advantageous. It makes the scheduling algorithm more efficient and it increases green delay without affecting their loss.

Observe that increasing green delay simultaneously reduces the blue delay, thus bringing the average delays of both classes of traffic closer together. Note that a blue packet that arrives after a green packet cannot be served before the green packet.

4.2.2 Green at back with blue pushed up

Here in addition to green packets being placed at the back of the serial queue, a blue that enters the serial queue, whether via the holding queue or directly upon arrival, is placed as far up the queue as possible while not violating the deadline of any packet. We illustrate this in Figure 4.6. Recall that in the original optimised version of serial DSD, the blue went at the back of the serial queue.

Consider a blue packet B. A search is performed from the back of the serial queue which terminates when either a green that *cannot wait* given B is found, a blue is found, or the search reaches the head of the queue. In the latter case, it is placed at the head of the queue, while in the two former cases it is placed behind the packet which has been found.

Again, a future packet's chance of acceptance is unaffected, as only the arrangement of packets within the serial queue is changed. Note that here it is possible for a blue packet that arrives after a green packet to be served before the green packet.

4.2.3 Other strategies?

The next question to ask is whether we can further increase the average green delay without subjecting the green to further losses. Unfortunately without known restrictions on the arrival process we cannot.

Imagine that the serial queue had only green packets, and that all packets in the serial queue *can wait* given B, the head of the holding queue. Without further information, we cannot know that letting B go ahead of these packets will not cause a future green loss. Immediately after B begins service, a large green packet may arrive. The serial queue is now longer than it would have been had B waited, and may be too large for the arriving green packet.

For a given maximum packet size and a bounded input rate to the queue, one could make a decision as to whether B could go ahead without possibly inflicting losses. In addition, one could ensure the probability of inducing a green loss is kept below some small value.

```
Serial DSD, variable $g$

Packet Serving Algorithm:
$p$ = first packet in serial queue
$b$ = first packet in holding queue
if $p$ == 0 or
  ($B^*$ == 0 and $b \neq 0$
              and $p$ can wait given $b$
              and with probability $1 - g$)
{
  if $b \neq 0$
  {
    serve $b$
    cancel previous timer
    if holding queue non-empty (after $b$'s removal)
      setHoldingQueueTimer()
    clean up any possible stale packets in serial queue
  }
}
else // $p$ exists and if necessary won the biased "toss"
{
  serve $p$
  if $p$ == $G^*$
    $G^*$ = 0 // no longer any green in serial queue
  else if $p$ == $B^*$
  {
    $t^*_B$ = now + $B^*$.transDelay
    $B^*$ = $B^*$.next
  }
}

addGreenToSerial()
{
  if $B^* \neq 0$ // blue behind $G^*$ that can possibly wait
  {
    $b = B^*$
    $t = t^*_B$
  }
  else if $G^* \neq 0$ // there is a green in serial queue
  {
    $b$ = $G^*$.next
    $t = t^*_G$ + $G^*$.transDelay
  }
  else
  {
    $b$ = 0
    $t$ = now
  }
  while $b \neq 0$ // while not reached back of serial queue
  {
    // if $b$ can wait given $p$ and $p$ wins biased "toss"
    if $b$.deadline − $b$.transDelay $\geq t$ + $p$.transDelay
      then with probability $g$:
    {
      insert $p$ in serial queue just in front of $b$
      recalcLastThatCanPossiblyWait()
      exit while loop
    }
    $t$ = $t$ + $b$.transDelay
    $b$ = $b$.next // try next blue in the queue
  } // end of while loop

  $G^*$ = $p$
  $t^*_G = t$
  if $b$ == 0 // $p$ can go at back
    add $p$ to the back of serial queue
}

addBlueToSerial(q)
{
  $r$ = last blue packet in queue
  if $r$ == 0
    $r$ =head of serial queue
  while $r \neq 0$
  {
    if $r$ can wait given $q$
      then with probability $1 - g$:
    {
      insert $q$ just behind $r$
      clean up stale packets (if any)
      exit while loop
    }
    $r$ = $r$.next
  }
  if $q$ was not inserted
    insert $q$ at back of serial queue

  if $B^*$ == 0 and $q$ can possibly wait
  {
    $B^* = q$
    $l_s$ = length of serial queue (in bits)
    $t^*_B$ = now + $l_s/c$
  }
}
```

Table 4.3: Pseudocode of *Serial DSD, g variable*. *now* is the current time, p.deadline denotes the latest time a packet p can remain in the queue, p.next denotes the next packet behind p *in the serial queue (which will be served after p)*, and p.transDelay denotes its transmission delay. The "packet arrival algorithm", timerExpires() and recalcLastThatCanPossiblyWait() are the same as for serial DSD, optimised (Table 4.2 on page 54). setHoldingQueueTimer() is the same as in Table 3.3 on page 45.

Chapter 5

Queueing Analysis of DSD

In this chapter, we:

- present a finite-state description model of DSD that lends itself to queueing analysis and verification (Section 5.1);
- analyse this model under the relatively simple but tractable case of Poisson distributed arrivals/service (Section 5.3, page 65).

The results that follow are an analysis of the DSD queue under the simple model of Poisson arrivals and Poisson service intervals. These are not realistic conditions in which ABE will operate, but it is a useful first step. Possible further more detailed analysis is discussed in the future work section in Chapter 9.

We first describe the finite-state model. We then look at the "rough and ready" notional implementation described in Section 3.1 on page 24 where some numerical results were already presented. Finally, we describe the method of calculation for the finite-state DSD model followed by some numerical analysis using the said method.

5.1 Finite-state description of DSD

We developed a model of DSD queueing in terms of a state description of the numbers of packets in the queue. The motivation for this model is that it provides an analytical tool for the study of the performance of DSD and the effect of the variations of DSD. As a first step in this direction, Section 5.3 on page 65 studies the performance of DSD for the simple but tractable case of Poisson arrivals and services.

The model mirrors serial DSD, simple version, described in Section 4.1.1 on page 49. The main simplification is that things are done on a packet level, so that the model is equivalent to DSD with $g = 1$ when packets have the same length.

A green packet is not accepted if there are K_g packets or more who cannot wait, as opposed to DSD, which guarantees a maximum delay of d to all green packets.[1] Recall Definition 3.3.1 (a packet can possibly wait) and Definition 3.3.2 (a packet *can wait*) on page 33. For this model "can possibly wait" and "can wait" are the same thing.

The current system state is described by means of the general vector

$$(N_v, N_{\not w}, N_w^1, \dots, N_w^{K-K_g}).$$

Now N_v is the current virtual queue length in number of packets. The real queueing system is divided up as follows:

- $N_{\not w}$: the number of blue packets that *cannot wait* plus all accepted green packets;
- N_w^i, $i = 1, \dots, K-K_g$: the number of blue packets that can wait until i green packets go ahead of it.

For convenience, we define $N_w^0 = N_{\not w}$, which considers green packets as able to wait 0 packets. The current number of packets in the real queue is

$$\mathcal{N} = \sum_{i=0}^{K} N_w^i.$$

The total number of packets that *can wait* is

$$\mathcal{N}_w = \sum_{i=1}^{K-K_g} N_w^i.$$

Note that the values N_v and $N_{\not w}$ include the packet that is currently in service. There are three possible events: a blue arrival, a green arrival or the finish of a packet service. The rules for packet acceptance are:

- an arriving blue packet is accepted neither in the real queue nor the virtual queue if the virtual queue is full, i.e. when $N_v = K$.
- an arriving green packet is not accepted into the virtual queue if $N_v = K$. It is not accepted into the real queue if $N_{\not w} \geq K_g$.

A service cannot take place when $N_v = 0$. For a given state of the system, the transitions are as given in Table 5.1.

[1] If L is the maximum length of a packet in the scheduler, then this model guarantees a worst case delay to green of $K_g L/c$ for a link of bandwidth c.

Event	**New state**
Blue acceptance	$N_v \to N_v + 1$. If $\mathcal{N} = 0$ then $N_{\not w} \to 1$; else $N_w^{N_v - \mathcal{N}} \to N_w^{N_v - \mathcal{N}} + 1$
Green acceptance in real queue	$(\max(N_v + 1, K), N_{\not w} + 1 + N_w^1, N_w^2, \dots, N_w^{K-K_g}, 0)$
Green duplicate accepted in virtual queue	$N_v \to N_v + 1$
Service (packet leaves system)	$N_v \to N_v - 1$. If $\mathcal{N} \neq 0$ then $N_{\not w} \to N_{\not w} - 1$. After that, if $N_{\not w} = 0$ and $\mathcal{N} \neq 0$ then let i be the smallest value in $\{1, \dots, K - K_g\}$ such that $N_w^i \neq 0$, and then $N_{\not w} \to 1$ and $N_w^i \to N_w^i - 1$.

Table 5.1: State Transitions for Model of DSD.

These transitions can be explained as follows:

- *Blue acceptance*: The virtual queue increases by one packet since the blue's duplicate was accepted in the virtual queue. If there are no packets in the queue, then this blue must enter the "cannot wait" part of the queue, since it will begin service straight away and pre-empting is not permitted. Otherwise, the arriving blue can allow the same number of green to overtake it as the last blue in the queue, since the same number of spaces exist for it.

- *Green acceptance in real queue*: A green is accepted even if its duplicate is rejected in the virtual queue. A green packet is placed at the back of the packets that *cannot wait*, hence this total increases by one packet. This green has skipped ahead of all blue that *can wait*, which now can wait for one less green packet.

- *Green duplicate accepted in virtual queue*: A green that does not make it in the real queue has its duplicate accepted in the virtual queue if there is space, in order to create a space for future green packets.

- *Service*: The virtual queue length is reduced by one packet. If there is a packet in the real system, the packet at the head of the queue leaves the system. If there are then only packets that *can wait*, the first one must enter the "cannot wait" section in order to receive service next. Note that by Theorem 3.3.3 on page 37, the size of real system is never larger than the size of the virtual one.

5.1.1 Example state space

For those that wish to understand the model through a worked example we provide an example state space for the simple case of $K = 5$ and $K_g = 2$. Starting from the state of an empty queue, one can iterate to produce the entire state space below. Note that for clarity, trailing zeros are suppressed.

0,0	1,1	2,2	3,3	3,2	4,4	4,3	2,1
4,2,1	4,2	5,5	5,4	5,3,1	5,3	1,0	3,1,1
5,2,2	5,2,1	3,1	5,2,0,1	5,2	4,1,2	4,1,1	2,0
4,1,0,1	4,1	5,1,3	5,1,1,1	5,1,0,2	3,0	5,1,0,0,1	

5.1.2 Green undergo virtual queue test

When including the virtual queue drop for green, a green is not accepted if its duplicate is dropped. Thus, the state update upon green acceptance in the real queue becomes $(N_v + 1, N_b + 1 + N_w^1, N_w^2, \ldots, N_w^{K-K_g-1}, 0)$.

5.2 Queueing analysis of "rough and ready"

5.2.1 Method

Recall the rough and ready proposed implementation as described in Section 3.1 on page 24. The model of an output queue for this system is as follows. The queue can store up to K packets, with blue and green packets arriving according to Poisson processes of rates λ_g and λ_b respectively. Packet lengths are exponentially distributed, represented by service times which are Poisson with rate μ.

An arriving green packet is dropped if it finds K_g or more packets in the queue The situation is thus a modified M/M/1/K queue.

Steady State Probabilities

Let $\{P_i, i = 0, \ldots, K\}$ be the steady-state probabilities.[2] The overall arrival rate of the process is $\lambda = \lambda_b + \lambda_g$. The intensity of the system is given by $\rho = \lambda/\mu$, the blue intensity by $\rho_b = \lambda_b/\mu$, and the green intensity by $\rho_g = \rho - \rho_b$. It is not hard

[2] Let $N(t)$ be the number of packets in the queue at time t; then $P_i = \lim_{t\to\infty} \mathbb{P}(N(t) = i)$.

to derive that[3]

$$P_i = \rho^i P_0, \quad i = 0, \ldots, K_g \tag{5.1}$$

and

$$P_i = \rho_b^{i-K_g} \rho^{K_g} P_0, \quad i = K_g{+}1, \ldots, K \tag{5.2}$$

where

$$P_0 = \frac{(1-\rho)(1-\rho_b)}{1-\rho_b-\rho^{K_g}(\rho-\rho_b-\rho_b^{K+1-K_g}(\rho-1))}.$$

Delay and Loss Calculations

The blue loss probability is P_K and the green loss probability is

$$\mathbb{P}(N \geq K_g) = \sum_{n=K_g}^{K} P_n = \rho^{K_g} P_0 \frac{1-\rho_b^{K+1-K_g}}{1-\rho_b}.$$

In steady-state, a typical arriving green will encounter the expected number of packets in the system. Thus, the expected time an accepted green will spend in the system is the amount of time it takes the queue to serve the expected number of packets in the system given that the green was accepted plus the expected time it will take to be served, so

$$\begin{aligned} \mathbb{E}[D_g] &= \frac{1}{\mu}(1+\mathbb{E}[N \mid N < K_g]) \\ &= \begin{cases} \frac{(1+K_g\rho^{K_g+1}-(K_g+1)\rho^{K_g}}{(\mu-\lambda)(1-\rho^{K_g})} & \rho \neq 1 \\ \frac{K_g+1}{2\mu} & \rho = 1 \end{cases}. \end{aligned} \tag{5.3}$$

Similarly the expected time an accepted blue will spend in the system is

$$\mathbb{E}[D_b] = \frac{1}{\mu} + \frac{1}{\mu}\mathbb{E}[N \mid N < K].$$

[3]e.g. follow p.438-440 of [40], with the balance equations given instead by

$$\begin{aligned} \lambda P_0 &= \mu P_1, \\ \{(\mu+\lambda)P_i &= \lambda P_{i-1} + \mu P_{i+1}, \quad i = 1, \ldots, K_g{-}1\}, \\ (\mu+\lambda_b)P_{K_g} &= \lambda P_{K_g-1} + \mu P_{K_g+1}, \\ \{(\mu+\lambda_b)P_i &= \lambda_b P_{i-1} + \mu P_{i+1}, \quad i = K_g{+}1, \ldots, K{-}1\}, \\ \mu P_K &= \lambda_b P_{K-1}. \end{aligned}$$

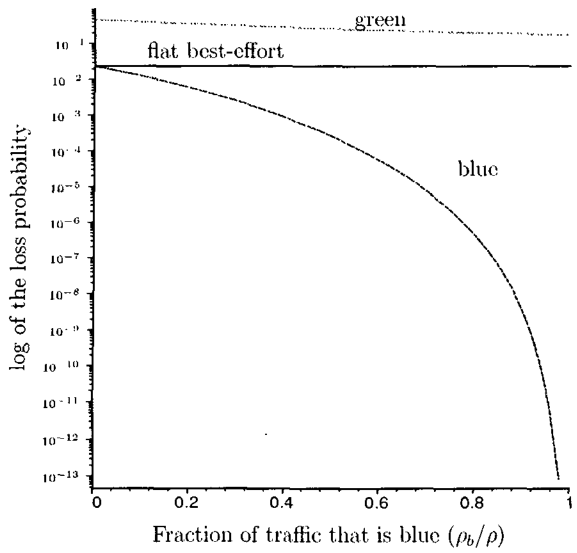


Figure 5.1: The loss probability (log scale) for green, blue and flat best-effort in the "rough and ready" notional implementation, as a function of the proportion of traffic that is green (ρ_g/ρ). The total intensity of traffic ρ is fixed at 0.8 (other parameters were $K = 10, K_g = 3$).

Flat Best-Effort

The results for flat best-effort follow directly from page 440 of [40]. The loss probability is

$$P_{\text{loss}} = \begin{cases} \frac{(1-\rho)\rho^K}{1-\rho^{K+1}} & \rho \neq 1 \\ \frac{1}{K+1} & \rho = 1 \end{cases} \tag{5.4}$$

and the expected time spent in the system is

$$\mathbb{E}[D_f] = \begin{cases} \frac{1+K\rho^{K+1}-(K+1)\rho^K}{(\mu-\lambda)(1-\rho^K)} & \rho \neq 1 \\ \frac{K+1}{2\lambda} & \rho = 1 \end{cases}. \tag{5.5}$$

5.2.2 Numerical results

We already showed in Section 3.1 on page 24 the poor performance in terms of loss the green receive under this notional implementation as a function of the traffic intensity ρ, when the proportion of blue and green traffic is the same (i.e. $\rho_g = 0.5\rho$).

Now consider Figure 5.1 where ρ is fixed but the fraction of traffic that is blue,

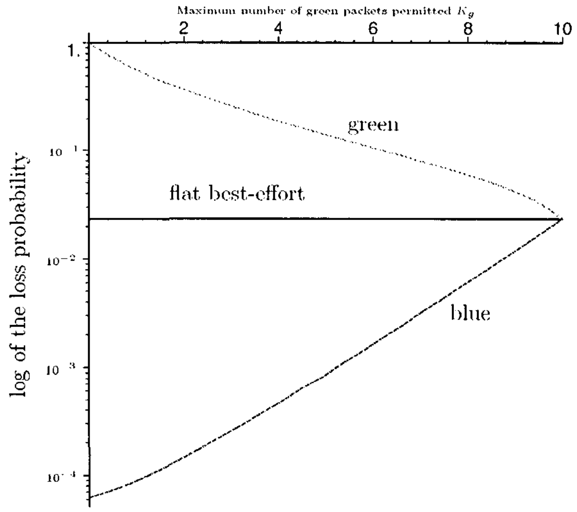


Figure 5.2: The loss probability (log scale) for green, blue and flat best-effort in the "rough and ready" notional implementation, as a function of K_g; a green is only accepted if there are $K_g - 1$ or less packets in the queue. (all other parameters were fixed as follows: $K = 10, \rho = 0.8, \rho_b = 0.5$).

ρ_b/ρ, is varied. The green loss ratio remains the same since P_{K_g} is independent of ρ_g, and the loss probability for green naturally decreases, albeit slowly, as the intensity of blue traffic decreases. Figure 5.2 shows that as K_g increases linearly, the loss probability decreases exponentially, so the lower the green delay, the much higher the tradeoff in terms of loss.

5.3 Queueing analysis of DSD

Now let us examine the DSD model as described in Section 5.1 on page 59 under Poisson arrival processes, with blue rate λ_b and green rate λ_g. Let $\lambda = \lambda_b + \lambda_g$, and let packets be served exponentially with service rate μ.

5.3.1 Method

Steady State Probabilities

Let $\mathcal{S}$ be the state-space generated from the state rules and $\mathbf{p} = (p_i)$ be the steady-steady probability vector of size $|\mathcal{S}|$, containing elements $1, \ldots, |\mathcal{S}|$. Let P be a

$|\mathcal{S}| \times |\mathcal{S}|$ matrix where each element $P_{i,j}$ represents going from state $i \in \mathcal{S}$ to state $j \in \mathcal{S}$, and defined as follows:

$$P_{i,j} = \begin{cases} \lambda_b + \lambda_g & j \text{ reachable from } i \text{ by blue or green arrival} \\ \lambda_b & j \text{ reachable from } i \text{ by blue arrival only} \\ \lambda_g & j \text{ reachable from } i \text{ by green arrival only} \\ \mu & j \text{ reachable from } i \text{ by packet service} \\ 0 & \text{otherwise} \end{cases}$$

where $i, j \in \mathcal{S}$.

This is not precisely the transition matrix but chosen for its suitability in calculation, and for convenience $P_{i,i} = 0$ for all $i \in \mathcal{S}$. The rate of entry into a state $s \in \mathcal{S}$ is

$$\sum_{i \in \mathcal{S}} p_i P_{i,s} \tag{5.6}$$

and the rate of leaving s is

$$p_s \sum_{i \in \mathcal{S}} P_{s,i}. \tag{5.7}$$

In steady-state these equations are equal, resulting in the balance equations. The input into all states, Equation (5.6), can be expressed by the vector $P^{\mathrm{T}}\mathbf{p}$, and the output from all states, Equation (5.7), by the vector $K\mathbf{p}$ where K is composed by summing the rows of P, and placing the resultant vector along the diagonal of an $|\mathcal{S}| \times |\mathcal{S}|$ matrix[4]. The steady-state probabilities $\mathbf{p}$ can then be obtained by solving

$$P^{\mathrm{T}}\mathbf{p} = K\mathbf{p}$$

together with the requirement that the probabilities sum to 1,

$$\sum_{s \in \mathcal{S}} p_s = 1.$$

[4]In "matlab" notation this would be expressed

$$K = \mathrm{diag}(sum(P^{\mathrm{T}})^{\mathrm{T}}).$$

Loss Calculations

The probability of a blue loss is obtainable directly by examining the virtual queue, and so by Equation (5.4),

$$P_{\text{loss}}^{\text{blue}} = P_{\text{loss}} = \frac{(1-\rho)\rho^K}{1-\rho^{K+1}}.$$

The probability of a green loss is

$$P_{\text{loss}}^{\text{green}} = \mathbb{P}(N_{\psi} \geq K_g).$$

Delay Calculations

An arriving green is accepted whenever $N_{\psi} < K_g$. If accepted, the expected delay is the time it takes to serve all packets ahead that *cannot wait* plus the expected time to serve the packet itself, i.e.

$$\mathbb{E}[D_g] = \frac{1}{\mu}(1 + \mathbb{E}[N_{\psi} \mid N_{\psi} < K_g])$$

which can be calculated from the steady-state probability vector $\mathbf{p}$.

