A Simulation Study of Traffic Conditioner Performance

by
Marthinus David Strauss

June 2005

Submitted in partial fulfilment of the requirements for the degree Master of Science (Computer Science) in the Faculty of Engineering, Built Environment and Information Technology, University of Pretoria, Pretoria

Soli Deo Gloria

A Simulation Study of Traffic Conditioner Performance

by
Marthinus David Strauss

Abstract

A traffic conditioner is an element of the Differentiated Services architecture. This architecture is used to regulate quality of service in computer networks. Five traffic conditioners were selected for the study. These include the token bucket marker and four conditioners described in RFCs. The contribution of this dissertation is two-fold. Firstly, it presents process algebra models of the five identified traffic conditioners. These models provide succinct descriptions of the conditioners thereby highlighting essential features. The models are, however, not intended for model checking purposes, but rather serve as a convenient pedagogical device. The second and main contribution of the dissertation is a simulation study to investigate the relative performance of the five traffic conditioners across a range of simulated scenarios in which traffic patterns and subscription levels are varied in a fixed network topology. Two performance measures—TargetRatio and GreenRatio—are defined, justified, and used to compare the traffic conditioners. The GreenRatio measure was found to be more discriminating than the TargetRatio measure. A variant of the GreenRatio measure was used to further illuminate the differences between conditioners. The simulation results suggest that the performance of the conditioners are sensitive to parameter values such as token bucket size and that bursty traffic patterns are particularly sensitive to these parameters. Under such bursty conditions, these parameters should be chosen with care.

Keywords: Quality of service, Diffserv, traffic conditioners, simulation, FSP.

Degree: Magister Scientia
Supervisor: Prof. D. G. Kourie
Department of Computer Science

Acknowledgements

I would like to express my gratitude to the following individuals who contributed to the eventual success of this work.

- To my supervisor, Prof. Derrick Kourie for invaluable guidance and support throughout the course of this work and for enriching conversation on topics ranging from religion to regression.
- To André Jordaan and Richard Ellerbrock for introducing me to the world of corporate networking and for being the best team-mates ever.

To my wife, Janet, I would like to express sincere appreciation for her never-ending inspiration and, above all, for her—sometimes tried—patience.

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

AF	Assured Forwarding
AFC	Aggregate Flow Control
ATM	Asynchronous Transfer Mode
AQM	Active Queue Management
BA	Behaviour Aggregate
CBM	Counters-Based Modified
CBR	Constant Bit Rate
CBS	Committed Burst Size
CDF	Cumulative Distribution Function
CIR	Committed Information Rate
COPS	Common Open Policy Service
COS	Class Of Service
CSP	Communicating Sequential Processes
CTR	Committed Target Rate
Diffserv	Differentiated Services
DS	Differentiated Services
DSCP	Differentiated Services CodePoint
DSField	Differentiated Services Field
DTFT	Discrete Time Fourier Transform
EBM	Equation-Based Marking

EBS	Excess Burst Size
ECN	Explicit Congestion Notification
EF	Expedited Forwarding
FGN	Fractional Gaussian Noise
FGM	Fractional Gaussian Motion
FTP	File Transfer Protocol
EXPOO	Exponential On/Off
FFT	Fast Fourier Transform
FGN	Fractional Gaussian Noise
FSP	Finite State Processes
FTP	File Transfer Protocol
IAB	Internet Architecture Board
ICMP	Internet Control Message Protocol
IntServ	Integrated Services
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ItswTCM	Improved time-sliding window Three Colour Marker
LAN	Local Area Network
LRD	Long-Range Dependance
MBTCM	Memory-Based Three Colour Marker
MF	Multi-Field
MIB	Management Information Base
MPLS	Multiprotocol Label Switching

OA	Ordered Aggregate
PAROO	Pareto On/Off
PBS	Peak Burst Size
PDB	Per-Domain Behaviour
PDF	Probability Distribution Function
PHB	Per-Hop Behaviour
PIR	Peak Information Rate
PTR	Peak Target Rate
QoS	Quality of Service
RED	Random Early Detection
RFC	Request For Comments
RIO	RED with In/Out bit
RSVP	Resource ReSerVation Protocol
RTT	Round Trip Time
SLA	Service Level Agreement
SLS	Service Level Specification
SNMP	Simple Network Management Protocol
srTCM	Single Rate Three Color Marker
TBM	Token Bucket Marker
TCA	Traffic Conditioning Agreement
TCB	Traffic Conditioning Block
TCS	Traffic Conditioning Specification
TCP	Transmission Control Protocol
TOS	Type Of Service
trTCM	Two Rate Three Colour Marker

TSW	Time Sliding Window
TSW2CM	Time Sliding Window Two Colour Marker
TSW3CM	Time Sliding Window Three Colour Marker
TTL	Time To Live
UDP	User Datagram Protocol
VBR	Variable Bit Rate
VoD	Video-on-Demand
WAN	Wide Area Network
WRED	Weighted RED

1. Introduction

> The last thing one knows when constructing a work is what to put first.
>
> *Blaise Pascal*

Quality of Service (QoS) in computer networks is, it seems, in some sense comparable to a rather strange animal—the duck-billed platypus. The platypus, eventually classified as an egg laying mammal, has various seemingly incompatible morphological features. It is the size of a mole, has little eyes, webbed feet (with claws), a tail, fur, and a beak like a duck. One scientist, on first seeing a stuffed specimen, was so amazed and puzzled by the appearance of the creature that he suspected that it was put together by artificial means. Is it a fish, bird, mammal? The classification of the platypus took time and generated a fair amount of debate since scientists with different backgrounds tended to focus on different features of this strange animal.

Similarly, networking professionals with different backgrounds expect different features from QoS. Those with a traditional telephony background tend to think of QoS differently from those with a pure data networking background. Some network engineers might argue for absolute guarantees through reservation of resources, whilst others might argue for prioritising traffic (differentiating between classes of traffic) and providing statisticaly based guarantees.

Differentiated Services (Diffserv) is a framework for providing different levels of service to various classes of traffic through the relative priorities assigned to the classes. The Diffserv architecture, described in Chapter 2, consists of a number of components, one of which is the so-called traffic conditioner. The traffic conditioner is located on the boundary of a Diffserv-capable network and is responsible for measuring incoming traffic and assigning the constituent packets to different service classes based on predetermined policy rules. Five different traffic conditioners are described in Chapters 2 and 3. The conditioners are used to determine whether an incoming traffic stream is in- or out-of-profile and to assign different 'colours' to packets. These are based on the traffic conditioning specification which forms part of the service level agreement that has been negotiated between the network subscriber and provider. Packets are prioritised based on the colours assigned to them. This means that, depending on their colour, some packets will have a greater probability of being discarded in the event of congestion.

Since there is a number of different traffic conditioners, a network designer is faced

with the challenge of deciding on one of the available conditioners to employ in a particular network. The designer might base her decision on the expected incoming traffic patterns and the subscription level at which the network will be operated. It is in this area of design that this dissertation aims to make a contribution.

Simulation-based experiments were performed to determine the relative performance of five traffic conditioners based on two related performance measures which are defined and motivated in Chapter 4. The performance of the traffic conditioners were measured in a number of difference scenarios based on the expected incoming traffic pattern as well as the subscription level of the network. The aim of the experiments was to determine whether some traffic conditioners perform "better" than others in the particular scenarios under investigation.

Simulation was employed since such models are relatively inexpensive to set up and fairly simple to manipulate. However, simulation suffers from some inherent weaknesses. Simulation is no substitute for reality, only an approximation of it; therefore the value of the results depend on the quality of the models used in the experiments. One should keep these weaknesses in mind when interpreting any observed simulation result.

The remainder of this present chapter contains an overview of the structure of the dissertation. The purpose and contents of each chapter is summarised at a high level to provide a concise bird's-eye view of the dissertation.

Chapter 2 provides relevant background information to describe the context for the study. Quality of service is defined and motivated and some approaches to achieve service quality are mentioned. One approach, Diffserv, is discussed in some detail since aspects of this approach are further examined in the later chapters of the dissertation. One feature of the queues in nodes in a Diffserv network is that they need to be actively managed. One such active queue management scheme is Random Early Detection (RED) and its operation is also described in the chapter. As mentioned before, an essential element in the Diffserv architecture is the so-called traffic conditioner that measures and marks packets arriving at the Diffserv boundary according to certain configuration parameters. A functional description of each of the five traffic conditioners under consideration in the rest of the dissertation is provided in the chapter. The chapter closes with descriptions of the models used to simulate the range of traffic patterns which were used to compare the traffic conditioners.

Since Diffserv traffic conditioners form such an integral part of the study, they should be well-understood. Process algebra models of the various conditioners were derived as the vehicle to facilitate this goal. The models of the relevant traffic conditioners, together with diagrams which illustrate the structure of the models, are presented in Chapter 3. These models aid in understanding the traffic conditioners by focussing on the essential actions of the conditioners and by abstracting away implementation

specifics. From the models, two classes of conditioners are easily identified: token-bucket based conditioners and time-sliding window based conditioners.

To further study the behaviour and the relative performance of the traffic conditioners, simulation based experiments were conducted. Chapter 4 contains descriptions of the different experiments that were conducted. Since the aim of the study is to determine which traffic conditioners are more appropriate in certain scenarios, different simulation scenarios were created by using a range of traffic patterns and subscription levels in the network. Each of the five traffic conditioners are employed in each of the different scenarios. The chapter describes the network topology employed in the simulations; the RED parameters are provided; the traffic conditioning parameters are provided; and the configurations of the different traffic generators are described. The simulation scenarios varied between three reservation levels, five traffic conditioners, and five traffic patterns. The performance measures which were used to determine the relative performance of the traffic conditioners are also defined and motivated in this chapter.

The results and statistical analysis of the experiments described in Chapter 4 are presented in Chapter 5. The chapter is structured around the two performance measures. The first measure determines how well subscribers achieve their target rates. The target rate is the average amount of traffic that a subscriber should be able to send through the network during some time interval. The second performance measure determines how well the network serviced that component of the subscriber's traffic that was within the contracted profile. The results of the experiments are discussed and further experiments which were done to obtain additional evidence for some arguments are described.

The dissertation closes with Chapter 6. The chapter contains a section on work relating to the area of traffic conditioners in the Diffserv context. The section is organised into a number of areas, viz. earlier work on the performance of Diffserv in general; work in which novel traffic conditioners are proposed; work on the dynamic configuration of Diffserv components; and work on the ancillary issue of pricing differentiated services. Further, the chapter contains a list of lessons learnt through the study. The list includes some high-level lessons on simulation in general as well as lessons pertaining to the observations from the experiments. Since it is natural to limit the scope of a study, the chapter concludes with some suggestions for further research.

Finally, to summarise the layout of the dissertation: Chapter 2 provides relevant background information, Chapter 3 presents the process algebra models of the traffic conditioners, Chapter 4 describes the experimental setup, whilst the results of the experiments are discussed in Chapter 5. The results of the statistical tests are provided in Appedix A. The dissertation is then concluded in Chapter 6. A list of abbreviations as well as a glossary are provided for the reader's convenience.

2. Background Information

> The skill of writing is to create a context in which other people can think.
>
> *Edwin Schlossberg*
>
> Learn as much by writing as by reading.
>
> *Lord Acton*

This chapter provides an overview of the most pertinent concepts that are to be used later in the dissertation.

Section 2.1 gives an overview of Quality of Service; Section 2.2 describes the Differentiated Services scheme for providing quality of service; Section 2.3 describes Random Early Detection, one form of active queue management; Section 2.4 describes the different traffic conditioners that will be of interest in this dissertation; and Section 2.5 describes some aspects of modelling and synthesising telecommunication network traffic.

2.1. Quality of Service

Quality of Service (QoS) refers to the ability of a network to provide selected traffic with better service. This section provides an introduction to QoS in the Internet domain. The need for QoS is firstly motivated and then some mechanisms for QoS provisioning are mentioned.

2.1.1. The Need for Quality of Service

From Figure 2.1 (Nua Ltd., 2003) it can be seen that the number of Internet users has increased significantly over the last few years. With the increasing number of Internet users, more resources are required to satisfy user requirements. Commercial ventures also take advantage of the increasing number of users to create new sources of income by creating novel applications that might interest Internet users. The users and their application requirements drive the advance of networks.

According to Firoiu et al. (2002) there are two likely drivers for network services with guaranteed QoS. One comes from applications that have strict QoS requirements such

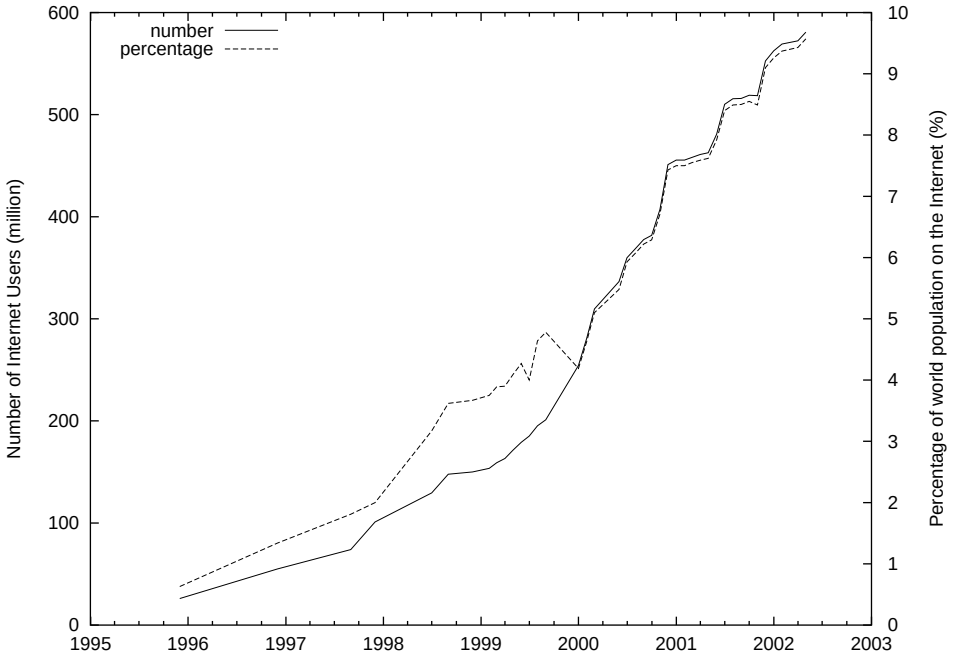


Figure 2.1.: Worldwide Internet Users.

as Video-on-Demand (VoD) over the Internet and Internet Protocol (IP) telephony. The second driver is the need for service differentiation. Due to the competitive nature of the Internet marketplace network service providers will try to offer their users better quality of service guarantees than their competitors.

The network protocol used predominantly in the Internet is IP. One of the reasons for the success of IP is the fact that is is relatively simple. The design principle for IP was derived from the "end-to-end argument" (Saltzer et al., 1984; Blumenthal and Clark, 2001; Clark et al., 2002). This argument states that intelligent functions are limited to the edges of the network and the core is relatively unintelligent. IP routers in the core of the network check the address of the IP datagram against a forwarding table to determine the correct next-hop interface for the datagram. If the queue for the next-hop is large, the datagram might experience delay. If the queue is full or unavailable, the router might discard the datagram. The result of this behaviour is a so-called "best-effort" service with unpredictable delays and data loss.

Support for different types of service was one of the original design priorities (Clark, 1988) of the Internet protocols. For a network service provider to be able to provide good quality service, it should be able to provide some guarantees to the subscribers of the service. QoS is the ability of a network element to have some level of assurance that its traffic and service requirements can be satisfied. QoS does not create bandwidth.

Rather, it manages the available bandwidth according to the needs of the applications.

By providing different levels of service, one creates an incentive to steal. One user might pay for a better service and another could try to steal some of that service. As a consequence, QoS requires policy enforcement and a policy management infrastructure. QoS also implies the need for accounting and billing. All these together—policy management, authentication, and accounting/billing—are essential to the success of QoS provision.

2.1.2. Providing Network QoS

Network QoS can be defined in various ways, depending on how different service requirements such as performance, availability, reliability, and security are factored into the definition. In the rest of this dissertation the focus will be on the performance aspect of network QoS. Typical performance metrics used in network QoS are bandwidth, throughput, delay, delay variation, and packet loss rate. One can use these performance metrics to specify network performance guarantees in various forms (Firoiu et al., 2002):

- absolute, such as a network connection of 1 Mbps bandwidth is guaranteed at all times;
- probabilistic, for example, network delay is guaranteed to be no more than 100 ms for 99% of the packets;
- time average, for example, packet loss rate is less than 10^{-5} measured over a month.

To be able to provide QoS guarantees, it is necessary to deploy appropriate QoS control mechanisms in the operations and management of a network. QoS mechanisms can be characterised by several aspects (Firoiu et al., 2002):

- The *time scale* at which a control mechanism operates. The different time-scales are listed below, in order of increasing time scale.
  - The fastest time scale is at the packet level, which is the smallest level at which a network can exert control. QoS control mechanisms that operate at the packet level include traffic conditioners, packet schedulers, and active queue management.
  - The next time scale is round-trip time at which scale feedback-based control mechanisms such as congestion control operate.

  - The session time scale is the time for which a user session lasts and during which admission control and QoS routing operate.
  - Beyond the session time scale there are a variety of long-term control mechanisms such as traffic engineering, time-of-day service pricing, and capacity planning.

- Another aspect of QoS mechanisms is the *granularity* of the control information used by the mechanism to make control decisions. The finest granularity of control is per-flow state information which can be used to exert control on single user sessions. Courser grain controls are based on aggregates of flows. The level of granularity depends on the level of aggregation (for example, per host, per network, or per application).

- The next aspect is the *carrier* of the control information, be it in the routers or in the packet header.

- Closely related to the previous aspect is the *location* of control. This aspect determines where control is exerted (such as at end-hosts, the network edge, or the network core).

The Internet Engineering Task Force (IETF) (Harris, 2001) has proposed numerous service models and mechanisms to provide QoS. Some of the more important of these include the Integrated Services (IntServ)/Resource ReSerVation Protocol (RSVP) model (Braden et al., 1994, 1997; Herzog, 2000), the Diffserv model (Nichols et al., 1998; Blake et al., 1998; Carpenter and Nichols, 2002), Multiprotocol Label Switching (MPLS) (Rosen et al., 2001b,a), Traffic Engineering (Awduche et al., 2002), and Explicit Congestion Notification (ECN) (Ramakrishnan et al., 2001).

The IntServ model is characterised by resource reservation. Applications need to set up paths and reserve resources before data is transmitted. RSVP is a signalling protocol used to set up paths and reserve resources. Routers in the core need to keep track of per-flow state information resulting in scalability challenges.

In the Diffserv model packets are marked differently to create traffic classes. Packets that are marked the same receive the same treatment in the router. Control is enforced on aggregates of flows resulting in better scalability than is obtained with IntServ.

MPLS is a packet forwarding scheme. Packets are assigned labels at the ingress of an MPLS domain. Packet classification and treatment are based on the value of the label.

Since Diffserv is the theme of the rest of the dissertation, it is described in a fair amount of detail in the following section. Diffserv was chosen since it is scalable, relatively easy to implement, and more recent than IntServ.

2.2. Differentiated Services

In this section the Differentiated Services model is described. An overview of the architecture, standards, and mechanisms to achieve service differentiation are discussed.

2.2.1. Overview

The Diffserv architecture was created to implement scalable service differentiation in the Internet. Scalability is achieved by maintaining classification state information based on aggregates of traffic flows instead of on the individual flows. The classification is conveyed by IP-layer packet marking using the Differentiated Services (DS) field in the IP header.

Packets are classified and marked to receive a particular forwarding behaviour at nodes on their path. Sophisticated classification, marking, policing, and shaping is only implemented at the boundary of the network or at end-hosts.

Network resources are allocated to traffic streams in terms of service provisioning policies that govern how traffic is marked and conditioned upon entering the DS capable network, and how traffic is forwarded in the network.

A 'service' defines some set of significant characteristics of packet transmission in one direction across a set of one or more paths within a network. These characteristics may be specified quantatively or statistically in terms of throughput, delay, jitter, or packet loss; or may otherwise be specified in terms of some relative priority of access to network resources. The Diffserv framework creates services within a DS domain by applying rules at the edges to create traffic aggregates, called Behaviour Aggregates (BAs). The traffic aggregates are then treated in some pre-defined manner by the network.

Service differentiation is desired to accommodate heterogeneous application requirements and to permit differentiated pricing of Internet services.

The Diffserv functionality is mapped to a number of functional components that are appropriately distributed across the various network nodes within the Diffserv domain.

2.2.2. Differentiated Services Architectural Model

This section relies on Blake et al. (1998) to define the Diffserv architecture. One of the most important properties of the architecture is that it is fairly simple. Simplicity allows for greater scalability and manageability.

The architectural features of Diffserv are encapsulated within a Diffserv domain. The following subsection outlines the various components in such a domain. The next

section explains the Diffserv activities known as traffic classification and conditioning. These activities take place at a network packet level.

Packets that enter a Diffserv domain are classified, may be conditioned, and are assigned to a BA. Such a BA may be thought of as a traffic aggregate which should be treated in some specified manner by the network. Packets in the same BA are marked with the same value in the DS field in the IP header. This value of the DS field is called the Differentiated Services CodePoint (DSCP). In the core of the Diffserv domain, the packets are forwarded according to the value of the DSCP.

The externally observable forwarding behaviour with which a BA is endowed at a DS node is called a Per-Hop Behaviour (PHB). PHBs are further discussed in Section 2.2.2.3.

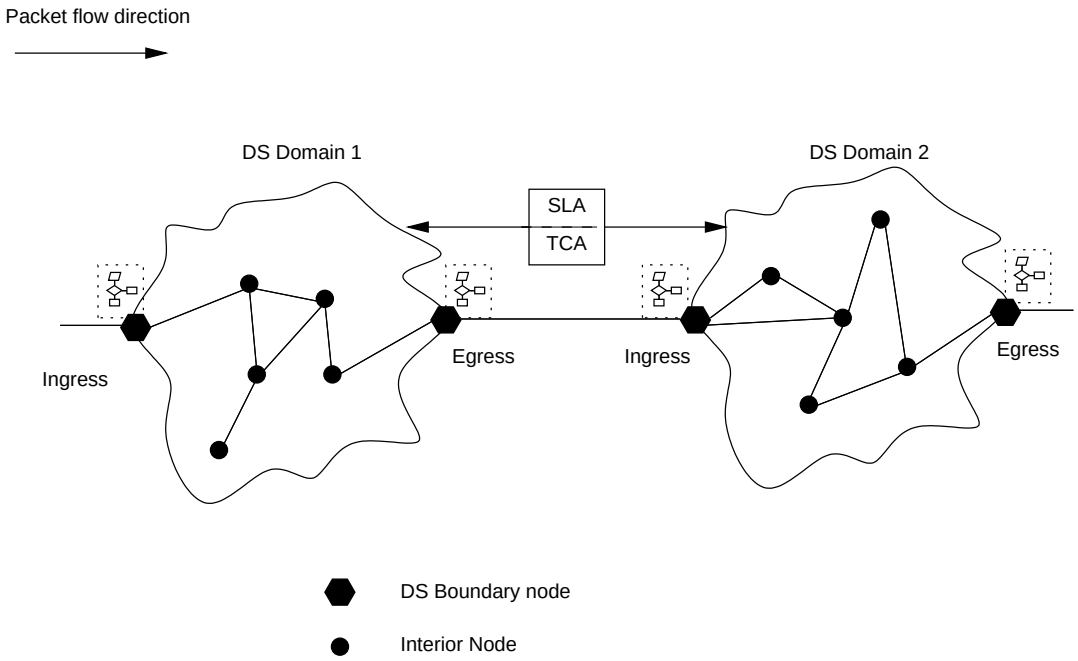


Figure 2.2.: Two Diffserv domains.

2.2.2.1. Differentiated Services Domain

A Diffserv domain is a contiguous set of DS nodes with the same service provisioning policy and set of PHBs. It has a well defined boundary. Boundary nodes apply rules to traffic that enter the DS domain. Traffic that enters or exits the domain is called ingress and egress traffic, respectively. Internal nodes select a forwarding behaviour (PHB) according to the value of the DS codepoint.

Figure 2.2 provides a diagram illustrating some of the components found in a DS domain.

Boundary and Interior Nodes A DS domain consists of both boundary and interior nodes. Boundary nodes connect the DS domain to other DS domains or to non-DS domains. Interior nodes only connect to other interior or boundary nodes in the same domain.

Boundary nodes must be able to perform traffic conditioning based on a Traffic Conditioning Specification (TCS). A TCS specifies, for example, service performance parameters (e.g. throughput and prop probability), traffic profiles within which the service is offered (e.g. token bucket parameters), the period over which the service is offered, and how packets are marked.

Ingress and egress nodes A boundary node can act as both an ingress and egress for traffic, depending on the direction of the stream.

A boundary node that acts as an ingress node must ensure that the traffic conforms to the TCS that was set up between the customer and provider domain.

The egress node can condition traffic exiting the domain to ensure that it conforms to the TCS of the downstream domain.

2.2.2.2. Traffic Classification and Conditioning

Differentiated services are extended across a DS domain boundary by establishing a Service Level Specification (SLS) between the upstream and the downstream domains. The SLS defines packet classification and re-marking rules, traffic profiles, and actions to be performed for in- and out-of-profile streams. The previously mentioned TCS forms an integral part of the SLS.

The subsequent paragraphs explain classifiers, traffic profiles, and traffic conditioning in more detail.

Classifiers A classifier selects packets in a traffic stream, based on the contents of a portion of the packet header. In the Diffserv context there are two types of classifiers: BA classifiers and multi-field classifiers. BA classifiers classify packets based on only the value of the DS codepoint. Multi-field classifiers use additional fields in the packet header to classify the packet.