Calculating the expected delay for blue in the same manner would not be straightforward since it involves consideration of the green packets that arrive after a blue but which receive priority whenever the blue *can wait*. Instead, we observe the conservation properties of the queueing system [23] and Little's law to obtain that

$$(1 - P_{\text{loss}}^{\text{blue}})\lambda_b \mathbb{E}[D_b] + (1 - P_{\text{loss}}^{\text{green}})\lambda_g \mathbb{E}[D_g] = \mathbb{E}[\mathcal{N}],$$

and thus that

$$\mathbb{E}[D_b] = \frac{\mathbb{E}[\mathcal{N}] - (1 - P_{\text{loss}}^{\text{green}})\lambda_g \mathbb{E}[D_g]}{(1 - P_{\text{loss}}^{\text{blue}})\lambda_b}. \tag{5.8}$$

(Recall from Section 5.1 on page 59 that $\mathcal{N}$ is the number of packets in the system)

Green undergo virtual queue test

The loss rate for green is

$$P_{\text{loss}}^{\text{vqtest}} = \mathbb{P}(N_{\psi} \geq K_g \cup N_v = K]$$

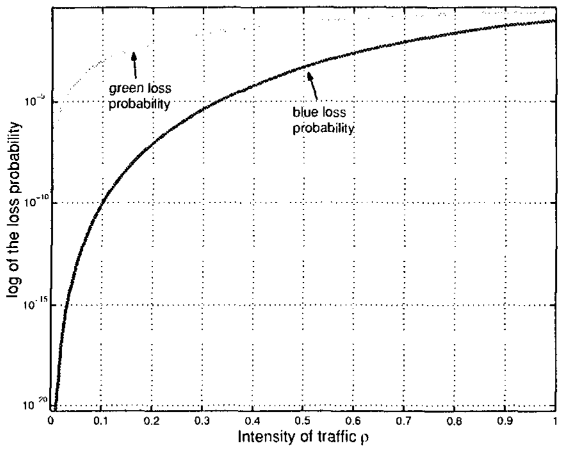


Figure 5.3: The loss probability (log scale) for green and blue traffic using DSD, as a function of the total intensity of traffic. The proportion of traffic is 50% green and 50% blue. Note that, due to the virtual queue, the loss probability in flat best-effort is the same as the blue loss probability. (parameters were $\rho_g = 0.5\rho, K = 10, K_g = 3, \lambda = 40$)

and the expected delay is

$$\frac{1}{\mu}(1 + \mathbb{E}[N_{\psi} \mid N_{\psi} < K_g \cap N_v < K]).$$

5.3.2 Numerical results

Figure 5.3 shows the loss probability as the total level of traffic increases, when there are equal proportions of green and blue traffic. Recall that the loss probability in flat best-effort is equal to the blue loss probability. The corresponding expected delay for green and blue is shown in Figure 5.4. The reduction in green delay necessitates a higher loss rate for green in order to provide local transparency, but the increase is the minimum possible.

When there is little traffic, the difference in loss rates is higher, at the same time as the queueing delays are small. In a region of medium to high levels of overall traffic, there is less difference between green and blue loss rates (although the overall number of losses increase). It is also at this point that green traffic needs the delay distinction. In summary, when the green need lower delay most, the price they pay in increased losses is not as high, justifying the trade-off and indeed the need for ABE.

Now consider Figure 5.5, which shows how the loss probability varies when the

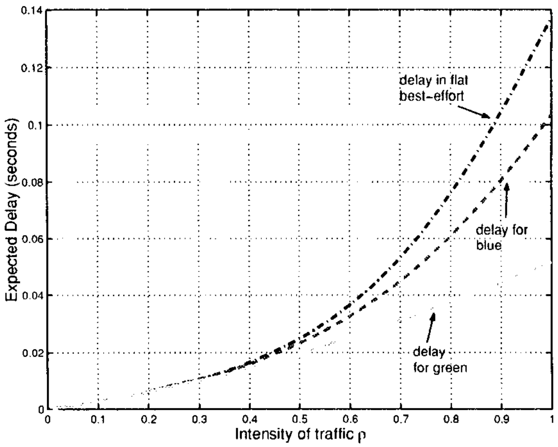


Figure 5.4: The expected delay for green and blue traffic using DSD and in flat best-effort traffic, as a function of the total level of traffic. The proportion of traffic is 50% green and 50% blue (parameters were $\rho_g = 0.5\rho, K = 10, K_g = 3, \lambda = 40$).

overall amount of traffic is constant but the percentage of green traffic is varied. The blue loss probability does not change, as it is a function of the total intensity of traffic. As the amount of traffic that becomes green increases, the overall green loss probability drops with decreasing rate of change. If there is little green traffic then there are less discriminatory drops for green, and less traffic to take advantage of them. Figure 5.6 shows the expected delay for blue and green using DSD compared to flat best-effort. The average blue delay decreases as it benefits from less amount of traffic being accepted. The green delay, seen more clearly in Figure 5.7, increases initially before decreasing, although the range of values is very small. There are three factors interacting: blue traffic decreasing which decreases green delay; green traffic increasing which increases green delay; and green losses increasing which decreases green delay. Initially, the extra amount of green traffic causes the increase but its effect becomes gradually less until eventually the other two factors dominate.

Finally, in Figures 5.8 and 5.9 we look at the case when the value of K_g, the maximum number of *cannot wait* packets permitted, is varied. The total intensity and proportions of blue and green traffic is fixed. The green traffic can pay a high price in terms of loss for a lower maximum delay which has ramifications for the choice of d by an operator.

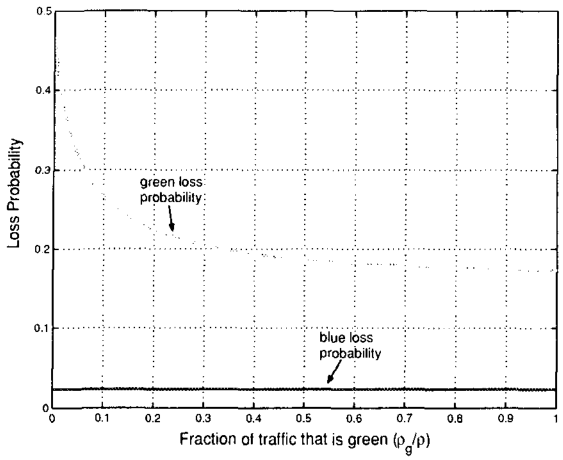


Figure 5.5: The loss probability for green and blue traffic using DSD, as a function of the fraction of traffic that is green. The total amount of traffic is constant. Note that the blue loss probability is only a function of the total amount of traffic and invariant to the amount of green and blue traffic. (parameters were $\rho = 0.8, K = 10, K_g = 3, \lambda = 40$)

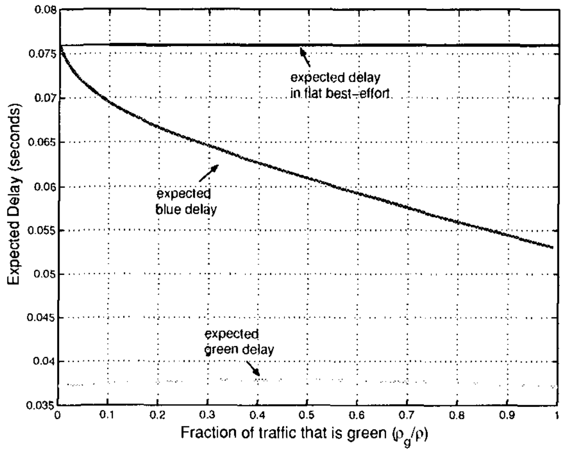


Figure 5.6: The expected delay for green and blue traffic using DSD and in flat best-effort traffic, as a function of the fraction of traffic that is green. The total amount of traffic is constant. In Figure 5.7, the delay for green is shown by itself in order to demonstrate its behaviour. (parameters were $\rho = 0.8, K = 10, K_g = 3, \lambda = 40$)

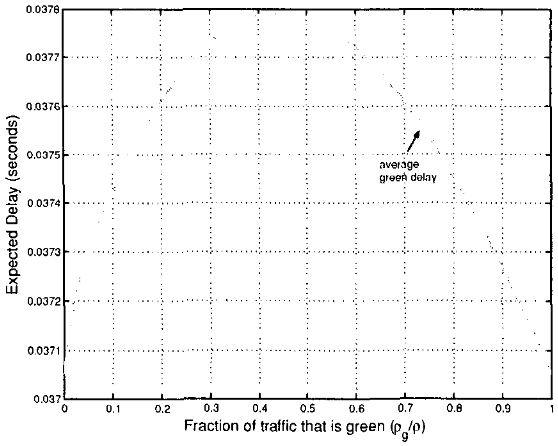


Figure 5.7: A close-up of the delay behaviour for green traffic from Figure 5.6. Initially all traffic is blue. Then, as the percentage of green traffic increases, so does the delay, up to a maximum level. Then the increase in the number of green losses means that the delay comes down.

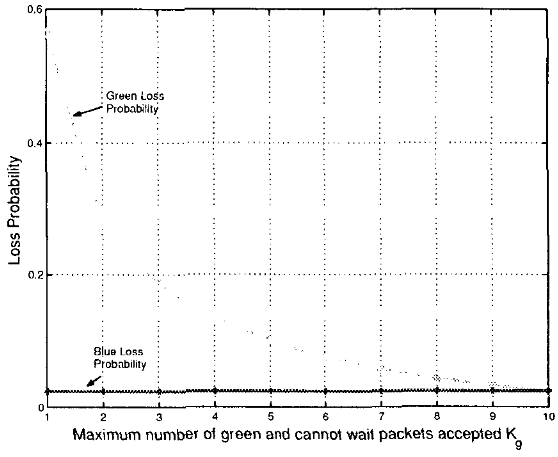


Figure 5.8: The loss probability for green and blue traffic using DSD, as a function of K_g, the size of the buffer green packets "see". The total intensity of traffic and proportion of it that is green is constant. (parameters were $\rho = 0.8, \rho_g = 0.4, K = 10, K_g = 3, \lambda = 40$)

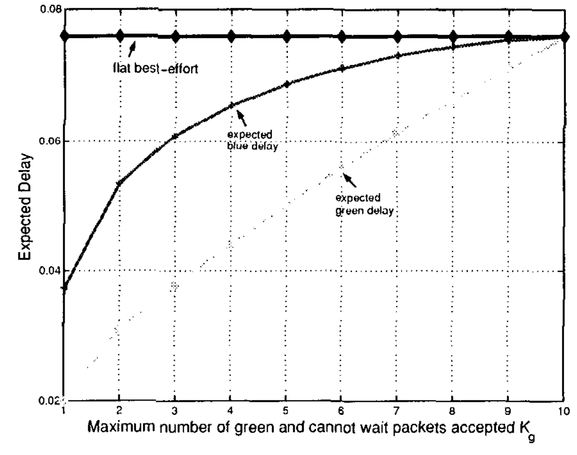


Figure 5.9: The expected delay for green, and blue traffic using DSD, as a function of K_g, the size of the buffer green packets "see". The total amount of traffic and proportion of it that is green is constant. (parameters were $\rho = 0.8, \rho_g = 0.4, K = 10, K_g = 3, \lambda = 40$)

Chapter 6

DSD Simulation Study

In this chapter we examine the performance of DSD and its variants under simulation given a large amount of different conditions. The simulation tool is ns-2 [36] which has become the *de facto* discrete-event simulator of Internet protocols and traffic control.

The fundamental things to establish are that:

- *green does not hurt blue*: the throughput for blue must be at least as good as it would in flat best-effort;
- green receive acceptable performance in which to operate.

We first look, in Section 6.1, at a scenario where there are blue and green TCP flows, sharing a bottleneck link, that all have the same non-queueing round-trip time. Then, in Section 6.2 on page 82 we look at the more general case where round-trip times are different, the green traffic uses a TCP friendly protocol, and there is non-adaptive traffic present.

The main finding is that DSD, with the control loop and the *green undergo virtual queue test*, ensures green does not hurt blue while still providing green with a reasonable level of throughput. Also, we recommend the *green at back* test as it reduces scheduling complexity of green packets but does not have a negative impact when used in conjunction with the control loop.

6.1 TCP and equal round-trip times

6.1.1 Simulation description

Figure 6.1 describes the general simulation topology. For each simulation run, n blue flows and m green flows compete along a bottleneck link of speed 10Mbps. All

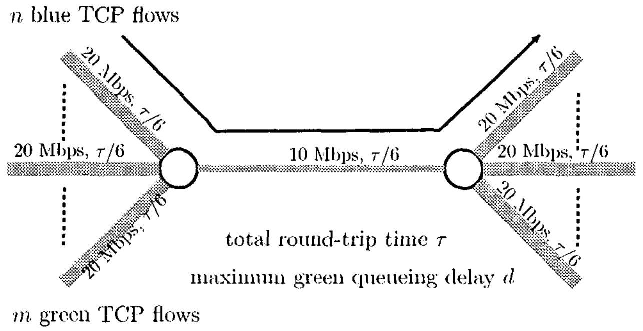


Figure 6.1: n blue TCP flows, m green TCP flows, all of non-queueing round-trip time τ.

---

flows are the SACK variant of TCP [30], have non-queueing round-trip time τ, and send packets of 1000 bytes. The maximum time green can spend in the system is d and the virtual queue buffer size is equal to the bandwidth-delay product (i.e. 1250τ packets).

The duration of each simulation is 300 seconds. To avoid synchronisation and to add an element of realism, every packet sent by an application is held for a period of time randomly distributed between 0 and 10 milliseconds. This is to represent random overhead in packet sending. Five simulation runs are performed and the average and confidence of the result ascertained. We omit the confidence interval whenever it is so small that it makes the graphs unreadable.

6.1.2 The "rough and ready" implementation

Firstly, we confirm that the "rough and ready" notional implementation is not practical. Figure 6.2 shows the average number of packets received by 30 green and 30 blue TCP flows as a function of time. In flat best-effort, both receive approximately the same throughput, with oscillations around the average as you would expect. Using "rough and ready", the blue traffic gets all of the throughput soon after the slow-start period. Most green packets are dropped since the buffer is nearly always too full for them to leave the system within $d = 0.1$ seconds of arrival.

6.1.3 Fixed amounts of traffic

Local transparency can mean throughput transparency

We show, in Figure 6.3, a case where throughput transparency is achieved while using DSD with the green bias fixed at $g = 1$. In fact, the blue sources get more

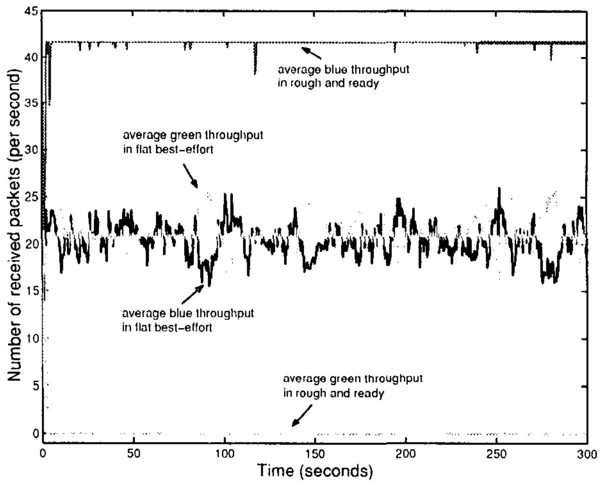


Figure 6.2: The throughput TCP sources received in the "rough and ready" notional implementation. The green flows eventually get no throughput because the buffer is too full and there is no green packets can make their deadline. (parameters were $n = 30, m = 30, \tau = 0.2, d = 0.1$)

throughput than in flat best-effort, while the green sources receive a reasonable share of the bandwidth and a lower bounded queueing which is shown by the delay histogram in Figure 6.4. The relative increase in loss rate for green was enough to compensate for the decrease in delay green received.

This was a very lightly loaded network where there were very few losses. The blue and green loss ratio was 0.13% and 0.28% respectively, extremely small; but green had twice as many losses. This reflects the situation as we saw in the numerical results of the modelling in Section 5.3.2 on page 68 where the less traffic in the network, the higher the relative loss ratio, and the lower the difference in average delay.

We now look at a relatively more heavily loaded network, where there there are $n = 30$ blue and $m = 30$ green flows and, without the control loop, the lower delay for the green TCP flows enables them to receive more throughput than their blue counterparts.

Local transparency does not imply throughput transparency

Figure 6.5 shows the average throughput received by 30 blue and 30 green flows in flat best-effort and using DSD when the green bias g is fixed at 1 (always favouring green

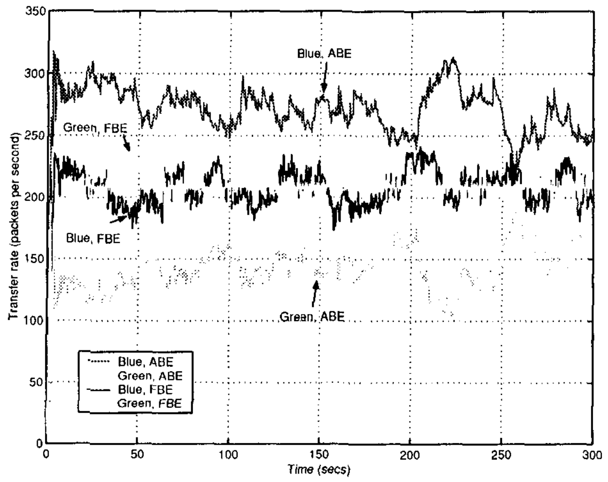


Figure 6.3: The throughput TCP sources receive using DSD with the green bias $g = 1$, and using flat best-effort. In this case, although not in general, local transparency is sufficient for throughput transparency. (parameters were $n = 3, m = 3, \tau = 0.2, d = 0.1$)

---

in both can wait situations). Here, throughput transparency does not hold, with green benefitting from substantially *increased* throughput and very low queueing delay.

Control loop ensures throughput transparency

Figure 6.6 shows that the average blue throughput is larger than the green throughput, ensuring that green does not hurt blue, when using the standard set-up for DSD, which is with the control loop and the *green undergo virtual queue test*. Figure 6.7 shows the how the *green undergo virtual queue test* increases throughput for blue, but reduces it for green. (Note that to make things visible, the smoothing filter of the transfer rate is larger than in Figure 6.6.)

The effect on green delay of the *green undergo virtual queue test* can be seen in Figure 6.8. It changes dramatically the green delay profile, because it causes the rejection of green packets at precisely the moments when the queue would be relatively full for green, namely when the virtual queue is full. Also, if there is a period where mostly green are accepted and not blue, a burst of green has a higher chance of being accepted.

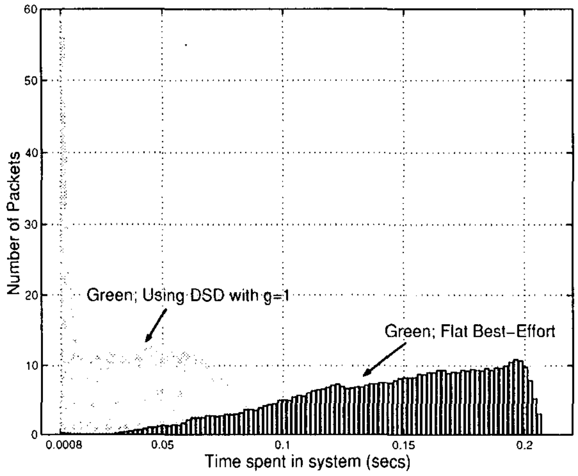


Figure 6.4: Histogram of delay experienced by green traffic using DSD with the green bias $g = 1$, and using flat best-effort. Note that the minimum delay possible is the transmission time of .8ms. Using DSD, the delay was bounded to 0.1 seconds. (bins of size 100, normalised to be of unit area, $n = 3, m = 3, \tau = 0.2, d = 0.1$)

Green at back, blue pushed up

Next we show the effect of the strategies *green at back* and *green at back, blue pushed up*.

Consider Figure 6.9, which shows the average throughput for blue and green flows with the control loop, with the *green at back* strategy either on or off. The throughput for blue is only slightly increased by employing the strategy. This is because the control loop drives the system into delaying green packets anyway. Thus, it appears that *green at back* is a good way to reduce scheduler complexity. Figure 6.10 shows the effect of this strategy and that of additionally *blue pushed up* have on green delay. *Blue pushed up* drives the green queueing delay towards d.

Effect of total amount of traffic

In Figure 6.11 we look at how DSD (with control loop) performs as the number of sources increases. The average number of packets per flow decreases as you would expect, and throughout green does not hurt blue.

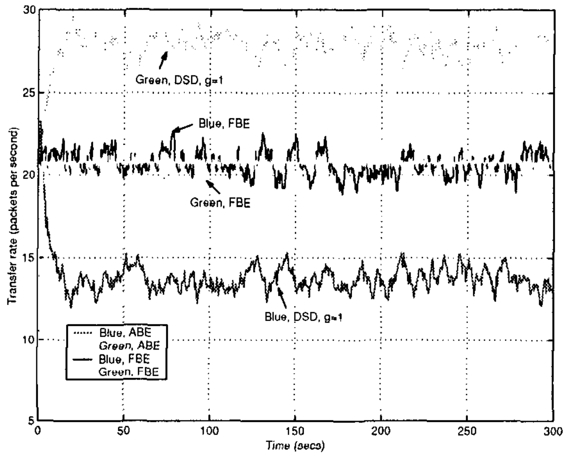


Figure 6.5: The average throughput for 30 blue and 30 green TCP sources under DSD with the green bias $g = 1$ and under FBE (flat best-effort). This shows that local transparency is not sufficient for throughput transparency. The green flows in this case receive substantially more throughput when $g = 1$, and hence the need for a control loop (parameters were $n = 30, m = 30, \tau = 0.2, d = 0.1$)

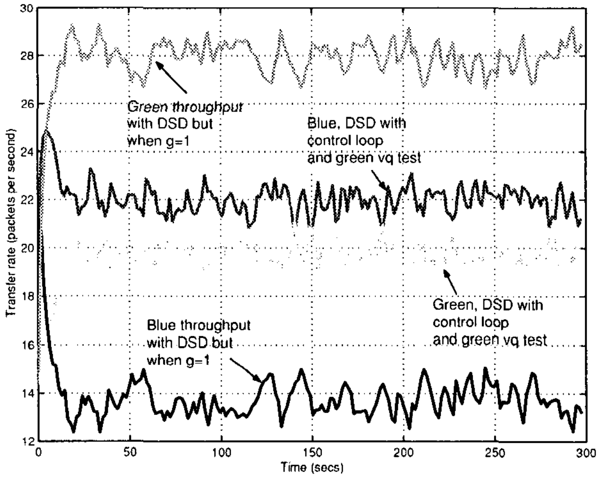


Figure 6.6: The throughput of 30 green and 30 blue TCP sources with and without the control on the green bias parameter. The margin γ of the control loop was set to give the blue 10% more throughput than the green. (parameters were $n = 30, m = 30, c = 10Mbps, \tau = 0.2, d = 0.1, \gamma = 1.1, T = 0.5, K = 1.1, \alpha = 0.4, \tau_e = 0.2$)

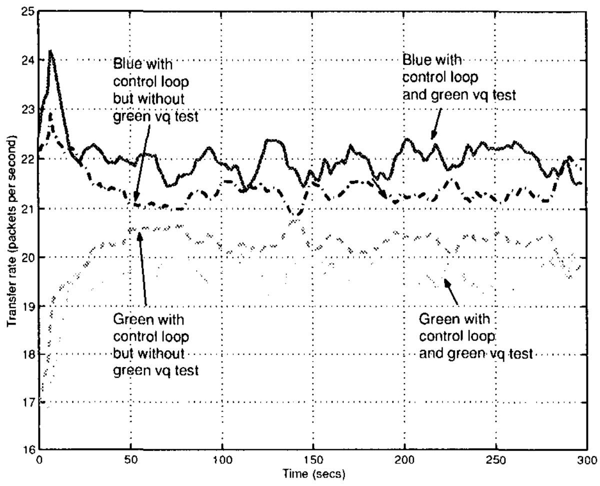


Figure 6.7: Performance effect of the *green undergo virtual queue test*. By preventing green from being accepted when blue cannot, the blue throughput increases.

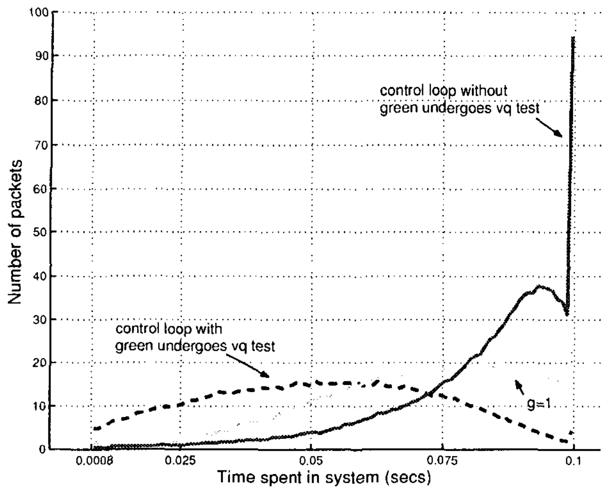


Figure 6.8: Density plot of queueing delay received by green traffic using DSD and under the following restrictions: $g = 1$ and using control loop with or without the *green undergo virtual queue test*. (parameters are the same as in Figure 6.6)

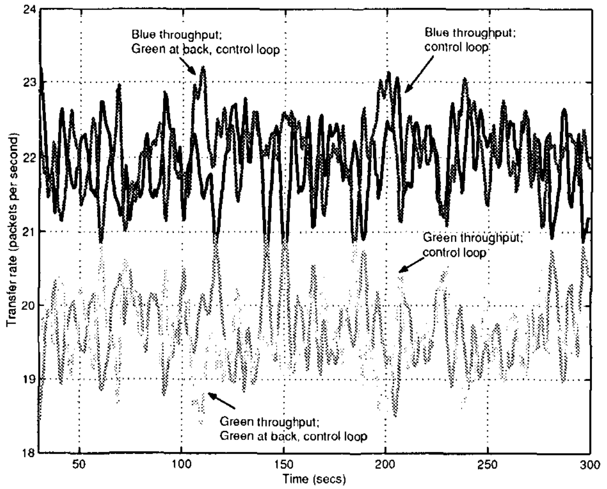


Figure 6.9: When using the control loop, green at back has no significant effect on performance. We thus recommend its use to reduce the complexity of the scheduler.

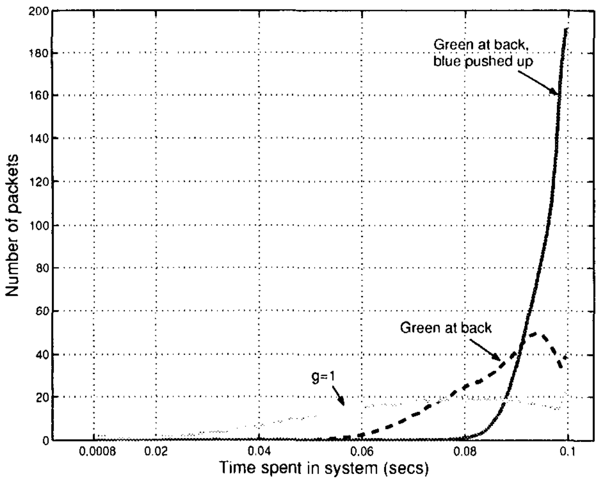


Figure 6.10: *Green at back, blue pushed up* drives the green queueing delay towards the maximum. *Green at back* on its own also does this, but less intensely. (parameters were $n = 30, m = 30, \tau = 0.2, d = 0.1$)

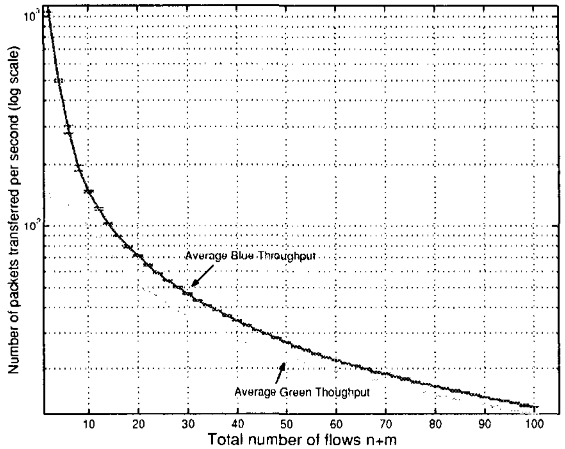


Figure 6.11: Average throughput (log scale) as a function of number of flows, when using DSD (with control loop and green undergo virtual queue test). The overall average throughput naturally decreases as the number of flows increase, but the control loop ensures that green does not hurt blue. There is an equal number of blue and green flows ($n = m$). Error bars are for 95% confidence.

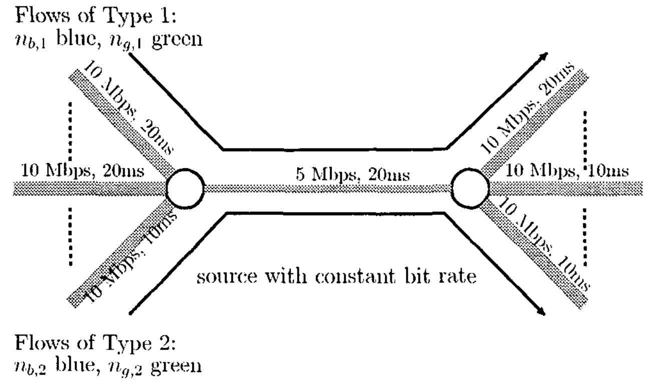


Figure 6.12: Simulation Topology II

6.2 TCP friendly and variable round-trip times

We now look at when there are flows of various round-trip times, and where green flows may be either TCP friendly or non-TCP friendly.

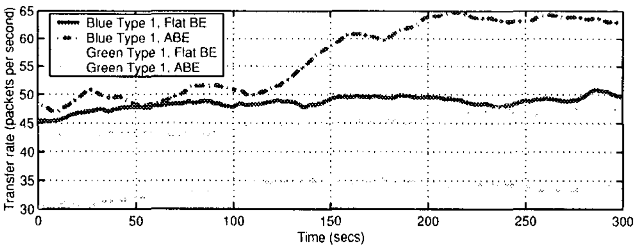
6.2.1 Simulation description

There are $n_{b,1}$ blue sources and $n_{g,1}$ green sources with an outgoing link propagation delay of 20ms (sources of type 1), and $n_{b,2}$ blue and $n_{g,2}$ green sources with an outgoing 10Mbps link of propagation delay 50ms (sources of type 2). All sources pass the 5Mbps link L of propagation delay 20ms, and terminate via a 10Mbps link of propagation delay 10ms. These blue sources are the SACK variant of TCP, and the green sources are the TCP friendly algorithm as described in [3]. There is also green traffic which sends a constant rate r (CBR) and passes through the link L.

The router buffer size was 60 packets (i.e. $Buff = 60$) and the maximum time a green can spend in the system, d, was $0.048s$. For simplicity, the size of all packets is fixed at 1000 bytes. The control loop updates its value of g every $0.5s$ (i.e. $T = 0.5$), the gain parameter α was 1.1, and the conservative value of $20ms$ was taken to be the round-trip time used for estimating throughput. Each simulation ran for 300 seconds of simulated time.

6.2.2 Equal fraction of green and blue

We first examine some scenarios when there are only TCP and TCP friendly flows. For the case where there are 5 blue TCP and 5 green TCP friendly flows of each type ($n_{b,1} = n_{b,2} = n_{g,1} = n_{g,2} = 5$), Figure 6.13 shows the average transfer rate for



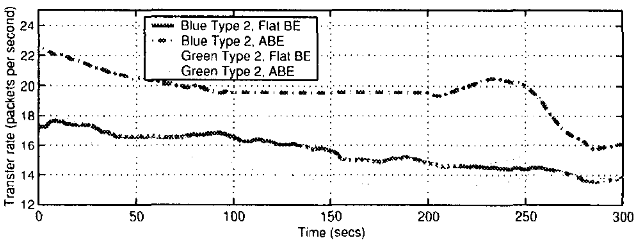


Figure 6.13: Average packet transfer rate for green and blue connections, as a function of time t, when the router implemented DSD and when it implemented flat best-effort. The results are obtained by simulating the network described on Figure 6.12, with 5 blue flows and 5 green flows of each type, namely $n_{b,1} = n_{b,2} = n_{g,1} = n_{g,2} = 5$, and no CBR traffic

each blue and green connection, of both types at each time t. Figure 6.14 shows the end-to-end delay distributions received for green packets under ABE and flat best-effort. Blue flows of each type receive more throughput with ABE than the did in flat best-effort, thus benefitting from the use of ABE. Green flows receive less, and in exchange, the green queueing delay is small and bounded by $d = 0.048s$. The green loss ratio was 4.97% when using ABE, and 3.3% in the flat best-effort, while the blue loss ratio decreased from 3.2% to 2.5% when moving to ABE. The extra throughput that blue flows of type 1 receive over type 2 flows follows from the lower round trip-time they experience.

6.2.3 Asymmetry in numbers of blue and green

ABE is designed to work independently of asymmetry in the amount of green and blue traffic. For the case where there are 5 blue TCP and 3 green TCP friendly flows of type 1 ($n_{b,1} = 5$, $n_{g,1} = 3$) and 3 blue TCP and 5 green TCP friendly flows of type 2 ($n_{b,2} = 3$, $n_{g,2} = 5$), Figure 6.15 shows that again green does not hurt blue.

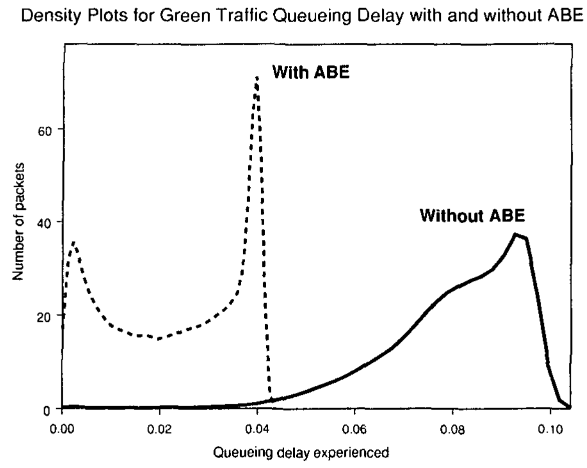


Figure 6.14: Density plot of waiting time for green packets under DSD and flat best-effort; 5 blue TCP and 5 green TCP friendly flows of each type

6.2.4 TCP and constant bit-rate sources

The situation where blue traffic is TCP, and green traffic is no longer TCP friendly, but a constant bit-rate source is now examined. Here there are 5 blue TCP flows of each type ($n_{b,1} = n_{b,2} = 5$) and CBR green traffic which sends at 1Mbps. The number of packets received for each blue traffic type and for the CBR source is shown as a function of time in Figure 6.16. What we see is that the blue traffic receives slightly more throughput with DSD than with flat best-effort, due to the local transparency property, and the non-TCP friendly CBR traffic receives less.

6.2.5 TCP, TCP friendly and constant bit-rate

We now look at the scenario where there is blue TCP traffic ($n_{b,1} = n_{b,2} = 5$), and green traffic is composed both of TCP friendly sources ($n_{g,1} = n_{g,2} = 5$) and CBR traffic of rate 1Mbps. The average packet transfer rate for the blue and green of type 1, and for the CBR source as a function of time is shown in Figure 6.17. The results for type 2 traffic is omitted for ease of reading.

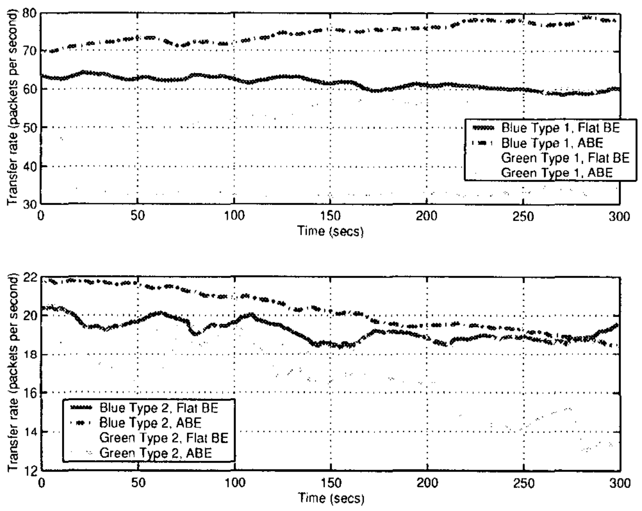


Figure 6.15: Average packet transfer rate per green and blue connection, as a function of time t, when the router implemented DSD and when it implemented flat best-effort. The results are obtained by simulating the network described on Figure 6.12, with $n_{b,1} = 5, n_{g,1} = 3, n_{b,2} = 3, n_{g,2} = 5$.

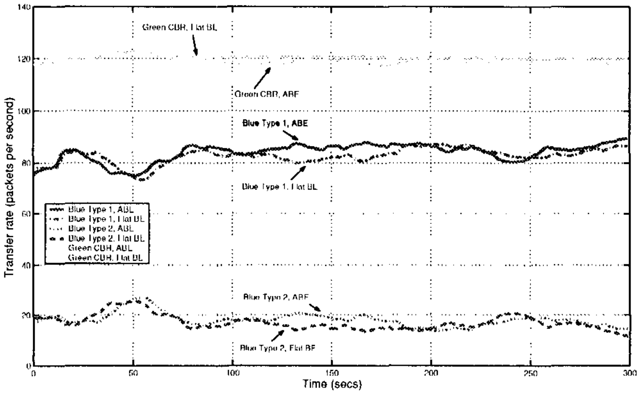


Figure 6.16: Average packet transfer rate per green and blue connection, as a function of time t, when the router implemented DSD and when it implemented flat best-effort. There are 5 blue flows of each type and a CBR flow of 1Mbps which is green.

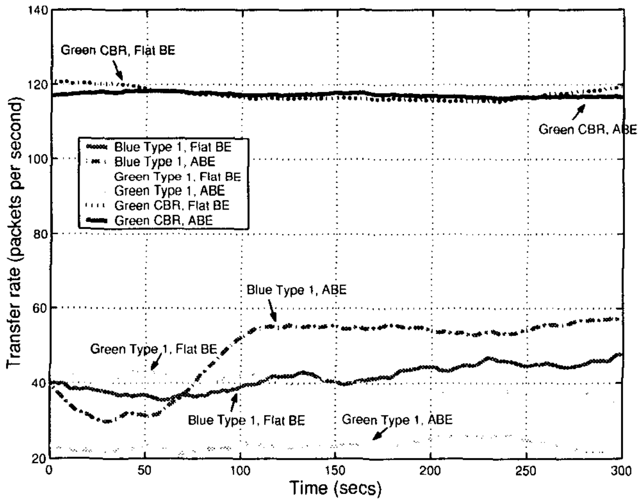


Figure 6.17: Average packet transfer rate per green and blue connection of Type 1, and for the CBR source as a function of time t, when the router implemented DSD and when it implemented flat best-effort. The CBR source sends at 1Mbps and there are 5 of each other type of source.

Chapter 7

ABE Implementation: EDF

We now describe ABE/EDF (Earliest Deadline First), the first ABE implementation that we created. It provides a low bounded queueing delay to green while ensuring throughput transparency.

It was developed prior to the invention of the concept of local transparency, and thus, in contrast to DSD in Chapter 3, it relies on flows being TCP friendly, which is a major disadvantage. We do note however that one advantage to implementing throughput transparency directly is that there is extra freedom to delay blue packets. In addition, one other difference is that number of blue packets accepted is on average higher than in the virtual queue.

7.1 Introduction

The design of ABE/EDF is outlined in Figure 7.1. It consists of two main modules: a Packet Admission Control (PAC) module and a scheduler. The PAC module man-

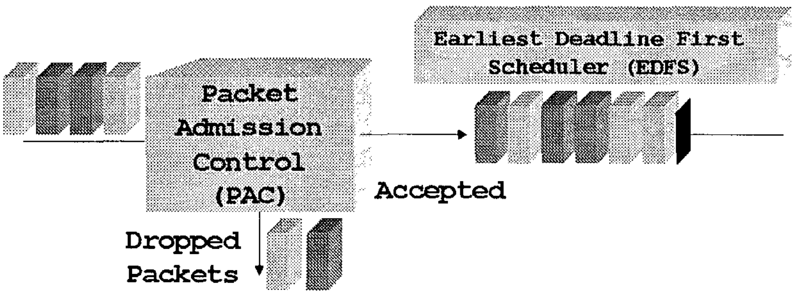


Figure 7.1: Overview of ABE/EDF's Router Support.