Classifiers are used to 'steer' certain packets (matching some rule) to an element of a traffic conditioner for further processing. Classifiers are configured by some management procedure according to a TCS.

Traffic Profiles A traffic profile specifies the properties of a traffic stream selected by a classifier. It also provides rules for determining whether a packet is in- or out-of-profile.

A profile could, for example, specify that all packets marked with codepoint X should be measured against a token bucket meter with rate r and burst size b.

Different so-called conditioning actions may be applied to in- and out-of-profile traffic. These actions are carried out by components called traffic conditioners, as discussed in the next section.

Traffic Conditioners A traffic stream is selected by a classifier which steers the packets to a logical instance of a traffic conditioner. A traffic conditioner may contain components called a meter, a shaper, a dropper, and a marker. A meter is used to measure the traffic stream's characteristics against a profile. The state of the meter regarding a specific packet is used to ascertain whether or not the packet is out-of-profile.

Out-of-profile traffic could be acted upon in a variety of ways. It might be queued until it is in-profile (shaped); it might be discarded (dropped or policed); it might be marked with a new codepoint (re-marked); or it might be forwarded without change but then also trigger an accounting procedure.

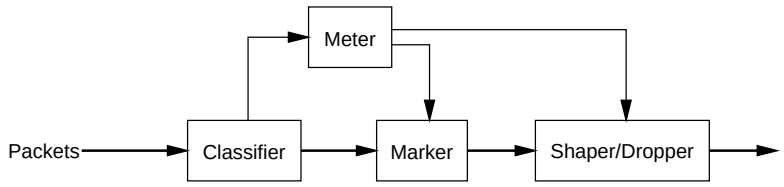


Figure 2.3.: Logical view of a Packet Classifier and Traffic Conditioner components.

Figure 2.3 gives a logical representation of a traffic classifier and the most important functional components of a conditioner. The various components in the figure are related as follows.

A meter measures the properties of packet stream that has been selected by a classifier against a profile. The meter then passes state information to other conditioning functions to trigger particular actions.

A marker sets the DS field of a packet to a particular codepoint, thereby adding it to a particular BA.

Shapers delay packets in a stream to bring them in compliance with a given profile. A shaper has finite buffer space and will discard packets if the buffer becomes full.

On the other hand, droppers discard packets in a stream to bring them in compliance with a profile. This is called 'policing' the stream.

Traffic conditioners are usually located within DS ingress and egress boundary nodes. The latter case usually occurs when one wants to ensure that the egress traffic conforms to some specification before it is sent to a downstream DS domain. Traffic conditioners can, however, also be located within interior nodes or in a non-DS-capable domain.

2.2.2.3. Per-hop Behaviours

When a DS node receives a packet, it examines the codepoint and takes certain actions, depending on the particular behaviour aggregate. The per-hop behaviour of the node is the resulting externally observable forwarding behaviour of the traffic at the node. The PHB may be specified in terms of the resources available to it (buffer, bandwidth); its priority relative to other PHBs; or the observable traffic characteristics (delay, loss) associated with it, expressed in relative terms.

PHBs are used as the building blocks for allocating resources in the forwarding path. They are implemented on nodes by buffer management and packet scheduling mechanisms.

Related to PHBs is the concept of Per-Domain Behaviour (PDB) (Nichols and Carpenter, 2001). While a PHB only describes behaviour at a single hop in die DS domain, a PDB describes how to configure a DS domain. The PDB is therefore the specification of how the DS domain is configured as well as the quantifiable behaviour that is expected from the domain.

The next section illustrates that PHBs are defined in terms of behaviour characteristics relevant to service provisioning policies and not in terms of implementation mechanisms.

2.2.3. Example Per-hop Behaviours

In this section two example PHB definitions are given. The one PHB allows for distinguishing between different classes of traffic while the other provides the ability to create a virtual leased-line.

2.2.3.1. Assured Forwarding

This section describes the Assured Forwarding (AF) PHB group as defined in Heinanen et al. (1999).

The requirement is to be sure that IP packets are forwarded with a high probability as long as the aggregate traffic does not exceed the specified traffic profile. Packets should also not be re-ordered within a micro-flow.

The AF PHB group provides different levels of forwarding assurance. Four AF classes are defined. Each class is allocated resources in the DS nodes. Within each class packets are marked with three drop precedence values. This indicates the relative importance of the packet within class. Packets with higher drop precedence will be discarded first in the case of congestion occurring.

In a DS node the level of forwarding assurance of a packet depends on the amount of forwarding resources allocated to the class, the current load of the class, and the drop precedence of the packet (when congestion occurs in the class).

The goal for the queuing behaviour is to minimise long-term congestion in each class while accommodating short-term congestion that results from traffic bursts. To achieve this goal an active queue management algorithm such as RED is required. RED is explained in Section 2.3.

2.2.3.2. Expedited Forwarding

In this section the Expedited Forwarding (EF) PHB is described as defined by Davie et al. (2002). It was originally described by Jacobson et al. (1999) but some problems (Bennet et al., 2001; Charny et al., 2002) in the specification led to an updated specification. An alternative specification of the original EF PHB based on delay bounds can be found in Armitage et al. (2002).

The EF PHB can be used to build a low loss, low latency, low jitter, assured bandwidth, end-to-end service through DS domains.

Loss, jitter, and latency are caused by the queues that traffic experience. To provide low loss, latency, and jitter, traffic should encounter small queues. To achieve this, the aggregate's maximum arrival rate must be less than its minimum departure rate. To create such a service two things are needed: an aggregate should have a well-defined minimum departure rate and the arrival rate should be less than the configured minimum departure rate.

The EF PHB provides the first requirement. By appropriately shaping, dropping or marking packets, the traffic conditioners attempt to provide for the second requirement.

One way of implementing the EF PHB is to use an output buffered device which delivers packet immediately to the appropriate output queue with a priority queue for EF traffic.

A recent proposal by Nichols et al. (2004) for a 'virtual wire' service across a Diffserv domain defines a per-domain behaviour based on the previous EF research.

2.2.4. A Management Model for Diffserv Routers

In this section an informal conceptual model of a Diffserv router is provided. The model is defined in Bernet et al. (2002).

2.2.4.1. Components of a Diffserv router

Figure 2.4 gives a graphical representation of the major functional blocks of a DS router.

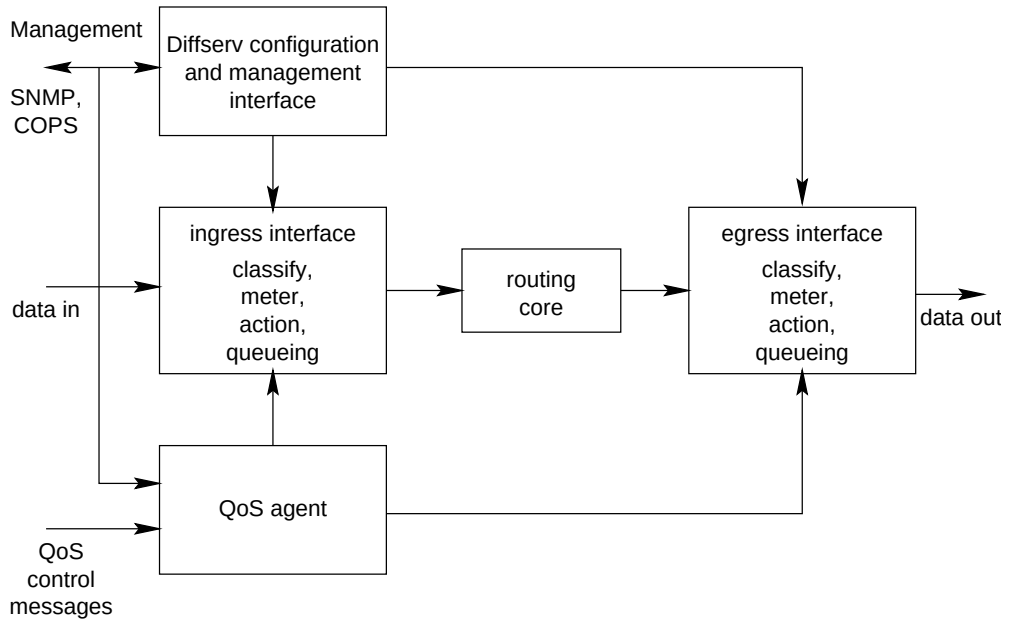


Figure 2.4.: Major functional blocks in a Diffserv router.

The inner block labelled 'routing core' represents the normal routing and switching functionality (outside Diffserv). In addition, however, the diagram shows ingress and egress interface blocks that may contain the following Diffserv functions:

- Traffic Classification elements.
- Metering functions.
- Action elements providing for marking, dropping, counting, and multiplexing.
- Queuing elements, including capabilities of dropping and scheduling.

- Certain combinations of the above functional datapath elements into higher-level blocks known as Traffic Conditioning Blocks (TCBs).

These building blocks of the router's functionality need to be managed by Diffserv configuration and management tools. An interface to support such management is depicted in the upper left block of the diagram.

Configuration and Management Interface Diffserv operating parameters are monitored and provisioned through this interface. Monitoring includes gathering statistics regarding traffic forwarded at different Diffserv service levels. These can be used for accounting purposes as well as for tracking compliance to Traffic Conditioning Agreements (TCAs).

Configuration parameters include parameters for classifiers and meters; for PHB; and parameters for action and queuing elements. The interface to these are provided by management protocols such as Simple Network Management Protocol (SNMP) or Common Open Policy Service (COPS).

The following section provides more detail about the remaining interface blocks in the diagram, namely the ingress and egress interfaces which provide certain Diffserv functions.

2.2.4.2. Diffserv Functions at Ingress and Egress

Figure 2.5 shows a high-level view of the ingress and egress interfaces on a Diffserv router.

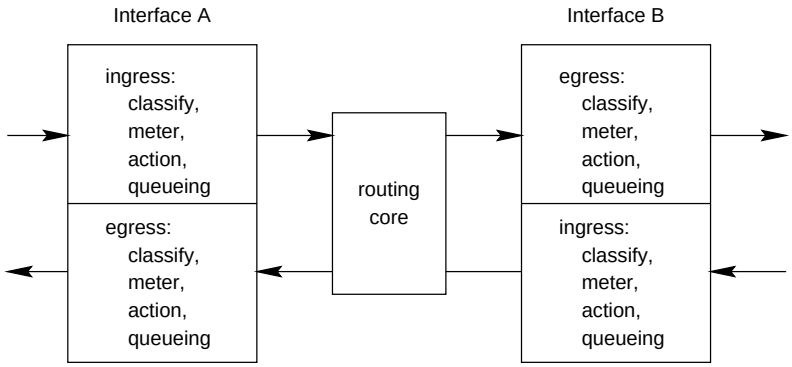


Figure 2.5.: Diffserv router ingress and egress interfaces.

The diagram illustrates two Diffserv interfaces. It also shows the functions that might be instantiated on each interface.

Each router should perform the following four QoS control functions on traffic in the datapath.

- Classify each packet according to some set of rules.
- Determine if the traffic stream to which the packet belongs is in- or out-of-profile by metering the stream.
- Perform a set of resulting actions. These might include applying a drop policy or marking of traffic with a DS codepoint.
- Enqueue the packet in the appropriate queue. The scheduling of the queue may lead to shaping of the packet stream or cause the packets to be forwarded at some minimum rate.

Not all functions will be implemented on all interfaces of all routers. Ingress and egress routers may differ in the same way that core and edge routers may differ.

2.3. Random Early Detection

One approach to congestion management, while avoiding the complications of global synchronisation is to discard packets pro-actively. The mainstream example of such a pro-active dropping scheme is RED (Floyd and Jacobson, 1993). RED is sometimes also referred to as *Random Early Discard* (Comer, 2000) or *Random Early Drop* (Clark and Fang, 1998).

Since RED is a form of active queue management, it functions on a packet queue by discarding packets randomly according to the scheme described later in this section. Although the location of the queue is irrelevant in the discussion of the behaviour of RED one would typically find RED queues at the egress interface of a DS node (possibly one for each class in the AF PHB).

For every packet that arrives, the RED gateway calculates the weighted moving average queue size (avg) relying, as discussed later, on a so-called low pass filter. It then compares the result with a minimum (min_{th}) and a maximum (max_{th}) threshold according to the following scheme.

- when $avg < min_{th}$ the packet is not dropped.
- when $min_{th} \leq avg < max_{th}$ the packet is dropped with probability p_a, where p_a is a function of p_b and $count$ (details are given below).
- when $max_{th} \leq avg$ the packet is dropped with probability 1.

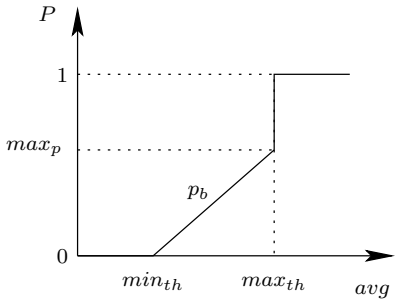


Figure 2.6.: The RED algorithm.

Figure 2.6 contains a graphical presentation of the above scheme. The x-axis indicates the average queue size and the y-axis, the probability that the packet is dropped.

The RED gateway has two algorithms: one for computing the average queue size and one for computing the packet drop probability, p_a.

The algorithm for computing the average queue size determines the degree of burstiness that will be allowed in the gateway queue. The drop-probability algorithm determines how many packets are dropped given the current congestion level.

The RED gateway uses a low-pass filter to calculate the average queue size. Such a filter is merely an exponential weighted moving average: $avg = (1 - w_q)avg + w_q q$, where w_q is a configured queue weight and q is the current queue size. The weight, w_q, determines the effect of the current queue size q on the new value for the average queue size.

As avg varies between min_{th} and max_{th}, the preliminary packet dropping probability p_b varies linearly from 0 to max_p: i.e. $p_b = max_p(avg - min_{th})/(max_{th} - min_{th})$. However, the final packet dropping probability, p_a, depends on p_b in relation to a variable, *count*, that is initialised to zero. The following scheme ensures that $p_a = p_b$ initially, that p_a increases as *count* increases, but that it returns to p_b after reaching a critical level.

While avg lies between min_{th} and max_{th}, then *count* is incremented by 1 each time a packet escapes discard but is reset to zero whenever a packet is discarded. The final packet drop probability is computed as: $p_a = p_b/(1 - count \cdot p_b)$ when $count < 1/p_b - 1$ and $p_a = 1$ otherwise. Thus, when $count \geq 1/p_b - 1$ the next incoming packet is discarded which, in turn, means that *count* is reset to zero.

Refer to Floyd and Jacobson (1993) for a discussion on configuring the parameters for a RED gateway. The comments of Floyd and Kohler (2003) are also relevant. This low-pass filter minimises the effect on the average queue size of short term fluctuations in the actual queue size (caused by bursty traffic or temporary congestion).

The RED gateway can also measure the queue length in bytes rather than in packets. When this option is used, the algorithm is modified to ensure that the probability that

a packet is marked is proportional to the size, in bytes, of the packet. These two modes are compared in Eddy and Allman (2003) and recommendations are made for when to use which mode.

In the so-called 'gentle_' modification to RED, the packet dropping probability p_b varies from max_p to 1, as avg varies from max_{th} to $2 \cdot max_{th}$. This modification makes RED more robust to the values of the parameters max_{th} and max_p.

The 'gentle_' variant of RED has been implemented and evaluated in Rosolen et al. (1999) for an ATM switch and has been recommended by Floyd (2000) for implementations and simulations.

Another variation on RED called RED with In/Out bit (RIO) is described in Clark and Fang (1998). Packets are marked as in- or out-of-profile. A gateway running RIO uses the same mechanisms as RED but with two sets of parameters, one for In packets and one for Out packets. When a packet arrives at the gateway, it is inspected to determine whether it is marked as In or Out. If it is In the gateway calculates avg_in, the average queue size for In packets. The gateway also calculates avg_total, the average queue size for all packets. The probability for dropping an In packet depends on avg_in and the probability for dropping an Out packet depends on avg_total.

2.4. Traffic Conditioners

This section introduces the set of traffic conditioners whose performance has been investigated in this dissertation. The material provided below summarises the literature sources on these topics, whereas Chapter 3 proposes process algebra abstractions of the respective traffic conditioners. The latter constitutes an elegant, succinct and unambiguous alternative to the less formal natural language descriptions given here and elsewhere in the literature. The set of these formally defined traffic conditioners is offered as a spin-off research contribution of this dissertation. The formal definitions were undertaken in the early part of this study in order to ensure that the traffic conditioner behaviour was understood, before proceeding to simulation of the traffic conditioners later in the study.

As seen in Section 2.2.2.2, the traffic conditioner is an important component in the Diffserv architecture. The traffic conditioners below implement the metering and marking functions of a Diffserv traffic conditioner. The parameters such as Peak Information Rate (PIR) are specified as part of the traffic conditioning specification that forms part of the agreement between the subscriber and the provider of the DS domain.

Token Bucket Before the conditioners are described, it is necessary to introduce the Token Bucket (Stallings, 1998, pg. 261). A token bucket is a device that can be used to

specify the characteristics of a traffic stream. One can also use it to determine whether a stream conforms to a traffic specification. The token bucket measures the traffic flow and then determines whether the flow is in- or out-of-profile.

A token bucket is specified by two parameters: a rate, ρ, at which tokens are placed in the bucket and the capacity of the bucket, β. When the bucket is filled to capacity, newly arriving tokens are discarded. $T(t)$ represents the number of tokens in the bucket at time t.

To determine whether a packet of size B arriving at time t is in- or out-of-profile the following logic is applied. If $T(t) - B \geq 0$ then the packet is in-profile; otherwise it is out-of-profile. When a packet is determined to be in-profile, the number of tokens in the bucket is reduced by the size of the packet, i.e. $T \leftarrow T - B$.

2.4.1. Token Bucket Marker

The Token Bucket Marker (TBM) meters a traffic stream based on two traffic conditioning parameters: Committed Information Rate (CIR) and Committed Burst Size (CBS). The marker marks the packets as either green, or red.

The meter is specified in terms of a token bucket, C. The maximum size of C is CBS. Initially the token bucket is full, i.e. $T_C(0) = CBS$, where T_C is the token count. T_C is updated CIR times per second:

```
if (T_C < CBS) then T_C ← T_C + 1 else
T_C not incremented
```

The marker uses the result of the meter to determine the colour of a newly arrived packet. Assume a packet of size B bytes arrives at time t. If $T_C(t) - B \geq 0$, the packet is marked as green and $T_C \leftarrow T_C - B$. If not, the packet is marked as red and T_C remains unchanged.

2.4.2. Time-sliding window three colour marker

The Time Sliding Window Three Colour Marker (TSW3CM) is described in Fang and Seddigh (2000). It is intended for use in the AF PHB. The TSW3CM meters a traffic stream according to two traffic conditioning parameters: Committed Target Rate (CTR) and Peak Target Rate (PTR), with $CTR < PTR$. The marker marks a packet as green, yellow, or red.

The TSW3CM is composed of two parts: a rate estimator and a packet marker. The specification (Fang and Seddigh, 2000) does not require a specific algorithm for the

rate estimator. The rate estimator should provide an estimate of the stream's arrival rate. This rate should approximate the running average bandwidth of the incoming traffic stream over a specific period of time. One example of such an algorithm is the Time Sliding Window (TSW) rate estimator (Clark and Fang, 1998). The algorithm is described in Figure 2.7.

Initially:

```
Win_length    = a constant
avg-rate      = CTR
t-front       = 0
```

Upon each packet arrival the rate estimator updates its variables:

```
Bytes_in_win = avg-rate * Win_length
New_bytes    = Bytes_in_win + pkt_size
avg-rate     = New_bytes/(now - t-front + Win_length)
t-front      = now
```

Where:

```
now           = The time of the current packet arrival
pkt_size      = The packet size in bytes of the arriving packet
avg-rate      = Measured arrival rate of the traffic stream
Win_length    = The window over which history is kept
```

Figure 2.7.: The TSW rate estimator algorithm.

Let avg be the estimated average rate of the traffic stream, as determined by the meter. Also let $P_0 = (avg - CTR)/avg$, $P_1 = (avg - PTR)/avg$, and $P_2 = (PTR - CTR)/avg = P_0 - P_1$. The marker then determines the colour of a packet based on the following scheme.

- If $avg \leq CTR$ the packet is marked green.
- If $CTR < avg \leq PTR$ the packet is marked yellow with probability P_0 and green with probability $1 - P_0$.
- If $PTR < agv$ then
  - with probability P_1 the packet is marked red,
  - with probability P_2 the packet is marked yellow, and
  - with probability $1 - (P_1 + P_2)$ the packet is marked green.

Thus, if CTR is chosen as relatively large compared to avg, then a large number of packets will go through as green. If PTR is chosen as relatively small compared to avg, then there will be many red packets. These matters should therefore be kept in mind when provider and subscriber negotiate a traffic conditioning specification.

2.4.3. Time-sliding window two colour marker

The Time Sliding Window Two Colour Marker (TSW2CM) is similar to the TSW3CM but instead of two rates, it is only configured with one rate and packets are marked as either green, or red.

It meters a traffic stream according to one traffic conditioning parameter: CTR.

The TSW2CM is also composed of a rate estimator and a packet marker. As discussed in the previous section, the rate estimator provides an estimate of the traffic stream's arrival rate. The algorithm of the TSW rate estimator was depicted in Figure 2.7.

The marker determines the colour of a packet based on the scheme given below.

Let avg again be the estimated average sending rate of the traffic stream, as determined by the meter, and let $P_0 = (avg - CTR)/avg$.

- If $avg \leq CTR$ the packet is marked green.
- If $CTR < avg$ the packet is marked red with probability P_0 and green with probability $1 - P_0$.

From the above it can be seen that the TSW2CM is a simplified version of the TSW3CM which only assigns two colours based on two ranges of values for avg.

2.4.4. Single rate three colour marker

This section describes the Single Rate Three Color Marker (srTCM) as presented in Heinanen and Guerin (1999a). It meters a traffic stream according to three traffic conditioning parameters: CIR, CBS, and Excess Burst Size (EBS). The values for the parameters are determined by the traffic conditioning specification.

The marker is useful for ingress policing of service where the length of a burst, and not its peak rate, determines the service level that is applicable.

The specification allows this traffic conditioner to take a previously assigned colour into account when assigning a colour to a newly arrived a packet; this is called colour-aware mode. When it is not running in colour-aware mode, then it is running in so-called colour-blind mode. Colour-aware mode could be used at an egress node of a DS domain

or perhaps at an ingress node of a domain expecting to receive pre-marked packets from an upstream DS domain.

The meter is specified in terms of two token buckets: C and E. The capacity of C is CBS and that of E is EBS. Initially, both token buckets are full, i.e. $T_C(0) = CBS$ and $T_E(0) = EBS$. Token counts T_C and T_E are updated CIR times per second:

```
if (T_C < CBS) then T_C ← T_C + 1
else if (T_E < EBS) then T_E ← T_E + 1
else T_C, T_E not incremented
```

A packet of size B arriving at time t is then coloured according to the subsequent rules. When the srTCM is operating in colour-blind mode the following apply:

- If $T_C(t) - B \geq 0$, the packet is marked as green and $T_C \leftarrow T_C - B$.
- If $T_C(t) - B < 0$ and $T_E(t) - B \geq 0$, the packet is marked as yellow and $T_E \leftarrow T_E - B$.
- If neither of the previous cases are valid the packet is marked as red and T_C and T_E remain unchanged.

For colour-aware mode the following are used:

- If the packet was premarked as green and $T_C(t) - B \geq 0$, the packet is marked as green and $T_C \leftarrow T_C - B$.
- If $T_C(t) - B < 0$, and the packet was premarked as green or yellow, and $T_E(t) - B \geq 0$; the packet is marked as yellow and $T_E \leftarrow T_E - B$.
- If neither of the previous cases are valid the packet is marked as red and T_C and T_E remain unchanged.

In the subsequent chapters, the srTCM is assumed to operate in colour-blind mode since the study was limited to a single DS domain.

2.4.5. Two-rate three colour marker

The Two Rate Three Colour Marker (trTCM) (Heinanen and Guerin, 1999b) meters a traffic stream according to four traffic conditioning parameters: PIR, CIR, CBS, and Peak Burst Size (PBS). The marker assigns one of three colours to a packet: green, yellow, or red.

The trTCM is similar to the srTCM; it only requires an additional rate (PIR) as configuration parameter and assigns one of three colours based on the states of two token buckets.

The meter is again specified in terms of two token buckets: P and C. The capacity of P is PBS and the capacity of C is CBS. At time 0, the token buckets are both full, i.e. $T_P(0) = PBS$ and $T_C(0) = CBS$. Token counts T_P and T_C are updated PIR and CIR times per second up to the respective maximum values:

```
if (T_P < PBS) then T_P ← T_P + 1
if (T_C < CBS) then T_C ← T_C + 1
```

Note that there is a slight difference here when comparing how token counts are updated in the srTCM. In the srTCM tokens are generated at single rate (CIR) increasing the number of tokens in the first bucket (C). When C 'overflows' the tokens are added to the second bucket E.

In the trTCM tokens are generated separately for each token bucket. C's token count is updated CIR times per second and P's PIR times per second.

For colour-blind mode the following scheme is used to assign a colour to a newly arrived packet of size B.

- If $T_P(t) - B < 0$, the packet is marked as red.
- If $T_P(t) - B \geq 0$ and $T_C(t) - B < 0$, the packet is marked as yellow and $T_P \leftarrow T_P - B$.
- If neither of the previous cases are valid the packet is marked as green and $T_P \leftarrow T_P - B$ and $T_C \leftarrow T_C - B$.