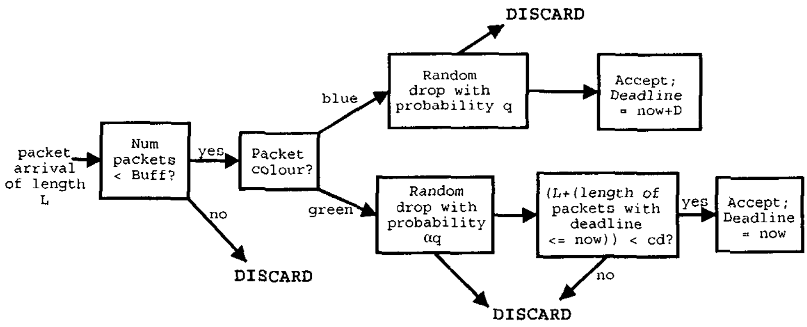

DISCARD
packet arrival of length L
Num packets < Buff?
yes
no
DISCARD
Packet colour?
blue
green
Random drop with probability q
Accept; Deadline = now+D
Random drop with probability αq
(L+(length of packets with deadline <= now)) < cd?
yes
no
Accept; Deadline = now
DISCARD

Figure 7.2: Summary of PAC Algorithm for ABE/EDF.

ages the queue by dropping packets whenever necessary or appropriate, acceptance being biased in favour of blue packets. *It must ensure that sufficient green packets* are dropped in order to prevent blue flows from suffering and must not accept green packets if they would experience delay greater than a specified bound d, where d is maximum time a green packet can spend in the system.

The scheduler determines which packet, if any, from the buffer should be sent next. Typically, it is biased towards giving green packets less queueing delay. A classifier is also required for identifying the traffic class of the incoming packet, i.e. whether it is green or blue. The PAC controls the dropping of green packets such that blue traffic receives as much throughput as if the network was flat best-effort, while trying to keep these green losses to a minimum. Let q_b and q_g be the drop ratio of blue and green traffic respectively. The PAC attempts to have the following relation hold,

$$q_g = \alpha q_b$$

where $\alpha \geq 1$ is a controlled parameter, called the *drop bias*, which provides a drop disadvantage to green packets.

7.2 Description

The PAC uses *Random Early Detection (RED)* [12] to obtain an initial probability q of dropping a packet. RED is a congestion avoidance mechanism, such that when the average queue size exceeds a pre-set threshold, the router drops each arriving packet with a certain probability which is a function of the average queue size. The modification is as shown in Figure 7.2. First, the RED dropping probability q is calculated. Then, if the packet is green, the probability of dropping q is multiplied by α.

The scheduling is Earliest Deadline First (EDF) [16]. Each packet is assigned a finishing service time deadline, a tag, and the packet currently having the lowest value is served first (i.e. earliest deadline). Each green packet arriving is assigned a finishing service time deadline equal to its arrival time, which is given by *now*. A blue packet is assigned a time equal to its arrival time, *now*, plus the value of the *offset bound* D, namely $now + D$. The goal of the offset bound is to limit the delay penalty imposed on blue.

The scheduling does impose a delay penalty on blue packets, which reduces throughput. We compensate for the delay penalty imposed on blues by artificially reducing their drop rate, and this is done by adjusting the value of the drop bias α. The determination and behaviour of the minimum α required to provide throughput transparency in a given network is analysed in Section 7.5. A sufficient α in the general case is described in Section 7.4.

A green packet can only be provisionally accepted at this point. To ensure the low delay guarantee to green is met, it is only accepted if the sum of the lengths of the packets in the queue with a deadline less than the current time, plus the length of the green packet, is less than cd, where c is the speed of the outgoing link. This bounds the queueing delay for green packets to d. In this way, if there are only green packets arriving, the system acts like a flat best-effort network with smaller buffers.

We summarise the PAC algorithm in Table 7.1. The queue size *Buff* and maximum queueing delay d are fixed, whereas the offset bound D and the drop bias α are adjusted automatically, on a slow time scale, by means of the control loop, which is now described. The question as to what drop bias α would be sufficient to protect blue flows is analysed in Section 7.5.

7.3 Satisfaction of router requirements

1. The per-hop delay bound for green, d, is enforced by the PAC.

2. Throughput transparency is obtained by the PAC ensuring that the delay penalty incurred by blue is compensated by a lower drop ratio. This is implemented by adjusting the offset bound D and the drop bias α.

3. Minimising the green drop ratio is performed by the control loop which adjusts the offset bound D and the drop bias α.

4. Packet sequence within blue and within green is preserved.

```
packet $p$ arrives at the output port
 if buffer full
  drop $p$
 else if $p$ is blue
  drop with random probability $q$
 if $p$ is still not dropped
  $p$.deadline $= now + D$
  accept packet
 else // $p$ is green
  drop with random probability $q\alpha$
  if $p$ is not dropped
   $l =$ sum of length of packets in queue with deadline $\leq now$
   if $l + p$.length $> cd$
    drop packet
   else
    $p$.deadline $= now$
    accept packet
```

Table 7.1: Pseudocode of ABE/EDF PAC. now is the current time, p.deadline determines when packet p will be served (whose value is tagged onto packet p), and p.length denotes the length of the packet

5. As described in the introduction (Section 7.1), it does not implement local transparency, because the delay for blue can be higher, violating the no higher delay requirement of Definition 2.4.1 on page 14. Note that the number of blue packets accepted is, on average, higher than in the virtual queue, since the average queue size is smaller in the real system.

7.4 The control loop

The control loop has two functions: to provide throughput transparency, and to ensure as few green losses as possible. We first look at the mechanism to ensure the former.

7.4.1 Motivation

In Section 7.5 on page 96, we determine, under modelling conditions, the required minimum α for a given network topology and number of flows in Section 7.5 on page 96. However, in general, knowledge of the network conditions to determine

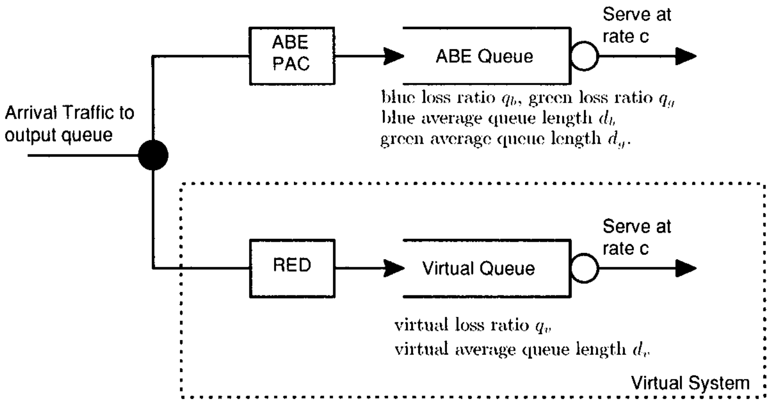

Arrival Traffic to output queue
ABE PAC
ABE Queue
Serve at rate c
blue loss ratio q_b, green loss ratio q_g
blue average queue length d_b
green average queue length d_g.
RED
Virtual Queue
Serve at rate c
virtual loss ratio q_v
virtual average queue length d_v
Virtual System

Figure 7.3: Virtual System used in ensuring throughput transparency.

the optimal drop bias α is impractical. Thus, in practice we must choose a safe α which works under general conditions for an unknown number of blue and green flows, unknown round-trip times, and unknown conditions flows through the router experience elsewhere. modelling relied on losses being packets.

Given our limited knowledge of the network, we cannot ascertain what would happen in a flat best-effort network, and thus the required minimum throughput each blue source would need to ensure throughput transparency is not known. We can however put our system in a state such that, provided certain general assumptions hold, the throughput blue sources receive is at least as high as in flat best-effort.

7.4.2 General mechanism

Consider the scenario as in Figure 7.3. It consists of measuring a virtual queue which is configured as in the ABE/EDF case but treats all packets as blue. This system is not the same as flat best-effort, since loss feedback to sources comes from the real system.

We use the following notation:

- q_b: the blue loss ratio
- q_g: the green loss ratio
- q_v: the loss ratio for packets in the virtual queue
- d_b: the average queueing delay for blue packets

- d_g: the average queueing delay for green packets
- d_v: the average delay for packets in the virtual queue.

We assume the established loss-throughput formula, Equation (2.1), holds.

Let τ_e be the lowest reasonable round-trip time a source could have through the router, excluding any queueing delays. This value is a parameter specified in the router. The higher τ_e can be chosen, the less green losses there will be. If it is chosen too high, there may be circumstances in which blue flows may not receive as much as they would in a flat best-effort network. Let

$$\begin{aligned} x_b(\tau) &= \theta(q_b, \tau + d_b) \\ x_g(\tau) &= \theta(q_g, \tau + d_g) \\ x_v(\tau) &= \theta(q_v, \tau + d_b) \end{aligned}$$

If a blue source with a round-trip time τ_e excluding queueing delay has a throughput higher than a green source, and it is higher than it would have in the virtual system, namely,

$$x_b(\tau_e) \geq \max(x_g(\tau_e), x_v(\tau_e)), \tag{7.1}$$

then throughput transparency holds for this blue source.

In addition, if Equation (7.1) holds, then throughput transparency for any flow with a higher round-trip time τ than τ_e also holds, namely that

$$x_b(\tau) \geq \max(x_g(\tau), x_v(\tau))$$

for all $\tau \geq \tau_e$. This result can be seen from Section 3.4.2 on page 42.

7.4.3 Control of α

The condition in Equation (7.1) motivates a control law which increases α if the blue throughput is less than either the green or the virtual throughput, and decreases α if the blue throughput is higher than both green and virtual throughput. We use one such control law here,

$$\alpha \leftarrow \max\left(1, \alpha + K_1 * (\max(x_g, x_v) - x_b)\right),$$

for gain parameter $K_1 > 0$. We prevent α from going below 1, because otherwise green could get more throughput than blue.

Now α responds to variations in queueing delays and losses, thus automatically adapting to variations in green and blue traffic load. Note that for a given α, the system will not drop exactly α times more green than blue packets due to: (a) the intrinsic randomness in the dropping mechanism, (b) the approximate modelling in the loss-throughput formula, and (c) the variations in the input process and green drops which occur in the second stage. The actual ratio of green to blue drop ratios achieved may be quite different from the α specified. The control law automatically adjusts α to reflect these inaccuracies.

For the purpose of determining α, the router acts as if it were the only bottleneck for all flows that pass through it. We show through simulation in Section 7.6, and through analysis in Section 7.5, that assuming one bottleneck results in a higher and thus sufficient α than if α was (able to be) chosen in conjunction with more global state.

7.4.4 Control of D

The current value of the offset bound D, which determines the level of delay priority given to green, is also controlled. The reasons to do so are two-fold. We would like to ensure there are few drops due to second stage drops (which is caused if D is too low), thus increasing the number of drops caused by the differential loss. Also, we seek to minimise green first stage losses, and having too high a D results in too many green losses to protect blue.

The lower D is, the less variations there are between the throughputs x_b, x_g and x_p. This keeps α low which reduces the number of first stage green drops. There can however be an increase in the number of second stage drops. Many second stage drops is a symptom of too low a D.

Conversely, if D is high, the throughputs are further apart, and thus α is high. This may result in an unnecessarily high number of green losses. The higher the α the lower the average number of green packets in the queue. To minimise green losses, the number of green packets in the queue should be maximised as much as possible while avoiding second stage drops. We control D in such a way as to drive it into this operating region.

The actions required can be thought of as follows. α should be increased if the delay boost increases or too many blue packets were dropped in the past. Conversely, α should be decreased if the delay boost decreases or too many green packets were dropped in the past. D should be decreased if the number of green packets in the queue has been small, and increased if there were many second stage drops.

We now describe the controlling mechanism for the offset bound D. Let w_l be the measured average number of bits with a deadline less than *now* over the time interval T_1. Let q''_g denote the ratio of the number of second stage green packet losses to the total number of green arrivals.

In the control law, a safety margin λ is used to avoid waiting for second stage losses to accumulate before increasing D where λ is a measure of the tolerance of how close we allow the green buffer to being full. Thus we consider $\lambda cd - w_l$ to be a measure as to how far we are from a full queue, or, if negative, by how much we have exceeded it. In controlling D we would like to ensure $q''_g \to 0$ and $w_l \to \lambda cd$. We arrive at the control law,

$$D \leftarrow D + K_2 * \left(\frac{w_l}{\lambda cd} - 1\right) + K_3 q''_g$$

for gain parameters $K_2, K_3 > 0$ and $\lambda \in (0, 1]$. D is restricted to lie in the range $[0, D_{\max}]$, where $D_{\max}$ is the highest tolerable offset bound, given by the value beyond which the system behaves effectively as if it gave absolute priority to greens. From our experiments, it was found that λ should lie between 0.4 and 0.6.

7.4.5 Implementation details

We now describe how the control loop is mplemented. The measurements α and D are updated after each time interval T_1, for some T_1. In a given time period the following are the measured values:

- q^m_b: the ratio of blue losses to number of blue arrivals
- q^m_g: the ratio of green losses to number of green arrivals
- q''^m_g: the ratio of second stage green losses to number of green arrivals
- q^m_v: the ratio of losses in the virtual system to total number of arrivals
- d^m_g: the average green queueing delay
- d^m_b: the average blue queueing delay
- d^m_v: the average queueing delay in the virtual system.

In order not to be too reactive to the instantaneous values of the measured loss ratios, we smooth using an exponentially weighted moving average (EWMA) filter with parameter $\gamma_1 \in [0, 1]$. The measured average queueing delays are also smoothed by an EMWA filter with parameter $\gamma_2 \in [0, 1]$, and $q_g^{''m}$ is smoothed by an EWMA filter with parameter $\gamma_3 \in [0, 1]$.

The average queueing delays and $q_g^{''m}$ are filtered using a different smoothing parameter to the loss ratios in order to react to higher frequency oscillations in these signals. We use the measured value of w_l without any filtering.

We now summarise the control action for the nth iteration (i.e. at time nT_1):

$$\begin{aligned}
q_b(n) &= \gamma_1 q_b(n-1) + (1-\gamma_1)q_b^m(n) \\
q_g(n) &= \gamma_1 q_g(n-1) + (1-\gamma_1)q_g^m(n) \\
q_g''(n) &= \gamma_3 q_g''(n-1) + (1-\gamma_3)q_g^{''m}(n) \\
d_b(n) &= \gamma_2 d_b(n-1) + (1-\gamma_2)d_b^m(n) \\
d_g(n) &= \gamma_2 d_g(n-1) + (1-\gamma_2)d_g^m(n) \\
d_v(n) &= \gamma_2 d_v(n-1) + (1-\gamma_2)d_v^m(n) \\
x_g(n) &= \theta(q_g(n), \tau_e + d_g(n)) \\
x_b(n) &= \theta(q_b(n), \tau_e + d_b(n)) \\
x_v(n) &= \theta(q_v(n), \tau_e + d_b(n)) \\
\alpha(n) &= \max\left(1, \alpha(n-1) + K_1 * (\max(x_g, x_v) - x_b)\right) \\
D(n) &= D(n-1) + K_2 * \left(\frac{w_l(n)}{\lambda cd} - 1\right) + K_3 q_g''(n)
\end{aligned}$$

Optimal settings for the parameters of the system are not necessary for the implementation to work, as we have found that the system is sufficiently robust. Examples of settings used are shown in the simulations section in Section 7.6 on page 101.

7.4.6 Initial values

Upon system commencement some initial values are required. Their choice is not vital as the system will settle to operating values.

- The intial loss ratios $q_b(0)$, $q_g(0)$, $q_v(0)$ are chosen based on observation, e.g. $q_b(0) = 0.01$, $q_g(0) = 0.04$ and $q_v(0) = 0.01$. The initial drop bias is then $\alpha(0) = q_g(0)/q_b(0)$.
- Initially it is reasonable to assume no second stage drops, so $q_g''(0) = 0$.
- $D(0)$ should be chosen to reflect a reasonable delay differential, e.g. $D(0) = 0.1$.

- The queueing delay measurements can be chosen to be $d_v(0) = 0$, $d_g(0) = 0$ and $d_b(0) = D(0)/2$.

7.5 Analysis of minimum drop bias α

Let b be the *delay boost* for an output queue to a link, i.e. the difference between the average queueing delay for blue packets and for green packets. Then b is $d_b - d_g$, for average blue queueing delay d_b, and average green queueing delay d_g.

In this section we analyse the drop bias α requirements for a given network for a given delay boost b.

We first describe a method to determine α for a given arbitrary network composed of a deterministic number of blue and green sources and links. We then demonstrate under an example topology, that:

(i) the minimum α required is dependent on the relative number of green and blue sources;

(ii) the determination of the minimum α for all flows is equivalent to choosing the highest α needed for sources with just one bottleneck;

(iii) the required α increases approximately exponentially as the boost b increases linearly.

The establishment of (ii) was shown on the example and not in general, a result which was also verified by simulation. Result (iii) has significance in the engineering of a router's delay advantage to green.

The results we show apply to long lived flows. However, it is known that shorter flows will suffer more than long term ones when there are losses. Therefore the worst case drop bias α needed is obtained by considering all flows as being long lived (since we protect blue flows more).

7.5.1 The model

Consider the following abstraction of a general network. It contains L links, a set of blue sources B and a set of green sources G. L_l is the set of all sources that use link l. A source s is on link l if $s \in L_l$. Assume each source s has a long term rate of x_s and a fixed round-trip time τ_s. The capacity of each link l is c_l. Whenever the probability of not accepting a blue packet on a link l is q, the probability of rejecting a green packet is $\alpha_l q$.

Assume a source s responds in an additive increase/multiplicative decrease way, i.e. in the event of no loss in a round-trip time τ_s, it increases its rate by r_s, and responds to loss detection by decreasing its rate by $\eta \in (0,1)$. This behaviour is a simplification of the congestion response of a TCP flow.

We need to use this method of global modelling, rather than simply applying known TCP equations [25, 31, 10, 37, 4] which relate loss rate to throughput for an individual source, with no concept of the number of bottlenecks a flow experiences.

The following theorem describes a method to determine under this model of a network, the distribution of long term rates between flows.

Theorem 7.5.1 *Let $\alpha = \alpha_l$ for all $l = 1 \dots L$ (the drop bias is the same on every link). Then the distribution of long term rates x_s can be determined by the maximisation of the utility function,*

$$\sum_{s \in B} \frac{1}{\tau_s} \log \frac{x_s}{r_s + \eta x_s} + \frac{1}{\alpha} \sum_{s \in G} \frac{1}{\tau_s} \log \frac{x_s}{r_s + \eta x_s} \tag{7.2}$$

subject to the link constraints,

$$\sum_{s \in L_l} x_s \leq c_l, \quad l = 1, \dots, L. \tag{7.3}$$

Proof: The derivation, which relies on losses being rare, is an easy consequence of the results in Chapter 8 and [43].

Let $\hat{\alpha}_s = \alpha$ if $s \in G$ (source is green) and $\hat{\alpha}_s = 1$ if $s \in B$ (source is blue). Equation (22) on page 4 of [43] changes to

$$\frac{dx_s}{ds} = \frac{r_s}{t_s} - x_s(r_s + \eta x_s)\hat{\alpha}_s \sum_{l \in L, s \in L_l} g_l(f_l)$$

since the dropping probability function for a green source is $\alpha g_l(f_l)$. From this, it follows that

$$J_A^h(\vec{x}) = \sum_{s \in B} \frac{1}{\tau_s} \log \frac{x_s}{r_s + \eta x_s} + \frac{1}{\alpha} \sum_{s \in G} \frac{1}{\tau_s} \log \frac{x_s}{r_s + \eta x_s} - G(\vec{x}).$$

The rest of the derivation is the same. □

The restriction that $\alpha = \alpha_l$ for all $l = 1, \dots, L$ is of course not true in general. Determining the distribution in this case is not as simple, and not necessary for the network which we analyse here. When $\alpha = 1$, the maximisation results in the flat best-effort distribution of rates.

For a blue source s and a delay boost b, the minimum drop bias α_s should be the smallest value such that:

1. Its throughput with ABE/EDF is no less than if the network were flat best-effort. This ensures throughput transparency is satisfied. The minimum in this case, α'_s, is given when both throughputs are the same i.e. it is the solution to $x_s(b, \alpha'_s) = x^f_s$ where $x_s(\alpha'_s)$ is the throughput from an ABE/EDF network with drop bias α_s and x^f_s is the throughput the flow receives in a flat best-effort network.

2. If the blue source were to become green it would not receive more throughput. The minimum in this case, α''_s, is given by the solution to $x_s(\alpha''_s) = x'_s(\alpha''_s)$ where $x'_s(\alpha''_s)$ is the throughput from the ABE/EDF network if the source was green rather than blue.

The minimum drop bias for a source s is thus given by $\alpha_s = \max(\alpha'_s, \alpha''_s)$. Note that in general $\alpha'_s \geq \alpha''_s$ may not hold, a point we illustrate in Section 7.5.3.

For a given network, the α that protects all blue flows at minimum cost to green flows is given by $\alpha = \max_{s \in B} \alpha_s$. Using this drop bias α ensures that all blue flows receive at least as much as they would if the network were flat best-effort, and that turning green cannot result in a throughput advantage.

7.5.2 Example topology

We now look at the parking lot scenario as depicted in Figure 7.4. There are I links each of capacity c. There are $n_{b,l}$ blue flows with throughput $x_{b,l}$ and $n_{g,l}$ green flows with throughput $x_{g,l}$ which traverse all links. These are the long flows. On each link there are $n_{b,s}$ blue flows with throughput $x_{b,s}$ and $n_{g,s}$ green flows with throughput $x_{g,s}$ which traverse just this link. These are the short flows.

When this is an ABE/EDF network, the long and short green flows have a round-trip time (RTT) of $\tau_{g,l}$ and $\tau_{g,s}$ respectively. The long and short blue flows have a round-trip time of $\tau_{b,l}$ and $\tau_{b,s}$ respectively. Let the average extra queueing delay at each link for a blue flow be the delay boost b. Thus, $\tau_{b,s} = \tau_{g,s} + b$ and $\tau_{b,l} = \tau_{g,l} + Ib$. The assumption that the drop bias α is the same on each link is valid given that the load on each link is the same. For the same reason, the delay boost b can also be considered to be the same on each link. The α sufficient to ensure blues do not suffer is given by $\max(\alpha_l, \alpha_s)$, where α_l and α_s are the minimum drop biases for long and short flows respectively.

When this network is flat best-effort, $\alpha = 1$. The round-trip times of the flows are unknown, unless queueing analysis is performed. To bypass this difficulty we

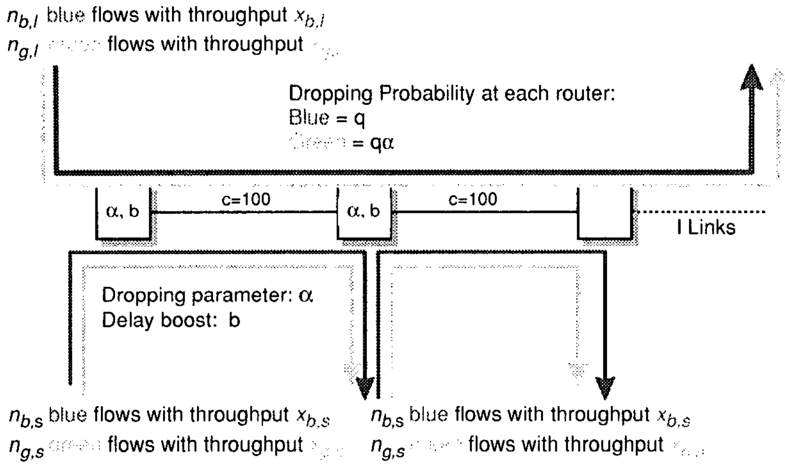

$n_{b,l}$ blue flows with throughput $x_{b,l}$
$n_{g,l}$ green flows with throughput $x_{g,l}$
Dropping Probability at each router:
Blue = q
Green = qα
α, b
c=100
α, b
c=100
l Links
Dropping parameter: α
Delay boost: b
$n_{b,s}$ blue flows with throughput $x_{b,s}$
$n_{g,s}$ green flows with throughput $x_{g,s}$
$n_{b,s}$ blue flows with throughput $x_{b,s}$
$n_{g,s}$ green flows with throughput $x_{g,s}$

Figure 7.4: Parking Lot Scenario used in analysis of α.

consider all possible values to determine a worse-case α.

Let $\eta = 0.5$. The additive increase parameters for a long and short blue flow are given by $r_{b,l}$ and $r_{b,s}$ respectively, and $r_{g,s}$ and by $r_{g,s}$ for a long and short green flow. Let $r_{b,l} = 1/\tau_{b,l}$, $r_{b,s} = 1/\tau_{b,s}$, $r_{g,l} = 1/\tau_{g,l}$ and $r_{g,s} = 1/\tau_{g,s}$. This means the sources react like a TCP source by decreasing their rate by two in the event of a lost packet, and increasing their rate accordingly in the event of no loss.

The throughput for the long blue and green flows is given by $x_{b,l}$ and $x_{g,l}$, and by $x_{b,s}$ and $x_{g,s}$ for the short blue and green flows. By maximisation procedures, we can determine from Equations (7.2) and (7.3) that the distribution is given by the solution to the following equations:

$$\tau_{b,l}x_{b,l}(2+\tau_{b,l}x_{b,l})I = \alpha x_{g,s}\tau_{g,s}(2+x_{g,s}\tau_{g,s}) \quad (7.4)$$

$$x_{g,l} = \frac{1}{\alpha\tau_{g,l}}\left(-\alpha+\sqrt{\alpha(\alpha+x_{b,l}\tau_{b,l}(2+x_{b,l}\tau_{b,l}))}\right) \quad (7.5)$$

$$x_{b,s} = \frac{1}{\tau_{b,s}}\left(-1+\sqrt{1+Ix_{b,l}\tau_{b,l}(2+x_{b,l}\tau_{b,l})}\right) \quad (7.6)$$

$$x_{g,s} = \frac{1}{n_{g,s}}\left(c-n_{b,l}x_{b,l}-n_{g,l}x_{g,l}-n_{b,s}x_{b,s}\right). \quad (7.7)$$

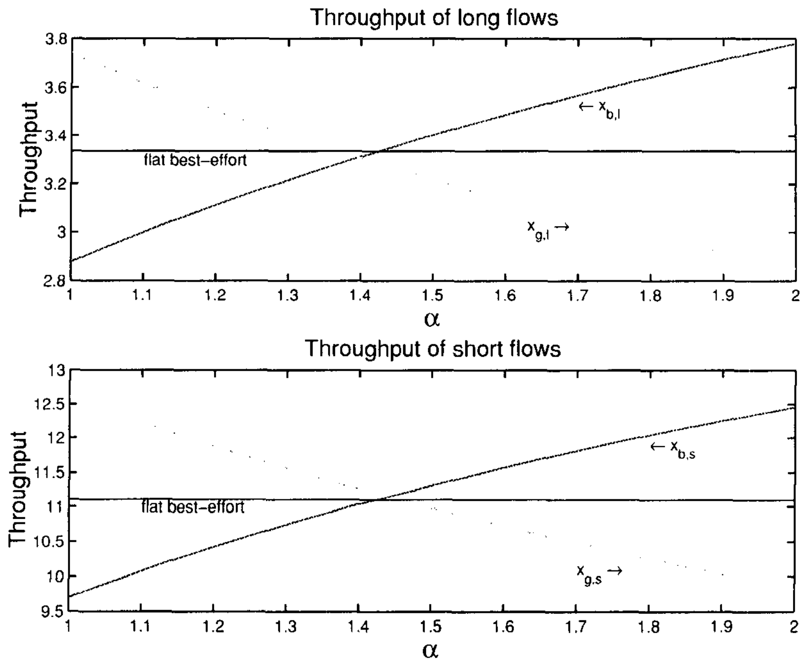


Figure 7.5: The throughput of long and short flows as a function of the drop bias α. Here, $n_{b,l} = n_{g,l} = 5$, $n_{b,s} = n_{g,s} = 3$, and $b = 0.03$.

7.5.3 Numerical results

The results we now present are determined by numerical solutions to Equation (7.7). Consider the case when $I = 2$, the capacity of each link $c = 100$, $\tau_{g,l} = 0.2$ and $\tau_{g,s} = 0.1$. We examine first the case of an equal number of green and blue flows, $n_{b,l} = n_{g,l} = 5$, $n_{b,s} = n_{g,s} = 3$. First let the boost be fixed at $b = 0.03$. Figure 7.5 shows the throughput of each source. The short flows get more throughput than the long ones due to their shorter round-trip times and less number of bottlenecks. In the long flow case, blue flows receive more than green flows when $\alpha > \alpha_l'' \approx 1.40$ but do not receive as high a rate as in a flat best-effort network until $\alpha \geq \alpha_l' \approx 1.44$. In the short flow case, blue flows receive at least as much as they would in a flat best-effort network when $\alpha \geq \alpha_s' \approx 1.42$, but do not receive more than green flows until $\alpha \geq \alpha_s'' \approx 1.46$. When the short blue flow receives as much as in the flat case, the greens still get more because the long green flows lose sufficient throughput to

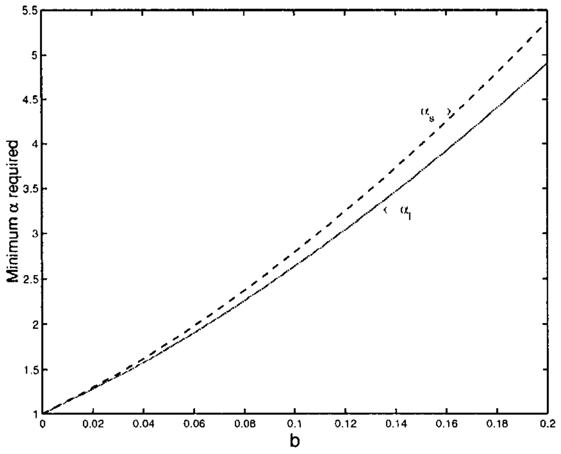


Figure 7.6: The minimum α for the long and short flows as a function of the delay boost b. $n_{b,l} = n_{g,l} = 5$, $n_{b,s} = 3$, $n_{g,s} = 1$, and $b = 0.03$.

the benefit of the short green flows. Overall, $\alpha = 1.46$ is sufficient to ensure both blue flow types receive enough throughput.

In Figure 7.6 we let the delay boost b vary, and measure the required α_s and α_l. We notice two things. Firstly, satisfying the short blue flows requires a higher α than the long blue flows. Secondly, as b increases, we must increase α exponentially to compensate.

Suppose we vary the number of short and long blue and green flows alternatively as in Figure 7.7. The number of other flows directly influences the minimum required drop bias. In this case, we see that increasing the number of short blue flows increases the size of α needed while in all other cases it reduces the α required.

Note that the delay boost is fixed at $b = 0.03$ here. In reality, when we change the number of flows, we also change b. To obtain the relationship between b and the number of flows, a queueing analysis would be needed.

7.6 Simulation study

The general test topology, shown in Figure 7.8, consists of: $n_{b,l}$ blue and $n_{g,l}$ green sources which traverse both links, the long flows; n_{b,s_1} blue and n_{g,s_1} green sources which traverse the first link, the type 1 short flows; and n_{b,s_2} blue and n_{g,s_2} green sources which traverse the second link, the type 2 short flows. The links from the

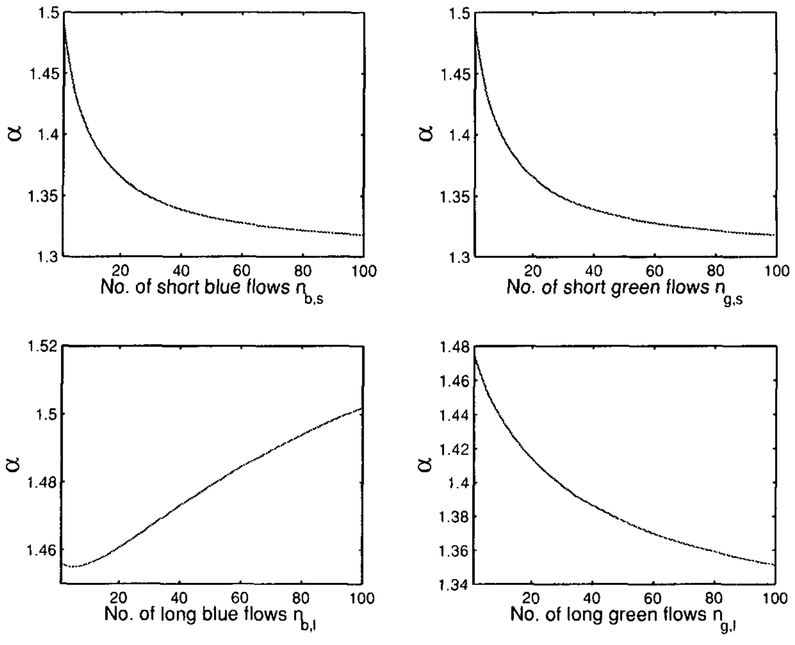


Figure 7.7: α required when we vary alternatively the number of flows of a given type and colour while. For fixed $b = 0.03$.

sources and sinks to the bottleneck links were 10Mbps and had propagation delays of 20ms.

Each router buffer size was 60 packets (i.e. $L_{tot} = 60$) and the maximum queueing delay for green d was 0.0176 seconds, which represents the transmission time of 11 packets. The set of RED parameters used, were as follows: The initial threshold $min_{th} = 10$, maximum threshold $max_{th} = 30$, the upper bound on the (blue) marking probability $max_p = 0.15$, and average queue weight $w_q = 0.002$. the success of ABE. Throughout, $\gamma_1 = 0.8$, $\gamma_2 = 0.4$, and $\gamma_3 = 0.4$, and the control loop updates α and D every $T_1 = 0.5s$. The gain parameters were $K_1 = 1.1$, $K_2 = 0.02$, $K_3 = 2.0$ and $\lambda = 0.4$. The initial values were $q_b(0) = 0.01$, $w_l(0) = 4$, and $D(0) = 0.1$, $d_b = D(0)/2$. The round-trip time τ_e, used in determining α, was 0.18, which is based on the propagation delays for the short flows.

Given the randomness in dropping, average values were obtained after four simulation runs and confidence interval results obtained. Throughput and delay measurements for the first 30s of simulation time are not measured as we allow the

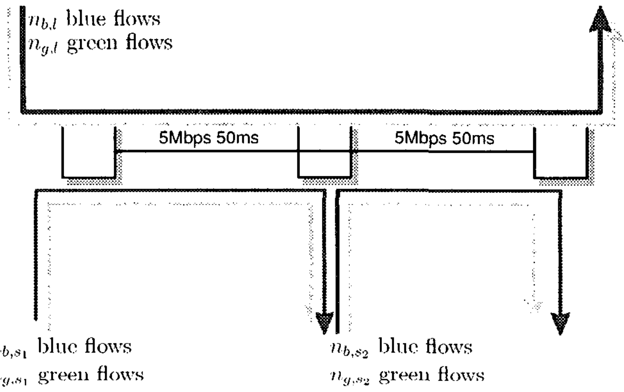


Figure 7.8: Simulation topology for ABE/EDF Experiments

---

control mechanism to warm up to reflect more realistic operation.

7.6.1 Equal numbers of blue and green flows

We first look at the case when there are an equal number of blue and green flows for each flow type, explicitly $n_{b,l} = n_{b,s_1} = n_{b,s_2} = 5$ and $n_{g,l} = n_{g,s_1} = n_{g,s_2} = 5$. Figure 7.9 shows the average number of packets received by each blue and green connection at each time t. The average shown at each time t is the average obtained over 4 simulation results for that time t. For clarity the confidence intervals are not shown. The worst-case interval for 95% confidence seen was small, 67 packets. Figure 7.10 shows the end-to-end delay distributions received for green packets of each type under ABE and flat best-effort. We show only the type 1 short flows delay distribution, since the type 2 short flow distribution is practically identical.

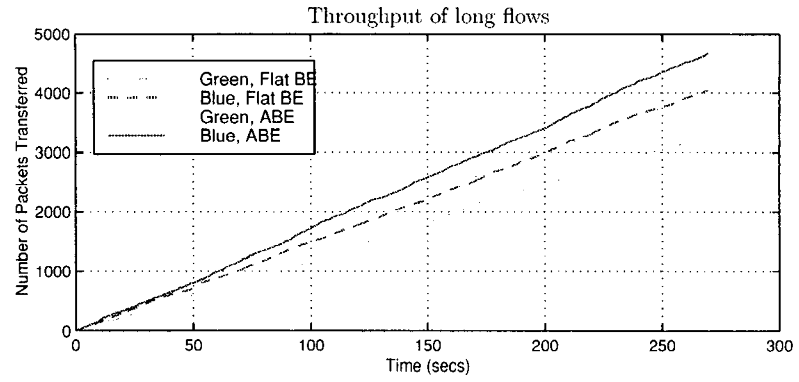
The delay for green packets is small and bounded, receiving the benefit of low bounded delay. The blue flows receive at least as much as they would as with flat best-effort. They actually receive more, thus receiving benefit from the use of ABE/EDF. The green sources receive not significantly less throughput than in flat best-effort.

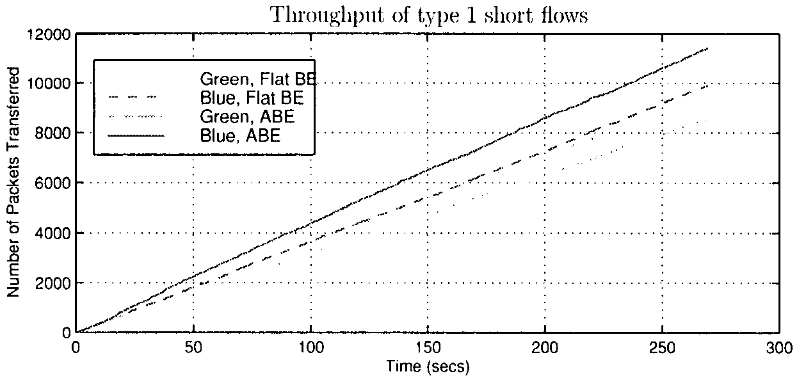
Figure 7.9 provides some additional observations. The long blue flows receive less throughput than the short blue flows due to their longer round-trip time and multiple bottlenecks. A similar observation hold for the green flows. The long blue flows receive proportionally more throughput benefit from ABE/EDF than the short blue flows do, which is expected. The minimum round trip-time value used in the calculation of the drop bias α was the short flows' propagation delay. Thus, the long flows benefitted from being considered, for the purposes of protecting it by dropping green packets, as having a lower round-trip time. Also, because of the multiple bottlenecks, the long green flows' losses receive a proportionally lower share.

7.6.2 Unequal numbers of blue and green flows

We now look at the case where there is the same number of long blue and green flows, but there are more blue than green flows of type 1, and more green than blue flows of type 2. The explicit values are $n_{b,l} = n_{g,l} = 5$, $n_{b,s_1} = 4$, $n_{g,s_1} = 4$, $n_{b,s_2} = 1$, and $n_{g,s_2} = 4$. Figure 7.11 shows how the average throughput received by each type varies. For brevity, we omit the end-to-end delay distribution which is similar to the previous case.

We observe that, when using ABE the type 2 short blue flow get a higher increase in throughput than the type 1 short blue sources. This is expected behaviour. Both compete with the same number of flows on its bottleneck link. However, the type 2 short blue flow has more since it competes with less blue flows and more green flows. If these greens were to become blue then the throughput of type 2 short blue flows would reduce.