The following rules determine the colour of the packet when the trTCM is operating in colour-aware mode.

- If the packet was premarked as red or $T_P(t) - B < 0$, the packet is marked as red.
- If the packet was not premarked as red, and $T_P(t) - B \geq 0$, and the packet was premarked as yellow or $T_C(t) - B < 0$; the packet is marked as yellow and $T_P \leftarrow T_P - B$.
- If neither of the previous cases are valid, the packet is marked as green and $T_P \leftarrow T_P - B$ and $T_C \leftarrow T_C - B$.

As with the srTCM the trTCM is assumed to operate in colour-blind mode in the subsequent chapters.

2.5. Traffic generation models

This section describes the different models used to generate network traffic in the subsequent simulation study.

Simulating the Internet and network traffic is difficult since the Internet is so heterogeneous and constantly changing (Floyd and Paxson, 2001). Various methods exist to generate network traffic. Amongst these are a structural approach (Willinger et al., 1998; Taqqu et al., 1997) employing strictly alternating 'ON' and 'OFF' periods; methods employing wavelets (Jeong et al., 1999a) which lend themselves to modelling complex patterns; and, more recently, a method (Salvador et al., 2004) utilising so-called discrete-time batch Markovian arrival processes (BMAPs).

The following sections contain descriptions of the traffic generation models used in the simulation study presented in this dissertation. The models were not chosen to represent specific applications or traffic scenarios. Rather, they were chosen to represent a range of different traffic patterns. This was done in order to compare the different traffic conditioners under various traffic pattern conditions, possibly matching real-world situations. The models presented here range from a fairly simple constant bit rate traffic generator to a fairly complex model used to create so-called self-similar, or long-range dependent traffic.

2.5.1. Constant bit rate

The Constant Bit Rate (CBR) traffic generator generates traffic at a constant rate. The generated packets are all the same size.

Randomised variation can be introduced in the inter-packet departure times. This will prevent different simulated sources that started sending at the same time from generating packets at exactly the same time for the duration of the simulation. The variation will ensure that packets are generated at slightly different moments in time.

This model simulates a traffic source that generates traffic at a constant rate, such as some video or audio applications.

2.5.2. Poisson Arrival Process

The Poisson process is frequently used in simulation and analytical studies as a model of packet arrivals. This section provides a basic description of the process.

Let the inter-arrival times of packets be modelled by a sequence of independent identically distributed exponential random variables, $\eta_1, \eta_2, \ldots$, with rate λ. Let $\xi_n = \eta_1 + \eta_2 + \cdots + \eta_n$, which represents the time that the n^{th} packet arrives ($\xi_0 = 0$). Let $N(t)$

be the number of packets that arrived up to time $t \geq 0$. $N(t)$ is then a Poisson process and has the Poisson distribution with parameter λt (Brzeźniak and Zastawniak, 1999). Thus, for an arbitrary value of t, the probability that $N(t) = n$ and the expected value of $N(t)$ are given by the following:

$$P(N(t) = n) = e^{-\lambda t}\frac{(\lambda t)^n}{n!} \tag{2.1}$$

$$E(N(t)) = \lambda t \tag{2.2}$$

Figure 2.8 provides a graphical representation of a sample path of $N(t)$.

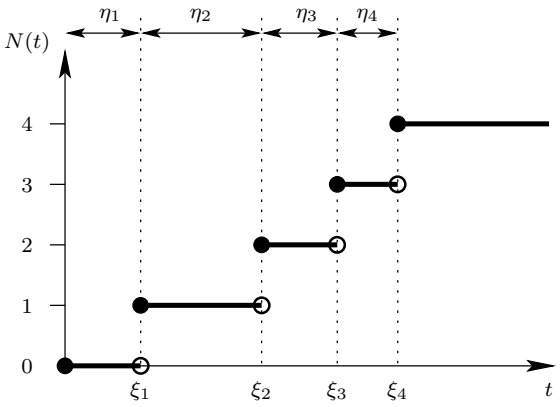


Figure 2.8.: Poisson process illustration.

Queuing analysis has been valuable to network designers when doing capacity planning and performance prediction. One of the basic assumptions in queuing analysis is that the arriving packets is Poisson of nature. This assumption has been shown (Paxson and Floyd, 1995) to not always hold in practice. Network traffic is often more complex and exhibits a so-called self-similar property (discussed later).

2.5.3. Exponential On/Off

This section describes the Exponential On/Off (EXPOO) traffic generator found in *ns*—the network simulator used in this study.

The model of operation is based on an ON/OFF process that utilises two Exponential distributions to produce traffic. Traffic is generated in two periods—ON and OFF. In the ON period traffic is generated and during the OFF period no traffic is generated. One distribution dictates the length of the OFF period, the other the length of the ON period. No packets are sent during the OFF period and packets are sent at a constant

rate during the ON periods. Constant rate in the current context means that if, say, the rate is ten packets per second, a packet will be sent every tenth of a second.

If X is a random variable from a Exponential distribution with rate λ, then the probability distribution function $f(x)$ and the expected value $E(X)$ are

$$f(x) = P(X = x) = \lambda e^{-\lambda x} \quad \text{for } x \geq 0 \tag{2.3}$$

$$E(X) = \frac{1}{\lambda} \quad \text{if } \lambda > 0 \tag{2.4}$$

where λ is called the *rate* parameter.

The traffic generator is configured with the following parameters:

- burst_time: The mean duration of ON periods
- idle_time: The mean duration of OFF periods
- rate: The constant sending rate during ON periods
- packetSize: The fixed size of data units sent

It is possible to describe the previous two models in terms of a EXPOO model. An EXPOO source with an infinite ON period and zero length OFF period would behave the same as a CBR source. If one creates an EXPOO source with a very short ON period sufficient to always only transmit one packet and some average OFF period, the source will exhibit the behaviour of a Poisson process.

2.5.4. Pareto On/Off

This section describes the Pareto On/Off (PAROO) traffic generator which is similar to the Exponential On/Off generator discussed in previous section.

The model of operation is also based on an ON/OFF process that utilises, in this case, two Pareto distributions to produce traffic. Packets are again generated during ON periods while no packets are generated during OFF periods. One distribution dictates the length of the OFF period and the other the length of the ON periods.

If X is a random variable from a Pareto distribution, then the probability distribution function $f(x)$ and the expected value $E(X)$ are

$$f(x) = P(X = x) = \frac{\alpha \beta^{\alpha}}{x^{\alpha+1}} \quad \text{for } x \geq \beta \tag{2.5}$$

$$E(X) = \beta \frac{\alpha}{\alpha - 1} \quad \text{if } \alpha > 1 \tag{2.6}$$

where α is called the Pareto *shape* parameter and β the *scale* parameter.

The traffic generator is configured with the same parameters as the Exponential On/Off model. In addition, the shape parameter for the Pareto distributions are also required:

- burst_time: The mean duration of ON periods
- idle_time: The mean duration of OFF periods
- rate: The sending rate during ON periods
- packetSize: The fixed size of data units sent
- shape: The Pareto shape parameter

The Pareto scale parameter is not given as input since the mean and the shape are both given. The scale parameter β can therefore be obtained from (2.6). Also note that the shape parameter for both Pareto distributions are the same; they only differ in their means.

The PAROO model is very similar to the EXPOO model. They only differ in the probability distribution used to obtain ON and OFF periods.

2.5.5. Self-Similar Traffic

According to (Schroeder, 1990, pg. 81) an object such as a geometric figure is invariant with scaling, or *self-similar*, if it is reproduced by magnifying some portion of it. Self-similarity in network traffic was first found in Local Area Network (LAN) traffic and documented in Leland et al. (1994). Subsequent studies (Crovella and Bestavros, 1997; Park et al., 1996; Paxson and Floyd, 1995; Garrett and Willinger, 1994; Willinger et al., 1997; Kalden and Ibrahim, 2004) have collected and analysed traffic traces from physical networks. These studies suggest that self-similarity is found not only in certain natural phenomena, but also in various network traffic patterns.

For these reasons it was decided that self-similar traffic sources would be simulated in the study. A brief overview of the theory relating to self-similarity is given below. It should be noted that a detailed knowledge of the theory is not essential. A "recipe" of how to synthesise such processes is presented later in this section.

Self-similar processes The descriptions here follows that of Willinger et al. (2002). More detail on self-similar and long-range dependent processes may be found in Leland et al. (1994); Abry et al. (2000); Crovella and Bestavros (1997); Erramilli et al. (2002); Grossglauser and Bolot (1999).

Let $X = (X_k : k = 1, 2, \ldots)$ be a time series representing the number of arriving packets in successive, non-overlapping time intervals of unit length. For modelling purposes assume X is a second-order stationary process and has zero mean. One can then define an aggregate process $X^{(m)}$ of X for integers $m > 1$:

$$X^{(m)}(i) = \frac{1}{m}(X_{(i-1)m+1} + \cdots + X_{im}), \quad i \geq 1.$$

$X^{(m)}$ is therefore obtained from X by splitting the intervals into non-overlapping blocks of size m intervals, and taking each block average as the new observations $X^{(m)}(i)$, where i is the block index. The family of aggregated processes $(X^{(m)} : m \geq 1)$ is well-suited for studying the properties of network traffic at different time scales. For example, one family member, say when $m = m_1$, might represent traffic during a relatively brief time period, whereas another family member, say $m = m_2$ represents traffic over a longer stretch of time.

The family of aggregated processes is also useful for studying concepts such as self-similarity that relate the statistical properties of X to those of $X^{(m)}$.

Let $r^{(m)} = (r^{(m)}(k), k \geq 0)$ denote the autocorrelation function of the aggregated process $X^{(m)}$. X is *asymptotically* second-order self-similar with self-similarity parameter (or Hurst parameter) $0 < H < 1$ if the second-order statistics of $m^{1-H}X^{(m)}$ converge as follows:

$$\lim_{m\to\infty} \text{var}(m^{1-H}X^{(m)}) = \sigma^2, \quad 0 < \sigma < \infty, \tag{2.7}$$

and

$$\lim_{m\to\infty} r^{(m)}(k) = \frac{1}{2}((k+1)^{2H} - 2k^{2H} + (k-1)^{2H}). \tag{2.8}$$

X is called *exactly* second-order self-similar with Hurst parameter $0 < H < 1$ if, for all $m > 1$, the $m^{1-H}X^{(m)}$ processes have the same second-order statistics as X, i.e. $\text{var}(m^{1-H}X^{(m)}) = \text{var}(X) = \sigma^2$ and $r^{(m)}(k) = r(k) = \frac{1}{2}((k+1)^{2H} - 2k^{2H} + (k-1)^{2H})$, where $(r(k), k \geq 0)$ is the autocorrelation function of X.

X is said to exhibit long-range dependence (LRD) if, for some $0 < \beta < 1$,

$$r(k) \sim c_1 k^{-\beta}, \quad \text{as } k \to \infty, \tag{2.9}$$

where c_1 is a positive constant and $\sim$ denotes that the expressions on the two sides are asymptotically proportional to each other. Also note that the autocorrelation function is non-summable, i.e. $\sum_k |r(k)| \to \infty$. Long-range dependence captures the phenomenon of pronounced clusters of consecutive large or consecutive small values that occur in many natural time series.

The form of the autocorrelation function in (2.8) points to the presence of long-range dependence. If one restricts the Hurst parameter H to values between 0.5 and 1, asymptotic or exact second-order self-similarity implies LRD with $\beta = 2 - 2H$.

One of the most widely studied self-similar processes is Fractional Gaussian Noise (FGN). Associated with FGN is Fractional Gaussian Motion (FGM), which is the sum of FGN increments.

Synthesis The rest of the section presents one method to synthesise self-similar network traffic that corresponds to an FGN self-similar process. The method is described in Paxson (1997) and compared to other methods in Jeong et al. (1999b). The study described in Jeong et al. (1999b) found that the method is appropriate to generate self-similar traffic for use in network simulations. Based on these findings, it was decided to rely on the method described below to generate various FGN self-similar processes to represent some of the traffic sources that were used in this study.

The following steps are used to compute the approximate FGN sample path. The inputs to the method are H, the desired Hurst parameter and n, the even number of observations in the sample path.

1. Construct a sequence of values $\{f_1, \ldots, f_{n/2}\}$, where $f_j = \widetilde{f}(\frac{2\pi j}{n}; H)$, as defined below in (2.10).
2. Create a new "fuzzed" sequence $\{\hat{f}_1, \ldots, \hat{f}_{n/2}\}$ by multiplying each f_i from the previous step by an independent Exponential random variable with mean 1.
3. Construct $\{z_1, \ldots, z_{n/2}\}$, a sequence of complex values such that $|z_i| = \sqrt{\hat{f}_i}$ and the phase of z_i is some variable that is uniformly distributed between 0 and 2π.
4. Construct $\{z_0^{'}, \ldots, z_{n-1}^{'}\}$, an "expanded" version of $\{z_1, \ldots, z_{n/2}\}$, such that:

$$z_j^{'} = \begin{cases} 0 & \text{if } j = 0 \\ z_j & \text{if } 0 < j \leq n/2 \\ \overline{z_{n-j}} & \text{if } n/2 < j < n \end{cases}$$

   where $\overline{z_{n-j}}$ denotes the complex conjugate of z_{n-j}.
5. Inverse-Fourier transform $\{z_0^{'}, \ldots, z_{n-1}^{'}\}$ to obtain the approximate FGN sample path $\{x_0, \ldots, x_{n-1}\}$. This sample path is an approximation (with n observations) of an FGN process with Hurst parameter H and zero mean.

The function referred to in step 1 above is given by:

$$\widetilde{f}(\lambda; H) = \mathcal{A}(\lambda; H)[|\lambda|^{-2H-1} + \mathcal{B}(\lambda; H)] \tag{2.10}$$

for $0 < H < 1$ and $-\pi \leq \lambda \leq \pi$, with

$$\begin{aligned}
\mathcal{A}(\lambda; H) &= 2\sin(\pi H)\Gamma(2H+1)(1-\cos\lambda) \\
\mathcal{B}(\lambda; H) &= a_1^d + b_1^d + a_2^d + b_2^d + a_3^d + b_3^d + \frac{a_3^{d'} + b_3^{d'} + a_4^{d'} + b_4^{d'}}{8H\pi} \\
d &= -2H - 1 \\
d' &= -2H \\
a_k &= 2k\pi + \lambda \\
b_k &= 2k\pi - \lambda
\end{aligned}$$

and Γ is the Gamma function:

$$\Gamma(x) = \int_0^\infty e^{-t}t^{x-1}dt$$

An implementation of the above procedure takes as arguments: n, the number of points to generate as well as the desired Hurst parameter H. It also accepts as parameters a mean and a variance which are used to transform the generated sequence. The implementation generates an approximate long-range dependent sample path using the supplied n and H. It then translates and scales the elements of the sample path to obtain the desired mean and variance, respectively.

Figure 2.9 on the following page contains plots of a sample path generated using the method described in this section. The sample path was generated with a mean of 44, a variance of 144, and a Hurst parameter of 0.95. The first graph contains the whole sample path with 4096 points. The second graph contains a plot of an aggregated version of the original sample path. The second plot was obtained by taking the average value of non-overlapping blocks of ten points and plotting these as new points. The third plot is simply the first 410 points in the original sample path. It is interesting to note how similar the second and third plots appear visually.

A sample path generated by using the preceding "recipe" can be used to represent a packet arrival process by letting each element in the sample path represent the number of packet arrivals per unit time-interval. Naturally, one would need to translate and scale the elements in the sequence to obtain a mean and a variance that are appropriate to the traffic pattern being simulated. For instance, if one wanted to generate a traffic stream with an average rate of approximately 2 Mbps using fixed-sized packets of 576 bytes, one could then use a sample path to represent the number of packet arrivals in a tenth of a second. The average number of arrivals per 0.1 s is then 43.3. The variance and the Hurst parameter should be chosen to fit the type of application being simulated. As will be seen in Chapter 4, this study used three different Hurst parameter values to simulate self-similar traffic.

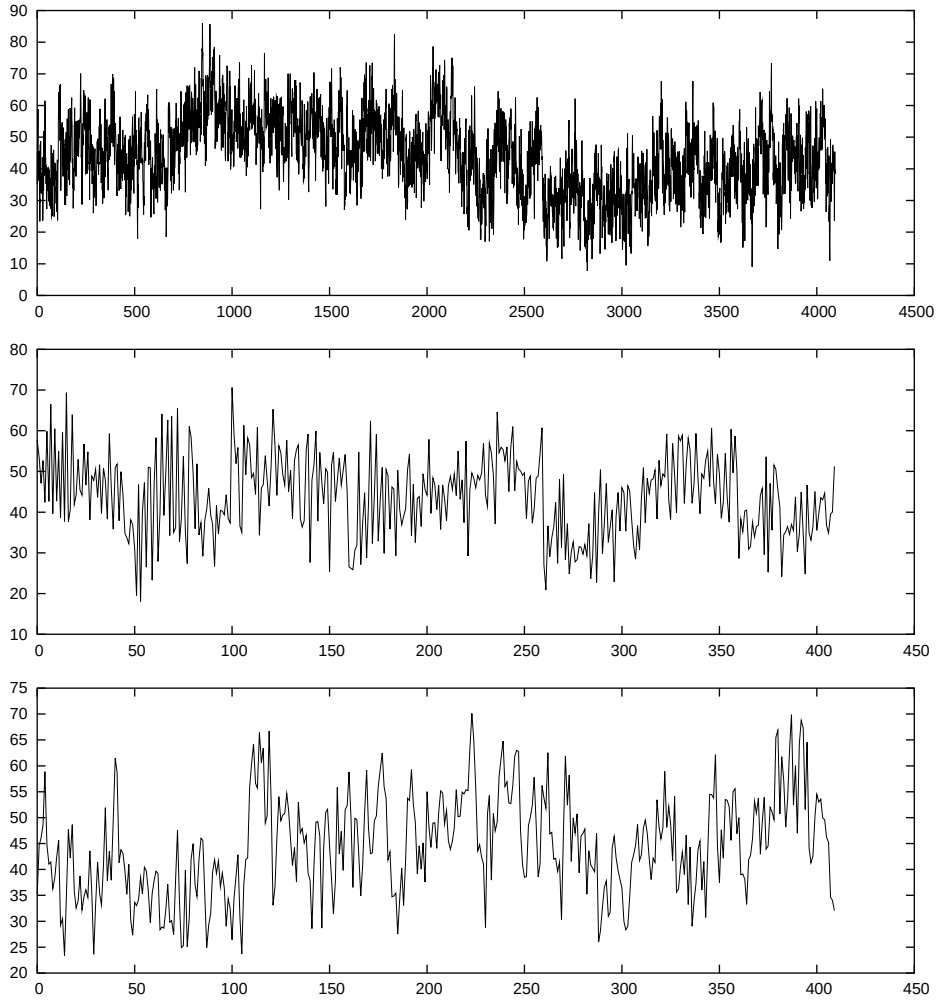


Figure 2.9.: An example LRD sample path.

3. Process Models for Conditioners

A good model can advance fashion by ten years.

Yves Saint-Laurent

In this chapter each of the Diffserv traffic conditioners discussed in Section 2.4 is modelled in Finite State Processes (FSP) (Magee and Kramer, 1999). FSP is a process algebra similar to Communicating Sequential Processes (CSP) (Schneider, 2000). In fact, it is a subset of CSP, having a syntax that relies on standard keyboard symbols only.

Each traffic conditioner will thus be modelled as set of communicating sequential processes. The motivation for doing this is to provide an abstract representation that aids in the understanding of the different conditioners. Each model aids understanding by focusing attention on the essential actions and interactions in the respective conditioner. Furthermore, the models, viewed collectively, provide a basis for cross-comparing the features and characteristics of the various conditioners at an abstract level, thereby highlighting the essential features that they share, as well as the features that distinguish one from the other. While these differences are obviously not unknown, the models represent an elegant and convenient pedagogical device that has not been encountered elsewhere in the literature.

However, unlike the use of process algebra models in other contexts, these models were not intended to be used for purposes such as model checking. Trace analysis, progress analysis, and deadlock detection will therefore not performed. There is also no intention to suggest that the architecture, or structure, of a model should necessarily imply that an implementation ought to have the same architecture. On the other hand, one could use these models as a starting point when creating a parallel processing implementation of a traffic conditioner. An implementation of a traffic conditioner should, however, have the same externally observable behaviour as its model, i.e. a model's trace set is a comprehensive description of the associated conditioner's behaviour (or semantics).

In Section 3.1 a brief overview of the FSP syntax is provided as an aid to understanding the subsequently provided process definitions. Certain primitive process definitions are given in Section 3.2. These process definitions are characterized as primitive, because they are later used as subprocesses in the definition of the models for the respective traffic conditioners. The TBM is modelled in Section 3.3, the srTCM in Section 3.6, and the trTCM in Section 3.7. The two markers employing a time-sliding window algorithm to estimate the packet arrival rate, the TSW3CM and the TSW2CM, are described in

Section 3.4 and in Section 3.5, respectively. Section 3.8 concludes the chapter by highlighting the common elements and differences of the various conditioners that were manifested in the process definitions.

3.1. Syntax Overview

This section provides an overview of the FSP syntax used in the modelling of the traffic conditioners. The section does not provide a complete language definition; it merely gives enough background to understand the descriptions in the subsequent sections.

An FSP process is defined by one or more local processes separated by commas. A full stop is used to indicate the end of a process definition. Process identifiers begin with an uppercase letter and action identifiers with a lowercase letter. Local processes and actions can be indexed to model multiple values. Indices can only assume a finite range of values.

Table 3.1 contains descriptions of the FSP language operators used later in the chapter. The table refers to actions. In FSP such actions (events in CSP) are regarded as atomic and instantaneous. An FSP specification thus abstracts away from the precise timing details, focussing instead on the possible ordering (i.e. traces) of actions.

A composite process is the parallel composition of one or more processes. Composite processes are distinguished from primitive processes by prefixing their definitions with '||'. Primitive processes are usually described using only action prefix, choice, and conditional operators. Composite processes are typically constructed using the parallel composition, relabelling, and interface operators on predefined named processes. These components of a composite process may be regarded as interacting or synchronised processes.

Interaction, or synchronisation, between processes is modelled in terms of shared actions in the processes, i.e. action labels that are the same in the interacting processes. It is often useful to apply relabelling and/or interface operators to interacting processes to ensure that certain action labels are transformed to common values, thus becoming shared actions that are relied upon for synchronisation.

In addition to specifying a process's (dynamic) behaviour in terms of FSP, a graphical notation, called a structure diagram, will be used below to communicate the static structure of a composite process. This notation is also borrowed from Magee and Kramer (1999). A structure diagram shows sub-processes that are used to produce a composite process as well as the interactions between these sub-processes.

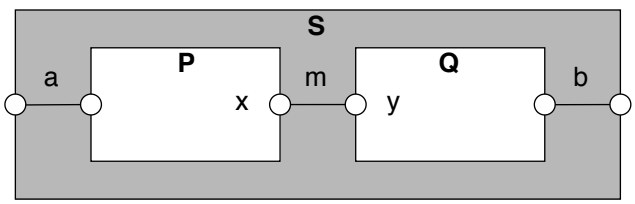
A process is represented as a box and externally visible actions (i.e. actions not hidden by the interface operator) are shown as circles on the border of the box. Shared actions are shown as lines connecting action circles. Relabelling might be necessary to define

Action Prefix	`->`	If `x` is an action and `P` a process, `(x->P)` describes a process that initially engages in `x` and then behaves as described in `P`.
Choice	`|`	If `x` and `y` are actions, `(x->P|y->Q)` describes a process that initially engages in either of `x` or `y`. If the first action that occurs is `x`, then the subsequent behaviour of the process is determined by `P`; otherwise the subsequent behaviour is determined by `Q`.
Conditional	`if then else`	The process `if B then P else Q` behaves as `P` if the Boolean condition `B` is true and as `Q` otherwise.
Parallel Composition	`||`	If `P` and `Q` are processes, `(P||Q)` represents the concurrent execution of `P` and `Q`. If an action is common to both processes, then it can only occur in one process if it simultaneously occurs in the other process, i.e. the processes synchronize on common actions and can only engage in such actions simultaneously. Where processes cannot synchronize on common actions, deadlock occurs.
Process Labelling	`:`	`a:P` prefixes every action label in the alphabet of `P` with `a`.
Relabelling	`/`	Relabelling changes the names of action labels. The form is: $/\{new_1/old_1, \ldots, new_n/old_n\}$
Interface	`@`	When applied to a process `P`, the interface operator `@{a1,...,ax}` hides all actions in the alphabet of `P` not listed in the set following the `@`.

Table 3.1.: Basic FSP process operators.

shared actions, in which case the new action label is shown as a label on the connecting line. A composite process is shown as a box enclosing a set of process boxes.

Consider the following example diagram for `||S=(P||Q)/{m/x,m/y}@{a,b}`.



The diagram is a clear and concise visual representation of process S. It shows that S is created from the composition of P and Q. It makes apparent that the alphabet of

P is the set of actions consisting of `a` and `x`. The line connecting P and Q indicates that these two processes interact. Since the action labels differ, the relabelling, `m`, to be applied in order to obtain a shared action is shown at the interaction point in the diagram. The actions that can take place at the (external) interface of S are indicated by the circles on the border of the outside block.

3.2. Primitive Processes

The models for the traffic conditioners, described in the sections that follow, are created using the primitive processes defined in this section. In this section the models of the primitive processes are explained. In the later sections the compositions, using these processes, will be discussed.

The following primitive process are discussed in this sections.

BUCKET — a process that models the behaviour of a token bucket used to determine whether an arriving packet is in- or out-of-profile. This process is used in the TMB, srTCM, and trTCM processes.