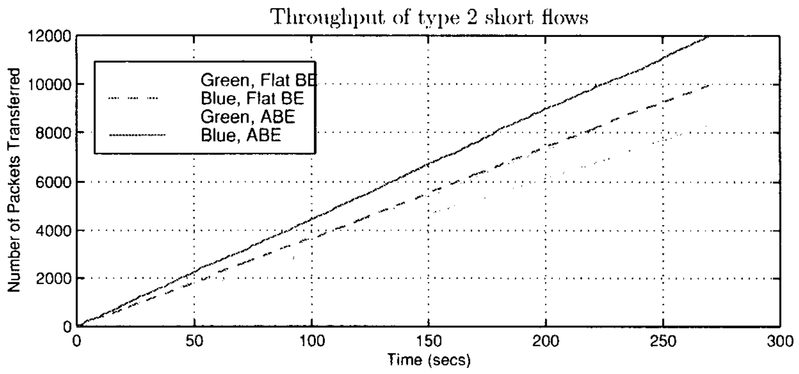
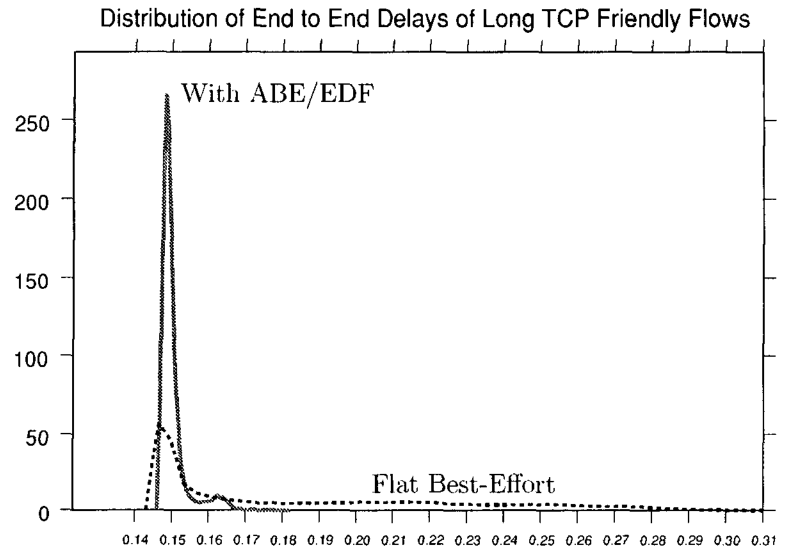


Figure 7.9: Average number of packets transferred per green and blue connection, for each traffic type, at each time t. When routers implement ABE/EDF and not (flat best-effort). There are 5 blue and green flows of each flow type.



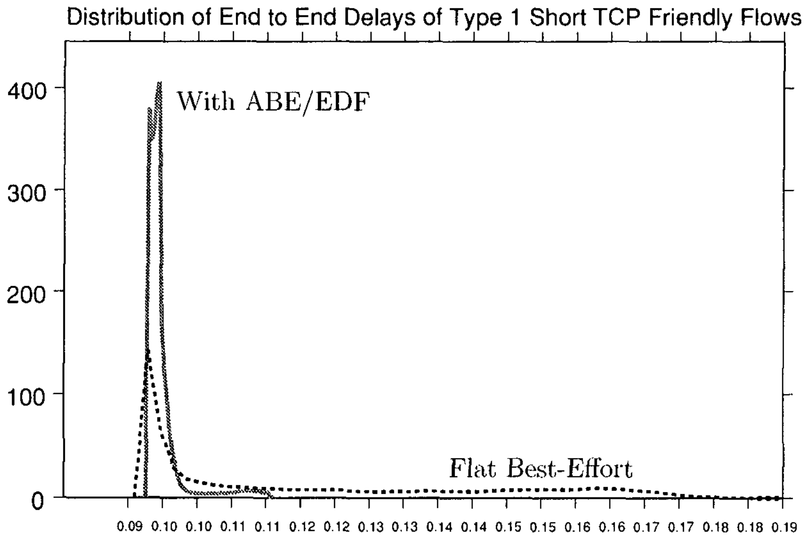


Figure 7.10: End-to-End Delay distributions received for green packets of each type under ABE and flat best-effort. 5 blue and green flows of each flow type.

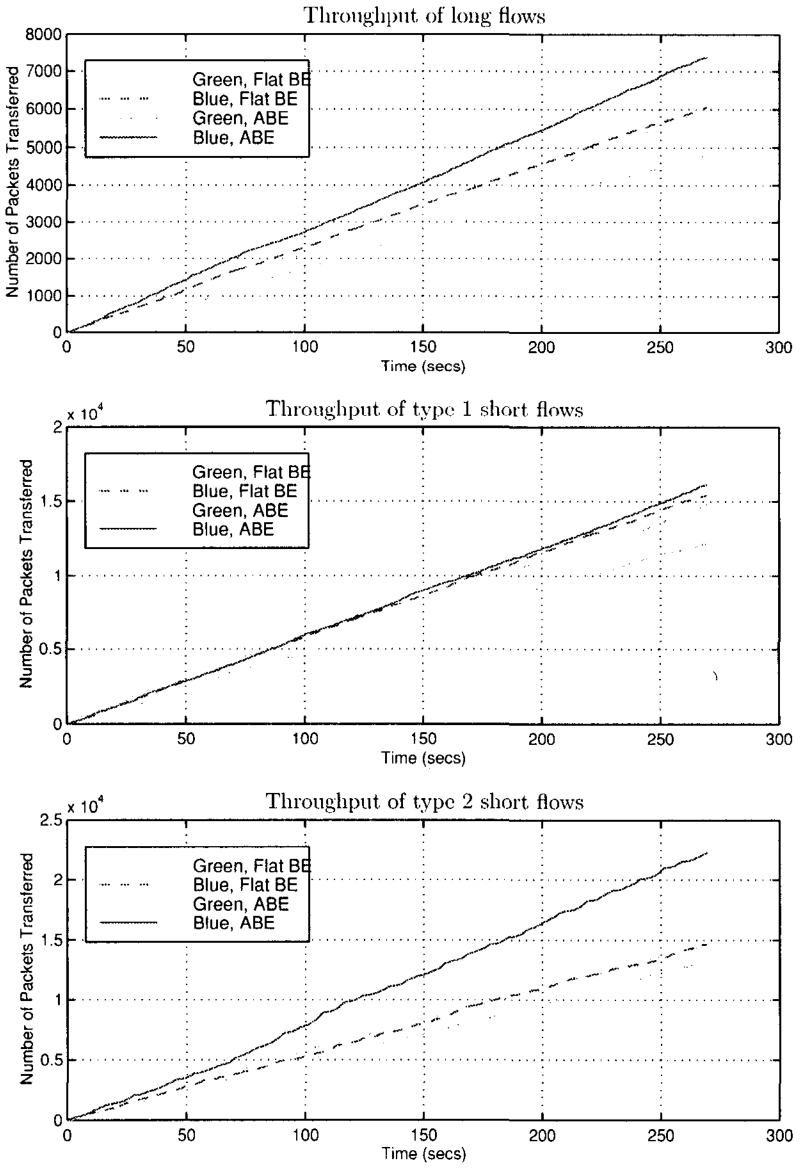


Figure 7.11: Average number of packets received per connection for each time t for ABE and flat best-effort when $n_{b,l} = n_{g,l} = 5$, $n_{b,s_1} = 1$, $n_{g,s_1} = 4$, $n_{b,s_2} = 4$, $n_{g,s_2} = 1$.

Chapter 8

Fairness Analysis

8.1 Introduction

In this chapter, we re-examine the topic of the rate allocation in packet networks to sources that adapt their rate according to the additive increase/multiplicative decrease algorithm. The network sends binary feedback (either implicitly by inferring the existence or absence of packet loss), or explicitly in the form of explicit congestion notification) to sources that adjusts their rate as follows. They decrease it multiplicatively (by some factor η) upon receipt of negative feedback and increase it linearly (by some value r_0) if there is positive feedback.

This algorithm [5] was originally believed to exhibit *max-min fairness*, an allocation favouring smaller rates. This is the allocation reached such that any further increase in the rate of one source results in the decrease of some smaller rate.

Results in [21, 29] suggest that for equal round-trip times TCP appears to provide *proportional fairness*. Proportional fairness is a form of fairness which distributes bandwidth with a bias in favour of flows using a smaller number of hops, in contrast to max-min fairness, which gives absolute priority to small flows.

Imagine there are I sources. An allocation of rates $\vec{x} = \{x_0, \ldots, x_I\}$ amongst these sources is said to be proportionally fair if and only if, $x_i \neq 0$ for all i and for any other permitted allocation $\vec{y}$,

$$\sum_{i=1}^{I} \frac{y_i - x_i}{x_i} \leq 0.$$

Let $A_{l,j}$ be the amount of traffic source j carried on link l. It was shown [21] that

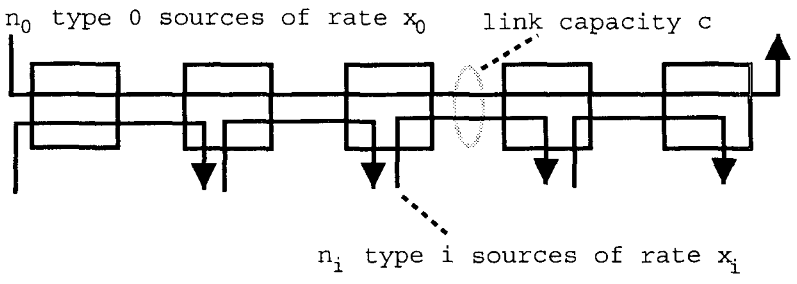


Figure 8.1: Parking lot Scenario with I links

this is equivalent to stating that the rates x_i maximise

$$F_0(\vec{x}) = \sum_{i=1}^{I} \log x_i \tag{8.1}$$

subject to the constraints that the link capacities in the network may not be exceeded, i.e. subject to

$$\sum_{j=1}^{I} A_{l,j} x_j \leq c_l \quad \text{for all links } l. \tag{8.2}$$

Consider, for example, what is called the parking lot scenario in Figure 8.1. It consists of I links each with capacity c. Sources of type 0 traverse the entire I links, while sources of type $i \geq 1$ only traverse the ith link. The number of sources of each type is given by $\vec{n} = (n_0, n_1, \ldots)$. The question posed is what is the rate allocation to each source type.

It can be shown that the distribution for max-min fairness in this network is given by

$$x_0 = \frac{c}{n_0 + \max_{i=1,\ldots,I} n_i}; \quad x_i = \frac{c - n_0 x_0}{n_i}, \quad i = 1, \ldots, I \tag{8.3}$$

and for proportional fairness by

$$x_0 = \frac{c}{\sum_{i=0}^{I} n_i}; \quad x_i = \frac{c - n_0 x_0}{n_i}, \quad i = 1, \ldots, I \tag{8.4}$$

From this it can be seen that the sources of type 0 get less in a proportional fairness allocation than a max-min one.

Outline

In what follows we shall argue that TCP connections of equal round-trip times do not converge to long term rates in agreement with proportional fairness. Rather, we show that in the event of rare negative feedback and equal round trip times, TCP distributes rates more closely in accordance with the fairness distribution algorithm derived here which we shall call F_A-fairness.

Even in the event where we have rate independent feedback we prove a result which closer reflects the convergence than proportional fairness. We use the framework of the ODE method to examine the development of long term rates for different sources. This establishes, in the event of rare negative feedback, convergence to F_A-fairness, as the multiplicative decrease factor and additive increase factor approach zero. We subsequently demonstrate by simulation that for large factors such as those specified by TCP, the average rate for each source converges around the value determined by F_A-fairness.

We demonstrate the behaviour of an F_A-fairness distribution in the context of the well-known example, the parking lot scenario. Finally, we establish that in the event of rate proportional feedback, our results maintain consistency with the well-known derivations relating TCP throughput as a function of loss ratio. However, this does not hold for the rate independent case, which is further validation of the assumption of rate dependent feedback.

8.2 Model

We consider a simplified network model, as follows. Traffic sources, labelled $1, \ldots, i, \ldots, I$, each send data to a given destination. The network is viewed as a collection of links labelled $1, \ldots, l, \ldots, L$, where the only resource consumed is link bandwidth. Every traffic source uses a fixed route. Let x_i be the sending rate for source i and assume that the amount of traffic from source i carried on link l is $A_{l,i}x_i$. The latter assumption amounts to assuming that losses are negligible. If source i sends traffic to one or several destinations over one single route, then $A_{l,i} = 0$ or 1 for all l, and those links l for which $A_{l,i} = 1$ constitute the route followed by the data. The general case where $A_{l,i}$ may have values between 0 and 1 allows traffic splitting over parallel paths. We assume that the rates of all sources are controlled by a mechanism of additive increase and multiplicative decrease as is encountered in TCP.

Modelling this mechanism is very complex because it contains both a random feedback (under the form of packet loss) and a random delay (the round trip time, including time for destinations to give feedback). We consider all round trip times to be constant and equal.

The system evolves as follows. Consider a number of time cycles of duration τ, where τ is the common round trip of all sources. During time cycle number t, the source sending rate for source i is assumed to be constant, and is given by $x_i(t)$. At the end of time cycle number t, source i receives a random, binary feedback $E_i(t)$ which is used to compute a new value of the sending rate. The binary feedbacks $E_i(t)$ for all i are independent Bernoulli random variables given the state of the system $\vec{x}(t) = (x_1(t), \ldots, x_i(t), \ldots, x_I(t))$. The sequence $\vec{x}(t)$ is thus a Markov chain. The feedback models packet losses in the Internet, or the congestion experienced bit in DecNet, Frame Relay or ATM. We assume rare negative feedback, and thus $E_i(t)$ takes values in the set $\{0, 1\}$.

Sources react to feedback by adjusting their rate; additively increasing it, when $E_i(t) = 0$ and multiplicatively decreasing it, when $E_i(t) = 1$. This yields the following relationship,

$$x_i(t+1) = x_i(t) + r_0(1 - E_i(t)) - \eta x_i(t) E_i(t) \tag{8.5}$$

or equivalently

$$x_i(t+1) = x_i(t) + r_0 - E_i(t)(r_0 + \eta x_i(t)). \tag{8.6}$$

r_0 is the additive increment for the rate and η the multiplicative decrease factor. For TCP, ignoring the effect of exponential increase during slow start, and assuming that all packets have the same size, $\eta = 0.5$ and $r_0 = 1/\tau$ (in packets per second) for no delayed acknowledgements and $r_0 = 1/(2\tau)$ for delayed acknowledgements.

We derive first a behaviour in an ideal case where, unlike real TCP implementations, r_0 and η are small. Afterwards we present simulation results which show that a TCP-like connection's average rate converges to a value in agreement with our results.

We also assume that all packets have the same fixed size. The amount of negative feedback received during one time cycle of duration τ is equal on average to $\mathbb{E}(E_i(t)|\vec{x}(t))$, which is the conditional expectation of $E_i(t)$ given $\vec{x}(t)$.

We consider two possible cases for the distribution of feedback (a) rate proportional and (b) rate independent as follows.

Case A: rate proportional feedback

The conditional expectation of $E_i(t)$ given $\vec{x}(t)$ is given by

$$\mathbb{E}(E_i(t)|\vec{x}(t)) = \tau \sum_{l=1}^{L} g_l(f_l(\vec{x}(t)))A_{l,i}x_i(t) \tag{8.7}$$

where

$$f_l(\vec{x}(t)) = \sum_{j=1}^{I} A_{l,j}x_j(t).$$

Now $f_l(t)$ represents the total amount of traffic flow on link l, while $A_{l,i}$ is the fraction of traffic from source i which uses link l. We interpret Equation (8.7) by assuming that $g_l(f)$ is the probability that a packet is marked with a feedback equal to 1 (*negative feedback*) by link l, given that the traffic load on link l is expressed by the real number f. In the regime of rare negative feedback, we assume that we can neglect the occurrence of a packet marked with a negative feedback on several links within one time cycle. Equation (8.7) simply then gives the expectation of the number of marked packets received during one time cycle by source i.

We believe that this models accurately the case where all flows receive the same loss rate independent of packet level statistics. This is believed to be achieved by using active queue management such as RED [12].

Case B: rate independent feedback

In this hypothetical case, the expectation of the amount of feedback received per cycle would have the form

$$\mathbb{E}(E_i(t)|\vec{x}(t)) = C \sum_{l=1}^{L} g_l(f_l(\vec{x}(t)))A_{l,i}$$

where C is a constant, and the rest is as per case A. We do not think that this case is a realistic model for congestion control under the assumption of rare negative feedback, and examine it partly because it implicitly underlies the findings in [21, 29, 15].

8.3 The ODE method

With our system model, $\vec{x}(t)$ is a Markov chain and the transition probabilities can be entirely defined using Equations (8.6) and (8.7) for case A, or (8.6) and (8.8) for case B. We use here an alternative tool, which gives some insight into the convergence of the system. The tool is the ODE method which was developed by Ljung [27] and Kushner and Clark [24]. The method applies to stochastic iterative algorithms of the form

$$\vec{x}(t+1) = \vec{x}(t) + \gamma \vec{H}(\vec{\xi}(t), \vec{x}(t)) \tag{8.8}$$

where $\vec{\xi}(t)$ is a sequence of random inputs and $\gamma > 0$ a small gain parameter, to which we associate the ODE,

$$\frac{d\vec{x}(s)}{ds} = h(\vec{x}(s)) \tag{8.9}$$

where

$$h(\vec{x}) = \mathbb{E}\{\vec{H}(\vec{\xi}, \vec{x}(t)) | \vec{x}(t))\}. \tag{8.10}$$

The result of the method is that the stochastic system in Equation (8.8) converges, in some sense, towards an attractor of the *ODE in Equation (8.9). The* attractor $\vec{x^*}$ of the ODE is $\vec{x^*} = \lim_{t\to\infty} \vec{x}(t)$ for solutions $\vec{x}(t)$ of Equation (8.9) given appropriate initial conditions. The case of interest is when the attractor is an equilibrium point.

Let $\vec{\xi} = \vec{E} = (E_1, E_2, \ldots E_I)$. Since r_0 and η are small, we can write

$$r_0 = k_r \gamma$$

and

$$\eta = k_\eta \gamma$$

where k_r and k_η are two positive constants. Then $\vec{H} = (H_1, \ldots, H_I)$ with

$$H_i(\vec{E}, \vec{x}) = k_r - E_i(k_r + k_\eta x_i).$$

The components of the mean vector field $\vec{h}(\vec{x})$ are therefore,

$$h_i(\vec{x}) = k_r - \tau x_i(k_r + k_\eta x_i) \sum_{l=1}^{L} g_l(f_l(\vec{x})) A_{l,i}$$

in Case A with a similar expression for Case B:

$$h_i(\vec{x}) = k_r - C(k_r + k_\eta x_i) \sum_{l=1}^{L} g_l(f_l(\vec{x})) A_{l,i}$$

As the components of the random feedback vector $\vec{E}(t)$ are independent variables depending only on the current value of $\vec{x}(t)$, and as the mean vector field satisfies the requirements of Theorem 3 of Chapter 2 from [1], we can apply this theorem, which we rephrase as follows:

Theorem 8.3.1 *If the ODE (8.9) is globally stable, with a unique stable equilibrium $\vec{x}^*$, then for $\gamma > 0$ sufficiently small, for all $\varepsilon > 0$, there exists a constant $C(\gamma)$ tending towards zero as γ tends to zero, such that*

$$\limsup_{t\to\infty} \mathbb{P}\{\|\vec{x}(t) - \vec{x}^*\| > \varepsilon\} \leq C(\gamma). \tag{8.11}$$

Note that multiplying the right-hand side of Equation (8.9) by $\gamma > 0$ does not modify the convergence properties of the ODE (it only amounts to a change of time scale). For simplicity of notation, we therefore study the equivalent ODE

$$\frac{d\vec{x}(s)}{ds} = \gamma h(\vec{x}(s)).$$

8.4 Application to the analysis of cases A and B

We now apply the ODE method to find some properties of our system in both cases.

Case A (rate proportional feedback)

Combining Equations (8.9), (8.10) with (8.6) and (8.7), we obtain:

$$\frac{dx_i}{ds} = r_0 - \tau x_i(r_0 + \eta x_i)\sum_{l=1}^{L} g_l(f_l)A_{l,i} \tag{8.12}$$

where $f_l = \sum_{j=1}^{I} A_{l,j}x_j$. In order to study the attractors of this ODE, we identify a Lyapunov function for it [33]. As such, we follow [21] and [15] and note that

$$\sum_{l=1}^{L} g_l(f_l)A_{l,i} = \frac{\partial}{\partial x_i}\sum_{l=1}^{L} G_l(f_l) = \frac{\partial G(\vec{x})}{\partial x_i}$$

where G_l is a primitive of g_l defined for example by

$$G_l(f) = \int_0^f g_l(u)du$$

and

$$G(\vec{x}) = \sum_{l=1}^{L} G_l(f_l).$$

Equation (8.12) then becomes,

$$\frac{dx_i}{ds} = x_i(r_0 + \eta x_i)\left\{\frac{r_0}{x_i(r_0 + \eta x_i)} - \tau\frac{\partial G(\vec{x})}{\partial x_i}\right\}. \tag{8.13}$$

Consider now the function J_A defined by

$$J_A(\vec{x}) = \sum_{i=1}^{I} \phi(x_i) - \tau G(\vec{x}) \tag{8.14}$$

where

$$\phi(x_i) = \int_0^{x_i} \frac{r_0 du}{u(r_0 + \eta u)} = \log\frac{x_i}{r_0 + \eta x_i}.$$

We can then rewrite Equation (8.13) as

$$\frac{dx_i}{ds} = x_i(r_0 + \eta x_i)\frac{\partial J_A(\vec{x})}{\partial x_i}. \tag{8.15}$$

Now it is easy to see that J_A is strictly concave and therefore has a unique maximum over any bounded region. It follows from this and from Equation (8.15) that J_A is a Lyapunov function for the ODE in (8.12), and thus the ODE in (8.12) has a unique attractor, which is the point where the maximum of J_A is reached.

Combined with Theorem 8.3.1 this shows that, for case A, the rates $x_i(t)$ converge at equilibrium towards a set of values that maximise $J_A(\vec{x})$, with J_A defined by

$$J_A(\vec{x}) = \sum_{i=1}^{I} \log\frac{x_i}{r_0 + \eta x_i} - \tau G(\vec{x}).$$

Case B (rate independent feedback)

The analysis follows the same line. The ODE is now

$$\frac{dx_i}{ds} = r_0 - C(r_0 + \eta x_i)\sum_{l=1}^{L} g_l(f_l)A_{l,i} \tag{8.16}$$

from which we derive that, for case B, the rates $x_i(t)$ converge at equilibrium towards a set of value that maximises $J_B(\vec{x})$, with J_B defined by

$$J_B(\vec{x}) = \frac{r_0}{\eta} \sum_{i=1}^{I} \log(r_0 + \eta x_i) \ - \ CG(\vec{x}).$$

Interpretation and comparison with previous results

In order to interpret the previous results, we follow [21] and assume that, calling c_l the capacity of link l, the function g_l can be assumed to be arbitrarily close to δ_{c_l} in some sense, where $\delta_c(f) = 0$ if $f < c$ and $\delta_c(f) = 1$ if $f \geq c$. Thus, in the limit, the method in [21] finds that, for case A, the rates are distributed so as to maximise

$$F_A(\vec{x}) = \sum_{i=1}^{I} \log \frac{x_i}{r_0 + \eta x_i}, \tag{8.17}$$

subject to the link constraints $\sum_{j=1}^{I} A_{l,j} x_j \ \leq c_l$ for all l. For case B, the rates tend to maximise

$$F_B(\vec{x}) = \sum_{i=1}^{I} \log(r_0 + \eta x_i), \tag{8.18}$$

subject to the link constraints $\sum_{j=1}^{I} A_{l,j} x_j \ \leq c_l$ for all l.

Let us now compare these results with those of [21] and [29] as discussed in Section 8.1. Both find, under the limiting case mentioned where g_l tends to δ_{c_l}, that the rates x_i are distributed according to proportional fairness, which is equivalent to stating that the rates x_i maximise Equation (8.1) subject to the link constraints Equation (8.2). *If we compare our results we find two differences.* Firstly, in [21] and [29], the model implicitly assumes case B, whereas we contend that case A is more realistic, in the regime of rare negative feedback.

Secondly, even for case B, our results do not exactly coincide. Indeed, in [21] and [29], the system is directly modelled with a differential equation, without using the intermediate stochastic modelling as we do in Section 8.3. The differential equation in [21] and [29] is

$$\frac{dx_i}{ds} = C \left(r_0 - \eta x_i \sum_{l=1}^{L} g_l(f_l) A_{l,i} \right)$$

which differs from Equation (8.16) by a missing term r_0 in the second part, and the constant C multiplied over the whole expression. It is our interpretation that our modelling method using the stochastic system more accurately reflects the real behaviour of the additive increase/multiplicative decrease algorithm, at least for the cases where our assumptions hold.

If we compare case B with proportional fairness, we find that, since r_0 is assumed to be small, the difference between F_B and F_0 is small, and thus if feedback is distributed independently of the sending rate, the rates tend to be roughly distributed according to proportional fairness. In some sense, this confirms the results in [21] and [29]. However, on the example of the next section, we find that case B tends to give less to sources that use several bottleneck links.

The situation is very different for case A, which we claim is more realistic. Here, the weight given to x_i tends to $-\log \eta$ as x_i tends to ∞. Thus, the distribution of rates will tend to favour small rates more than proportional fairness. In the next section we find an example that is indeed between proportional and max-min fairness.

8.5 Examples of F_A and F_B fairness

F_A-fairness and F_B-fairness are defined to be the distribution of rates given by maximising F_A and F_B respectively as shown in Equations (8.17) and (8.18). In this section we show for the example of the parking lot scenario F_A-fairness allocates more to sources that receive a small rate allocation from proportional fairness, and less to these sources than max-min fairness. When there is very little capacity (c is small), it approximates proportional fairness. For large c, F_A-fairness varies between max-min and proportional fairness.

We also show that F_B-fairness always allocates less than proportional fairness to sources that get small rates from proportional fairness.

The distribution of rates determined by F_A-fairness favours giving smaller rates more than proportional fairness and more closely reflects max-min fairness. We show also that F_B-fairness allocates less to sources with smaller rates than proportional fairness.

8.5.1 Analysis of F_A-fairness

Consider again the parking lot scenario from Figure 8.1. We now analyse the nature of the rate allocations obtained by F_A-fairness when using this network. The fraction of capacity distributed by F_A-fairness is not independent of the capacity, unlike the proportional and max-min fairness cases.

We show here how F_A-fairness varies with capacity i.e. we illustrate what we have already mentioned: for higher capacities, max-min approximates F_A while for lower capacities, F_A resembles proportional fairness.

Since $n_0x_0 + n_ix_i = c$, F_A can be expressed in terms of x_0,

$$F_A(x_0) = n_0 \log\left(\frac{x_0}{r_0 + \eta x_0}\right) + \sum_{i=1}^{I} n_i \log\left(\frac{c - n_0x_0}{r_0n_i + \eta(c - n_0x_0)}\right). \tag{8.19}$$

Note that $F_A(x_0)$ goes to $-\infty$ as x_0 goes to 0 and also as x_0 goes to $\frac{c}{n_0}$. This guarantees at least one maximum in the valid range $x_0 \in (0, \frac{c}{n_0})$. We can thus determine the distribution of $\vec{x}$ by solving $F_A'(x_0) = 0$. For general $\vec{n}$ maximising this directly is hard, as it involves solving a polynomial of order up to $2I$. So we focus on the case when $\vec{n} = (v, w, w, \ldots)$, for which the follow result may be derived:

Lemma 8.5.1 *The F_A-fairness distribution for the parking lot scenario where $\vec{n} = (v, w, w, \ldots)$ is given by*

$$x_0 = \frac{1}{2\eta(v^2 - Iw^2)}\Big(v(2c\eta + r_0w) + Iw^2r_0 - \sqrt{(v(2c\eta + r_0w) + Iw^2r_0)^2 - 4(v^2 - Iw^2)c\eta(\eta c + r_0w)}\Big) \tag{8.20}$$

when $v^2 - Iw^2 \neq 0$, and

$$x_0 = \frac{c(\eta c + r_0w)}{Iw^2r_0 + v(2c\eta + r_0w)} \tag{8.21}$$

when $v^2 - Iw^2 = 0$. x_i is then given by

$$x_i = \frac{c - vx_0}{w}, \quad i = 1, \ldots, I.$$

Proof: See Section 8.9

From Equations (8.3) and (8.4), when $\vec{n} = (v, w, w, \ldots)$, the distribution for max-min fairness and proportional fairness is given by $x_0 = \frac{c}{v+w}$ and $x_0 = \frac{c}{v+Iw}$ respectively. To examine how F_A-fairness distribution varies with c we examine the fraction of capacity source 0 receives, x_0/c, as capacity increases.

For F_A-fairness, x_0/c is increasing in c and it can be determined from Equation (8.20) that,

$$\lim_{c\to\infty} \frac{x_0}{c} = \frac{1}{v + \sqrt{I}w} \quad \text{and} \quad \lim_{c\to 0} \frac{x_0}{c} = \frac{1}{v + Iw} \quad \text{for all } v, I, w. \tag{8.22}$$

	For c small	For c large
F_A-fairness	$\frac{c}{v+Iw}$	$\frac{c}{v+\sqrt{I}w}$
Proportional fairness	$\frac{c}{v+Iw}$	$\frac{c}{v+Iw}$
Max-min fairness	$\frac{c}{v+w}$	$\frac{c}{v+w}$

Figure 8.2: Comparison of throughput given to type 0 sources by different fairness derivations.

We can see that F_A-fairness, in this case, allocates more of the fraction of capacity to sources of type 0 than proportional fairness, getting further away from proportional fairness as capacity increases, and exactly equalling it in the case of zero capacity.

We can also see that F_A-fairness allocates less capacity than max-min fairness for any capacity. When capacity is large we can see from Equation (8.22) that the distribution to type 0 sources can be approximated by $\frac{c}{v+\sqrt{I}w}$.

We show in Figure 8.2 a summary of the relationship between the three fairness criteria.

A graph of $\frac{x_0(c)}{c}$ for F_A-fairness alongside graphs for proportional and max-min fairness is shown in Figure 8.3 for the example when $\eta = 0.5, r_0 = 5, I = 2, v = 3, w = 2$. This graph is representative of any parameter settings.

If there are a lot of non-type 0 sources relative to type 0 sources (Iw is large compared to v), the difference between this and what max-min fairness would allocate is smaller than that of proportional fairness. Thus, for sources which would receive small rates from proportional fairness (large number of competing sources, many bottlenecks) F_A-fairness is better approximated by max-min fairness.

8.5.2 F_B analysis

Lemma 8.5.2 *The F_B-fairness distribution for the parking lot scenario where $\vec{n} = (v, w, w, \ldots)$ is given by*

$$x_0 = \max\left(\frac{c}{v+Iw} - \frac{(I-1)wr_0}{\eta(v+Iw)}, 0\right). \tag{8.23}$$

Proof: See Section 8.9.

x_0 is strictly increasing in c. $\lim_{c\to\infty} x_0/c = \frac{1}{v+Iw}$. Thus, when $I = 1$, F_B-fairness' fraction of capacity is the same as that for proportional fairness (and max-min fairness). When $I > 1$, the fraction of capacity allocated is always less than proportional fairness.

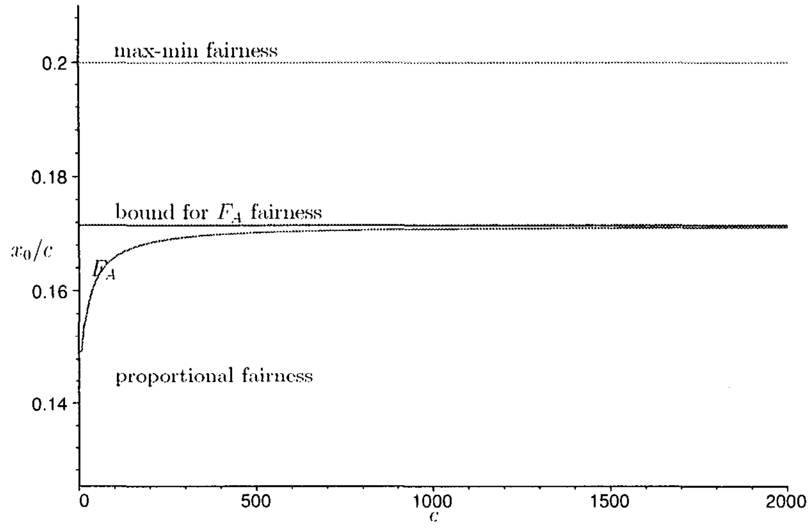


Figure 8.3: Numerical illustration of Section 8.5.1: x_0/c as a function of c.

In the limiting case, i.e. for very small capacity relative to the number of competing sources, F_B-fairness allocates zero to type 0 sources.

8.6 Verification by simulation

In this section, we investigate the convergence of the average rate of the time series for the sources for small values of η and r_0, and also for more TCP-like settings for the parameters. This is done both for the cases of rate proportional feedback and rate independent feedback.

We do this by simulation of the stochastic process in the parking lot scenario where $\vec{n} = (v, w, w, \ldots)$.

8.6.1 Rate proportional feedback

We first verify that the convergence holds for small increments of η and r_0. We then show that the series converges for TCP-like settings. More precisely, we show that in a regime of rare negative feedback, the average of the series converges to that expected from F_A-fairness for TCP-like settings of η and r_0 i.e. the distributed rates eventually oscillate around the value determined by F_A-fairness.

For the simulations, we consider the family of g_l functions,

$$g_l(f,d,p) = \begin{cases} 1 & f \geq c \\ 0 & f \leq dc \\ \left(\frac{\frac{f}{c}-d}{1-d}\right)^p & \text{otherwise} \end{cases} .$$

These functions are 0 when the link usage is less than dc, an increasing function from 0 to 1 for link usage between dc and c, and 1 when the link usage exceeds capacity available on the link. p is representative of how steep the increase between 0 and 1 should be.

At the start of each simulation, each x_i is assigned a random number from a *uniform distribution on* (0,c). At each iteration, the expectation, E_i for each source i is calculated. Then a random number is drawn from a uniform distribution on (0,1). If this number is greater than or equal to the calculated expectation, a value of $E_i = 0$ is assumed to have occurred, and x_i is linearly increased by r_0. Otherwise, x_i is multiplicatively decreased by η. The system continues to evolve until the total average capacity allocated does not change by a given tolerance.

The available simulation parameters are η, r_0, τ, I, v, w, d and p. For each chosen parameter set, the simulation is run four times, and the average of all four are calculated along with determined confidence intervals.

With additive increase/multiplicative decrease, the aggregate average rate allocated on a link will always be less than a link's nominal capacity c. Thus the sum of the average rates of all sources converges to a value, c', below this nominal rate c. How close c' is to c is determined by the efficiency of the g_l function in maximising overall throughput.

So, for each source, we consider the proportion of its average rate that it has of c'. This value is what we refer to as the *scaled average*. We obtain the F_A-fairness distribution from Equation (8.20).

Small values of η and r_0

Here we consider values of $\eta = r_0 = 0.01$ and $\tau = 0.2$. We varied the parameters as follows: $I = 2, 5$, v and $w = 1, 2, 6, 12$, $c = 250, 625$, $d = 0, 0.5, 1$, and $p = 1, 2, 5, 10$.

In all cases except when $d = 1$, we found the scaled average to converge to that expected from F_A-fairness, which can be seen in Figure 8.4. When $d = 1$, the assumption of rare negative feedback no longer held because every source was receiving a large amount of negative feedback at the same time.

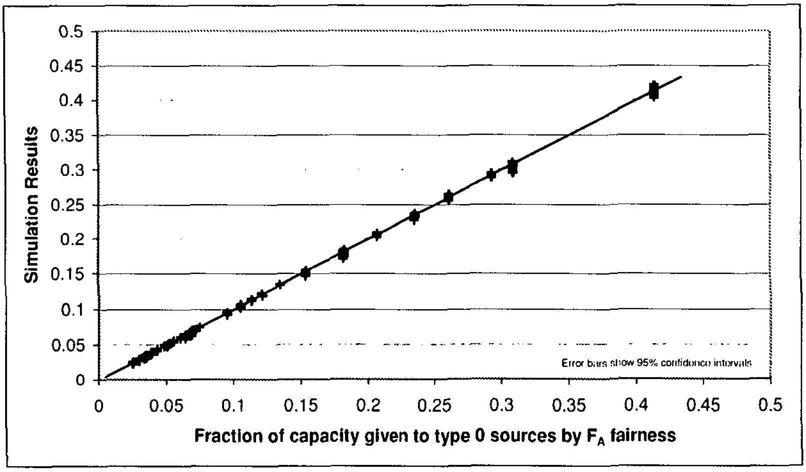


Figure 8.4: Scatter plot of F_A versus simulation results of type 0 sources' fraction of capacity. For small values of η and r_0. Each point represents a simulation run for values of the parameters v, w, I, d and p.

TCP-like parameter settings

Here we set $\eta = 0.5$, $\tau = 0.2$ and $r_0 = \frac{1}{\tau}$. We varied the parameters as in the previous case. As before, we found the scaled average to converge to that expected from F_A-fairness except for the case $d = 1$. This is illustrated by the scatter plot in Figure 8.5 for simulation values not including the $d = 1$ case. The error bars for 95% confidence are there, but perhaps not too visible given that the highest confidence interval is ± 0.002.

To summarise, we have established that F_A-fairness is a realistic model for TCP-like connections with equal round-trip times.

We can see the evolution of each of the source's time series in Figure 8.6 for the case of sources of type 0 and type i when $I = 2$, $d = 0.5$ and $p = 5$. They each start from random values and then oscillate. Convergence in the sense of Theorem 8.3.1 does not occur because η and r_0 do not tend to zero. However, it can observed that the time averages converge towards the rates predicted by F_A-fairness.

8.6.2 Rate independent feedback simulation

The case when the feedback is assumed to be rate independent as described in Section 8.2 was also simulated. This was done for small and TCP-like values of η

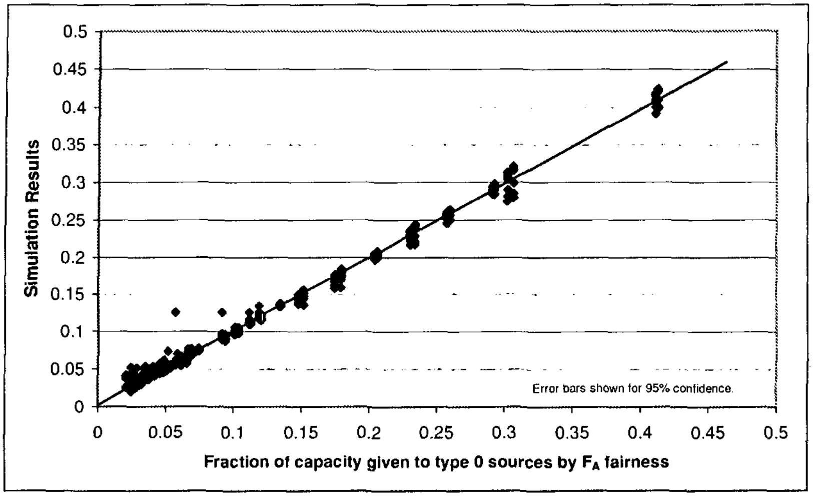


Figure 8.5: F_A versus simulation results of type 0 sources' fraction of capacity. For TCP-like parameter settings.

---

and r_0 and the results compared with the values as detemined by F_B-fairness.

We found that in both cases, the results agree with that anticipated from F_B-fairness, the main finding being that even with TCP-like parameter settings, the average rate converges in agreement with F_B.

We preserve the same conditions for simulation as in the rate proportional feedback case. The only difference is that the expectation of $E_i(t)$ is given by Equation (8.8) rather than Equation (8.7).

Small values of η and r_0

Again we consider values of $\eta = r_0 = 0.01$, where $\tau = 0.2$ and for the same range of parameters as in the previous simulations. We found the scaled average to converge to that expected from F_B-fairness. This is shown in Figure 8.7.

TCP-like parameter settings

Here we set $\eta = 0.5$, $\tau = 0.2$ and $r_0 = \frac{1}{\tau}$. Again the same parameter set was used. Figure 8.8 shows the converged rate of x_0 sources versus results from calculating F_B-fairness.

Even when F_B-fairness determines that sources of type 0 should be allocated

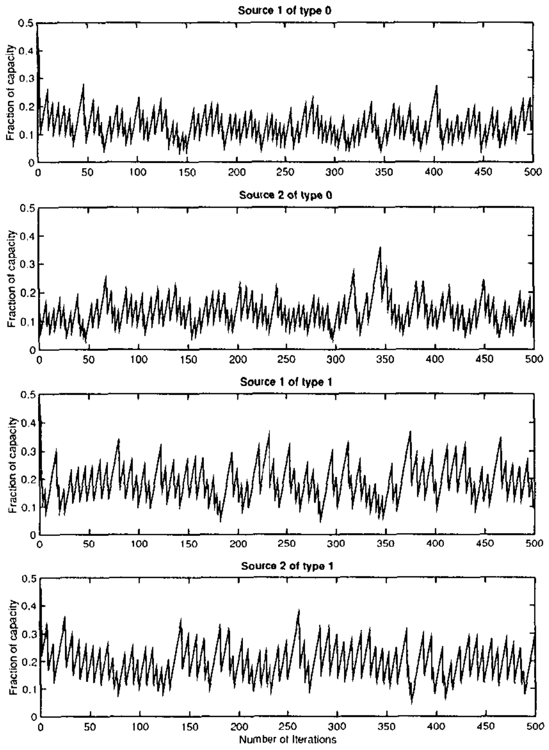


Figure 8.6: Example trace of the fraction of capacity time-series for two TCP-like sources of type 0 and two of type 1. ($I = 2, v = 2, w = 2, d = 0.5, p = 5$)

a rate of zero, the result converges to almost zero. This is in contrast to the rate allocated by proportional fairness. For a typical example, in one case the simulation average rate for type 0 sources converged to approximately 0.0000006. Here, F_B would allocate 0 to type 0 sources, while proportional fairness would allocate 0.08333323.