TOKEN — a token generating process used to add tokens to a token bucket. All the composite processes that include a BUCKET process will have one of these to generate tokens for the bucket.

RATE_EST — a process that models a rate estimator used to determine the current rate of an incoming traffic stream. This process is used in conjunction with the MARKER process.

MARKER — a process that assigns colours to packets based on the outcome of the RATE_EST process. This process is used in the TSW2CM and TSW3CM processes.

TOKEN The simplest process definition is that of the TOKEN process. The definition of the process is given in the box below.

Token generation process

```
TOKEN = (token -> TOKEN).
```

The TOKEN process models the generation of tokens to be added to the token bucket. It does this by simply repeating the `token` action.

Note that the rate of token replenishment is not modelled by an FSP specification. Instead the replenishment rate is regarded as an implementation detail.

Token Bucket Process

```
1  BUCKET(N=20) = T[N],      /* Bucket is initially full */
2  T[b:0..N] = (in.packet[p:P]->
3                 if (b-p >= 0) then
4                    (inProfile.packet[p] -> T[b-p])
5                 else
6                    (outProfile.packet[p] -> T[b])
7              |token -> if b < N then
8                           T[b+1]
9                        else
10                          (stay -> T[b])
11             ).
```

BUCKET The BUCKET process models the behaviour of a token bucket (see Sec. 2.4). The actions in which the process can engage (i.e. its alphabet) represent the following: a packet of size `p` arriving (`in.packet[p:P]`); a token to be added to the bucket (`token`); a packet that has been determined to be out of profile (`outProfile.packet[p]`); and a packet that has been determined to be in profile (`inProfile.packet[p]`).

The process definition above is for a token bucket with capacity N tokens. The process is defined in terms of an indexed local sub-process, T[b:0..N], representing the behaviour when `b` tokens are in the bucket.

Initially the BUCKET process can engage in one of two actions: `in.packet` (line 2) or `token` (line 7). When a `token` action occurs, the number of tokens in the bucket is incremented (`T[b+1]`) up to the maximum value (N). The `stay` action (line 10) is required for synchronisation purposes. The precise reason for introducing it into the definition of BUCKET will be explained in Section 3.6.

When a packet arrives, it must be determined whether the packet is in profile or not. A packet of size p is in profile when there are enough tokens in the bucket, i.e. when $b - p \geq 0$ where b is the current number of tokens in the bucket (refer to lines 3 and 4). If the packet is determined to be in-profile, then the number of tokens in the bucket is decreased by an amount that corresponds to the size of the packet. (line 4)

Rate Estimator Process

```
RATE_EST = (estimate -> rate[r:R] -> RATE_EST).
```

RATE_EST The RATE_EST process abstracts away from the details of rate estimation calculations. It simply models the entire calculation activity as a single action, `estimate`, which is followed by a `rate` action that has an index to represent the current estimated rate of a traffic stream.

Probabilistic Packet Marker

```
1  MARKER(PIR=10,CIR=5) =  RATE[0],
2  RATE[i:R] = (in.packet -> if i > PIR then
3                (red    -> RATE[i]
4                |yellow -> RATE[i]
5                |green  -> RATE[i]
6                )
7              else
8                if i > CIR then
9                  (yellow -> RATE[i]
10                 |green  -> RATE[i]
11                 )
12               else
13                 (green  -> RATE[i])
14             |rate[r:R]->RATE[r]).
```

MARKER The MARKER process above is used to assign a colour to an arriving packet based on the current estimated rate. The process takes two rates as arguments; `PIR` which represents a peak rate; and `CIR` which represents a committed, or guaranteed, rate. Based on these rates together with the current rate, it determines which colour to assign to a packet.

The process is defined by an indexed local sub-process definition RATE[i:R], whose index corresponds to the currently estimated rate.

Initially the process can engage in one of two actions, `in.packet` (line 2) or `rate` (line 14). The former corresponds to an arriving packet and the latter sets the process state to reflect the current estimated rate.

Note that FSP—and, in fact, all process algebras—abstract away from probabilistic issues with regard to alternative actions that may occur in the presence of the choice operator. Instead, a process definition is intended to define the set of *all* action traces in which the process may engage. As a result, the MARKER definition abstracts away from the fact that at an implementation level, colour assignments are probabilistically based on the current rate. Consider, for example, the case where the current rate is between the two configured rates, i.e $CIR < i < PIR$ (line 8). In this case, according to the model, the packet can be assigned either `yellow` or `green`. However, the model does not provide any information about the probability with which either of the actions will take place.

3.3. Token Bucket Marker

This section describes the composite process for the TBM. The structure diagram for the model is shown in Figure 3.1.

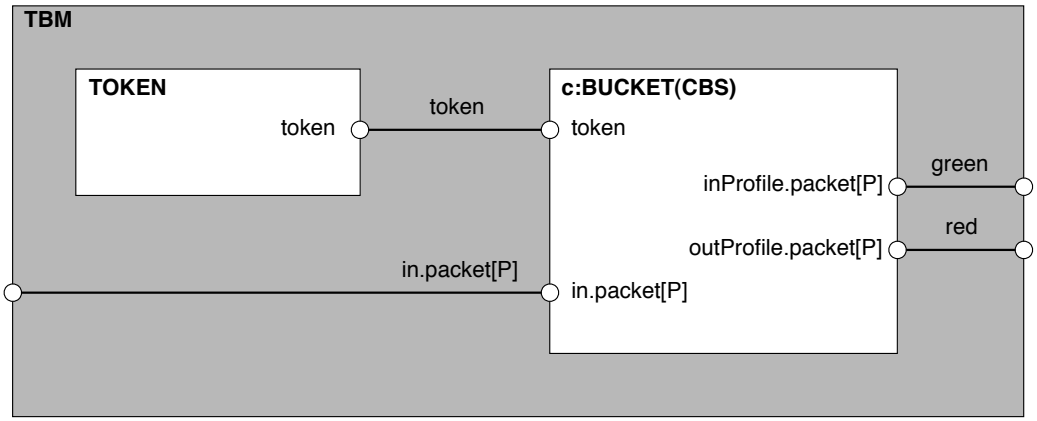


Figure 3.1.: Structure diagram for the Token Bucket Marker.

From the structure diagram, one can see that the TBM process is composed of a TOKEN and a BUCKET process. These processes interact through the shared action `token`. The TBM process assumes that some packet arrival process in its environment provides packets to it, these being represented by the `in.packet[P]` actions. Of course, at the current level of abstraction, the arrival rate of packets is not relevant.

The interface of the process is provided by the `in.packet[P]`, `green`, and `red` actions. The first action should be regarded as a packet that the process 'consumes' as input, and the latter two actions signify the packet marked by a specific colour as output.

The box below contains the FSP for the composition of the TBM.

Token Bucket Marker Composition

```
1 ||TBM = (TOKEN||c:BUCKET(CBS))
2        /{green/c.inProfile.packet[P],
3          red/c.outProfile.packet[P],
4          in.packet/c.in.packet,
5          token/c.token}
6       @{in.packet[P],green,red}.
```

The model for the TBM is specified as the parallel composition of a TOKEN process and a BUCKET process. The argument to the BUCKET process determines the capacity of the token bucket. The size of the bucket controls the allowed burst size

of a packet stream. Though not modelled, a TOKEN process implementation would produces tokens at a rate given by the CIR.

The labelling of processes is only necessary when multiple copies of a process are needed in a composition. In the TBM process there is only one BUCKET process so the labelling, c:BUCKET, is not necessary. The labelling is, however, not incorrect; it was done merely as a matter of style in order to be consistent with the definitions of SRTCM and TRTCM in which BUCKET processes are also used and in which labels are indeed necessary.

The relabelling in lines 2 and 3 is needed to assign a 'colour action' to an in- or out-of-profile packet. In lines 4 and 5 the action names are changed to obtain 'shared' actions so that the processes can synchronise on these actions. The interface exposed by the TBM process is made explicit in line 6.

3.4. Time Sliding Window Three Colour Marker

This section explains the FSP model of the TSW3CM. The structure diagram for the process can be found in Figure 3.2.

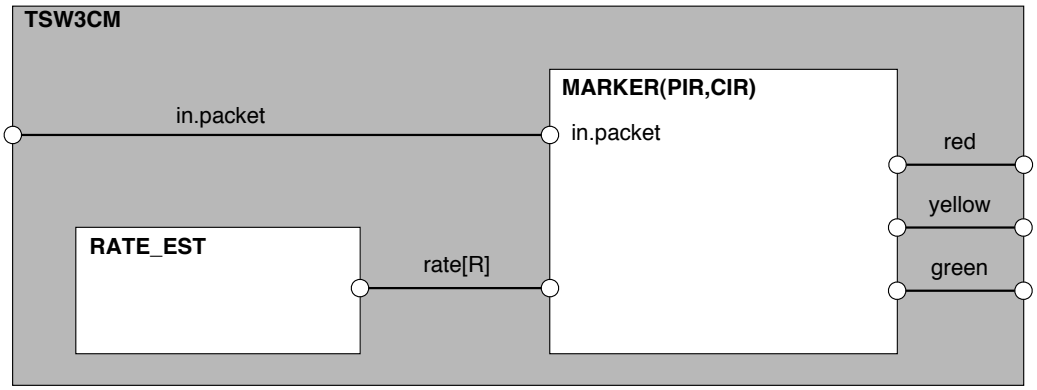


Figure 3.2.: Structure diagram for the Time Sliding Window Three Colour Marker

The TSW3CM process is specified as the composition of a RATE_EST and a MARKER process. The RATE_EST process provides an estimate of the current rate of packet arrivals to the MARKER process and the MARKER process determines the colour to assign to the arrived packet. The details of these processes are specified in Section 3.2.

Time Sliding Window Three Colour Marker Composition

```
1 ||TSW3CM = (RATE_EST||MARKER(PIR,CIR))
2            /{in.packet/in.packet[P]}
3            @{in.packet,green,yellow,red}.
```

The MARKER process uses two arguments to specify the rates that are used to determine the possible colours that can be assigned to a packet. The first argument is a maximum, or peak, rate; the second is a guaranteed, or committed, rate. If the current rate exceeds PIR, then the packet is marked as 'red', 'yellow', or 'green'. If the current rate is between PIR and CIR, then the packet is marked as 'yellow' or 'green'. Lastly, if the current rate is less than CIR, the packet is marked as 'green'.

Recall that the colours are assigned with different probabilities, but the notion of probability is not available in the model. The model only indicates that one of several colours might be assigned to the packet.

3.5. Time Sliding Window Two Colour Marker

The structure of the TSW2CM process is given in Figure 3.3. From the diagram it can be seen that the TSW2CM process is very similar to the TSW3CM process. The only changes are that the `yellow` action in the MARKER process is relabelled to be `red` and one of the rate parameters of the MARKER process is set to infinity[1].

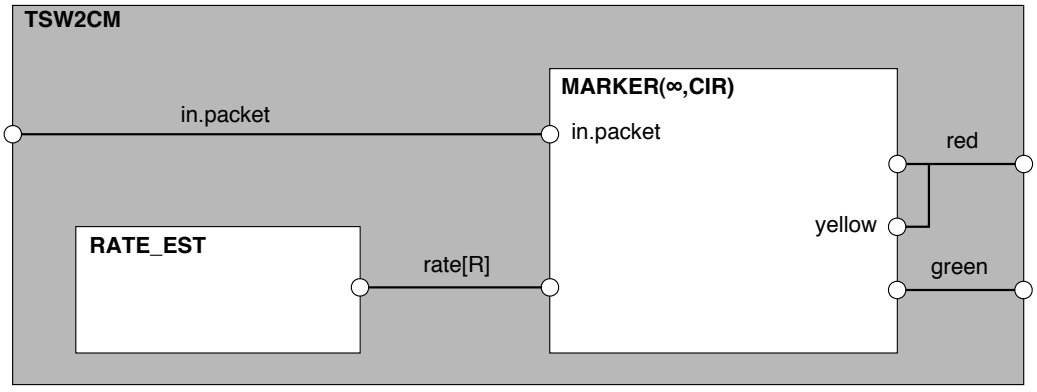


Figure 3.3.: Structure diagram for the Time Sliding Window Two Colour Marker

[1] In the structure diagram and subsequent FSP definitions, the ∞ symbol is used to represent infinity even though it is not valid FSP syntax; a constant of adequate size would typically be used to represent infinity.

In the TSW3CM the current estimated rate fell into one of three intervals: less than CIR, between the CIR and PIR, and greater than the PIR. By setting the first rate parameter (PIR) of the MARKER process to infinity the current rate can only fall into one of two intervals: less than the CIR or greater than the CIR. Furthermore, in TSW2CM, packets arriving while the current estimated rate exceeds the CIR are all subject to being marked 'red' with some probability, i.e. the possibility of packets being marked 'yellow' does not arise. This is the reason for relabelling the `yellow` actions that appear in the definition of the MARKER process to `red` the definition of the TSW2CM process.

Time Sliding Window Two Colour Marker Composition

```
1 ||TSW2CM = (RATE_EST||MARKER(∞,CIR))
2            /{in.packet/in.packet[P],red/yellow}
3            @{in.packet,green,red}.
```

The relabelling mentioned earlier can be seen in line 2 of the preceding FSP specification. Note that in line 3, only two colours (red and green) are part of the process's interface, hence the *two colour* marker nomenclature.

3.6. Single Rate Three Color Marker

The model for the srTCM is created by the parallel composition of a TOKEN process and two BUCKET processes. The structure diagram for the composite process can be seen in Figure 3.4.

The single TOKEN process generates new tokens for both BUCKET processes. Though not modelled, the TOKEN process generates tokens at a rate equal to the CIR. The relabelling has been designed to ensure that the initial tokens that are generated interact with the c:BUCKET process (line 8). If, however, that bucket is full, then the relabelling redirects interaction to the e:BUCKET process via the dummy event in the BUCKET process called 'stay' (line 7).

Single Rate Three Color Marker Composition

```
1 ||SRTCM = (TOKEN||c:BUCKET(CBS)||e:BUCKET(EBS))
2           /{yellow/e.inProfile.packet[P],
3             green/c.inProfile.packet[P],
4             e.in.packet[i:P]/c.outProfile.packet[i],
5             red/e.outProfile.packet[P],
6             in.packet/c.in.packet,
7             e.token/c.stay,
8             token/c.token}
9           @{in.packet[P],green,yellow,red}.
```

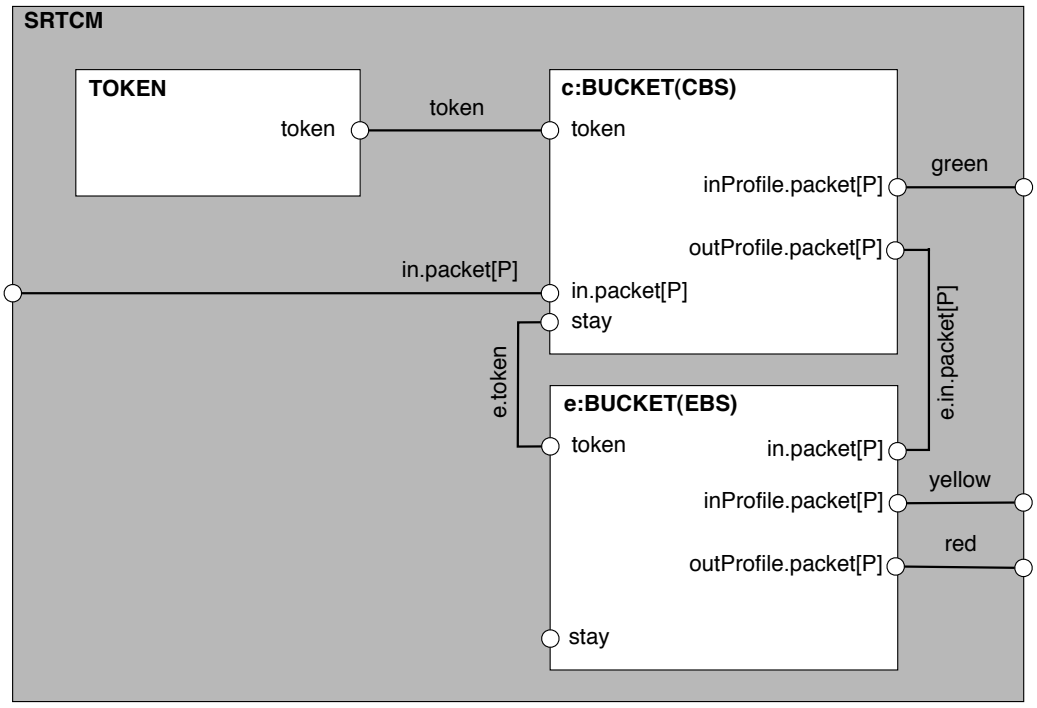


Figure 3.4.: Structure diagram for the Single Rate Three Color Marker.

The arriving packet is marked 'green' if the first bucket (c:BUCKET) finds it to be in profile (line 3). If the first bucket finds the packet to be out-of-profile, then it transfers the packet on to the e:BUCKET process, this transfer being modelled here by a relabelling operation (line 4). The e:BUCKET process marks in profile packets as 'yellow' (line 2) and marks out-of-profile packets as 'red' (line 5).

In this model, there is only one TOKEN process and it is assumed to generate tokens at a fixed (i.e. single) rate. For this reason, and because the marker colours packets into one of three colours, it is called a *single rate* three colour marker.

3.7. Two Rate Three Colour Marker

The trTCM process is specified as the composition of two TOKEN processes and two BUCKET processes. The structure of the composition can be seen in Figure 3.5. The processes p:BUCKET and c:BUCKET correspond to the two token buckets P and C, respectively.

The TRTCM process is similar to the SRTCM process since two BUCKET processes are used to determine the colour of an arriving packet. The difference lies in the fact

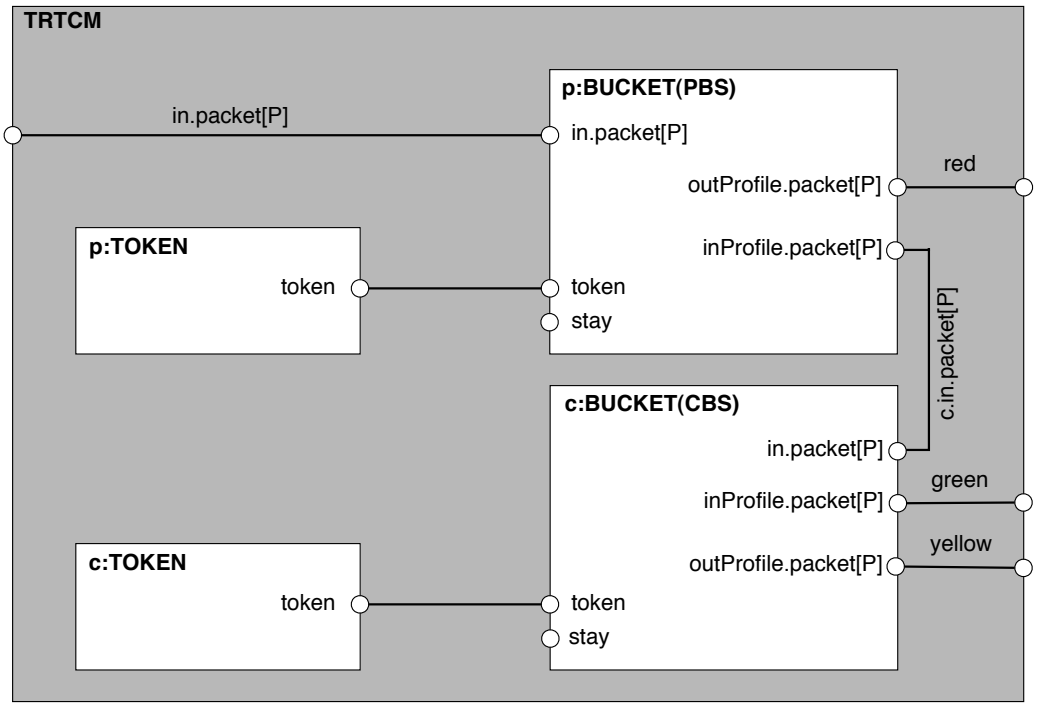


Figure 3.5.: Structure diagram for the Single Rate Three Color Marker.

that the TRTCM uses two TOKEN process to generate tokens as opposed to the single TOKEN in the SRTCM process. The c:TOKEN process generates tokens at the CIR and the p:TOKEN process at the PIR.

The c:TOKEN process interacts with the c:BUCKET process and the p:TOKEN process interacts with p:BUCKET. The tokens can be generated at differing rates, whence the *two rate* three colour marker.

Two Rate Three Colour Marker Composition

```
1 ||TRTCM = ({p,c}:TOKEN||c:BUCKET(CBS)||p:BUCKET(PBS))
2           /{yellow/c.outProfile.packet[P],
3             green/c.inProfile.packet[P],
4             c.in.packet[i:P]/p.inProfile.packet[i],
5             red/p.outProfile.packet[P],
6             in.packet/p.in.packet}
7           @{in.packet[P],green,yellow,red}.
```

The packet is marked as 'red' if the p:BUCKET process finds the packet to be out-of-profile (line 5). If it is in profile the c:BUCKET process determines whether it is marked 'yellow' (out-of-profile, line 2) or 'green' (in profile, line 3).

3.8. Comparing the Traffic Conditioners

Models for the TBM, srTCM, trTCM, TSW2CM, and TSW3CM conditioners were defined in the preceding sections of this chapter. The FSP definitions described the dynamic behaviour of the processes and the structure diagrams conveyed the static structure or architecture of the models. This section provides an overview of the differences and similarities between the different types of traffic conditioners, as highlighted by their respective models.

The conditioner comparison is based upon the primitive processes utilised in their respective models, the number of possible colours assigned to packets, and the parameters used for configuration. Table 3.2 shows which columns are relevant to each of the traffic conditioner models. The filled circles represent items present in the models of the traffic conditioners and the empty circles represent items not explicitly part of the models but required in an implementation of the model.

	Primitives				*Colours*		*Parameters*			
	TOKEN	BUCKET	RATE_EST	MARKER	Two	Three	CIR	CBS	PIR	EBS/PBS
TBM:	●	●			●		○	●		
TSW3CM:			●	●		●	●		●	
TSW2CM:			●	●	●		●			
SRTCM:	●	●●				●	○	●		●
TRTCM:	●●	●●				●	○	●	○	●

Table 3.2.: Traffic Conditioner Model Properties.

The information in Table 3.2 suggests various dimensions in which conditioners can be classified or grouped together. Firstly, the constituent primitive processes of the conditioner models suggest two groups of conditioners in this dimension: those that are token bucket based and those that are rate estimator based. The TBM, SRTCM, and TRTCM conditioner processes are token bucket based while the TSW3CM and TSW2CM conditioner processes are rate estimator based.

The number of possible colours that can be assigned to a packet offers another dimension in which the conditioners can be classified, again into two groups: processes in the first group assign one of two colours to packets; and the processes in the other group assign one of three colours to packets. From the table it is evident that TBM and TSW2CM assign two colours and that TSW3CM, SRTCM, and TRTCM assign three colours.

If the parameters required to configure the traffic conditioners are considered, then, yet again, the conditioners can be classified into two groups in two different ways. The one way to group is on whether or not marking is done exclusively based on the 'committed' parameters CIR or CBS. Since the TBM and TSW2CM models are only configured with CIR or CBS, they belong to one group. The other models TSW3CM, SRTCM, and TRTCM belong to the other group since they are configured with additional parameters.

The second way to group the conditioners based on configuration parameters, is on whether or not the conditioner is configured with a notion of 'burst size' (CBS, EBS, or PBS). TBM, SRTCM, and TRTCM utilise burst parameters while TSW3CM and TSW2CM do not.

There is some redundancy in Table 3.2. The classification can be simplified to reflect only two conditioner properties. The one property indicates whether the conditioner's process model is token bucket or rate estimator based; the other indicates whether it assigns two or three colours. Table 3.3 shows where each of the conditioners fits into such a classification scheme.

		Colours	
		Two	*Three*
Scheme	*Token Bucket*	TBM	srTCM trTCM
	Rate Estimator	TSW2CM	TSW3CM

Table 3.3.: Process Classification.

The foregoing provides an elegant basis for comparing key features of the various conditioners. In each case, the model is a summary of all the possible traces of actions in which a given process can engage. However, such a model does not indicate which set of traces will predominate in the face of a certain traffic pattern. For example, a model does not indicate what proportion of incoming packets (modelled above as `in.packet[P]` actions) will be labelled a particular colour by a particular conditioner. To study these characteristics of conditioners, simulation is employed. This will be the theme of forthcoming chapters.

4. Experimental Configuration

> Life is short, the art long, opportunity fleeting, experiment treacherous, judgment difficult.
>
> *Hippocrates*

The purpose of this chapter is to communicate the configuration of the simulation-based experiments carried out in the study.

Section 4.1 provides a motivation for the study and background to the simulation environment used. In Section 4.2 the simulation topology is described, the source traffic generators are explained in Section 4.3, and the differentiated domain configuration is provided in Section 4.4. Section 4.5 provides a description of the different scenarios that were simulated. The chapter is concluded in Section 4.6 by a summary of the data collected for analysis in the next chapter.

4.1. Background

This section provides background to the problem investigated through the subsequent simulation study. It also provides information on the simulation package used to perform the simulations.

4.1.1. Motivation and Goal

The motivation of the study originates from the domain of computer network design. A network designer is often confronted with design decisions ranging from determining a network topology to deciding on the values of various, sometimes seemingly insignificant, parameters. When designing a DS domain, or DS node for that matter, one needs to decide on which traffic conditioner to use in a specific environment. Clearly, the expected traffic characteristics and subscription level of the network would be two important factors to consider in making such a decision.

In Chapters 2 and 3, five traffic conditioners were described. These correspond to conditioners described in RFCs. Other research (refer to Section 6.2) has investigated conditioners under fairly fixed circumstances (such as using only one traffic type, for example), or proposed new conditioners not described in RFCs (at time of writing).

Thus, no study of conditioners against traffic characteristics were encountered. The forthcoming chapters investigate whether these five traffic conditioners are more suited to certain environments than to others. The environments to be considered are varied based on the expected type of traffic and the level of subscription in the network. These scenarios are explained in more detail later in Section 4.5. The performance measures are also provided later in Section 4.6.2.

4.1.2. The *ns-2* Network Simulator

The experiments were performed in a discrete event simulation environment namely *ns-2* version 2.26 (McCanne and Floyd, 2003; Fall et al., 2003; Breslau et al., 2000). The simulations were done in a slightly modified version of the package. The modification is given in 4.6.1.

Models in *ns* are created by defining nodes and connecting them with links to form some network topology. For the network to actually do something, network traffic needs to be generated and transferred between nodes. Traffic is generated at nodes in *ns* using so-called traffic generation applications.

Traffic generating applications reside on top of transport agents corresponding to transport layer protocols such as User Datagram Protocol (UDP) (Postel, 1980) and Transmission Control Protocol (TCP) (Postel, 1981).

The following traffic generating applications exist in *ns*.

CBR An implementation of a traffic source that generates traffic at a constant rate.

Exponential This application generates traffic according to an Exponential On/Off model.

Pareto The application generates traffic using a Pareto On/Off scheme.

Traffic Trace A trace file is used to generate traffic originating at the node. The trace file contains a series of pairs, each consisting of two fields. One field represents the size of the next packet to be generated and the other represents the time until the next packet is generated.

It is possible to simulate a DS domain in *ns* using its built-in Diffserv module. The Diffserv module supports up to four classes of traffic, each with up to three dropping precedence levels. Packets in a single class of traffic are enqueued into one physical RED queue, which contains virtual queues for each drop precedence level. Each virtual queue is configured with its own RED parameters.

The Diffserv module in *ns* has three major components: policy, edge routers, and core routers. Policy is determined by an agreement with the customer and specifies the

level of service that a class of traffic should receive in the network. Edge routers mark packets with a DSCP according to the policy specified and the properties of the traffic stream. Core routers examine packets' DSCPs and forward them accordingly. Both core and edge routers contain the functionality to forward packets based on the DSCP. In fact, core routers are edge routers with the traffic measuring and marking functions removed.

The Diffserv module keeps track of how many packets of each code-point value are received, sent, and dropped in the network. The module distinguishes between two types of drops—early drops and late drops. Late drops are due to physical buffer overflow, i.e. there is no more space in the buffer to receive packets. Early drops are due to RED dropping packets. Recall that RED queues drop packets probabilistically based on their configuration parameters.

4.2. Simulation Topology

The nodes in the simulation were connected according to a simple symmetrical dumb-bell topology. The simulation topology can be seen in Figure 4.1.

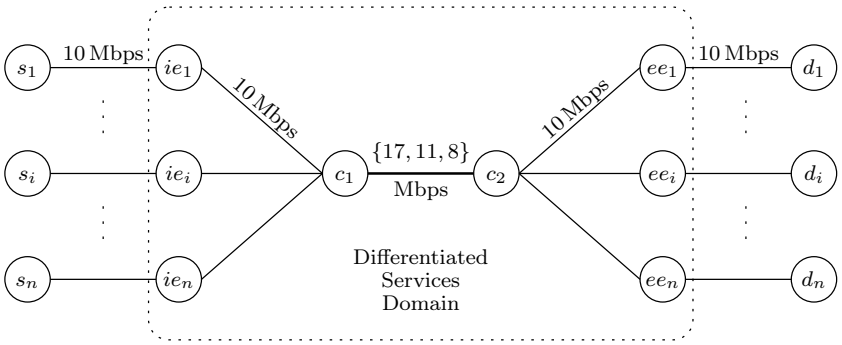


Figure 4.1.: Simulation Topology.

There are five different types of nodes in the simulation. Each source node (s_i) sends data to each destination node (d_i). Each source is connected to the Diffserv domain by a direct connection to an ingress edge node (ie_i). The destination nodes are similarly connected to egress edge nodes (ee_i). There are two core nodes (c_1 and c_2) in the simulation. The core nodes are connected via a single bottle-neck link. This link is assigned capacities from the set $\{8\,\text{Mbps}, 11\,\text{Mbps}, 17\,\text{Mbps}\}$.

For the purpose of this study $n = 10$ identical sources with identical TCSs were used in each simulation scenario. The enforcement of the TCS of source s_i is done at the ingress boundary through ie_i.

4.3. Source Traffic Generation

This section provides details of the configuration of the traffic generating applications used in the simulations. Each traffic type is discussed in Section 2.5. For this reason, the description here focusses on the values of the parameters and not on the detail operation of the models.

Let $S = \{\text{CBR}, \text{EXPOO}, \text{PAROO}, \text{POISSON}, \text{H1}, \text{H2}, \text{H3}\}$ denote the set of source models used in the simulations. Details of the elements of S are provided below.

All sources generate packets of size 576 bytes using UDP as transport layer protocol. 576 byte packets were chosen since they are common in the Internet (Thompson et al., 1997). UDP was chosen over TCP since it does not contain complex flow- or congestion control mechanisms. The observed behaviour is therefore not influenced by transport layer protocol mechanisms but only by the Diffserv mechanisms. Another reason why UDP is more appropriate than TCP is that the source traffic application is not necessarily a simulated application running on a single node. The source application simulates a traffic stream. This stream could consist of multiple application 'flows'. If these 'flows' were running over TCP, each would need its own TCP connection, which is not provided in the present scenario.

Each source generates traffic at an approximate average rate of 2 Mbps. There is therefore ten 2 Mbps traffic streams, each transmitted over a 10 Mbps line, which then converge on a channel of, at most 17 Mbps. The bottle-neck link will therefore be heavily congested while the 10 Mbps links will not be. Packet discard is therefore only expected at the bottle-neck link.

CBR The source generates traffic at a constant rate of 2 Mbps. The sources were configured to introduce some small random variation between packets. For more details on the CBR traffic model and the reason for random variation between packets, refer to Section 2.5.1 on page 24.

POISSON The Poisson arrival processes used in the simulations were created using the technique described in Section 2.5.2 on page 24. An inter-arrival process was created in which the times between packets are drawn from an Exponential distribution with mean 2304 μs. As a result, the number of packets arriving per unit of time is described by a Poisson process with an average packet arrival rate of 434 pps. (This rate is, of course, the inverse of the mean of the Exponential distribution, with the appropriate conversion from micro-seconds to seconds.) Since all packets are 576 bytes in size, the average sending rate is $434 \times 576 \times 8 = 2$ Mbps. Thus, the Poisson process parameters were chosen so that the average sending rate at each source is directly comparable to the sending rate used in the CBR simulation.

The inter-arrival process was implemented as an external program and used to generate a set of data pairs as input data for a Traffic Trace application in *ns*.

EXPOO This Exponential On/Off model has already been outlined in Section 2.5.3 on page 25. It is one of the source generation models built into *ns*. For the purpose of this study it was configured as follows. The average length of both ON and OFF periods was set to 500 ms. The sending rate during ON periods was 4 Mbps. As a result, this model is designed to reflect a bursty traffic situation in which, for approximately half of the time in a given time period, about twice as many packets are emitted as in the CBR model, while no packets at all are emitted during the remaining time. This, again, results in an expected average rate of 2 Mbps.

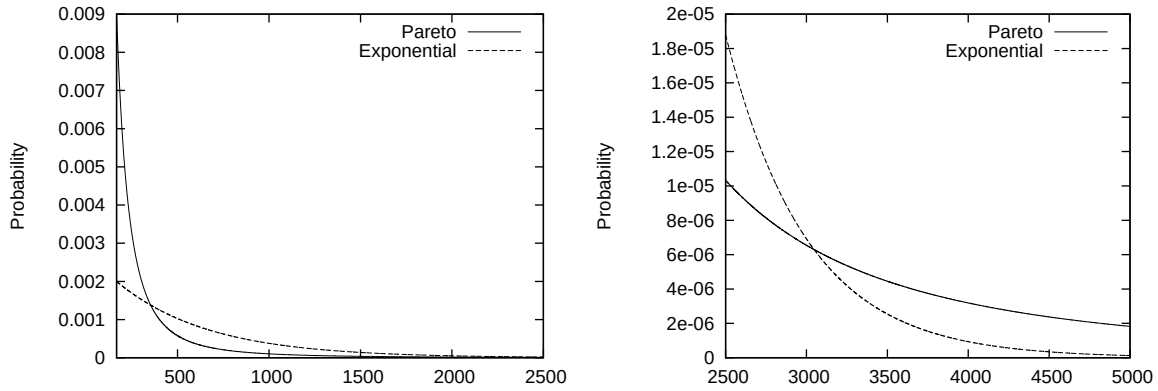


Figure 4.2.: A Pareto and Exponential distribution.

PAROO This Pareto On/Off model is another of the source traffic generation models built into *ns*. It was described in Section 2.5.4 on page 26. For the purpose of this study it was configured along similar lines to the previous model. The average length of both ON and OFF periods was set at 500 ms, and the sending rate during ON periods was set to 4 Mbps.

Note that when the shape parameter of a Pareto distribution is between one and two, a distribution mean exists and the distribution's variance is infinite. The shape parameter was left at the system default of 1.5, which is in the middle of the acceptable range. Figure 4.2 shows the Probability Distribution Functions (PDFs) of a Pareto and a Exponential distribution with the same mean. The shape parameter of the Pareto distribution in the figure is set to 1.5. The figure therefore shows the probability that a given duration ON (or OFF) period will occur for the respective distributions.

From the plot on the right it can be seen that the probability of obtaining very long periods is greater for the Pareto distribution than for the Exponential distribution.

H1, H2, H3 Three self-similar traffic sources (H1, H2, and H3) were used in the simulation study. The source traffic processes were created using an implementation by Paxson (1997) of the method detailed in Section 2.5.5 on page 27 and only differ in the value of the Hurst parameter (H). The sample paths were created with mean 44, variance 144, and Hurst parameter one of $\{0.5, 0.7, 0.95\}$ respectively. H1 corresponds to $H = 0.5$, H2 to $H = 0.7$ and H3 to $H = 0.95$. The values of the Hurst parameter creates traffic streams with varying degrees of long range dependance. As the value of H increases the degree of long-range dependence also increases.

The program used to create the sample paths, produces an arrival process, i.e. it gives the number of arrivals in a time interval. *ns*, however, requires an inter-arrival process (time between arrivals) as input for the Traffic Trace application. For this reason the arrival process was translated into a inter-arrival process according to the following scheme. Each entry x_i in the arrival process represents the number of packets arriving in the time-interval t_i of duration 0.1 s. Since the average number of arrivals in t_i is 44 packets of 576 bytes each, the average rate is again close to 2 Mbps. The number of arrivals per interval t_i was then converted to inter-arrival times by distributing all x_i arrivals evenly across the interval t_i and specifying the size of each arrival to be 576 bytes. The resulting inter-arrival process was used as a Traffic Trace application in *ns*.

4.4. Differentiated Services Domain

This section details the configuration of the nodes in the Diffserv domain. The simulated scenario consisted of a single Diffserv domain with multiple nodes representing different subscribers connected to the domain. Sources send data through the provider network to destinations which are also connected to the DS domain.

The first facet of the DS domain that should be determined is that of policy. Policies should be formulated that specify the network resources to which each class of subscriber is entitled. These policies should then be policed to determine to what extent the subscriber adheres to its side of the agreement. From the policing process, packets are assigned different precedence levels based on the current traffic characteristics and the TCS. Thus, under conditions of high channel utilisation, the policing process will be inclined to assign low precedence levels to packets from subscribers whose traffic generation activity does not conform to the agreed-upon policy. The packets with different precedence levels are enqueued into different virtual queues, each queue with its own set of RED parameters.

4.4.1. Policies

As described in Section 2.2, policies are defined between the customer and the provider in a TCS. The TCS is policed at the ingress boundary nodes using traffic conditioners.

In the study different ways of defining and policing customer traffic streams were used. Let $P = \{\text{TBM}, \text{TSW3CM}, \text{TSW2CM}, \text{srTCM}, \text{trTCM}\}$ be the set containing the different traffic conditioners used in the experiments. The configuration parameters and functioning of these traffic conditioners are described in Section 2.4. Table 4.1 contains the values of the configuration parameters for each of the elements of P.

Parameter	TBM	TSW3CM	TSW2CM	srTCM	trTCM
CIR (Mbps)	1	1	1	1	1
CBS (B)	5760	—	—	5760	5760
PIR (Mbps)	—	1.5	—	—	1.5
EBS/PBS (B)	—	—	—	2880	2880

Table 4.1.: Policy parameter values.

All the traffic conditioners are configured with a committed rate (CIR) of 1 Mbps. This rate is the average rate that the provider commits to provide to the subscriber. This committed rate will also be referred to as a subscriber's 'reserved' rate (rr_i).

The traffic conditioners based on a token bucket scheme include the notion of a committed burst size (CBS). The burst size was set to 5760 bytes which translates into ten packets.

The peak rate that is allowed (PIR) is set at 1.5 Mbps and the excess and peak burst sizes (EBS and PBS) are set to 2880 bytes, or five packets.

Notice that the source applications' sending rates all exceed the rates specified in the policies. One would therefore expect to observe dropped packets. This study therefore intentionally simulates a situation of severe abuse of network resources, precisely in order to monitor the performance of various conditioners in such a case.

As described earlier, packets in a Diffserv domain are marked with a DSCP according to the outcome of the metering process in the traffic conditioner. Recall that the DSCP is a value in the DS field in the IP packet header. The value determines the per-hop behaviour of a packet at a DS node. The PHB is created by the values of configuration parameters of the RED queues in the DS node.

The initial code-point used for packets entering the simulated DS domain is '10'. The DS field of packets that are determined to be 'green' at the ingress nodes (ie_i) is also set to '10'. The DS field of packets determined to be 'yellow' is set to '11' and 'red' packets are marked '12'. The core nodes (c_1 and c_2) only use the DSCP to determine into which queue to place a packet; they do not manipulate the DSCP.

4.4.2. Queue Configuration

This section provides the configuration details of the RED queues used in the simulated scenarios.

In the simulated network, packets with differing DSCPs are put into the same physical queue but separate virtual queues with different RED parameters for each virtual queue. These queues exist at the edge nodes as well as the core nodes. At an edge node, the traffic stream is monitored and a DSCP is assigned to a packet; based on the DSCP the packet is placed into an appropriate queue and forwarded. At a core node, the packet is inspected to determine its DSCP, it is placed into an appropriate queue and then forwarded.

The RED queues at the edge nodes and the core nodes are configured identically. The nodes have one physical queue of finite capacity and three virtual queues within the physical queue. There is one virtual queue for each DSCP (10, 11, 12) in the DS domain. The lower priority packets (DSCPs 11 and 12) are more aggressively managed through stricter RED parameters.

The maximum physical queue size is 600 packets. There can therefore only be 600 packets in a queue; if any more packets arrive, they are discarded irrespective of their DSCPs.

The RED parameters used for the virtual queues are listed in Table 4.2.

Colour	DSCP	min_{th}	max_{th}	max_p
Green	10	60	120	0.02
Yellow	11	40	60	0.5
Red	12	25	50	0.5

Table 4.2.: DSCPs and the corresponding RED parameters.

Recall from Section 2.3 that if the current average queue size of a RED queue is less than min_{th}, the packet is not discarded and if it is greater than max_{th}, the packet is discarded. When the average queue size is between min_{th} and max_{th}, however, the probability of discard grows as the queue size increases, up to a maximum probability equal to max_p.

Packets with DSCP 10 have less aggressive RED parameters than those with different DSCPs. The minimum and maximum thresholds for DSCP 10 packets are set to 60 and 120 packets, respectively, and max_p is set to a fairly low 0.02. The other virtual queues are managed more aggressively with both max_p's set to 0.5. The management also starts earlier with the min_{th} for DSCP 11 set to 40 packets and max_{th} to 60 packets. The management starts even earlier for DSCP 12 packets; min_{th} is set to 25 and max_{th} to 50 packets.

From the RED configurations above, it is clear that packets which are in-profile (DSCP 10) are policed less aggressively than out of profile packets (DSCP 11 and 12), with DSCP 12 packets given the lowest priority.

4.5. Simulation Scenarios

Recall from Section 4.2 that three different bottle-neck link capacities (8 Mbps, 11 Mbps, 17 Mbps) are used in the simulations. Since there is a reserved rate (rr_i) of 1 Mbps associated with each of the $n = 10$ subscribers, one can obtain a so-called reservation level for the network by calculating how much of the bottle-neck link capacity is reserved for, or committed to, subscribers. The reservation level is therefore $\sum_i rr_i$ divided by the link capacity (C). The reservation level for a network with a 8 Mbps link is then 125%, for an 11 Mbps link it is 90%, and for a 17 Mbps link it is 60%. These levels respectively correspond to an oversubscribed, nearly fully subscribed, and an undersubscribed network. In fact, the values 8, 11 and 17 were specifically chosen to reflect these three scenarios. Let $R = \{125\%, 90\%, 60\%\}$ be the set of reservation levels used in the simulation study.

Recalling that $S = \{\text{CBR}, \text{EXPOO}, \text{PAROO}, \text{POISSON}, \text{H1}, \text{H2}, \text{H3}\}$ and $P = \{\text{TBM}, \text{TSW3CM}, \text{TSW2CM}, \text{srTCM}, \text{trTCM}\}$, the different scenarios that were simulated can be represented by the Cartesian product of the sets mentioned before: $SIMS = R \times S \times P$. Thus, for example, one simulation scenario assumed a 125% reservation level on the bottle-neck link, where CBR traffic was generated (at 2 Mbps) at each of the n source nodes, this traffic being policied at the ingress nodes according to, say, the TBM policing policy; another corresponded to the foregoing, but assumed a 90% reservation level, etc. This means that a total of $3 \times 5 \times 7 = 105$ scenarios were simulated.

Every simulation scenario element in $SIMS$ was run ten times. The simulations were run for 60 s (simulation time).

4.6. Data Collection

This section contains information about the data collected for the simulation study.

4.6.1. Terminology and Notation

The following terms and symbols are used in the subsequent discussions relating to the experimental observations and results. Note that the packet counts and average rates are measured or calculated over the complete simulation period of sixty seconds. Also note that packets are always 576 bytes in size.

- $N_{\text{in},i}$: the number of packets produced by source s_i after the simulation completed.
- $N_{\text{out},i}$: the number of packets received at destination d_i.
- $N_{10,i}$: the number of packets with DSCP 10 received at destination d_i.
- $N_{11,i}$: the number of packets with DSCP 11 received at destination d_i.
- $N_{12,i}$: the number of packets with DSCP 12 received at destination d_i.
- $\nu(x)$: a function which translates an average bit rate, x, into a packet count. Here, and in $\rho(x)$ below, it is assumed that the time over which the average is taken, is the simulation time of 60 s. Further are all packets 576 bytes in size.
- $\rho(x)$: a function which translates a packet count, x, into an average bit rate.
- rr_i: the reserved bit rate of source s_i. This rate is equal to the configured committed bit rate between source s_i and destination d_i.
- tr_i: the target bit rate of source s_i. The target rate is the average bit rate that the subscriber should achieve during the simulation. The formulation is given below. Keep in mind that C is the bottle-neck link capacity and n is the number of subscribers in the network.

$$tr_i = rr_i + \frac{C - \sum_{j=1}^{n} rr_j}{n}.$$

  Thus the target rate is equal to the node's reserved rate plus its equal share of the excess capacity (or shortfall of capacity) on the bottle-neck link. If the network is oversubscribed, i.e. $\sum rr_j > C$, the target rate is less than the reserved rate.
- ar_i: the achieved bit rate of source s_i. This is the actual average bit rate that source s_i achieved through the simulation and is $\rho(N_{\text{out},i})$.
- sr_i: the actual average bit rate of source s_i, i.e. $\rho(N_{\text{in},i})$.

Figure 4.3 illustrates how the theoretical reserved rate (rr) and target rate (tr) correspond to the observable packet counts. In an ideal environment, the number of packets with DSCP 10 should—when converted to an average rate—equal the reserved rate of the subscriber. Similarly, the number of packets that arrived at the destination should equal the subscriber's target rate, i.e. ar_i should equal tr_i.

After each simulation run the following data were collected: the number of packets generated by each source ($N_{\text{in},i}$); the number of packets per DSCP that arrived at each

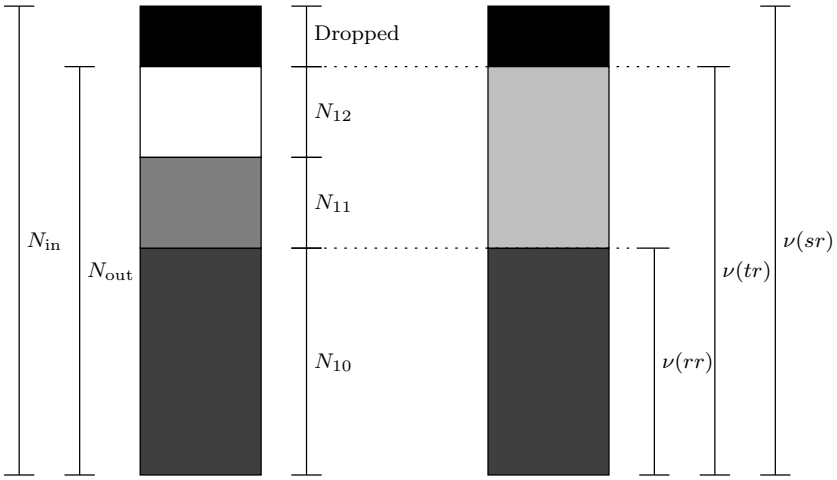


Figure 4.3.: Observed packet counts related to configured rates.

destination ($N_{10,i}$, $N_{11,i}$, $N_{12,i}$); as well as the number of packets of each DSCP that was dropped.

As mentioned earlier, *ns*'s DiffServ module distinguishes between two kinds of drops—early drops and late drops. The *ns* manual (Fall et al., 2003) describes early drops as packets dropped due to RED and late drops are packets dropped due to link overflow. It was found, however, that *ns* reported as late drops, those packets dropped by RED when the current queue size exceeded max_{th}. The *ns* code was subsequently modified to count those packets as early drops. Late drops are therefore only packets dropped because the physical queue's capacity is exceeded. The number of packets of each DSCP that were early drops, as well as the number of packets of each DSCP that were late drops, was counted.

4.6.2. Performance Measures

Since the objective of the study was to determine which traffic conditioners perform better than others in the given environments, a measure for comparing relative performance was needed.

Two performance measures were considered in the study. The one measure relates to what extent the subscriber achieved its target rate. The other measure determines how well the number of DSCP 10 packets corresponds to the reserved rate.

The first measure's relevance is fairly obvious. The provider network should provide the subscriber with a fair share of capacity as agreed upon. The subscriber should therefore be able to achieve its target rate. To determine how well the target rate is

achieved, one divides the average achieved rate by the target rate:

$$\text{TargetRatio}_i = \frac{ar_i}{tr_i}.$$

The second measure requires additional motivation. Recall that the subscriber and provider enter into an agreement regarding the service to be expected from the provider. The provider commits to deliver packets from the subscriber while the offered traffic rate is less than, or equal to, the agreed upon CIR (also referred to as reserved rate). The subscriber is allowed, however, to offer more traffic to the network on the understanding that the traffic exceeding the CIR will receive treatment different from the traffic not exceeding the CIR. For this reason it is important that the correct number of packets are marked with DSCP 10, the correct number being a number that corresponds to the reserved rate. To determine how well the reserved rate was achieved with 'green' packets, one calculates the following ratio:

$$\text{GreenRatio}_i = \frac{\rho(N_{10,i})}{rr_i}.$$

These two performance measures are used in the subsequent chapter to determine whether some traffic conditioners are better than others in the scenarios considered. The closer the observed value of the ratio is to one, the better the traffic conditioner performed.

These results of the simulations and subsequent statistical analysis are presented in the following chapter.

5. Experimental Observations

> Get your facts first and then you can distort them as much as you please.
>
> *Mark Twain*

This chapter contains the results, observations, and statistical analysis of the experiments performed as described in the previous chapter.

Statistical calculations were done using R (R Development Core Team, 2004), a system for statistical computation and graphics.

This chapter is organised as follows. Section 5.1 contains observations regarding the performance of the traffic conditioners based on the target rate performance measure. In Section 5.2 the performance of the conditioners are analysed, based on how well they monitored the reserved rate. The chapter is concluded in Section 5.3 which reports on the results of an additional analysis that was catalysed by observations discussed earlier in the chapter.

5.1. Target Rate

Recall from Chapter 4 that the i^{th} subscriber's target rate is its reserved rate plus its share of the excess capacity. Thus: $tr_i = rr_i + \frac{1}{n}(C - \sum_{j=1}^{n} rr_j)$. This target rate is the average transfer rate that a subscriber should achieve during a simulation run. Also recall that the sources generate traffic at an average rate of approximately double their reserved rates which will cause the network to be congested. To determine how well a subscriber achieved its target rate, a $\text{TargetRatio}_i = \frac{1}{tr_i}\rho(N_{\text{out},i})$ was defined in Subsection 4.6.2. This ratio is used as a performance measure to compare the different traffic conditioners. The closer to one that the observed TargetRatio_i is, the better the traffic conditioner is deemed to have performed. If the ratio is larger than one, then the subscriber achieved an average rate greater than it is allowed. If, under stress conditions, one subscriber obtains a better rate, then another will typically achieve a lower than one ratio, since there is a fixed sized bottleneck link in the network. For some subscribers to win, some have to lose.