8.7 Rate as a function of packet loss ratio

The analysis also provides a simple means to derive the source rates as a function of the packet loss ratio experienced by the source. For a given rate distribution vector

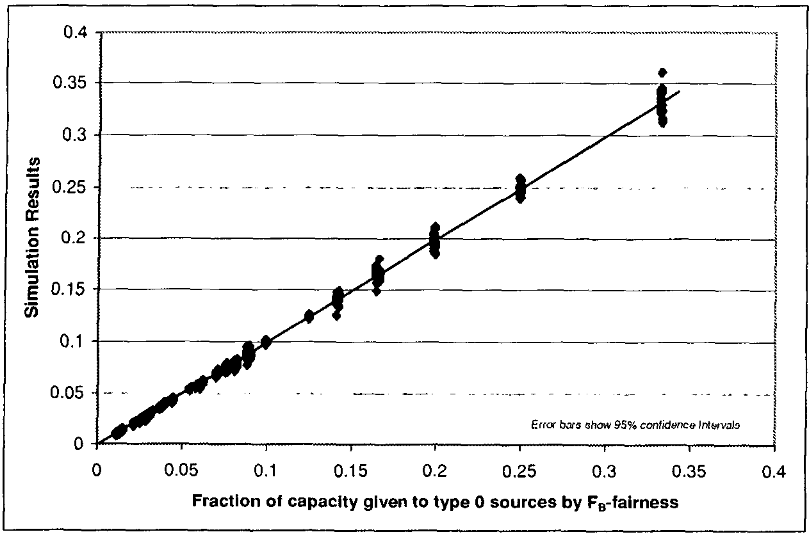


Figure 8.7: F_B versus simulation results of type 0 sources' fraction of capacity. For small values of η and r_0.

$\vec{x}$, the packet loss ratio $q_i(t)$ over the path of source i is

$$q_i(t) = \sum_{l=1}^{L} g_l(f_l(\vec{x}(t)))A_{l,i}$$

and we interpret Equation (8.7) by observing that, with case A, the expected feedback over one time cycle of duration τ is proportional to the number of packets sent which is given by $x_i(t)\tau$. With the hypothetical case B, we would say that the feedback is proportional to the packet loss ratio, but independent of the number of packets sent over one time interval (Equation (8.8)).

In the limit, we have for case A that

$$\lim_{t\to\infty} \frac{dx_i(t)}{dt} = 0$$

which combined with Equation (8.12) yields

$$r_0 - \tau x_i^*(r_0 + \eta x_i^*)q_i^* = 0$$

where $x_i^* = \lim_{t\to\infty} x_i(t)$ and $q_i^* = \lim_{t\to\infty} q_i(t)$.

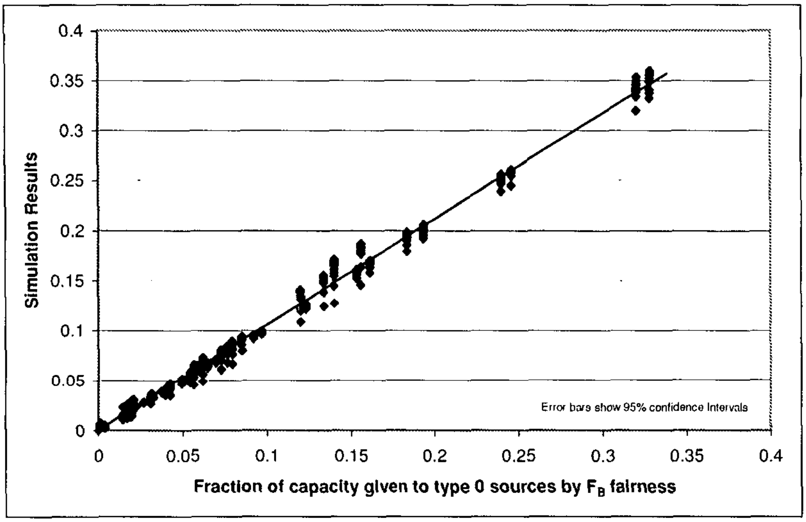


Figure 8.8: F_B versus simulation results of type 0 sources' fraction of capacity. For TCP-like parameter settings.

Solving for x_i^* yields

$$x_i^* = \frac{-\tau q_i^* r_0 + \sqrt{4 r_0 \tau q_i^* \eta + \tau^2 q_i^{*2} r_0^2}}{2\tau q_i^* \eta}. \tag{8.24}$$

For very small loss ratio q_i^*, the leading term in Equation (8.24) is given by

$$x_i^* \approx \sqrt{\frac{r_0}{\tau q_i^* \eta}}.$$

In the case of a TCP connection, we have $r_0 = \frac{1}{\tau}$ (packets per second) and $\eta = 0.5$. The previous equations give rates in packets per seconds; calling MSS the packet size in bits, we obtain the rates in bits per second from the previous equation:

$$x_i^* \approx \frac{MSS}{\tau} \frac{C}{\sqrt{q_i^*}} \text{ b/s}$$

with $C = \sqrt{2}$. This last result is in line with a family of similar results [31, 10, 25]. Our results differs in the value of C, which we attribute to the fact that we have assumed a fluid model converging towards some equilibrium, whereas in reality the TCP window size oscillates around some equilibrium.

If we did the same analysis with the modelling of case B, we would find that the leading factor in x_i^* would be in $\frac{1}{q_i^*}$, which does not match the previous results. We interpret this as a further confirmation that model A is closer to reality than model B.

8.8 Conclusions

TCP compliant sources with equal round trip times competing for bandwidth do not, as was previously thought, end up with a distribution of rates in accordance with proportional fairness.

Rather, we show that when feedback is rate dependent and negative feedback rare, the distribution agrees with F_A-fairness. In addition, we confirm this by derivation of the standard TCP throughput as a function of loss formula.

Even in the cases where feedback could no longer be assumed to be rate dependent, we have shown that proportional fairness would only approximate the long term rate distribution, and would be reflected closer by F_B-fairness.

An assumption of rare negative feedback is valid when the increments are small (i.e. the round-trip time τ is small) and the losses relatively low. It is our belief that these results essentially hold when we remove the assumption of rare negative feedback, but this remains to be verified.

Finally we mention that in [43] the authors extend the modelling described here to the case of heterogeneous round-trip times, resulting in a rate allocation given by the maximisation of

$$F_A^h(x) = \sum_{i \in S} \frac{1}{\tau_i} \log \frac{x_i}{r_i + \eta_i x_i}$$

subject to the usual link constraints

$$\sum_{j=1}^{I} A_{l,j} x_j \leq c_l \text{ for all } l$$

where τ_i is the round-trip time of source i. This differs from F_A-fairness by the round-trip time factor $1/\tau_i$.

8.9 Proofs

Proof of Lemma 8.5.1

$$F_A(x_0) = v \log \left(\frac{x_0}{r_0 + \eta x_0} \right) + wI \log \left(\frac{c - v x_0}{r_0 w + \eta (c - v x_0)} \right).$$

Solving the differential equation $F'_A(x_0) = 0$ results in a quadratic equation, $Ax_0^2 + Bx_0 + C = 0$ where,

$$A = \eta(v^2 - Iw^2), \quad B = -(Iw^2r_0 + v(2c\eta + r_0w)), \quad \text{and} \quad C = c(\eta c + r_0w).$$

If $A = 0$ (i.e. $v^2 = Iw^2$) then $x_0 = \frac{-C}{-B}$, which yields Equation (8.21).

If $A \neq 0$ ($v^2 \neq Iw^2$), we get the usual quadratic solution, $x_0 = \frac{-B\pm\sqrt{B^2-4AC}}{2A}$. Since we have only two extrema, only one of these solutions can lie in $(0, \frac{c}{v})$, and this must be the maximum. Denote the plus and minus roots by x_0^+ and x_0^- respectively. We have two cases.

case $A > 0$: Here $x_0^+ > 0$ since $\sqrt{B^2 - 4AC} > B$ because $B < 0$.

$x_0^- > 0$ if and only if $B + \sqrt{B^2 - 4AC} < 0$ which is true since $A, C > 0$. So $x_0^- > 0$.

Since both roots are greater than zero and only one of the roots can be less than c/v, the smallest of them, x_0^- must be the maximum.

case $A < 0$: $x_0^+ > 0 \iff \sqrt{B^2 - 4AC} < B$ which is false since $B > 0$ i.e. $x_0^+ < 0$.

$x_0^- > 0 \iff \sqrt{B^2 - 4AC} > B$ which is true since $B < 0$. So $x_0^- > 0$.

Thus, in both cases, x_0^- is the only possible solution, and so Equation (8.20) maximises $F_A(x_0)$ for $x_0 \in (0, \frac{c}{v})$. □

Proof of Lemma 8.5.2

$$F_B(x_0) = v\log(r_0 + \eta x_0) + Iw\log\left(r_0 + \frac{\eta(c - vx_0)}{w}\right).$$

Solving $F'_B(x_0) = 0$ results in Equation (8.23). This x_0 maximises $F_B(x_0)$, since $F''_B(x_0) < 0$, and is less than or equal to $\frac{c}{v}$. However, for certain values, $F'_B(x_0) = 0$ results in $x_0 < 0$, which is not in the valid range. Since $F_B(x_0)$ is a decreasing function in this case, $F_B(x_0)$ is maximised when $x_0 = 0$ for this case.□

Chapter 9

Conclusions

- We propose that *ABE be introduced as a replacement for the existing best-effort Internet service.* It enables best-effort traffic to experience a low queueing delay, at the expense of maybe less throughput, with no concept of reservation or signalling and while retaining the spirit of flat rate pricing. It does this with no deterioriation in the performance of traffic that does not need the lower delay.

  The design of a multimedia adaptive application that exploits the new degree of freedom offered by ABE is an important next step. To this end, Boutremans and Le Boudec [3] describe an ABE-aware audio application which combines colour adaptation, forward error correction control, and TCP friendly rate adaptation for the purpose of maximising a utility function.

- The *current state of the art implementation of ABE* is the use of serial DSD together with the *holding queue*, control loop, *green undergo virtual queue test*, and the *green at the back* algorithm. This, we believe, has the lowest level of complexity and still provides green traffic with the best service subject to local and throughput transparency.

  If a fix in TCP to remove the bias against flows with longer round-trip times [18] were to become widespread, then the *green undergo virtual queue test* in vanilla or serial DSD would be sufficient to satisy both local and throughput transparency and hence sufficent to implement ABE.

  The simulation results of Chapter 6 show the benefit of the new degree of freedom offered by ABE. We found that, under ABE, blue flows received more throughput than under a flat best-effort network while green flows received a low bounded delay. DSD thus facilitates multimedia adaptive applications to increase, in many cases, their utility.

Possible Future Work

Many avenues of future work arise of which we particularly note the following:

- *ABE in a class-based queueing environment*: We described the implementation of DSD for a constant rate server. It would useful to extend DSD to work in the more general environment where the link is shared [13] and ABE is but one class amongst potentially many.

- *ABE with per-flow queueing:* A scheduler which combines providing some flows with low-delay using ABE and a form of fair queueing [38] would have interesting fairness and flow isolation properties. One starting point would be the work on supporting TCP with per-flow queueing by Suter *et al* [42].

- *Increasing average green delay*: In Section 4.2 on page 53, we described algorithms which do not affect green loss rate but reduce the difference in delay between blue and green flows. This work could be extended to ensure the probability of inducing a green loss, by making it wait, is kept small.

- *Extension of DSD queueing analysis*: The queueing analysis could be extended to consider other arrival and service processes and to examine the structure of the green and blue loss distributions. The Markov state description could be broadened to incorporate strategies such as *green at the back*.

- *Fairness analysis of local transparency*: The extension of the work in Chapter 8 on the fairness of additive increase/multiplicative decrease to determine the long-term distribution of rates between blue and green traffic under optimal local transparency is a natural follow on. The difficulty lies in determining the relative delays and loss rates for blue and green traffic under realistic models that capture the bursty nature of TCP traffic. One possible approach would be the use of Markov modulated Poisson processes or the N-burst arrival model of TCP traffic as described by Schwefel in [39].

  A further interesting, and admittedly hard, open problem that arises is the determination of a distributed algorithm, in the spririt of Low and Lapsley [28], that optimises the distribution of rate amongst blue and green flows subject to the constraint of local transparency and the maximum queueing delay d.

Bibliography

[1] A. Benveniste, M. Metivier, and P. Priouret. *Adaptive Algorithms and Stochastic Approximations*. Springer Verlag, Berlin, 1990.

[2] Jean Bolot, S. Fosse-Parisis, and Don Towsley. Adaptive FEC-based error control for interactive audio in the internet. In *IEEE Infocom*, 1999.

[3] Catherine Boutremans and Jean-Yves Le Boudec. Adaptive delay aware error control for internet telephony. In *Proceedings of 2nd IP-Telephony workshop*, pages 81 92, Columbia University, New York, April 2001.

[4] Neal Cardwell, Stefan Savage, and Thomas Anderson. Modeling TCP latency. In *infocom*, Tel Aviv, Israel, March 2000.

[5] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17:1–14, June 1989.

[6] Jon Crowcroft. All you need is just 1 bit, keynote presentation. In *IFIP inproceedings on Protocols for High Speed Networks*, October 1996.

[7] Christophe Diot, Christian Huitema, and T. Turletti. Multimedia application should be adaptive. In *HPCS*, August 1995.

[8] C. Dovrolis, D. Stiliadia, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. In *ACM Sigcomm*, 1999.

[9] T. Ferrari, Werner Almesberger, and Jean-Yves Le Boudec. SRP: a scalable resource reservation protocol for the internet. *Computer Communications:Special issue on 'Multimedia networking'*, 21(14):1200 1211, September 1998.

[10] Sally Floyd. Connections with multiple congested gateways in packet switched networks, part 1. *ACM Computer Communication Review*, 22(5):30 47, October 1991.

[11] Sally Floyd. TCP and explicit congestion notification. *ACM Computer Communication Review*, 24(5):10 23, October 1994.

[12] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397 413, August 1993.

[13] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4), August 1995.

[14] TCP friendly web site. http://www.psc.edu/networking/tcp_friendly.html.

[15] S. Golestani and S. Bhattacharyya. End-to-end congestion control for the internet: A global optimization framework. In *ICNP*, October 1998.

[16] Roch Guerin and Vinod Peris. Quality-of-service in packet networks: Basic mechanisms and directions. *Computer Networks and ISDN Systems. Special issue on multimedia communications over packet based networks*, 31(3):169 189, February 1999.

[17] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured forwarding PHB group. Request for Comments 2597, Internet Engineering Task Force, June 1999.

[18] Tom Henderson, E. Sahouria, Steven McCanne, and Randy Katz. Improving fairness of TCP congestion avoidance. In *IEEE Globecom*, November 1998.

[19] Christian Huitema. Quality today in the Internet. Technical report, Telecordia, February 2000.

[20] Van Jacobson, Kathy Nichols, and K. Poduri. An expedited forwarding PHB. Internet Draft, Internet Engineering Task Force, February 1999. Work in progress.

[21] Frank P. Kelly, A. K. Maulloo, and D.K.H. Tan. Rate control for commmunication networks: Shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[22] K. Kilkki and J. Ruutu. Simple integrated media access - an internet service based on priorities. In *6th International inproceedings on Telecommunication Systems*, 1998.

[23] Leonard Kleinrock. *Queueing Systems — Computer Applications*, volume 2. Wiley-Interscience, New York, New York, 1976.

[24] H.J. Kushner and D.S. Clark. Stochastic approximations for constrained and unconstrained systems. *Applied Mathematical Sciences*, 26, 1978.

[25] T. V. Lakshman and U. Madhow. The performance of TCP for networks with high bandwidth delay products and random loss. *IEEE/ACM Transactions on Networking*, 5(3):336 350, June 1997.

[26] Jean-Yves Le Boudec and Patrick Thiran. Network calculus viewed as a min-plus system theory applied to communication networks. Technical Report SSC/1998/016, EPFL-DSC, April 1998.

[27] L. Liung. Analysis of recursive stochastic algorithms. *IEEE Transactions on Automatic Control*, 22:551–575, 1977.

[28] Steven H. Low and David E. Lapsley. Optimization flow control–i: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, December 1999.

[29] Laurent Massoulie and James Roberts. Fairness and QoS for elastic traffic. In *CNET*, 1998.

[30] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. Request for Comments 2018, Internet Engineering Task Force, October 1996.

[31] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behaviour of the TCP congestion avoidance algorithm. *Computer Communications Review*, 3, July 1997.

[32] Martin May, Jean Bolot, Christophe Diot, and A. Jean-Marie. 1-bit schemes for service discrimination in the internet: Analysis and evaluation. Technical Report 3238, INRIA, 1997.

[33] R.K. Miller and A.N. Michell. *Ordinary Differential Equations*. Academic Press, 1982.

[34] Yan Moret and Serge Fdida. A proportional queue control mechanism to provide differentiated services. In *International Symposium on Computer Systems*, October 1998.

[35] T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. Relative delay differentation and delay class adaptation in core-stateless networks. In *Infocom*, 2000.

[36] ns v2 simulator. http://www.isi.edu/nsnam/ns.

[37] J. Padhye, Victor Firoiu, Dan Towsley, and James Kurose. Modeling TCP throughput: a simple model and its empirical validation. In *ACM Sigcomm*, 1998.

[38] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE/ACM Transactions on Networking*, 1(3):344 357, June 1993.

[39] Hans-Peter Schwefel. Behaviour of TCP-like elastic traffic at a buffered bottleneck router. In *Infocom*, 2001.

[40] Sheldon M. Ross. *Introduction to Probability Models. Seventh Eedition.* Academic Press, 2000.

[41] I. Stoica and H. Zhang. Providing guaranteed services without per flow management. In *ACM Sigcomm*, pages 81 94, 1999.

[42] Bernhard Suter, T. V. Lakshman, Dimitrios Stiliadis, and Abhijit K. Choudhury. Buffer management schemes for supporting TCP in gigabit routers with per-flow queueing. *IEEE Journal on Selected Areas in Communications*, 17(6):1159–1169, June 1999.

[43] Milan Vojnovic, Jean-Yves Le Boudec, and Catherine Boutremans. Global fairness of additive-increase and multiplicative-decrease with heterogeneous roundtrip times. In *IEEE Infocom*, March 2000.

[44] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. In *IEEE*, volume 83, October 1995.

List of Figures

List of Tables

Publications

- P. Hurley, M. Kara, J. Y. Le Boudec, P. Thiran. "ABE: Providing a Low Delay Service Within Best-Effort". *IEEE Network*, May 2001.
- P. Hurley, M. Kara, J. Y. Le Boudec, P. Thiran. "A Novel Scheduler For a Low Delay Service Within Best-Effort". *IwQoS 2001*, Karlsruhe, Germany, June 2001.
- P. Hurley, J. Y. Le Boudec, P. Thiran. "The Asymmetric Best-Effort Service". *Proceedings of IEEE Globecom 1999*, Rio de Janeiro, Brazil, December 1999.
- P. Hurley, J. Y. Le Boudec, P. Thiran. "A Note on the Fairness of Additive Increase and Multiplicative Decrease". *Proceedings of ITC-16*, Edinburgh, June 1999.
- P. Hurley, J. Y. Le Boudec. "A Proposal for an Asymmetric Best-Effort Service". *Proceedings of IEEE/IFIP IWQoS '99*, London, England, May 1999.
- P. Hurley, J. Y. Le Boudec, P. Thiran. "The Fairness of Additive Increase and Multiplicative Decrease". *10th INFORMS Applied Probability Conference*, University of Ulm, Germany, July 1999.
- P. Hurley, G. Iannaccone, M. Kara, J. Y. Le Boudec, P. Thiran. "The Alternative Best-Effort Service". Internet-Draft, December 2000.
- P. Hurley, M. Kara, J. Y. Le Boudec, P. Thiran. "Packet Scheduler with Deadline-based Service Algorithm and Virtual Queue". Patent in progress.

Curriculum Vitae

Paul Hurley graduated with first class honours in computer and mathematical science from University College, Galway, Ireland in 1995. He then worked as a design engineer both for Videologic, London, and for Digital (now Compaq) in Galway, Ireland. In October 1996, he joined the Network Communications Laboratory at EPFL which later merged into the Institute for computer Communications and Applications (ICA). He initially worked on a project on a project dealing with network protocols over satellite link, in the first year, with Swisscom and then, for the second year, with the European Space Agency. He then worked on the ABE *project in conjunction with Sprint* Advanced Technology Labs where, in the summer/autumn of 2000, he also undertook an internship.