In this section the goal is to determine whether the different traffic conditioners yield different observed target ratios for the scenarios under consideration. For each reser-

vation level and traffic model combination, the five traffic conditioners are compared to determine whether they yield differing target ratios.

Let η_k denote the median of the ten times repeated observations TargetRatio_i for $i = 1 \ldots 10$ of traffic conditioner k for a given reservation level and traffic model combination. The null hypothesis can then be stated as H_0: $\eta_1 = \eta_2 = \cdots = \eta_5$ stating that the different traffic conditioners (or treatments in statistical parlance) achieve the same target rate results. The alternative hypothesis is given by H_1: at least one η_k differs from the others.

The Kruskal-Wallis non-parametric test (Steyn et al., 1994) was used to test the hypotheses.[1] The results of the tests are summarised in the Table 5.1. Note that = indicates that the null hypothesis could not be rejected at a 5% confidence level and $\neq$ that the null hypothesis was rejected.

	CBR	POISSON	EXPOO	PAROO	H=0.5	H=0.7	H=0.95
125%	=	=	=	=	=	=	=
90%	=	=	=	=	=	=	=
60%	=	$\neq$	=	=	=	=	=

Table 5.1.: Results of target ratio based comparisons.

From that table it can be seen that in all but one of the cases under consideration there was insufficient evidence to reject the null hypothesis. The only scenario where the null hypothesis was rejected is for Poisson traffic at reservation level 60%. Box plots of the observed target ratios are given in Figure 5.1.

Perusing the plots one finds that, on average, the TSW2CM produced lower target ratios than the other traffic conditioners. Even though this is the case, the differences are fairly small in absolute terms. It therefore seems reasonable to assume that the performance of the different traffic conditioners within the different scenarios have similar performance, at least when viewed from the perspective of how well the target rate was achieved. Within the limits of the current simulation experiment, it seems that, on average, subscribers achieve the same target rate, irrespective of the traffic conditioner chosen.

Although the observed target rate performance of the traffic conditioners is essentially the same, there might be other differences that matter to both the subscriber as well as to the provider. For example, in an extreme case, a subscriber might achieve her target rate even though almost none of the packets are marked as in-profile. This could

[1] It seemed appropriate to use this non-parametric test since (i) its only assumption is a continuous distribution of the underlying population, (ii) the sample size is relatively small, and (iii) no a priori knowledge if the underlying population's statistical distribution was available.

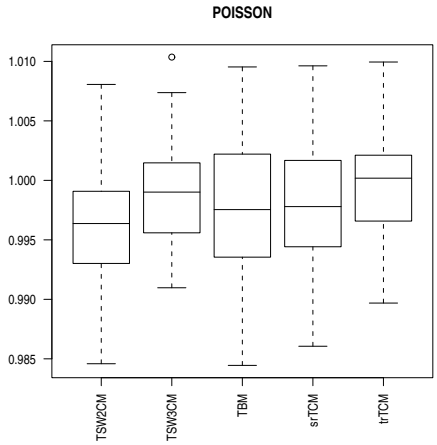


Figure 5.1.: TargetRatios for POISSON traffic at 60% reservation.

potentially have negative cost implications for the subscriber, depending, of course, on exactly how the Service Level Agreement (SLA) was specified. The subscriber might have to pay more for the out-of-profile packets, since she might be penalised for exceeding the agreed upon reserved rate.

From the provider perspective, the same issues are relevant, although the subscriber and provider have opposing commercial interests. When a subscriber achieves her target rate by using more than her reserved rate, then the provider would typically prefer to charge the subscriber differently for delivering the packets that exceeded the reserved rate.

For the above mentioned reasons a second performance measure was used to compare the traffic conditioners.

5.2. Reserved Rate

In this section the traffic conditioners are compared on the basis of how well they monitored the reserved rated of the set of subscribers. Recall from Chapter 4 that a subscriber's reserved rate, rr_i, is the average rate at which the provider commits to providing a service to the subscriber. Naturally, when one talks of an average rate, then the time period over which the average is taken is relevant. In the experiments, the average rate was, as mentioned in Chapter 4, calculated over the entire simulation period of sixty seconds.

Ideally, if a subscriber's average sending rate is less than, or equal to, the reserved rate, then all of the packets originating from said subscriber ought to arrive as green at the

destination. However, there will inevitably be short term variations in sending rate, which are determined by the way in which the source generates traffic. These variations will affect the way in which a traffic conditioner determines whether a particular packet should be marked as in-profile or not. As a consequence, even if the average sending rate taken over the full simulation falls below the reserved rates, these short term variations may nevertheless cause a proportion of the packets to be marked as out-of-profile, so that the ratio of green packet rate transmission to reserved rate is less than it might have been.

The performance of the different traffic conditioners were compared using $\text{GreenRatio}_i = \frac{1}{rr_i}\rho(N_{10,i})$. Similarly to what was done for the target rates, the performance of the traffic conditioners were compared in each of the scenarios under consideration.

The ideal value for the ratio is one. Since the simulated sources generate traffic exceeding their reserved rates, the ratio should not be less than one. Of course, if a source generates traffic at an average rate less than the reserved rate, the ratio will be less than one. The ratio should also not be greater than one since that would mean that the subscriber obtained 'better' service than specified in the service level specification.

Let η_k this time denote the median over ten repetitions of $i = 1 \ldots 10$ of GreenRatio_i for traffic conditioner k in a particular reservation level and traffic model combination. The null hypothesis is again that the traffic conditioners perform the same, i.e. H_1:$\eta_1 = \eta_2 = \cdots = \eta_5$. The null hypothesis was tested against the alternative that at least one η_k differs from the rest using the Kruskal-Wallis test as before. Table 5.2 contains a summary of the results of the tests.

	CBR	POISSON	EXPOO	PAROO	H=0.5	H=0.7	H=0.95
125%	$=$	$=$	$\neq$	$\neq$	$=$	$=$	$=$
90%	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$
60%	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$

Table 5.2.: Results of green ratio based comparisons.

From the table it can be seen that in the majority of the simulated scenarios, at least one traffic conditioner yielded a mean green ratio that was significantly different from the mean green ratio of other traffic conditioners. In fact, it is only in conditions of oversubscription where, for the CBR, Poisson, and self-similar source traffic types, there appears to be no significant difference between the mean green ratio.

Refer to Figure 5.2 on page 63 for a graphical representation of all the observed green ratios in the different scenarios under consideration. The plots are small—the idea is to give a visual impression of the relative performance of the conditioners within a given scenario. Note, however, that the scales of the graphs differ. Thus, one should only make visual comparisons within a plot and not across the different plots. In all plots

the first column represents the spread of the observations for the TSW2CM, column two for the TSW3CM, column three for the TBM, column four for the srTCM, and column five for the trTCM.

Having established that there are indeed differences between the performances of the various traffic conditioners, it seems appropriate to investigate further the nature of the differences. In particular, it is of interest to determine which specific conditioners performed better under the various scenarios. Note that this is not evident in Figure 5.2 on the next page because there is no indication in the plots whether a given conditioner is over- or undershooting the ideal value of 1. To achieve the goal of visually assessing conditioner performance by viewing such plots, the following transformation was done on the observed data: $|1 - \text{GreenRatio}_i|$ for all the observations. Better performance of a conditioner is now determined by how small the value is—the smaller, the better.

Visual inspection of the plots in Figure 5.3 on page 64 indicates that, where there are differences, the performance of both the time-sliding window conditioners well as of the token bucket conditioners appear to cluster around similar behaviour profiles, i.e. the differences are primarily related to time-sliding versus token bucket conditioner models. The respective performances arising from these two cases are therefore considered in greater detail below.

From the plots in Figure 5.3 it can be seen that the time sliding window conditioners perform better than the token bucket ones in the EXPOO or PAROO traffic generation model cases. In the cases where the other traffic generation models were used, the token bucket based conditioners performed better than the time-sliding window ones. These observations are verified as being pair-wise statistically different using statistical tests as provided in Appendix A.

From Figure 5.2 it can be seen that the green ratios produced by the time-sliding window conditioners are usually greater than what is produced by the token bucket based conditioners. For example, in the 60%, EXPOO case the GreenRatios obtained by the time sliding window based conditioners (first two columns) are greater than those of the token bucket based conditioners (the other three columns). The time-sliding window based conditioners are therefore more lenient towards subscribers since they allow more packets to be marked as in-profile, i.e. green.

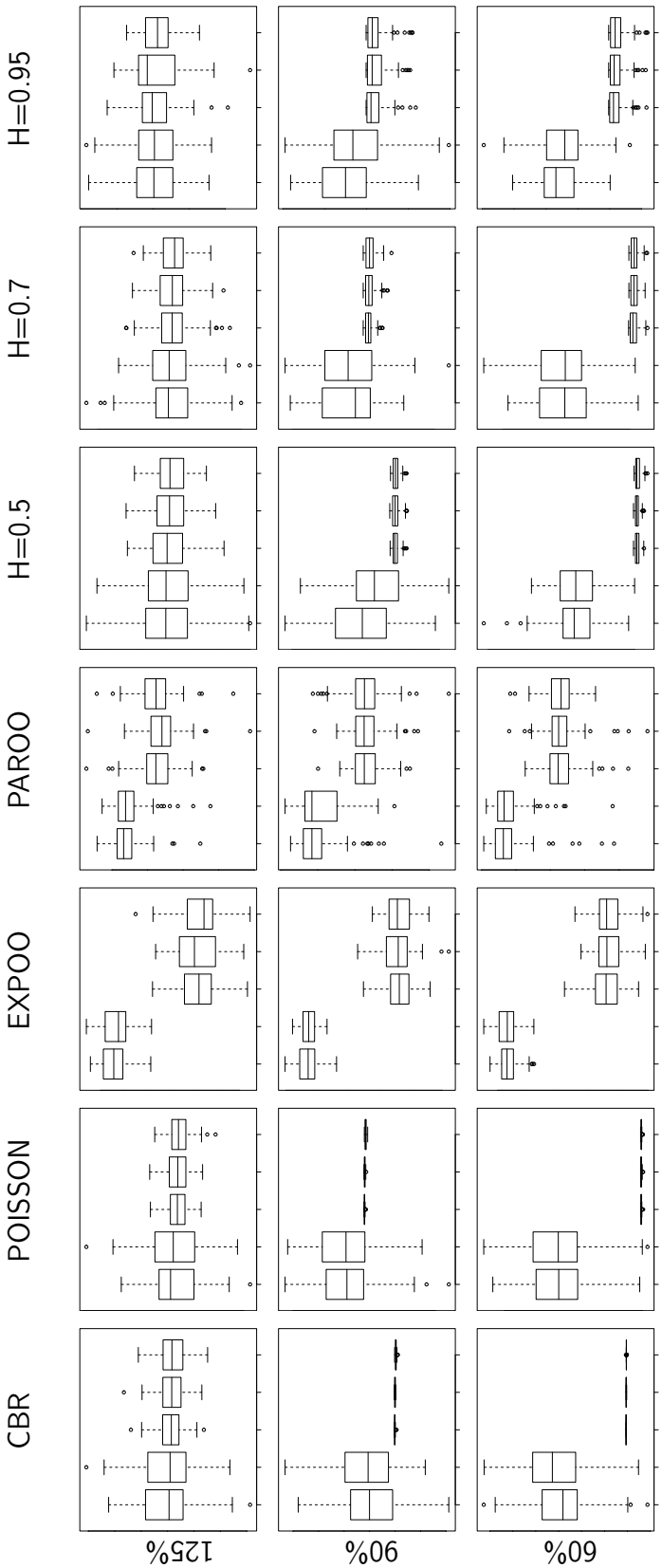


Figure 5.2.: Observed green ratios.

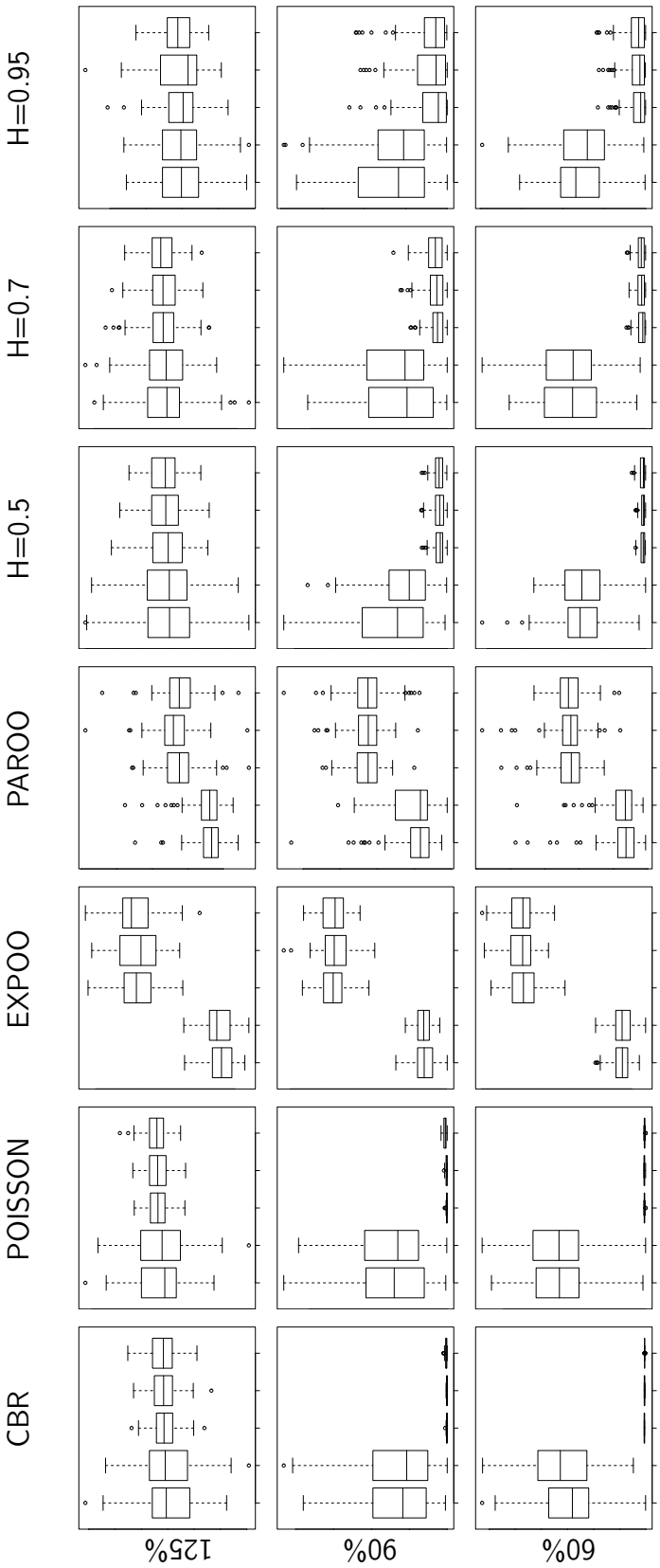


Figure 5.3.: Observed $|1 - \text{GreenRatio}_i|$

5.3. Discussion of Results

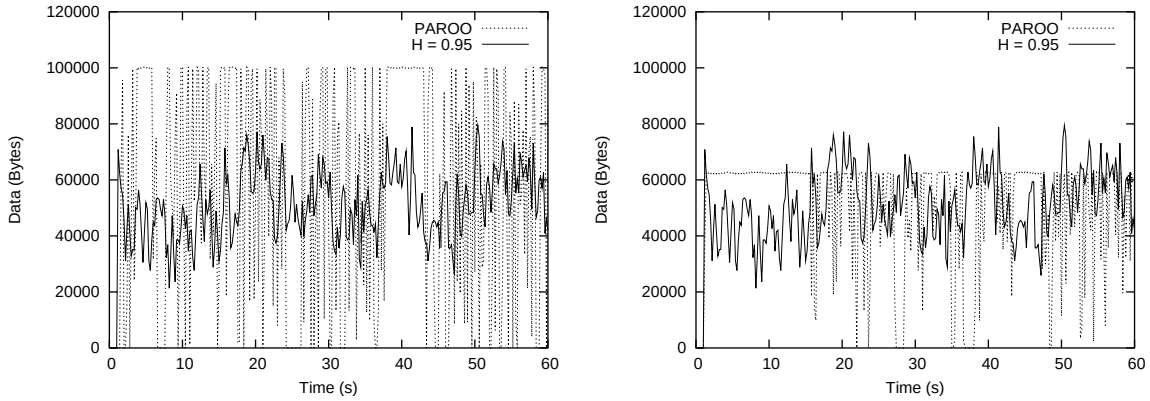
The results presented in the preceding sections suggest that for the very specific scenarios under study, the experiments did not yield significantly different target rate results for the different traffic conditioners. This constitutes prima-facie evidence that the choice of traffic conditioner does not have a significant impact on the extent to which the target rate is achieved. Of course, to gain greater confidence in this assertion, a more widespread study would have to be carried out that considered a range of scenarios not covered by the present simulation.

However, the reserved rate analysis points to a different conclusion. The observations in the preceding section indicated that the two time-sliding window traffic conditioners yielded higher GreenRatio values than did the three token bucket conditioners. This should not be construed to mean that the time-sliding window conditioners are always a better choice. Indeed, it was seen that they are sometimes too lenient, i.e. they result in green ratios that are greater than one. This is potentially to the detriment of the service provider, since it implies a potential loss of revenue in that the client is getting more of a service than was agreed upon per contract.

The reason for the lenience in the time-sliding window conditioners might be due to the smoothing features to estimate the current traffic rate that are inherent in the TSW algorithm (Figure 2.7). In the case of a token bucket conditioner, one would expect that the bucket's size determines the extent to which the short-term fluctuations in the traffic stream can be absorbed: a larger token bucket allows for larger short-term fluctuations in the sending rate before packets are marked as out-of-profile.

To compare the traffic conditioners, two scenario were selected based on the $|1 - \text{GreenRatio}_i|$ metric for further analysis: one in which the time-sliding window conditioners performed better; and another in which the token bucket conditioners performed better. Note in Figure 5.3 that the time-sliding window conditioners performed better in the PAROO case while the token bucket ones were better in the H=0.95 case. It was therefore decided to select the 60%, PAROO and 60%, H=0.95 scenarios respectively, as representative scenarios for further analysis. Also notice from Figure 5.2 that the time-sliding window conditioner achieved higher average GreenRatios in both these cases.

Figure 5.4(a) shows a plot of the traffic pattern generated by one single PAROO source superimposed on the traffic pattern generated by one single H=0.95 source. The graph shows the average number of bytes sent in 0.2 s intervals. From the plot it is clear that the PAROO traffic stream contains more variation than the H=0.95 traffic stream. The fact that the time-sliding window conditioners performed better in the PAROO case (as seen in Figure 5.3), suggests that when the traffic pattern is highly variable, a conditioner based on a time sliding window might be more effective than a token bucket conditioner. Of course, this can only be a tentative conclusion and further study of the



(a) Original (b) Modified

Figure 5.4.: PAROO and H=0.95 traffic patterns at a single source.

matter is required.

To gather additional evidence as to whether the traffic pattern plays a role in the relative performance of the conditioners, an additional experiment was carried out. The 60%, PAROO simulations were repeated with slightly different parameter values for the PAROO traffic generator. Specifically, the parameters were chosen to reduce the short-term variation in the PAROO traffic streams. The PAROO traffic model was configured with a 800 ms average ON period and a 200 ms OFF period. The constant sending rate during the ON period was set to 2.5 Mbps. Figure 5.4(b) shows the resulting modified PAROO traffic stream with the original H=0.95 traffic stream superimposed on it. It is clear from the plot that the modified PAROO traffic stream contains less variation than the original traffic stream.

The performance of the traffic conditioners in the modified case was compared to the original performance. The time-sliding window conditioners still performed better than the token bucket ones. The difference in performance was, however, smaller. This indicates not only that the traffic pattern does indeed affect the performance of the various traffic conditioners, but provides additional evidence to suggest that time sliding window technology is perhaps better than token bucket technology in conditions of high variability.

The foregoing conclusions were predicated on token bucket experiments that had a given bucket size of $5760 = 10 \times 576$ bytes. It was decided to determine whether the token bucket traffic conditioners' performance would improve if the bucket size was modified. Consequently, the following simulation scenarios were repeated using a range of token bucket sizes.

The experiments were limited to 60% reservation level with PAROO and H=0.95 traffic using the TBM traffic conditioner. The bucket size started at 2880 bytes (5 packets)

and was doubled up to 1 474 560 bytes (2560 packets). The case of a bucket size equal to the size of one packet, 576 bytes, was also simulated. Each simulation scenario was again repeated ten times. Figure 5.5 contains graphical representations of the observed green ratios. Note that the x-axis is in units of packet size, where 1 packet is 576 bytes.

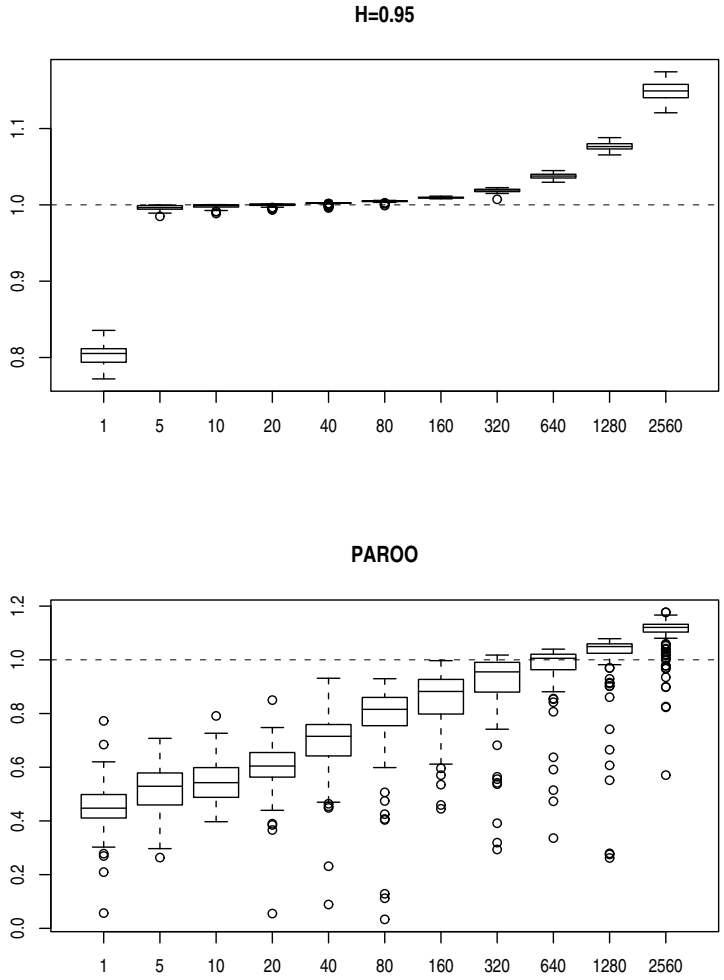


Figure 5.5.: Green performance for different bucket sizes.

From Figure 5.5 it can clearly be seen that the bucket size does, as expected, influence the green ratio performance of the token bucket traffic conditioner. However, it affects the performance in the two traffic patterns differently. From the plots it can be seen that a bucket size of approximately 640×576 bytes is required in the PAROO case to achieve a green ratio of one, while this ratio is already attained in the H=0.95 model

when the bucket size is very small—approximately 5×576 bytes.

In fact, in the H=0.95 traffic case the performance is not very sensitive to changes in the bucket size for sizes less than 320×576 bytes.

It can also be seen that the token bucket based conditioners can be as lenient as the time-sliding window based conditioners. This is demonstrated in the simulation scenarios described below.

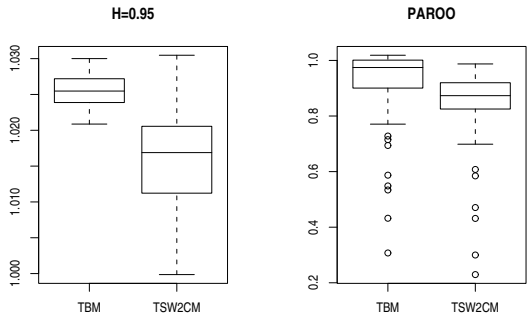


Figure 5.6.: Green ratios for TBM and TSW2CM.

Figure 5.6 contains box plots of the green ratios observed for both PAROO and H=95% traffic patterns, when conditioned by both a TBM conditioner with a 250 000 byte bucket (i.e. approximately 434 packets) and by a TSW2CM conditioner. The rationale for choosing this bucket size is that it corresponds to the number of bytes that is generated in one second, assuming a traffic generation rate of 2 Mbps—the average rate at which sources generated traffic in the simulations. One second is the length of the smoothing window in the time sliding window algorithm used in the TSW2CM.

From the plot it can be seen that the TBM achieved, on average, greater green ratios than the TSW2CM. This is true for both the PAROO and H=95% traffic patterns. It is therefore possible to obtain similar levels of leniency when using a token bucket based conditioner than time sliding window conditioner

6. Related Work & Conclusions

> Nothing is my last word on anything.
>
> *Henry James*

This chapter concludes the dissertation. The major findings are presented here as well as related work and potential future research possibilities.

The chapter is organised as follows. Section 6.1 lists the lessons learnt from the simulation study that was presented in previous chapters. Section 6.2 presents work relating to the area studied, while Section 6.3 suggests possible avenues for future work. The dissertation is concluded in Section 6.4 with a few closing comments.

6.1. Lessons Learnt

This section lists lessons learnt through conducting the simulation study described in the previous chapters. The lessons are based on the observations made whilst utilising a fairly limited set of scenarios. One should be wary about generalising the results to include other situations.

- When performance was measured in terms of the TargetRatio metric, the various traffic conditioners were seen to perform similarly under the different scenarios. There was no convincing evidence to suggest that the choice of traffic conditioner affects a source's achieved rate. Therefore, if a network designer is only interested in the relative target rate performance of the traffic conditioners, she is, based on the evidence at hand, free to choose any one of the studied traffic conditioners since they performed similarly.

- Traffic conditioner performance differed when compared in terms of the GreenRatio metric. The experimental results suggest that the traffic patterns do indeed play a role in how well the various traffic conditioners perform. A network designer should be aware of this fact and should choose an appropriate traffic conditioner. In general, the various versions of token bucket conditioners performed similarly to one another in a specific traffic pattern context. The same was true for the various versions of time sliding window conditioners.

- However, within the context of the initial experiments, the time sliding window conditioners tended to achieve more easily a green ratio greater than one than the token bucket conditioners. Later experiments suggested that the same ratios could be attained by token bucket conditioners if the bucket size is made large enough. The attainment of such ratios are an indication that the subscriber is being treated too leniently and that the network is possibly being deployed non-optimally from the provider's point of view. The network provider should therefore be cognisant of this observed difference between conditioners and should monitor the green ratios to ensure that appropriate levels are attained.

- Chapter 5 indicates that the performance of the traffic conditioners be can influenced by manipulating parameter values such as the token bucket size. It is therefore important to configure a traffic conditioner according to the service level specification, while taking the expected traffic pattern into consideration. Some parameters such as the CIR are easily determined from the service level specification. Other parameters, such as a token bucket's size, are less easily determined. Although this study did not explore in detail how to determine these values, it clearly showed that the token bucket size influences how well the traffic conditioner performs based on GreenRatio as metric.

- The provider network should be engineered (sized) to accommodate the contracted rates if the subscribers usually exceed, or equal, their reserved rates. If subscribers usually under-utilise the network, the subscriber could consider over-subscribing the network. This issue was not explored in detail in this study.

- From the simulation results it was observed that, in regard to GreenRatios, the two time-sliding window based conditioners performed in the same manner and that the three token bucket based conditioners also performed similarly. From the models discussed in Chapter 3, this observation is to be expected when the traffic conditioners are configured with the same parameter values.

  Thus, the time-sliding window conditioners will behave similarly when configured with the same CIR. From the model, a packet will be assigned a colour other than 'green' when the the current estimated rate exceeds the CIR.

  Similarly, the token bucket based conditioners will behave in very much the same manner when configured with the same CIR and 'green' token bucket size. Recall from Chapter 3 that all the token bucket based conditioners contain a c:BUCKET process; it is this token bucket that is referred to in the previous sentence. Since this token bucket is used to determine whether a packet is assigned the colour 'green' or not, the conditioners will behave similarly, based on the GreenRatio metric.

- Although the FSP models and associated structure diagrams were non-trivial to derive, the result is a concise and elegant behavioural description of the various

conditioners. The abstract representations highlight the similarities and differences between the traffic conditioners. The models aid the researcher in obtaining insight into the operation of the various traffic conditioners, as well as how they relate to each other. The models also provide a convenient way in which to communicate the operation of the traffic conditioners. For these reasons, the exercise of deriving these models is considered worthwhile even though not absolutely necessary for the simulation study.

- Simulation can be useful in uncovering likely conditioner behaviour in response to various traffic patterns. The *ns-2* simulator appeared to be adequate for the purposes of the study. However, due to the large number of possible parameters, it is difficult to know, or decide, which parameters to vary and which to keep constant. Further, when one has decided on which parameters to vary, it is also difficult to decide on the specific values to use for the fixed, as well as the varied, parameters.

6.2. Related Work

This section contains high-level summaries of previous work relating to the work described in this dissertation. This dissertation's research focused on traffic conditioners that are described in RFCs and compared them under various traffic pattern conditions. Other research typically studied a novel traffic conditioner in a particular environment.

Earlier work (Ibanez and Nichols, 1998; Yeom and Reddy, 1999b,a; Goyal et al., 1999) has studied the performance and usefulness of the Diffserv assured forwarding per-hop behaviour.

Ibanez and Nichols (1998) used *ns-2* to simulate so-called infinite File Transfer Protocol (FTP) traffic sources using TCP as transport layer protocol. The Diffserv environment was implemented with a token bucket marker and RED as the queue management scheme. The authors decided against a time-sliding window conditioner since the token bucket was deemed to be more suitable for the typically bursty TCP traffic. The authors investigated the performance of the sources against their reserved rates under various round-trip times and types of competing nodes. They concluded that the assured service lacks consistent rate guarantees and that the benefit of introducing it is not quantifiable.

Yeom and Reddy (1999b,a) performed similar studies and also proposed strategies for improving TCP throughput. The authors also used *ns-2* to generate FTP traffic over TCP. A time-sliding window algorithm was used to determine whether packets are in- or out-of-profile. This study focussed on the impact of aggregated sources on the service provided by the Diffserv network. The authors showed that aggregation reduces the impact of round-trip time variation on throughput guarantees. In addition, the authors

proposed two new marking algorithms to improve the fairness among flows within an aggregation.

Goyal et al. (1999) studied the number of drop precedence levels that is useful in assured forwarding. They also used *ns-2* to simulate the Diffserv environment. They simulated aggregate congestion sensitive (TCP) traffic sources and one congestion insensitive (UDP) source, the traffic conditioning was done by token buckets and RED queues were used for the packets. The authors found that, when there is excess capacity available in the network, three drop precedence levels are required to distinguish between congestion sensitive and congestion insensitive traffic. Congestion insensitive traffic tends to be 'greedy' and consumes more of the excess capacity than the more well-behaved congestion sensitive sources, leading to unfairness. However, when the network is operating close to capacity, three levels are redundant since there is not much excess capacity to share. This observation seems consistent with the results of the present study.

More recent studies (Herrería-Alonso et al., 2004; El-Gendy and Shin, 2003; Cano et al., 2003; Kumar et al., 2002; Su and Atiquzzaman, 2001) propose alternative marking mechanisms or traffic conditioners for use in Diffserv domains. Herrería-Alonso et al. (2004) studied the performance of the so-called aggregate flow control (AFC) mechanism. The aim of the mechanism is to improve fairness among flows within an aggregate. AFC works by creating control TCP connections between two network edges; these connections inject control packets into the network to detect congestion; the aggregate associated with the control TCP connection is then throttled based on the connection's dropped control packets. The authors conducted a simulation study with AFC added to Diffserv, using a TSW3CM and RED, and found that AFC improves fairness in assured forwarding.

El-Gendy and Shin (2003) evaluated the performance of a packet marking algorithm called equation-based marking (EBM). They found it to improve on the fairness experienced by TCP flows when compared to other markers. EMB works similarly to TCP, but on the level of packet marking. It estimates the current round-trip time and current loss rate and uses an analytical method to calculate the target loss probabilities from the estimates and the target rates (CIR and PIR). The target loss probabilities are used by a marking function to assign colours to packets.

Cano et al. (2003) study a new traffic conditioner called the counters-based modified (CBM) traffic conditioner. The conditioner was born from the idea that if each source injects the same number of out-of-profile packets, the sources will obtain equal shares of the excess bandwidth. The CBM conditioner keeps count of the number of out-of-profile packets between two consecutive in-profile packets. When the counter is below a minimum threshold, the out-of-profile packet is injected into the network; when the counter is greater than a maximum threshold the packet is dropped; and when the counter is between the thresholds, the packet is dropped with a probability based on

the contracted rate. Their analysis found it to be a feasible option for use in assured forwarding since it guarantees target rates and shares excess capacity fairly.

Kumar et al. (2002) present a TCP-friendly memory-based three colour marker (MBTCM). The marker takes the previously determined average rate into consideration when determining the marking probability, whence the memory-based label. The marker is presented as simple and insensitive to parameters. The authors report on a simulation-based comparison of the MBTCM to the TSW3CM. They suggest the deployment of the MBTCM for Diffserv edge routers.

Su and Atiquzzaman (2001) propose an improved time-sliding window three colour marker. The intention of the improved marker is to achieve greater fairness in sharing excess bandwidth among aggregates. The basic idea behind the ItswTCM is to allow an aggregate to inject 'yellow' packets at a rate proportional to the source's CIR. The authors report on a simulation study in which the ItswTCM is compared to srTCM, trTCM, and TSW3CM; they found that their ItswTCM performs better (in terms of fairness) than the other three traffic conditioners.

Another more recent area of research is the dynamic configuration of network resources within a Diffserv domain (Liao and Campbell, 2004; Krief, 2004; Mahajan et al., 2004). The idea is to dynamically, and also automatically, reallocate resources based on the current requirements and state of the network. Liao and Campbell (2004) propose an approach to dynamically provision interior and core nodes. A node provisioning algorithm predicts service level violations and automatically adjusts the service weights of weighted fair queuing schedulers at core nodes. A core provisioning algorithm adjusts core bandwidth in response to signals from node provisioning modules. The authors also provide results of a simulation study that suggests that dynamic provisioning is superior to static provisioning.

Krief (2004) describes an architecture for self-aware management of QoS enabled IP networks by using policy-based management and multi-agent systems. The architecture includes three mediation levels, each with monitoring functions to be able to adjust its behaviour to the environment under its control. The access mediator mediates between the end-user and service provider; the service mediator informs the access mediator of available services and it supervises the management of access to services via the underlying resource mediator. The resource mediator is associated with the underlying network. It is responsible for determining which policy rules to apply to network elements in order to satisfy the service mediator.

Mahajan et al. (2004) present a mechanism for active resource management in a Diffserv environment. The scheme makes use of a so-called bandwidth broker agent which keeps a database of flow parameters such as reservations, allocations, and DSCP mappings. According to these parameters, the broker agent makes a reservation and assigns a DSCP for the service provided to a client. When the broker agent notices, through the meter component in a traffic conditioner, that a client is using less bandwidth than

reserved, it re-allocates the excess bandwidth without degrading service. The authors present simulation results and find that network resources are used more efficiently with active resource management added to the Diffserv environment.

Also of importance, especially to network providers, is how to price for differentiated services. Pricing affects not only the expected revenue of a network provider, but also the way subscribers interact with the network. Soursos et al. (2003); Paschalidis and Liu (2002); O'Donnell and Sethu (2002); Stiller et al. (2001); Marbach (2001) report on some pricing strategies and find that pricing does influence the interaction of subscribers with the network.

The foregoing indicates that, though there has been ongoing research into various aspects of Differentiated Services, the relative behaviour of the different standard traffic conditioners in the presence of different traffic patterns has not received much attention. It is hoped that the present study will stimulate further research in this context. The next section suggests some possible areas to advance this research.

6.3. Future Work

Since performance will be affected by the choice of RED parameters, one should also investigate the sensitivity of performance to these parameters. The RED queues do the actual dropping. In this study the actual drop statistics were not studied.

A range of different traffic models were used in this study. These correspond to the more commonly used statistically based models that are frequently encountered in network traffic simulation studies, and include self-similar traffic patterns, often considered to be realistic models of network traffic. However, it would be of interest to extend the study to the use of traffic models that more realistically reflect specific applications such as, for example, VoD (Koucheryavy et al., 2003).

In this study only the 'green' packets were studied—the 'yellow' and 'red' packet data were not explicitly analysed. The behaviour of the various traffic conditioners in relation to these latter colours might exhibit more differences. It would also be of interest to consider the impact of various pricing strategies for different coloured packets and to compare the 'costs' that different traffic conditioners would assign to the same traffic pattern. The aim would be to discover whether some conditioners generate more revenue for the provider whilst using the same parameter values as others.

The level of subscription to a network is also very important to a provider. If the provider can commit more of its infrastructure without investing more in it, it will be able to increase its profitability. The provider should be able to honour its commitments with as little as possible infrastructure.

Creating and comparing analytical models of traffic conditioner performance would be advantageous to network designers. If one could calculate the requirements analytically, it would be easy to design a near-optimal network. This is, however, no simple task.

6.4. Closing Remarks

It should be clear that there are just too many degrees of freedom in such a simulation study to be able to explore all possible avenues in a one person-year project such as this. The previous section thus suggests possible ways of broadening the work.

Although the results presented in Chapter 5 are not spectacular, they do suggest—as alluded to in Section 6.1—that setting parameters for traffic conditioners appropriately is important. The results emphasise that network designers should take cognisance not only of the fact that traffic patterns and traffic conditioner parameter values affect the performance of the network; but also of the fact that these may ultimately impact on both provider revenue and on subscriber satisfaction. The methodology employed in this study—simulation of traffic sources in a simple network topology to study the impact of various traffic conditioners, characterised by various parameters—would appear to be a relatively inexpensive and practical approach to grappling with the real-world problem of arriving at an optimal network design.

Glossary

Admission control the decision process of whether to accept a request for resources.

Behaviour Aggregate (BA) a collection of packets with the same codepoint crossing a link in a particular direction.

BA classifier a classifier that selects packets based only on the contents of the DS field.

Boundary link a link connecting the edge nodes of two domains.

Classification the process of sorting packets based on the content of packet headers according to defined rules.

Classifier an entity which selects packets based on the content of packet headers according to defined rules.

DS behaviour aggregate a collection of packets with the same DS code point crossing a link in a particular direction.

DS boundary node a DS node that connects one DS domain to a node either in another DS domain or in a domain that is not DS-capable.

DS-capable capable of implementing differentiated services as described in RFC2475

DS codepoint a value which is encoded in the DSField, and which each DS Node must use to select the PHB which is to be experienced by each packet it forwards.

DS-compliant enabled to support differentiated services and functions and behaviour as defined in RFC 2474 and RFC 2475 and other Diffserv documents.

DS domain a DS-capable domain; a contiguous set of nodes which operate with a common set of service provisioning policies and PHB definitions.

DS egress node a DS boundary node in its role in handling traffic as it leaves a DS domain.

DS ingress node a DS boundary node in its role in handling traffic as it enters a DS domain.

DS interior node a DS node that is not a DS boundary node.

DS field the six most significant bits of the (former) IPv4 TOS octet or the (former) IPv6 Traffic Class octet.

DS node a DS-compliant node.

DS region a set of contiguous DS domains which can offer differentiated services over paths across those DS domains.

Downstream DS domain the DS domain downstream of traffic flow on the boundary link.

Dropper a device that performs dropping.

Dropping the process of discarding packets based on the specified rules; policing.

Global Synchronisation When congestion occurs and a queue fills up, packets from multiple TCP connections might be dropped. The affected TCP connections will enter slow-start state at the same time. This causes a significant reduction in network traffic so that the network is under-utilised for that time. Since the TCP connections entered slow-start at the same time, they will increase window sizes at approxamately the same rate resulting in congestion and another cycle of synchronised slow-start.

Internet Architecture Board (IAB) A small group of people who set policy and direction for TCP/IP and the global Internet. The IAB was formarly known as the *Internet Activities Board*.

Internet Engineering Task Force (IETF) A group of people under the IAB who work on the design and engineering of TCP/IP and the global Internet. The IETF is divided into areas and areas are further divided into workin groups.

Legacy node a node which implements IPv4 Precedence as defined in RFC791,1812 but which is otherwise DS-compliant.

Management Information Base (MIB) The set of variables (database) that a system running an SNMP agent maintains. Managers can fetch and store into these variables.

Marker a device that performs marking.

Marking the process of setting the DS codepoint in the packet based on defined rules; pre-marking, re-marking.

Mechanism a specific algorithm or operation (e.g., queuing discipline) that is implemented in a node to realise a set of one or more per hop behaviours.

Meter a device that performs metering.

Metering the process of measuring the temporal properties (e.g., rate) of a traffic stream selected by a classifier. The instantaneous state of this process may be used to affect the operation of the marker, shaper, or dropper, and/or may be used for accounting and measurement purposes.

Micro-flow a single instance of an application-to-application flow of packets which is identified by source address, source port, destination address, destination port, and protocol id.

MF Classifier a MF classifier which selects packets based on the content of some arbitrary number of header fields; typically some combination of source address, destination address, DS field, protocol ID, source port, and destination port.

Ordered Aggregate (OA) a set of BAs that share an ordering constraint. The set of PHBs that are applied to this set of BAs constitutes a PHB scheduling class.

Per-Domain Behaviour (PDB) the expected treatment that an identifiable or target group of packets will receive from 'edge-to-edge' of a DS domain. A particular PHB (or, if applicable, list of PHBs) and traffic conditioning requirements are associated with each PDB.

Per-Hop Behaviour (PHB) the externally observable forwarding behaviour applied at a DS-compliant node to a DS behaviour aggregate.

PHB group a set of one or more PHBs that can only be meaningfully specified and implemented simultaneously, due to a common constraint applying to all PHBs in the set such as a queue servicing or queue management policy. A PHB group provides a service building block that allows a set of related forwarding behaviours to be specifies together (e.g., four dropping priorities). A single PHB is a special case of a PHB group.

PHB Scheduling Class A PHB group for which a common constraint is that, ordering of at least those packets belonging to the same micro-flow must be preserved.

Policing the process of discarding packets (by a dropper) within a traffic stream in accordance with the state of a corresponding meter enforcing a traffic profile.

Pre-mark to set the DS codepoint of a packet prior to entry into a downstream DS domain.

Provider DS domain the DS-capable provider of services to a source domain.

Queue management controlling the length of packet queues by dropping packets when necessary or appropriate.

Re-mark to change the DS codepoint of a packet, usually performed by a marker in accordance with a TCA.

Scheduling the process of deciding which packet to send first in a system with multiple queues.

Service the overall treatment of a defined subset of a customers traffic within a DS domain or end-to-end.

Service Level Agreement (SLA) a service contract between a customer and a service provider that specifies the forwarding service a customer should receive. A customer may be a user organisation (source domain) or another DS domain (upstream domain). A SLA may include traffic condition rules which constitute a TCA in while or in part.

Service Level Specification (SLS) a set of parameters and their values which together define the service offered to a traffic stream by a DS domain.

Service Provisioning Policy a policy which defines how traffic conditioners are configured on DS boundary nodes and how traffic streams are mapped to DS behaviour aggregates to achieve a range of services.

Shaper a device that performs shaping.

Shaping the process of delaying packets within a traffic stream to cause it to conform to some defined traffic profile.

Simple Network Management Protocol (SNMP) A protocol used to manage devices such as hosts, routers, and printers.

Slow-start A congestion avoidance scheme in TCP in which TCP increases its initially small window size as acknowledgements arrive.

Source domain a domain which contains the node(s) originating the traffic receiving a particular service.

Traffic Aggregate a collection of packets with a codepoint that maps to the same PHB, usually in a DS domain or some subset of a DS domain.

Traffic conditioner an entity which performs traffic conditioning functions and which may contain meters, markers, droppers, and shapers. Traffic conditioners are typically deployed in DS boundary nodes only. A traffic conditioner may remark a traffic stream or may discard or shape packets to alter the temporal characteristics of the stream and bring it into compliance with a traffic profile.

Traffic conditioning control functions to enforce rules specified in a TCA, including metering, marking, shaping, and policing.

Traffic Conditioning Agreement (TCA) an agreement specifying classifier rules and ant corresponding traffic profiles and metering, marking, discarding and/or shaping rules which are to apply to the traffic streams selected by the classifier. A TCA encompasses all of the traffic conditioning rules explicitly specified within the SLA along with all of the rules implicit from the relevant service requirements and/or from the DS domain's service provisioning policy.

Traffic Conditioning Specification (TCS) a set of parameters and their values which together specify a set of classifier rules and a traffic profile. A TCS is an integral element of an SLS.

Traffic profile a description of the temporal properties of a traffic stream such as rate and burst size.

Traffic stream an administratively significant set of one of more micro-flows which traverse a path segment. A traffic stream may consist of the set of active micro-flows which are selected by a particular classifier.

Traffic trunk An aggregation of flows with the same service class that can be put into an MPLS label-switched path.

Upstream DS domain the DS domain upstream of traffic flow on a boundary link.

Bibliography

There are no answers, only cross-references.

Weiner's Law of Libraries

P. Abry, P. Flandrin, M. Taqqu, and D. Veitch. Self-similarity and long-range dependence through the wavelet lens. In P. Doukhan, G. Oppenheim, and M. Taqqu, editors, *Theory and Applications of Long-range Dependence*. Birkhäuser, 2000.

G. Armitage, B. Carpenter, A. Casati, J. Crowcroft, J. Halpern, B. Kumar, and J. Schinizlein. A delay bound alternative revision of RFC 2598. *RFC 3248*, Mar 2002.

D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. Overview and principles of internet traffic engineering. *RFC 3272*, May 2002.

J. Bennet, K. Benson, A. Charny, W. Courtney, and J.-Y. LeBoudec. Delay jitter bounds and packet scale rate guarantee for expedited forwarding. In *Proceedings of IEEE INFOCOM 2001*, volume 3, pages 1502–1509, Apr 2001.

Y. Bernet, S. Blake, D. Grossman, and A. Smith. An informal management model for Diffserv routers. *RFC 3290*, May 2002.

S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. *RFC 2475*, Dec 1998.

M. S. Blumenthal and D. D. Clark. Rethinking the design of the internet: the end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1):70–109, 2001.

R. Braden, D. D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview. *RFC 1633*, Jun 1994.

R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation protocol (RSVP) – version 1 functional specification. *RFC 2205*, Sep 1997.

L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

Z. Brzeźniak and T. Zastawniak. *Basic Stochastic Processes: a course through exercises*. Springer, 1999.

M.-D. Cano, F. Cerdan, J. Garcia-Haro, and J. Malgosa-Sanahuja. Performance analysis of the counters-based modified traffic conditioner in a diffserv network. In *Proceedings of ISCC'03*, pages 305–311, 2003.

B. Carpenter and K. Nichols. Differentiated services in the internet. *Proceedings of the IEEE*, 90(9):1479–1494, Sep 2002.

A. Charny, F. Baker, B. Davie, J. Bennett, K. Benson, J. L. Boudec, A. Chiu, W. Courtney, S. Davari, V. Firoiu, C. Klamanek, K. Ramakrishnan, and D. Stiliadis. Supplemental information for the new definition of the EF PHB (expedited forwarding per-hop behavior). *RFC 3247*, Mar 2002.

D. D. Clark. The design philosophy of the DARPA Internet protocols. In *Proceedings of ACM SIGCOMM '88*, pages 106–114, Aug 1988.

D. D. Clark and W. Fang. Explicit allocation of best effort packet delivery service. *IEEE/ACM Transactions on Networking*, 6(4):362–373, Aug 1998.

D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: defining tomorrow's internet. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 347–356. ACM Press, 2002.

D. E. Comer. *Internetworking with TCP/IP: Principles, protocols, and architectures*, volume 1. Prentice Hall, 4$^{\text{th}}$ edition, 2000.

M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, Dec 1997.

B. Davie, A. Charny, J. Bennett, K. Benson, J. L. Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An expedited forwarding PHB (Per-Hop Behavior). *RFC 3246*, Mar 2002.

W. M. Eddy and M. Allman. Comparison of RED's byte and packet modes. *Computer Networks*, 42(3):261–280, Jun 2003.

M. A. El-Gendy and K. G. Shin. Assured forwarding fairness using equation-based packet marking and packet seperation. *Computer Networks*, 41(4):435–450, Mar 2003.

A. Erramilli, M. Roughan, D. Veitch, and W. Willinger. Self-similar traffic and network dynamics. *Proceedings of the IEEE*, 90(5):800–819, May 2002.

K. Fall, K. Varadhan, and the VINT project. The ns manual, 2003.

W. Fang and N. Seddigh. A time sliding window three colour marker (TSWTCM). *RFC 2859*, Jun 2000.

V. Firoiu, J.-Y. Le Boudec, D. Towsley, and Z.-L. Zhang. Theories and models for Internet quality of service. *Proceedings of the IEEE*, 90(9):1565–1591, Sep 2002.

S. Floyd. Recommendation on using the "gentle_" variant of RED, Mar 2000. URL `http://www.icir.org/floyd/red/gentle.html`.

S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug 1993.

S. Floyd and E. Kohler. Internet research needs better models. *ACM SIGCOMM Computer Communications Review*, 33(1):29–34, 2003.

S. Floyd and V. Paxson. Difficulties in simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, Aug 2001.

M. Garrett and W. Willinger. Analysis, modeling and generation of self-similar VBR video traffic. In *ACM SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 269–280, 1994.

M. Goyal, A. Durresi, P. Misra, C. Liu, and R. Jain. Effect of number of drop precedences in assured forwarding. In *Proceedings of GLOBECOM'99*, pages 188–193, 1999.

M. Grossglauser and J. Bolot. On the relevance of long range dependence in network traffic. *IEEE/ACM Transactions on Networking*, 7(5):629–640, Oct 1999.

S. Harris. The Tao of IETF - A Novice's Guide to the Internet Engineering Task Force. *RFC 3160*, Aug 2001.

J. Heinanen and R. Guerin. A single rate three color marker. *RFC 2697*, Sep 1999a.

J. Heinanen and R. Guerin. A two rate three color marker. *RFC 2698*, Sep 1999b.

J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured forwarding PHB group. *RFC 2597*, Jun 1999.

S. Herrería-Alonso, A. Suáres-González, M. Fernández-Veiga, R. F. Rodríguez-Rubio, and C. López-García. Improving aggregate flow control in differentiated services networks. *Computer Networks*, 44(4):499–512, Mar 2004.

S. Herzog. RSVP extensions for policy control. *RFC 2750*, Jan 2000.

J. Ibanez and K. Nichols. Preliminary simulation evaluation of an assured service. *IETF Draft*, Aug 1998. Work in Progress.

V. Jacobson, K. Nichols, and K. Poduri. An expedited forwarding PHB. *RFC 2598*, Jun 1999.

H.-D. Jeong, D. McNickle, and K. Pawlikowski. Fast self-similar teletraffic generation based on FGN and wavelets. In *Proceedings of ICON '99*, pages 75–82. IEEE, 1999a.

H.-D. J. Jeong, D. McNickle, and K. Pawlikowski. A comparative study of three self-similar teletraffic generators. In *Proceedings of ESM'99*, volume 1, pages 356–362, 1999b.

R. Kalden and S. Ibrahim. Searching for self-similarity in GPRS. In *Proceedings of PAM2004*, pages 83–92, Antibes Juan-les-Pins, France, April 2004.

Y. Koucheryavy, D. Moltchanov, and J. Harju. A top-down approach to VoD traffic transmission over diffserv domain using AF PHB class. In *Proceedings of ICC'03*, pages 243–249. IEEE, May 2003.

F. Krief. Self-aware management of IP networks with QoS guarantees. *International Journal of Network Management*, 14(5):351–364, Sep 2004.

K. R. Kumar, A. Ananda, and L. Kacob. TCP-friendly traffic conditioning in diffserv networks: a memory-based approach. *Computer Networks*, 38(6):731–743, Apr 2002.

W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transaction on Networking*, 2(1): 1–15, Feb 1994.

R. R.-F. Liao and A. T. Campbell. Dynamic core provisioning for quantative differentiated services. *IEEE/ACM Transactions on Networking*, 12(3):429–442, Jun 2004.

J. Magee and J. Kramer. *Concurrency: state models & Java programs*. Wiley, 1999.

M. Mahajan, A. Ramanathan, and M. Parashar. Active resource management for the differentiated services environment. *International Journal of Network Management*, 14(3):149–165, May 2004.

P. Marbach. Pricing differentiated services networks: bursty traffic. In *Proceedings of INFOCOM 2001*, pages 650–658. IEEE, 2001.

S. McCanne and S. Floyd. ns Network Simulator, Feb 2003. URL `http://www.isi.edu/nsnam/ns/`.

K. Nichols and B. Carpenter. Definition of differentiated services per domain behaviors and rules for their specification. *RFC 3086*, Apr 2001.

K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS Field) in the IPv4 and IPv6 headers. *RFC 2474*, Dec 1998.

K. Nichols, V. Jacobson, and K. Poduri. A per-domain behavior for circuit emulation in IP networks. *ACM SIGCOMM Computer Communications Review*, 34(2):71–83, Apr 2004.

Nua Ltd. Nua internet how many online, 2003. URL `http://www.nua.ie/surveys/how_many_online/`. LAST VIEWED: Nov 2004.

A. O'Donnell and H. Sethu. A novel, practical pricing strategy for congestion control and differentiated services. In *Proceedings of IEEE ICC 2002*, volume 2, pages 986–990, 2002.

K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proceedings of the International Conference on Network Protocols*, pages 171–180, Oct 1996.

I. C. Paschalidis and Y. Liu. Pricing in multiservice loss networks: Static pricing, asymptotoc optimality, and demand substitution effects. *IEEE/ACM Transactions on Networking*, 10(3):425–438, Jun 2002.

V. Paxson. Fast, approximate synthesis of fractional Gaussian noise for generating self-similar network traffic. *ACM SIGCOMM Computer Communications Review*, 27(5):5–18, Oct 1997.

V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions of Networking*, 3(3):226–244, Jun 1995.

J. Postel. User datagram protocol. *RFC 768*, Aug 1980.

J. Postel. Transmission control protocol. *RFC 793*, Sep 1981.

R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. URL `http://www.R-project.org`. 3-900051-07-0.

K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN) to IP. *RFC 3168*, Sep 2001.

E. Rosen, D. T. G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, and A. Conta. MPLS label stack encoding. *RFC 3032*, Jan 2001a.

E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. *RFC 3031*, Jan 2001b.

V. Rosolen, O. Bonaventure, and G. Leduc. A RED discard strategy for ATM networks and its performance evaluation with TCP/IP traffic. *ACM SIGCOMM Computer Communications Review*, 29(3):23–43, Jul 1999.

J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

P. Salvador, A. Pacheco, and R. Valadas. Modeling IP traffic: joint characterization of packet arrivals and packet sizes using BMAPs. *Computer Networks*, 44(3):335–352, 2004.

S. Schneider. *Concurrent and Real-time Systems: The CSP Approach*. Wiley, 2000.

M. R. Schroeder. *Fractals, Chaos, Power Laws: Minutes from an infinite paradise*. W. H. Freeman and Co., 1990.

S. Soursos, C. Courcounetis, and G. C. Polyzos. Pricing differentiated services in the GPRS environment. *Wireless Networks*, 9(4):331–339, 2003.

W. Stallings. *High Speed Networks: TCP/IP and ATM Design Principles*. Prentice-Hall, 1998.

A. Steyn, C. Smith, S. du Toit, and C. Strasheim. *Modern Statistics in Practice*. Van Schaik, 1994.

B. Stiller, J. Gerke, P. Reichl, and P. Flury. Management of differentiated services usage by the Cumulus pricing scheme and a generic Internet charging system. In *Integrated Network Management Proceedings*, pages 93–106. IEEE/IFIP, 2001.

H. Su and M. Atiquzzaman. ItswTCM: a new aggregate marker to improve fairness in DiffServ. In *Proceedings of GLOBECOM '01.*, volume 3, pages 1841–1846. IEEE, 2001.

M. S. Taqqu, W. Willinger, and R. Sherman. Proof of a fundamental result in self-similar traffic modeling. *ACM SIGCOMM Computer Communications Review*, 27 (2):5–23, Apr 1997.

K. Thompson, G. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11(6):10–23, Nov/Dec 1997.

W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, Feb 1997.

W. Willinger, V. Paxson, and M. S. Taqqu. *Self-Similarity and Heavy Tails: Structural Modeling of Network Traffic*, pages 27–53. Birkhauser, Boston, 1998.

W. Willinger, V. Paxson, R. H. Riedi, and M. S. Taqqu. Long-range dependence and data network traffic. In P. Doukhan, G. Oppenheim, and M. S. Taqqu, editors, *Theory and Applications of Long-range Dependence*. Birkhäuser, 2002.

I. Yeom and A. Reddy. Realizing throughput guarantees in a differentiated services network. In *Proceedings of the International Conference on Multimedia Computing and Systems*, volume 2, pages 372–376. IEEE, Jul 1999a.

I. Yeom and A. L. N. Reddy. Impact of marking strategy on aggregated flows in a differentiated services network. In *Proceedings of IWQoS '99*, pages 156–158, 1999b.

A. Statistical Test Output

> Statistics is like a bikini. What they reveal is suggestive, what they conceal is vital.
>
> *Aaron Levenstein*

This Appendix includes the numerical results of the statistical tests referred to in Chapter 6.

Section A.1 contains the results of the Kruskal-Wallis tests for equality in the observed TargetRatio medians. Section A.2 contains the results of the Kruskal-Wallis test for equality in the GreenRatio medians, as well as the results of the Wilcoxon pair-wise comparisons for the different traffic conditioners. Lastly, Section A.3 presents the pair-wise comparisons of the traffic conditioners using the $|1 - \text{GreenRatio}_i|$ performance measure.

A.1. Target Rate

This section contains the p-values obtained for the statistical tests described in Section 5.1.

Note that η_k for $k = 1, 2, \ldots, 5$ is obtained by determining the median of 100 observations. Each TargetRatio_i with $i = 1, 2, \ldots, 10$ (corresponding to source i) was recorded in each of ten repetitions of a given reservation level, traffic model, and traffic conditioner simulation scenario. The median of these 100 TargetRatio values was then determined and assigned to η_k for each of the five traffic conditioners.

The following table contains the p-values obtained when testing $H_0 : \eta_1 = \cdots = \eta_5$ against H_1 : at least one η_k differs from the rest, using the Kruskal-Wallis non-parametric test for every reservation level and traffic model combination.

	CBR	POISSON	EXPOO	PAROO	H=0.5	H=0.7	H=0.95
125%	0.9887	0.9265	0.8092	0.917	0.9964	0.9906	0.8654
90%	0.9856	0.9759	0.8831	0.9035	0.9942	0.9943	0.9544
60%	0.9484	6.5×10^{-5}	0.9546	0.9434	0.1962	0.8635	0.991

A.2. Reserved Rate

This section contains the observed p-values for the tests described in Section 5.2.

The following table contains the p-values obtained when testing $H_0 : \eta_1 = \cdots = \eta_5$ against H_1 : at least one η_k differs from the rest. The η_k values are obtained from the observed GreenRatio values.

	CBR	POISSON	EXPOO	PAROO	H=0.5	H=0.7	H=0.95
125%	0.6692	0.2606	2×10^{-16}	2×10^{-16}	0.5576	0.4415	0.8373
90%	2×10^{-16}	2×10^{-16}	2×10^{-16}	2×10^{-16}	2×10^{-16}	3×10^{-16}	2×10^{-16}
60%	2×10^{-16}	2×10^{-16}	2×10^{-16}	2×10^{-16}	2×10^{-16}	2×10^{-16}	2×10^{-16}

Pairwise comparisons

Since the results of the tests in the preceding section showed that the null hypotheses were rejected, it was decided to also perform pair-wise comparisons using one-sided alternatives.

The R output for the Wilcoxon non-parametric test is show here for every reservation level and traffic model combination. For each combination, the median of the observed GreenRatios obtained by each traffic conditioner is compared to all the others. H_0 : $\eta_m = \eta_n$ against $H_1 : \eta_m < \eta_n$ and $H_1 : \eta_m > \eta_n$.

Scenario: 125%, CBR

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

          TSW2CM TSW3CM TBM srTCM
TSW3CM 1.0000000     NA  NA    NA
TBM    1.0000000      1  NA    NA
srTCM  1.0000000      1   1    NA
trTCM  0.8791178      1   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 125%, POISSON

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

          TSW2CM    TSW3CM       TBM     srTCM
TSW3CM 0.9769227        NA        NA        NA
TBM    0.5175291 0.8501825        NA        NA
srTCM  0.5175291 0.8501825 0.9769227        NA
trTCM  0.2762210 0.6005337 0.9769227 0.9769227

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 125%, EXPOO

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM       TBM      srTCM
TSW3CM 3.446669e-01           NA        NA         NA
TBM    1.280639e-33 1.280639e-33        NA         NA
srTCM  1.280639e-33 1.280639e-33 0.8689665         NA
trTCM  1.280639e-33 1.280639e-33 0.3446669 0.06807192

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 125%, PAROO

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM        TBM srTCM
```

```
TSW3CM 7.020442e-01           NA          NA     NA
TBM    2.454014e-21 1.209107e-19          NA     NA
srTCM  7.890380e-26 9.627942e-24 0.05715339     NA
trTCM  5.059784e-22 3.322775e-20 1.00000000      1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM      srTCM
TSW3CM      1     NA  NA         NA
TBM         1      1  NA         NA
srTCM       1      1   1         NA
trTCM       1      1   1 0.05260236
```

Scenario: 125%, H=0.5

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

         TSW2CM TSW3CM TBM srTCM
TSW3CM 1.000000     NA  NA    NA
TBM    1.000000      1  NA    NA
srTCM  1.000000      1   1    NA
trTCM  0.608138      1   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 125%, H=0.7

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

          TSW2CM   TSW3CM TBM srTCM
TSW3CM 1.0000000       NA  NA    NA
TBM    1.0000000 1.000000  NA    NA
srTCM  1.0000000 1.000000   1    NA
trTCM  0.6990222 0.495357   1     1
```

```
> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 125%, H=0.95

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 90%, CBR

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM          TBM        srTCM
TSW3CM 7.663663e-01           NA           NA           NA
TBM    1.208080e-11 1.469377e-14           NA           NA
srTCM  1.262196e-11 1.301051e-14 5.351080e-03           NA
trTCM  1.262196e-11 3.383186e-15 1.852343e-23 2.838536e-17

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
```

```
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 90%, POISSON

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM          TBM        srTCM
TSW3CM 6.430115e-01           NA           NA           NA
TBM    7.092247e-10 3.771109e-09           NA           NA
srTCM  7.092247e-10 3.633638e-09 3.710865e-02           NA
trTCM  7.092247e-10 7.357052e-10 2.902657e-24 1.796339e-20

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 90%, EXPOO

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM TBM     srTCM
TSW3CM 3.050794e-01           NA  NA        NA
TBM    1.279918e-33 1.279918e-33  NA        NA
srTCM  1.279918e-33 1.279918e-33   1        NA
trTCM  1.279918e-33 1.279918e-33   1 0.5212212

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM       TBM srTCM
TSW3CM      1     NA        NA    NA
TBM         1      1        NA    NA
srTCM       1      1 0.9681964    NA
trTCM       1      1 1.0000000     1
```

Scenario: 90%, PAROO

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM TBM srTCM
TSW3CM 5.790070e-01           NA  NA    NA
TBM    1.141075e-26 6.194749e-25  NA    NA
srTCM  1.615798e-26 6.194749e-25   1    NA
trTCM  4.015217e-25 3.508913e-23   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 90%, H=0.5

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM      TBM    srTCM
TSW3CM 4.278047e-03           NA       NA       NA
TBM    2.386076e-13 9.300351e-07       NA       NA
srTCM  2.386076e-13 9.300351e-07 0.612727       NA
trTCM  2.002627e-13 8.417836e-07 0.612727 0.612727

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 90%, H=0.7

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM       TBM    srTCM
TSW3CM 6.778930e-01           NA        NA       NA
TBM    1.319524e-07 1.940359e-07        NA       NA
```

```
srTCM  3.732699e-08 6.437225e-08 0.4567462       NA
trTCM  1.597750e-08 3.838275e-08 0.1341487 0.488273

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 90%, H=0.95

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM       TBM    srTCM
TSW3CM 6.097720e-02           NA        NA       NA
TBM    8.315006e-15 8.988311e-08        NA       NA
srTCM  9.135949e-16 8.752855e-09 0.1651983       NA
trTCM  1.307436e-15 2.026649e-08 0.1521396 0.446039

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 60%, CBR

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM          TBM        srTCM
TSW3CM 9.064936e-01           NA           NA           NA
TBM    3.426394e-31 1.378966e-33           NA           NA
srTCM  1.978227e-31 7.006966e-34 4.718366e-06           NA
trTCM  7.212079e-33 1.803613e-35 2.447070e-19 3.958787e-08

> pairwise.wilcox.test(Green, Policer, alternative='g')
```

```
          TSW2CM TSW3CM TBM srTCM
TSW3CM 0.9391527     NA  NA    NA
TBM    1.0000000      1  NA    NA
srTCM  1.0000000      1   1    NA
trTCM  1.0000000      1   1     1
```

Scenario: 60%, POISSON

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM        TBM      srTCM
TSW3CM 6.548135e-01           NA         NA         NA
TBM    5.163735e-34 1.303126e-31         NA         NA
srTCM  5.392052e-34 1.303126e-31 0.65481353         NA
trTCM  5.392052e-34 1.303126e-31 0.03025202 0.09524705

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 60%, EXPOO

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM       TBM srTCM
TSW3CM 1.000000e+00           NA        NA    NA
TBM    1.280350e-33 1.280350e-33        NA    NA
srTCM  1.280350e-33 1.280350e-33 0.6937095    NA
trTCM  1.280350e-33 1.280350e-33 0.9861323     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 60%, PAROO

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM      TBM    srTCM
TSW3CM 6.399965e-01           NA       NA       NA
TBM    2.900778e-28 1.824292e-29       NA       NA
srTCM  3.225626e-28 2.113602e-29 0.743137       NA
trTCM  3.681015e-28 2.113602e-29 0.743137 0.743137

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 60%, H=0.5

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM       TBM     srTCM
TSW3CM 9.405911e-01           NA        NA        NA
TBM    1.250616e-33 1.250616e-33        NA        NA
srTCM  1.250616e-33 1.250616e-33 0.9405911        NA
trTCM  1.250616e-33 1.250616e-33 0.5901403 0.4209981

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 60%, H=0.7

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM       TBM     srTCM
```

```
TSW3CM 5.549017e-01           NA         NA         NA
TBM    1.020246e-32 7.922377e-33         NA         NA
srTCM  1.020246e-32 7.922377e-33 0.4912120         NA
trTCM  1.020246e-32 7.922377e-33 0.2653483 0.5418764

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 60%, H=0.95

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

              TSW2CM       TSW3CM       TBM     srTCM
TSW3CM 3.532438e-01           NA        NA        NA
TBM    4.541217e-33 1.856152e-31        NA        NA
srTCM  4.142049e-33 1.856152e-31 0.5365824        NA
trTCM  4.142049e-33 1.560791e-31 0.3603069 0.5365824

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

A.3. Green Differences

This section contains the pair-wise comparisons for the medians of the $|1-\text{GreenRatio}_i|$ observations.

Pairwise comparisons

Scenario: 125%, CBR

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

          TSW2CM TSW3CM TBM srTCM
TSW3CM 1.0000000     NA  NA    NA
TBM    1.0000000      1  NA    NA
srTCM  1.0000000      1   1    NA
trTCM  0.8791178      1   1     1
```

Scenario: 125%, POISSON

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

          TSW2CM    TSW3CM       TBM     srTCM
TSW3CM 0.9769227        NA        NA        NA
TBM    0.5175291 0.8501825        NA        NA
srTCM  0.5175291 0.8501825 0.9769227        NA
trTCM  0.2762210 0.6005337 0.9769227 0.9769227
```

Scenario: 125%, EXPOO

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
```

```
srTCM       1       1   1    NA
trTCM       1       1   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

          TSW2CM TSW3CM       TBM      srTCM
TSW3CM 0.3446669     NA        NA         NA
TBM    0.0000000      0        NA         NA
srTCM  0.0000000      0 0.8689665         NA
trTCM  0.0000000      0 0.3446669 0.06807192
```

Scenario: 125%, PAROO

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

       TSW2CM TSW3CM TBM      srTCM
TSW3CM      1     NA  NA         NA
TBM         1      1  NA         NA
srTCM       1      1   1         NA
trTCM       1      1   1 0.05260236

> pairwise.wilcox.test(Green, Policer, alternative='g')

          TSW2CM TSW3CM        TBM srTCM
TSW3CM 0.7020442     NA         NA    NA
TBM    0.0000000      0         NA    NA
srTCM  0.0000000      0 0.05715339    NA
trTCM  0.0000000      0 1.00000000     1
```

Scenario: 125%, H=0.5

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')
```

```
         TSW2CM TSW3CM TBM srTCM
TSW3CM 1.000000     NA  NA    NA
TBM    1.000000      1  NA    NA
srTCM  1.000000      1   1    NA
trTCM  0.608138      1   1     1
```

Scenario: 125%, H=0.7

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

          TSW2CM   TSW3CM TBM srTCM
TSW3CM 1.0000000       NA  NA    NA
TBM    1.0000000 1.000000  NA    NA
srTCM  1.0000000 1.000000   1    NA
trTCM  0.6990222 0.495357   1     1
```

Scenario: 125%, H=0.95

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 90%, CBR

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM TBM srTCM
TSW3CM 1.000000e+00           NA  NA    NA
TBM    2.808562e-34 1.554442e-32  NA    NA
srTCM  5.678098e-34 2.890697e-32   1    NA
trTCM  1.813177e-33 2.547105e-31   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM          TBM        srTCM
TSW3CM      1     NA           NA           NA
TBM         1      1           NA           NA
srTCM       1      1 1.000000e+00           NA
trTCM       1      1 2.746799e-06 1.404626e-06
```

Scenario: 90%, POISSON

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM TBM srTCM
TSW3CM 1.000000e+00           NA  NA    NA
TBM    9.727182e-34 9.702627e-33  NA    NA
srTCM  1.292295e-33 2.000972e-32   1    NA
trTCM  3.736138e-31 1.183326e-29   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM       TBM srTCM
TSW3CM      1     NA        NA    NA
TBM         1      1        NA    NA
srTCM       1      1 0.9565626    NA
trTCM       1      1 0.0000000     0
```

Scenario: 90%, EXPOO

```
> pairwise.wilcox.test(Green, Policer, alternative='l')
```

```
       TSW2CM TSW3CM       TBM srTCM
TSW3CM      1     NA        NA    NA
TBM         1      1        NA    NA
srTCM       1      1 0.9681964    NA
trTCM       1      1 1.0000000     1
```

```
> pairwise.wilcox.test(Green, Policer, alternative='g')

          TSW2CM TSW3CM TBM     srTCM
TSW3CM 0.3050794     NA  NA        NA
TBM    0.0000000      0  NA        NA
srTCM  0.0000000      0   1        NA
trTCM  0.0000000      0   1 0.5212212
```

Scenario: 90%, PAROO

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

```
> pairwise.wilcox.test(Green, Policer, alternative='g')

         TSW2CM TSW3CM TBM srTCM
TSW3CM 0.579007     NA  NA    NA
TBM    0.000000      0  NA    NA
srTCM  0.000000      0   1    NA
trTCM  0.000000      0   1     1
```

Scenario: 90%, H=0.5

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM TBM srTCM
TSW3CM 4.481111e-02           NA  NA    NA
TBM    1.148290e-23 3.207460e-21  NA    NA
srTCM  2.817182e-23 3.881235e-21   1    NA
trTCM  4.436877e-23 6.169923e-21   1     1
```

```
> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 90%, H=0.7

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM TBM srTCM
TSW3CM 1.000000e+00           NA  NA    NA
TBM    7.974370e-15 2.537896e-21  NA    NA
srTCM  8.987581e-13 6.878630e-19   1    NA
trTCM  7.035437e-12 3.322208e-18   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

          TSW2CM TSW3CM       TBM srTCM
TSW3CM 0.8638086     NA        NA    NA
TBM    1.0000000      1        NA    NA
srTCM  1.0000000      1 1.0000000    NA
trTCM  1.0000000      1 0.3353719     1
```

Scenario: 90%, H=0.95

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM TBM srTCM
TSW3CM 5.657343e-01           NA  NA    NA
TBM    1.196993e-15 6.564283e-15  NA    NA
srTCM  3.408095e-13 4.923613e-12   1    NA
trTCM  5.735375e-14 2.891996e-13   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM       TBM srTCM
TSW3CM      1     NA        NA    NA
```

```
TBM         1      1        NA    NA
srTCM       1      1 1.0000000    NA
trTCM       1      1 0.5465455     1
```

Scenario: 60%, CBR

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM           TBM        srTCM
TSW3CM 9.069011e-01           NA            NA           NA
TBM    8.625561e-34 5.520155e-35            NA           NA
srTCM  4.675199e-34 2.758198e-35 4.718366e-06           NA
trTCM  1.624741e-35 6.845221e-37 2.447070e-19 3.958787e-08

> pairwise.wilcox.test(Green, Policer, alternative='g')

          TSW2CM TSW3CM TBM srTCM
TSW3CM 0.9350642     NA  NA    NA
TBM    1.0000000      1  NA    NA
srTCM  1.0000000      1   1    NA
trTCM  1.0000000      1   1     1
```

Scenario: 60%, POISSON

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM         TBM     srTCM
TSW3CM 6.584963e-01           NA          NA        NA
TBM    5.163735e-34 7.012082e-33          NA        NA
srTCM  5.392052e-34 7.012082e-33 0.65849627        NA
trTCM  5.392052e-34 7.012082e-33 0.02798909 0.0873892

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 60%, EXPOO

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM       TBM srTCM
TSW3CM      1     NA        NA    NA
TBM         0      0        NA    NA
srTCM       0      0 0.6937095    NA
trTCM       0      0 0.9861323     1
```

Scenario: 60%, PAROO

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

          TSW2CM TSW3CM      TBM    srTCM
TSW3CM 0.6399965     NA       NA       NA
TBM    0.0000000      0       NA       NA
srTCM  0.0000000      0 0.743137       NA
trTCM  0.0000000      0 0.743137 0.743137
```

Scenario: 60%, H=0.5

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM TBM srTCM
```

```
TSW3CM 1.000000e+00           NA  NA    NA
TBM    1.689337e-33 1.932149e-32  NA    NA
srTCM  1.689337e-33 1.932149e-32   1    NA
trTCM  2.061298e-33 1.932149e-32   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM TBM srTCM
TSW3CM      1     NA  NA    NA
TBM         1      1  NA    NA
srTCM       1      1   1    NA
trTCM       1      1   1     1
```

Scenario: 60%, H=0.7

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM TBM srTCM
TSW3CM 1.000000e+00           NA  NA    NA
TBM    2.574466e-33 9.116207e-33  NA    NA
srTCM  2.574466e-33 9.270442e-33   1    NA
trTCM  2.574466e-33 9.270442e-33   1     1

> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM       TBM srTCM
TSW3CM      1     NA        NA    NA
TBM         1      1        NA    NA
srTCM       1      1 1.0000000    NA
trTCM       1      1 0.6385444     1
```

Scenario: 60%, H=0.95

```
> pairwise.wilcox.test(Green, Policer, alternative='l')

             TSW2CM       TSW3CM TBM srTCM
TSW3CM 3.740277e-01           NA  NA    NA
TBM    1.194039e-29 2.889738e-30  NA    NA
srTCM  4.312726e-29 8.133034e-30   1    NA
trTCM  6.472906e-29 2.485563e-29   1     1
```

```
> pairwise.wilcox.test(Green, Policer, alternative='g')

       TSW2CM TSW3CM       TBM srTCM
TSW3CM      1     NA        NA    NA
TBM         1      1        NA    NA
srTCM       1      1 1.0000000    NA
trTCM       1      1 0.8045312     1
```